SGOU
Higher Education for All

NATURAL LANGUAGE PROCESSING

Course Code: B24DS04DE
BSc Data Science and Analytics
Discipline Specific Elective Course
Self Learning Material



SREENARAYANAGURU OPEN UNIVERSITY

SREENARAYANAGURU OPEN UNIVERSITY

The State University for Education, Training and Research in Blended Format, Kerala

SREENARAYANAGURU OPEN UNIVERSITY

SGOU

Vision

To increase access of potential learners of all categories to higher education, research and training, and ensure equity through delivery of high quality processes and outcomes fostering inclusive educational empowerment for social advancement.

Mission

To be benchmarked as a model for conservation and dissemination of knowledge and skill on blended and virtual mode in education, training and research for normal, continuing, and adult learners.

Pathway

Access and Quality define Equity.

Natural Language Processing

Course Code: B24DS04DE

Semester - V

Discipline Specific Elective Course
Undergraduate Programme
BSc Data Science and Analytics
Self Learning Material
(With Model Question Paper Sets)

SGOU

SREENARAYANAGURU
OPEN UNIVERSITY

SREENARAYANAGURU OPEN UNIVERSITY

The State University for Education, Training and Research in Blended Format, Kerala

SREENARAYANAGURU
OPEN UNIVERSITY

NATURAL LANGUAGE PROCESSING

Course Code: B24DS04DE
Semester - V
Discipline Specific Elective Course
BSc Data Science and Analytics

Academic Committee

Dr. T. K. Manoj
Dr. Smitha Dharan
Dr. Satheesh S.
Dr. Vinod Chandra S.S.
Dr. Hari V. S.
Dr. Sharon Susan Jacob
Dr. Ajith Kumar R.
Dr. Smiju I.S.
Dr. Nimitha Aboobaker

Development of the Content

Alaka Krishnan R.U.

Review

Dr. Leena Silvoster M.

Edit

Dr. Leena Silvoster M.

Linguistics

Sujith Mohan

Scrutiny

Greeshma P.P.
Dr. Kanitha D.K.
Sreerekha V.K.
Shamin S.
Subi Priya Laxmi S.B.N.
Aswathy V.S.

Cover Design

Jobin J.

Co-ordination

Director, MDDC :
Dr. I.G. Shibi
Asst. Director, MDDC :
Dr. Sajeevkumar G.
Coordinator, Development:
Dr. Anfal M.
Coordinator, Distribution:
Dr. Sanitha K.K.

Scan this QR Code for reading the SLM on a digital device.

Edition
October 2026

Copyright
© Sreenarayanaguru Open University

ISBN 978-81-180900-6-9

9 788118 090069

SGOU

All rights reserved. No part of this work may be reproduced in any form, by mimeograph or any other means, without permission in writing from Sreenarayanaguru Open University. Printed and published on behalf of Sreenarayanaguru Open University by Registrar, SGOU, Kollam.

www.sgou.ac.in

Visit and Subscribe our Social Media Platforms

MESSAGE FROM VICE CHANCELLOR

Dear learner,

I extend my heartfelt greetings and profound enthusiasm as I warmly welcome you to Sreenarayanaguru Open University. Established in September 2020 as a state-led endeavour to promote higher education through open and distance learning modes, our institution was shaped by the guiding principle that access and quality are the cornerstones of equity. We have firmly resolved to uphold the highest standards of education, setting the benchmark and charting the course.

The courses offered by the Sreenarayanaguru Open University aim to strike a quality balance, ensuring students are equipped for both personal growth and professional excellence. The University embraces the widely acclaimed "blended format," a practical framework that harmoniously integrates Self-Learning Materials, Classroom Counseling, and Virtual modes, fostering a dynamic and enriching experience for both learners and instructors.

The University is committed to providing an engaging and dynamic educational environment that encourages active learning. The Study and Learning Material (SLM) is specifically designed to offer you a comprehensive and integrated learning experience, fostering a strong interest in exploring advancements in information technology (IT). The curriculum has been carefully structured to ensure a logical progression of topics, allowing you to develop a clear understanding of the evolution of the discipline. It is thoughtfully curated to equip you with the knowledge and skills to navigate current trends in IT, while fostering critical thinking and analytical capabilities.The Self-Learning Material has been meticulously crafted, incorporating relevant examples to facilitate better comprehension.

Rest assured, the university's student support services will be at your disposal throughout your academic journey, readily available to address any concerns or grievances you may encounter. We encourage you to reach out to us freely regarding any matter about your academic programme. It is our sincere wish that you achieve the utmost success.



Regards,
Dr. Jagathy Raj V. P.

01-10-2026

SGOU

Contents

Block 01	**Introduction to NLP**	**1**
Unit 1	Foundations of NLP	2
Unit 2	NLP Building Blocks	15
Unit 3	Core NLP Tasks	30
Unit 4	Approaches to NLP	44
Block 21	**Pre-processing and Representation Models**	**58**
Unit 1	NLP System Pipeline	59
Unit 2	Text Preprocessing for NLP	76
Unit 3	Text Representation	95
Unit 4	Text Vectorization	109
Block 03	**Classification and Information Extraction**	**131**
Unit 1	Text Classification	132
Unit 2	Designing Text Classifiers	151
Unit 3	Text Classification Algorithms	169
Unit 4	Information Extraction (IE)	191
Block 04	**Data Integration, Data Quality Assessment**	**214**
Unit 1	Relation Detection and Classification	215
Unit 2	Lightly Supervised Approaches to Relation Analysis	233
Unit 3	Evaluation of Relation Analysis Systems	250
Unit 4	Information Retrieval	267
Block 05	**Pre-processing and Representation Models**	**289**
Unit 1	Question Answering Systems	290
Unit 2	Answer Processing	304
Unit 3	Translation Techniques	325
Unit 4	Symmetrizing Alignments for Phrase-Based MT	343
Model Question Paper Sets		**357**

SGOU



Introduction to NLP

SGOU

Unit 1
Foundations of NLP

Learning Outcomes

After completing this unit, learner will able to:

- define Natural Language Processing and explain its significance in modern computing
- identify major NLP tasks and describe real-world applications based on them
- distinguish between natural language and computer language
- explain unstructured data and discuss why it poses challenges for traditional computational systems
- describe the need for specialized techniques for processing natural language

Prerequisites

This introductory unit assumes no prior knowledge of Natural Language Processing (NLP). This unit does not require a background in programming or advanced mathematics. Basic computer literacy is sufficient. Familiarity with tools such as search engines, email, and voice assistants is helpful. It is also advisable to learn basic Python for practical NLP applications, as many NLP tools involve coding and testing.

An elementary understanding of different kinds of data, how computers follow instructions, and how they recognize patterns helps when learning NLP. This unit introduces the foundations of NLP, explains the differences between natural and computer languages, and discusses the complexities of processing unstructured text.

Keywords

Natural Language Processing (NLP), Unstructured Data, Machine Learning, Artificial Intelligence, Computer Language, Corpus

SGOU



Discussion

"Language is the mirror of the mind and NLP is the science of reading that mirror."

By Daniel Jurafsky

1.1.1 Introduction to NLP

Think about the last time you asked your phone a question. Maybe you said *"Hey Siri, set an alarm for 7 am"* or typed *"best pizza near me"* into a search bar. Within seconds, you got exactly what you needed. There was no form to fill in, and no special commands to learn- just plain, everyday language, and the machine understood.

Figure 1.1.1 illustrates this kind of everyday interaction. It may seem simple, but it is not. Human language is one of the most complex forms of information computers must process.

Language is how we share ideas, learn from each other, and connect with the world. Natural languages such as English, Malayalam, Hindi, or French are full of complexity, variation, and ambiguity. They don't follow the strict and well-defined rules of programming languages. Instead, natural languages allow meanings to change depending on context, flexible grammatical structures, and multiple ways to express the same idea. That's why it's so tough to teach computers to understand how we talk. Unlike structured data, often unstructured and highly context-dependent. Even a single sentence can mean different meanings depending on how it is expressed, who says it, and the context of the preceding conversation.

Thanks to breakthroughs in Artificial Intelligence (AI), Machine Learning (ML), and computational methods for language processing, machines have become much better at understanding human speech and text. The field that makes this possible is called Natural

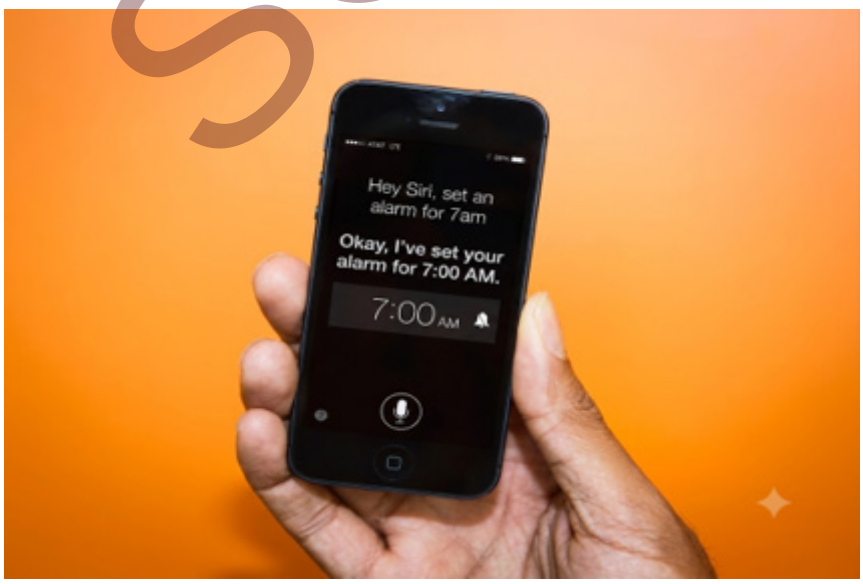


Fig 1.1.1 Example of Natural Language Processing in a virtual assistant responding to a spoken user command

SGOU

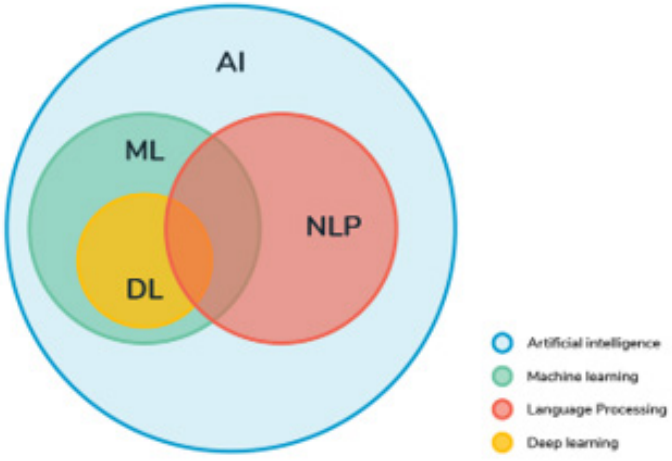


Fig 1.1.2 AI and Subdivisions

Language Processing (NLP). It combines ideas from computer science, linguistics, and data science to enable computers to process, understand, interpret, and generate human language, as illustrated in Figure 1.1.2.

Today, NLP is integrated into many applications that we use in everyday life, including search engines, digital assistants, translation applications, chatbots, spam filters, and automatic text summarization tools. In this unit, you'll learn the basics of NLP, its major tasks and real-world applications, the differences between natural and computer languages, and the role of different types of data in language processing. Understanding these concepts provides a strong foundation for studying more advanced NLP techniques in later units.

1.1.1.1 What is NLP?

Natural Language Processing (NLP) is a branch of artificial intelligence that enables computers to understand, interpret, analyze, generate, and respond to human language in a meaningful and intelligent manner. It combines concepts from computer science, linguistics, and machine learning to bridge the communication gap between humans and computers. Figure 1.1.2 illustrates the relationship of Artificial Intelligence (AI) and its related fields. NLP is a subfield of AI and often uses Machine Learning (ML) and Deep Learning (DL) techniques to process and understand human language.

1.1.2 Key Characteristics and Capabilities of NLP

Natural language processing differs from other areas of computing and data processing in several ways. Unlike traditional text processing, which treats words as isolated units, NLP considers grammatical structure, semantic relationships, and contextual information to interpret language more effectively. The key characteristics and capabilities of NLP are described below.

a. Human Language Understanding

NLP enables computers to process and interpret text and speech expressed in natural

SGOU

languages. It identifies words, phrases, grammar, and context to determine the intended meaning.

b. Language Generation

NLP enables computers to generate meaningful and grammatically appropriate human language. This capability is used in applications such as chatbots, virtual assistants, and automated report generation.

c. Context Awareness

The meaning of a word or expression often depends on its context. NLP techniques analyze surrounding words and sentences to determine the appropriate interpretation

d. Multilingual Processing

Modern NLP systems can process multiple languages, supporting applications such as machine translation, cross-lingual communication, and multilingual information retrieval.

e. Learning from Data

Many NLP systems use machine learning and deep learning algorithms to learn patterns from large text collections and improve performance on language-related tasks.

f. Ambiguity Resolution

Natural language is often ambiguous. NLP techniques use linguistic information, contextual cues, and statistical or machine learning models to determine the most appropriate meaning of words and sentences.

g. Real -Time Processing

Many NLP applications process text or speech in real time, providing immediate responses to user queries. Examples include voice assistants, live transcription systems, and customer-support chatbots.

1.1.3 Key NLP Tasks

Natural Language Processing includes a wide range of tasks that enable computers to process, interpret, and generate human language. Rather than examining words only as isolated units, NLP considers their grammatical structure, meaning, and context. The major NLP tasks are described below.

a. Text Classification

Text classification is the task of automatically assigning a predefined category or label to a piece of text based on its content. It is widely used in applications such as spam detection, topic classification, language identification, and intent recognition.

Examples include:

i. **Sentiment classification:** Determining whether a review expresses a positive, negative, or neutral opinion.

SGOU

ii. **Spam detection:** Classifying an email as spam or non-spam.

iii. **Language identification:** Determining the language in which a text is written.

iv. **Intent classification:** Identifying the purpose of a user's query or message.

b. Sentiment Analysis

Sentiment analysis identifies the opinions, attitudes, or emotions expressed in text. It is commonly used to determine whether a piece of text conveys a positive, negative, or neutral sentiment. Applications include customer-review analysis, social media monitoring, and survey analysis.

c. Machine Translation

Machine translation automatically converts text or speech from one language into another while preserving its meaning as accurately as possible. Modern machine translation systems consider grammatical structure, semantic information, and context to produce fluent and meaningful translations.

d. Named Entity Recognition (NER)

Named Entity Recognition (NER) is the process of identifying and classifying important entities in text into predefined categories such as persons, organizations, locations, dates, times, and monetary values.

e. Text Summarization

Text summarization produces a shorter version of a document while preserving its main ideas and important information. It helps users understand lengthy content quickly without reading the entire document.

f. Generating Text Content

Text generation is the task of generating meaningful, coherent text from a given input or prompt. It is used in conversational AI, content creation, email drafting, and automatic report generation.

g. Question Answering Systems

Question-answering systems interpret a user's question and provide an appropriate answer by retrieving information from relevant sources or generating a response based on available knowledge. These systems are commonly used in search engines, virtual assistants, chatbots, and educational applications.

h. Speech Recognition and Generation

Speech recognition converts spoken language into written text (speech-to-text), and speech generation converts written text into natural - sounding speech (text - to - speech). These technologies enable voice interaction between humans and computers.

1.1.4 Real World Applications

NLP has become an integral part of our daily lives, often working behind the scenes

SGOU

in the digital services we use. It enables computers to process, interpret, and generate human language efficiently. Below are some common real-world applications of NLP.

a. Chatbots and Virtual assistants

Many businesses use NLP-powered chatbots and virtual assistants to interact with users and provide instant support. These chatbots can understand user queries, answer frequently asked questions, provide recommendations, and assist with routine tasks. This reduces response time and improves user experience.

Example: Siri, Alexa, and Google Assistant use speech recognition and NLP to interpret spoken commands and generate appropriate responses.

b. Email spam filters

NLP can analyze email content, including words, phrases, linguistic patterns, and context, to help determine whether a message is legitimate or spam. It is commonly combined with other information, such as sender characteristics and message metadata.

Example: Gmail uses automated classification techniques to identify and filter unwanted or suspicious emails.

c. Search Engines

Search engines use NLP to understand the meaning and intent behind user queries. Instead of relying only on exact keyword matching, modern search systems consider context and semantic relationships to provide more relevant results.

Example: When a user searches for "best restaurants near me," the search engine interprets the user's intent and location to provide suitable restaurant recommendations.

d. Language Translation

Language translation in NLP, often called Machine Translation (MT), is the automated process of converting text or speech from one human language into another while attempting to preserve the original meaning and context.

Example: Google Translate, iTranslate, DeepL

e. Social media monitoring

NLP tools can analyze social media posts, comments, reviews, and discussions to identify opinions and public sentiment toward products, services, brands, or events. Organizations can use these insights to understand customer feedback and public perception.

Example: Companies can analyze customer comments on social media to determine whether reactions to a newly launched product are positive, negative, or neutral.

f. Healthcare

Healthcare professionals use NLP to process electronic health records, clinical notes,

SGOU

Table 1.1.1: Comparison between Natural Language and Computer Language

Natural Language (NL)	Computer Language (CL)
Used by humans for communication	Used to provide instructions to computers
Primarily intended for human communication	Primarily used by programmers to develop computer programs
Example: English, Malayalam, Hindi	Example: Python, Java, C++
Rules can be flexible and evolve over time	Follows strict and predefined syntax rules
Words or sentences may have multiple meanings	Statements are designed to have precise meanings
Meaning depends on context	Meaning is determined by defined syntax and semantics
Minor grammatical errors can often be understood by humans	Syntax errors may prevent successful program execution
Used for expressing ideas, emotions, and information	Used for Programming and software development

medical reports, and research articles. It helps extract patient information, identify diseases, and support clinical decision-making.

Example: NLP systems can summarize lengthy patient records to help healthcare professionals quickly identify important clinical information.

g. Document Processing and Information Extraction

NLP can be combined with Optical Character Recognition (OCR) to process scanned documents and images containing text. OCR first converts visible text into machine-readable form, after which NLP techniques can analyze, classify, translate, or extract information from the resulting text.

Example: An application can use OCR to extract text from an image and then use machine translation to translate it into another language.

1.1.5 Natural Language vs. Computer Language

Natural language and computer language are fundamentally different, with each serving a distinct purpose. People use natural language for everyday communication. It is flexible, expressive, and highly context-dependent. The meaning of a word or sentence often depends on the situation, the speaker's intention, tone, and previously shared information. Humans use contextual knowledge and experience to interpret such variations.

In contrast, computer language is a formal system designed to provide instructions

SGOU

to computers. It follows well-defined syntax and grammatical rules that allow precise interpretation. Even minor errors, such as a missing semicolon, an incorrect keyword, a missing symbol, or improper syntax, can prevent a program from executing correctly. Understanding the differences between natural and computer languages is important in Natural Language Processing (NLP), whose major objective is to bridge the gap between human communication and machine processing.

1.1.6 Data

Data is the fundamental input used by NLP systems to understand, analyse, and generate human language.

In NLP, a corpus (plural: corpora) is a large, organized collection of text or speech data used for linguistic analysis, training machine learning models, and evaluating NLP systems. A corpus may contain books, newspapers, websites, emails, social media posts, research articles, or spoken conversations.

Data can broadly be categorized into three types:

1.1.6.1 Structured Data

Structured data is data stored in a pre-defined format, usually organized into rows and columns with clearly defined fields.

Key Characteristics:

- **Predefined schema**: The structure of the data and the type of information stored in each field are defined in advance.
- **Consistent formatting**: Data follows a specific format, such as a standardized date format.
- **Ease of storage & querying:** Structured data can be efficiently stored in relational data-bases and retrieved using query languages such as SQL.

Example: Spreadsheets (Excel /Google sheets)

1.1.6.2 Semi-Structured Data

Semi-structured data has some organizational structure through tags, labels, or key-value pairs but does not follow the rigid format of structured databases.

Key Characteristics:

- **Flexible Schema**: Data does not require a fixed table structure.
- **Self- Describing**: It contains metadata, tags, or labels that provide information about the data.
- **Human Readability:** Many semi-structured formats are text-based and can be understood without specialized tools.

Example: XML documents, HTML pages, JSON files

SGOU

1.1.6.3 Unstructured Data

Unstructured data does not follow a predefined format or schema. It represents information without a fixed organizational structure and requires specialized techniques to extract meaningful information.

In NLP, much data is unstructured, including text, speech, and multimedia content.

Examples:

Emails, Word documents, PDF files, social media posts, images, videos, and chat messages.

Challenges of Unstructured data

Processing unstructured human language is challenging because natural language is complex, varied, and context-dependent.

- **Lack of Structure and Noisy Data**

Real-world text data is often noisy and inconsistent. It may contain spelling mistakes, abbreviations, slang, emojis, and informal expressions, which can reduce NLP accuracy.

- **Ambiguity**

Human language is ambiguous at different levels, including lexical ambiguity (multiple meanings of words), syntactic ambiguity (different sentence structures), and semantic ambiguity (different interpretations of meaning). Resolving these ambiguities requires contextual understanding.

- **Large Volume of Data**

The continuous generation of massive amounts of textual information through social media, customer feedback, news articles, and online platforms creates challenges in storage, processing, and analysis.

- **Scalability and Computational Cost**

Processing large volumes of text using modern machine learning and deep learning models requires significant computational resources, which can be challenging for small organizations and researchers.

- **Variability in Languages**

People express the same idea in different ways using different words, sentence structures, writing styles, and levels of formality. This variation makes it difficult for computers to interpret language consistently.

- **Language Diversity and Low-Resource Languages**

Languages such as English have large digital resources and well-developed NLP tools. However, many languages, including Malayalam and other Dravidian languages, lack sufficient annotated datasets and computational resources, limiting NLP performance.

SGOU

- **Multilingual Processing and Code-Mixing**

In multilingual communities, people often mix multiple languages within the same sentence or conversation. This code-mixing creates challenges for NLP systems trained primarily on a single language.

Example:

A Malayalam-English mixed sentence:

"Ente phone battery drain aavunnu"

("My phone battery is draining")

1.1.7 Why are specialized NLP techniques necessary?

Human language is complex, ambiguous, and highly context-dependent, making it difficult for traditional computer programs to process effectively. Natural Language Processing (NLP) employs specialized techniques and approaches to analyze language at different linguistic and computational levels. These techniques enable computers to identify patterns, understand contextual meaning, extract valuable information, and generate appropriate responses from unstructured language data. The following units explore the fundamental concepts of NLP, how these techniques work, and their practical applications.

Recap

- Natural Language Processing (NLP) is a branch of Artificial Intelligence (AI) that enables computers to understand, interpret, analyze, and generate human language.
- NLP combines concepts from computer science, linguistics, and machine learning to establish effective communication between humans and computers.
- Major NLP tasks, including sentiment analysis, machine translation, text summarization, speech recognition, and question answering, support the development of intelligent language-based applications.
- Data used in NLP can be categorized as structured, semi-structured, and unstructured.
- Since a large portion of human language exists as unstructured data, NLP techniques are essential for extracting meaningful information and insights from it.
- Natural language is flexible, context-dependent, and often ambiguous, whereas computer language follows strict syntax and predefined rules for communicating instructions to computers.

SGOU

- A corpus is a large and organized collection of text or speech data used for linguistic analysis, training NLP models, and evaluating their performance.
- Specialized NLP techniques transform unstructured language data into meaningful structured representations that can be effectively processed and analyzed by computers.
- Natural Language Processing (NLP) enables computers to understand, interpret, and process human language for various applications, such as language translation, question answering, and spam filtering.

Objective Type Questions

1. What does NLP stand for in the context of this course?
2. Which type of data has no predefined format or structure?
3. Which NLP task is primarily used in email spam filters?
4. Google Translate is a real-world application of which NLP task?
5. Give one example of programming and natural language.
6. Natural language follows strict syntax rules and cannot be ambiguous True/ False.
7. Which NLP application is used to answer user queries automatically?
8. What is the primary purpose of Optical Character Recognition (OCR)?
9. Documents stored in XML and JSON formats are examples of which kind of data?
10. A large and organized collection of text or speech data is called…….
11. Give two examples of unstructured data
12. What is sentiment analysis used for in NLP?
13. Which technology is used to convert spoken language into written text?
14. Which NLP application converts long articles into shorter summaries?
15. Which NLP technique helps computers understand the meaning of words based on their context?

SGOU

Answers to Objective Type Questions

1. Natural Language Processing
2. Unstructured data
3. Text Classification
4. Machine translation
5. Python, English
6. False
7. Chatbot or Question Answering System
8. To convert printed or handwritten text in images into machine-readable text
9. Semi-structured data
10. Corpus
11. Images, speech data
12. To determine the emotional tone of text
13. Speech recognition system
14. Text summarization
15. Word sense disambiguation

SGOU

Assignments

1. Describe NLP and five real-world applications of NLP
2. Distinguish between natural language and computer language with suitable examples.
3. What is unstructured data? Explain, with examples, the key challenges it poses for traditional data processing systems, and discuss why natural language is considered a form of unstructured data.
4. Explain the three types of data used in NLP. Structured, semi-structured, and unstructured, with examples of each. Why is most NLP research and applications focused on unstructured data?

Reference

1. D. Jurafsky and J. H. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, 2nd ed. Pearson Prentice Hall, 2009.

2. S. Vajjala, B. Majumder, A. Gupta, and H. Surana, *Practical Natural Language Processing: A Comprehensive Guide to Building Real-World NLP Systems*. O'Reilly Media, 2020.

3. DeepLearning.AI, "Natural Language Processing Resources," DeepLearning.AI. [Online]. Available: https://www.deeplearning.ai/resources/natural-language-processing [Accessed: 6 September 2026].

Suggested Reading

1. Oracle, "Natural Language Processing (NLP)," Oracle. [Online]. Available: https://www.oracle.com/uk/artificial-intelligence/natural-language-processing/

2. Hugging Face, "Introduction," *LLM Course*. [Online]. Available: https://huggingface.co/learn/llm-course/en/chapter1/1

3. NLP4SS, "Lecture 1," NLP4SS. [Online]. Available: https://nlp4ss.jeju.ai/en/session01/lecture1.html

SGOU

Unit 2
NLP Building Blocks

Learning Outcomes

After completing this unit, learner will able to:

- identify and explain the fundamental components of Natural Language Processing (NLP), including phonology, morphology, syntax, semantics, and pragmatics
- describe the specific role each linguistic level plays in facilitating language understanding
- explain the primary challenges associated with the computational processing of human language
- discuss how these foundational elements and challenges influence the design of real-world NLP systems

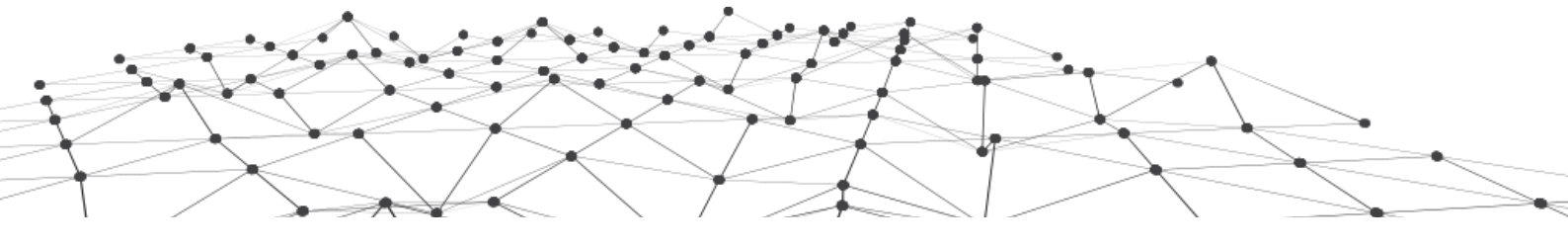
Prerequisites

Before a machine can effectively process human language, the language must be divided into manageable components, including sounds, words, structures, and meanings, which are then systematically analyzed. Understanding these building blocks is crucial to understanding how Natural Language Processing (NLP) systems transform raw text into structured representations. This unit extends the foundational concepts introduced in Unit 1, particularly the distinction between natural and computer languages and the characteristics of unstructured text data.

To engage effectively with this unit, learners should have a basic understanding of the definition of NLP, the various of NLP tasks and applications, and the general challenges associated with ambiguity and context dependency discussed earlier. With this foundational knowledge, learners will be able to explore the specific linguistic building blocks that NLP systems and examine the complex challenges with each linguistic level.

Keywords

Token, Corpus, Phonology, Morphology, Morpheme, Syntax, Semantics, Agglutinative Language, Unstructured Data



SGOU

Discussion

1.2.1 Introduction

Imagine asking your smartphone, *"What will the weather be like tomorrow?"* Within seconds, your virtual assistant understands your question, retrieves the latest forecast, and answers. The same process occurs when you search for *"best restaurants near me"* or receive suggestions while typing an email. These intelligent systems appear to understand human language effortlessly. However, behind these simple interactions are advanced Natural Language Processing (NLP) techniques that enable computers to understand, interpret, and process human language.

It may seem like a magic trick, and we are so accustomed to these technologies that we rarely consider how they work. A young child learns language naturally through listening and interaction with others. They can identify where one word ends and another begins, detect sarcasm in a parent's voice, and understand that *"Can you close the door?"* is a request rather than a question about ability. They cannot automatically recognize word boundaries, interpret emotional tone, or determine that a word like *"bank"* may have different meanings depending on the context in which it appears. Before a computer can analyze even a single sentence, it must divide the text into smaller parts, determine the role of each word, understand their meanings, and identify the relationships among them. These fundamental steps form the building blocks of Natural Language Processing and apply to every NLP application, from your smartphone voice assistants to email spam filters.

Processing human language is always challenging because language is constantly evolving. The same idea can be expressed in multiple ways, and word meanings can change depending on context. Everyday communication includes idioms, abbreviations, sarcasm, spelling errors, and multilingual expressions. This becomes particularly challenging when people switch between languages, such as Malayalam and English, within the same sentence. Because of these complexities, NLP systems must use specialized techniques rather than relying only on simple rule-based approaches.

This unit introduces the fundamental components of NLP and examines the challenges involved in analyzing human language. It discusses key NLP concepts, including text, documents, sentences, words, tokens, and corpora. The unit further explores different levels of linguistic analysis, such as *morphology, syntax, semantics, pragmatics, and discourse analysis.* The major challenges covered include ambiguity, context dependency, multilingual text, noisy data, and code-mixed language. A strong understanding of these foundational concepts is essential for developing intelligent language processing systems and is the basis for studying advanced NLP techniques in subsequent units.

1.2.2 NLP Building Blocks

Picture your friend sending you a long WhatsApp message explaining why they are running late. You can quickly understand the reason, the apology, and the updated plan. However, when a computer processes the same message, it initially sees only a

SGOU

sequence of characters. It doesn't understand tone, intent, or meaning of the message unless until NLP techniques help it analyze and interpret the information.

1.2.2.1 Text

Text refers to a collection of written characters, words, sentences, or paragraphs that convey information, ideas, or emotions. In natural language processing (NLP), text serves as the primary data source from which computers extract meaningful information and knowledge.

Text can be obtained from various sources, including books, newspapers, websites, emails, research articles, customer reviews, chat messages, and social media platforms.

Key Characteristics of Text

- Text is composed of characters, words, and sentences.
- Text encodes human language in written form.
- Text can be categorized as structured, semi-structured, or unstructured.
- Text may contain grammatical structures, punctuation marks, numerical values, and various symbols.
- Text often includes ambiguity, spelling errors, abbreviations, slang, and informal expressions.

For example: " I liked visiting the museum because the exhibits were educational."

Although humans can easily understand this sentence, a computer must first segment the text into individual words, analyse their meanings, and determine the overall sentiment before it can comprehend the message.

1.2.2.2 Corpus

Imagine trying to learn a new language using only a single sentence. You would have very little information to understand the language. Now consider learning from thousands of books, articles, and conversations written in that language. This is the type of resource that an NLP model requires: a large, organized collection of text from which it can learn patterns and relationships. This collection is called a *corpus*.

A corpus (plural: corpora) is a large, organized collection of text. It is used for linguistic analysis, training NLP models, and evaluating their performance.

Types of Corpora

- **Monolingual corpus:** Contains text in a single language, such as a Malayalam news corpus.
- **Multilingual corpus:** Contains text in multiple languages and is commonly used for tasks such as multilingual NLP and machine translation.

SGOU

- **Parallel corpus:** Contains the same text in two or more languages, aligned sentence by sentence. This type of corpus is essential for machine translation.
- **Annotated corpus:** Contains text labelled with additional linguistic information, such as part-of-speech tags, syntactic structures, or named entities.
- **Domain - specific corpus:** Contains text collected from a specific field or domain, such as medical, legal, or financial documents.

1.2.2.3 Document

A document is a self-contained piece of text. In natural language processing (NLP), documents are the primary units for tasks such as classification and summarization.

A document collection refers to a group of related documents, such as all medical reports from a clinic or all blog posts on a specific topic. In practical NLP applications, document collections are essential for developing systems such as search engines and recommendation tools.

A document constitutes a complete unit of text that conveys information on a specific topic. A document may range in length from a single sentence to an extensive report.

For example, a set of patient records collected from hospitals may be utilised to develop models that identify health trends.

1.2.2.4 Sentence

Consider the sentence: "*The chicken is ready to eat.*" This sentence can have multiple interpretations. It may mean that the chicken has been prepared as food, or that the chicken itself is hungry and ready to eat. Even short sentences can convey multiple meanings, which is why identifying where sentences begin and end is an essential first step in NLP.

Sentence segmentation, also known as sentence boundary detection, refers to the process of dividing a block of text into individual sentences. Although this may appear straightforward, punctuation marks can sometimes be misleading. For example, in "*Prof. Lee arrived at 10 a.m. on Monday,"* the periods do not always indicate the end of a sentence.

Accurate sentence segmentation is important because many NLP tasks, such as sentiment analysis, machine translation, and text summarization, operate at the sentence level. Errors during this step can propagate through the NLP pipeline and reduce the accuracy of subsequent processing stages.

1.2.2.5 Word

A word is a basic unit of language that carries meaning and can usually stand independently, such as *"tree"* or *"run",* In Natural Language Processing (NLP), words must be converted into numerical representations, such as embeddings or vectors, because computers cannot directly understand the meaning of raw text.

SGOU

Vocabulary

In NLP, vocabulary refers to the complete set of unique words or tokens recognized by an NLP system from its training data. You can think of it as the system's internal dictionary. Words that are not present in this vocabulary are known as "out-of-vocabulary" (OOV) words. Handling OOV words is challenging, particularly for low-resource languages with limited training data.

1.2.2.6 Token

Tokenization is the process of dividing text into smaller units called tokens. A token can be complete word, a part of a word, a character, or even a punctuation mark, depending on the tokenization method used. In NLP systems, tokenization is usually one of the first preprocessing steps because computers process text in smaller units rather than as a complete block.

Example: The sentence "Learning is fun!" can be divided into tokens as:

["Learning", "is", "fun", "!"].

Types of Tokenization

- **Word tokenization:** Divides text into individual words. For example, *"Dogs bark loudly"* becomes:

  ["Dogs", "bark", "loudly"].

- **Sentence tokenization**: Divides a paragraph or document into individual sentences. For example, "The sun is out. Let's go play." becomes:

  ["The sun is out.", "Let's go play."].

- **Subword tokenization:** Divides words into smaller meaningful units. This is useful for handling new, rare or unusual words. For example, "playfully" can be divided into:

  ["play", "ful", "ly"].

- **Character tokenization**: Divides text into individual characters. For example, "cat" becomes:

  ["c", "a", "t"].

This approach is useful for languages with complex word formations or applications where handling unknown words is important.

1.2.2.7 Lexicon

Dictionary and Lexicon

In NLP, a lexicon is a structured collection of words along with additional linguistic information associated with them. Unlike a general dictionary that mainly provides word meanings, an NLP lexicon may include word categories, usage patterns, semantic

SGOU

relationships, synonyms, and sentiment values. These resources help computers understand not only individual words but also their meaning and usage in different contexts.

Vocabulary Resources

To help computers analyze the meaning, relationships, and emotional characteristics of words, NLP systems use specialised word lists and lexical databases. Some commonly used resources include:

- **WordNet:** A large lexical database that groups English words into sets of synonyms called *synsets* and provides relationships among words based on their meanings.
- **Sentiment lexicons:** Collections of words labelled with sentiment information, such as positive, negative, or neutral, which help NLP systems identify the emotional tone of text.
- **Language-specific resources:** Lexical resources developed for specific languages. For example, IndoWordNet provides WordNet-like resources for Indian languages, including Malayalam, Hindi, and other regional languages.

1.2.3 Phases of Language Analysis

Levels of language analysis represent the stages through which a computer analyses and interprets human language. Each stage focuses on a different aspect of language, starting with word structure and progressing to the meaning of complete sentences and conversations. These levels work together to enable computers to understand and process natural language effectively.

1.2.3.1 Lexical and Morphological Analysis

Lexical analysis identifies and processes words by dividing text into meaningful units called tokens. It focuses on analysing the vocabulary and basic structure of language.

Key Components

- **Tokenization**: Divides text into smaller units known as **tokens**.

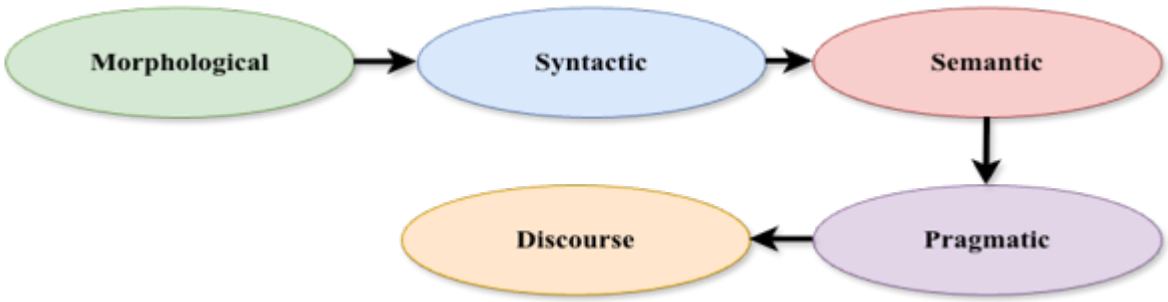


Fig 1.2.1 Five phases of Language Analysis

SGOU

- **Part-of-speech (POS) tagging**: Assigns grammatical categories, such as noun, verb, adjective, or adverb, to individual words based on their usage in a sentence.
- Lexical analysis helps identify and organise words for subsequent NLP tasks.

Morphology is the study of the internal structure and formation of words. It examines how words are constructed from the smallest meaningful units of language, called morphemes.

Example

Consider the word: "disagreement":

- **Root:** "agree" – refers to having the same opinion
- **Prefix:** "dis" – indicates negation or opposition.
- **Suffix:** "ment" – converts the verb into a noun.

1.2.3.2 Syntax

Syntax refers to the set of rules that determine how words are arranged to form meaningful and grammatically correct sentences. Through syntactic analysis, also known as parsing, NLP systems identify the grammatical structure of sentences and determine the relationships between words.

Key components

- **Part-of-Speech (POS) Tagging**: Assigns grammatical categories such as noun, verb, and adjective, or adverb, to words based on their role in a sentence.
- **Ambiguity Resolution**: Determines the intended interpretation of words or sentence structures when multiple possibilities exist.

Example:

In the sentence "Sara baked a cake," the word "Sara" functions as the subject and "cake" functions as the object.

This relationship is identified based on the grammatical structure and word arrangement of the sentence.

1.2.3.3 Semantics

Semantic analysis enables computers to understand the meaning of words, phrases, and sentences. It ensures that text interpretation is meaningful and consistent with context. This process involves analysing word meanings, semantic relationships such as synonyms and antonyms, and how context affects interpretation.

Key components

- **Named Entity Recognition (NER)**: Identifies named entities such as names

SGOU

of people, locations, organizations, and temporal expressions such as dates and times.

- **Word Sense Disambiguation (WSD)**: Determines the appropriate meaning of a word when it has multiple possible interpretations.

Example:

In the sentence "The dog drove the car," the grammatical structure is correct; however, the meaning is unrealistic because a dog cannot drive a car.

1.2.3.4 Pragmatics

Pragmatics focuses on understanding the intended meaning of language by considering the context, the speaker intention, and the situation in which communication occurs. It enables computers to interpret conversations more naturally, understand implied meanings, and handle expressions such as idioms, indirect requests, and polite statements.

Example:

When someone says, "It's cold in here," the intended meaning may be a request to close a window rather than a simply a statement about the temperature.

1.2.3.5 Discourse

Discourse analysis focuses on how sentences connect to form a meaningful, coherent text. It helps maintain consistency and interpret relationships between sentences and paragraphs.

Key Components:

- **Anaphora Resolution**: Identifies pronouns and other referring expressions and links them to the correct entities mentioned earlier in the text.
- **Contextual References**: Determines the meaning of words or phrases by considering the surrounding sentences and overall context.

Example:

"Taylor went to the store. She bought groceries."

In this example, "She" means Taylor. Anaphora resolution helps an NLP system establish this connection.

1.2.4 Challenges of Processing Human Languages

Computers face significant challenges in understanding language because words and sentences often have multiple interpretations depending on context. For instance, the sentence "Please close the window, it's chilly" is a polite request to shut the window rather than merely a comment about the weather. Similarly, the expression "That test was a piece of cake" indicates that the test was easy; it does not refer to a dessert. While humans can easily interpret such meanings using their background knowledge

SGOU

and context, computers require specialized methods and algorithms to achieve similar understanding. Therefore, enabling computers to comprehend everyday language remains one of the major challenges in Artificial Intelligence.

1.2.4.1 Ambiguity

Ambiguity occurs when a word, phrase, or sentence has more than one possible interpretation. Humans use context and prior knowledge to determine the intended meaning, whereas NLP systems require computational methods to identify the correct interpretation.

Example

"He watched the bat fly at night."

Here, "**bat"** refers to the animal.

"She swung the bat at the baseball."

Here, "**bat**" refers to sports equipment.

1.2.4.2 Context Dependence

The meaning of a sentence often depends on the surrounding context, not just the words themselves.

Example: "She's on fire!"

In an emergency, it may mean that someone is literally burning and in danger.

In a sports context, it may mean that she is performing extremely well.

NLP systems must analyse contextual information to determine the intended meaning of a sentence.

1.2.4.3 Morphological Complexity

Languages differ significantly in the way they form words. For example, Malayalam is an agglutinative language, where multiple morphemes can be combined to form a single word. As a result, one word in Malayalam may convey information that requires multiple words in English. This increases the complexity of morphological analysis in languages with rich word formation systems.

Example,

A single Malayalam word can express information related to tense, number, and grammatical relationships through attached suffixes.

പഠിക്ക്(paṭhikk) + ഉന്നു (unnu) → പഠിക്കുന്നു (paṭhikkunnu)

- Root: പഠിക്ക് – study
- Suffix: ഉന്നു– indicates present tense

SGOU

1.2.4.4 Syntactic Variation

Languages follow different grammatical rules and word-order patterns. For example, English generally follows the *Subject-Verb-Object (SVO)*structure, whereas Malayalam and Hindi commonly follow the *Subject-Object-Verb (SOV)* structure. Additionally, the same meaning can often be expressed using different sentence structures within a language. These variations make it challenging for NLP systems to accurately analyse sentence structure and meaning.

Example : Word Order Across Languages

English (SVO): Rahul plays football.

Malayalam (SOV): റാഹുൽ ഫുട്ബോൾ കളിക്കുന്നു. *(Rāhul phuṭbōḷ kaḷikkunnu.)*

Hindi (SOV): राहुल फुटबॉल खेलता है।

1.2.4.5 Polysemy

Polysemy occurs when a single word has multiple **related meanings** depending on the context.

Example:

"The school is near the river bank."

"I deposited money in the bank."

The word **"bank"** has different but related meanings: the side of a river and a financial institution. NLP systems must use context to determine the intended meaning.

1.2.4.6 Code-Mixed and Multilingual Language

People often use multiple languages within the same conversation, especially in multilingual societies.

Example:

"Let's meet after the class. പിന്നീട് coffee കുടിക്കാം."

This sentence demonstrates code-mixing between English and Malayalam.

Such multilingual communication creates additional challenges for NLP systems because they must identify, process, and interpret multiple languages within the same text.

1.2.4.7 Large Volume of Unstructured Data

Every day, people generate millions of emails, social media posts, news articles, blogs, and online reviews. Most of this information exists in an unstructured format and cannot be directly analysed using traditional database techniques. NLP systems must process large volumes of text efficiently, maintain accuracy, and extract meaningful information.

SGOU

1.2.4.8 Sarcasm and Irony

Sarcasm expresses a meaning that is different from, or opposite to, the literal meaning of words.

Example: "Great! My laptop crashed just before the presentation."

Although the word "Great" expresses a positive emotion, the speaker actually conveys frustration or disappointment.

Detecting sarcasm and irony remains one of the challenging tasks in NLP because it requires understanding tone, context, and speaker intention.

1.2.4.9 Spelling Errors and Informal Language

Real-world text often contains spelling mistakes, abbreviations, emojis, and informal expressions.

Example: "Gud mrng ", "I'll c u tmrw."

Humans can easily interpret such messages, but NLP systems often require text normalization techniques before further processing.

1.2.4.10 Coreference Resolution

Coreference resolution refers to the task of determining which entities words or phrases refer to in a text. This remains challenging, especially in long documents and conversations where multiple possible references may exist.

***Example**:*

"*Meera met Anjali after class. She invited her to the seminar."*

In this example, the pronoun "She" could refer to either Meera or Anjali, creating ambiguity. NLP systems must use contextual information and linguistic knowledge to resolve such references correctly.

SGOU

Recap

- Natural Language Processing (NLP) relies on several fundamental components, includingtext, documents, sentences, words, tokens, corpora, and lexicons, to facilitate the analysis and understanding of human language.
- A corpus is a large, systematically organized collection of text or speech data used for training, testing, and evaluating NLP systems.
- Tokenization involves dividing text into smaller units known as tokens and serves as one of the initial steps in many NLP applications.
- The different levels of language analysis, including **morphological, syntactic, semantic, pragmatic, and discourse analysis**, enable computational systems to interpret language at different levels of complexity.
- The diversity and complexity of human language create significant challenges, making it considerably more difficult to process than structured data or programming languages.
- Major challenges in processing human language include ambiguity, context dependency, syntactic variation, idiomatic expressions, and multilingual communication, and code-mixed language.
- NLP systems must process noisy and unstructured data, includingspelling errors, abbreviations, slang, emojis, and code-mixed text, which are common in real-world communication.
- A strong understanding of the foundational concepts of NLP and the challenges associated with human language processing provides a solid foundation for learning advanced NLP techniques and developing intelligent language-based applications.

SGOU

Objective Type Questions

1. What constitutes the primary input for a Natural Language Processing (NLP) system?
2. What term is used to describe a large collection of texts in NLP?
3. What is the term for a single unit of text within a corpus?
4. What is the process of dividing text into individual sentences called?
5. What is the smallest unit of language that carries meaning?

6. What are the smaller units of text produced through the tokenization process called?

7. What is commonly used initial preprocessing step in Natural Language Processing tasks?

8. What term refers to an organized collection of words and their associatedmeanings?

9. Which level of linguistic analysis focuses on the study of word formation?

10. Which level of linguistic analysis examines sentence structure?

11. Which level of linguistic analysis focuses on the meaning of words and sentences?

12. Which level of linguistic analysis considers context and speaker intention?

13. What linguistic challenge occurs when a word, phrase, or sentence has multiple possible meanings?

14. What is the phenomenon called when two or more languages are mixed within a single sentence?

15. Which Natural Language Processingtask identifies words or phrases that refer to the same entity?

Answers to Objective Type Questions

1. Text
2. Corpus
3. Document
4. Sentence segmentation
5. Morpheme
6. Token
7. Tokenization
8. Lexicon
9. Morphology

SGOU

10. Syntax

11. Semantics

12. Pragmatics

13. Ambiguity

14. Code-mixing

15. Coreference resolution

Assignments

1. Describe the fundamental building blocks of Natural Language Processing (NLP), with relevant examples that are different from those presented in the course material.

2. Explain the different phases of language analysis in NLP and discuss how each phase contributes to comprehensive language understanding.

3. Discuss the major challenges involved in processing human language, supporting your explanation with relevant examples.

4. Distinguish between a word and a token, and explain the importance of tokenization in Natural Language Processing with suitable examples.

5. Justify the statement that human language is more challenging for computers to process than computer languages by discussing the characteristics of natural language and the role of specialised NLP techniques.

SGOU

Reference

1. D. Jurafsky and J. H. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, 3rd ed., draft. Pearson. [Online].

2. C. D. Manning and H. Schütze, *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.

3. S. Bird, E. Klein, and E. Loper, *Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit*. O'Reilly Media, 2009.

Suggested Reading

1. GeeksforGeeks, “Phases of Natural Language Processing (NLP),” GeeksforGeeks. [Online]. Available: https://www.geeksforgeeks.org/machine-learning/phases-of-natural-language-processing-nlp/ [Accessed: 6 September 2026].

SGOU

Unit 3

Core NLP Tasks

Learning Outcomes

After completing this unit, learner will able to:

- define the five main NLP tasks and explain how they differ: Text Classification, Machine Translation, Text Summarization, Question Answering, and Named Entity Recognition
- describe the typical steps involved in performing each task
- identify real-world examples where each of these tasks is used
- recognize the main challenges associated with each task, especially when working with low-resource languages such as Malayalam

Prerequisites

Applications that can understand and process human language have become a regular part of everyday life. For example, search engines find web pages that match user queries, email services filter spam, translation tools help users read text in different languages, and virtual assistants provide quick answers to questions. Although these applications appear simple, they rely on several specialized **Natural Language Processing (NLP)** tasks working together behind the scenes.

To understand how these applications work, it helps to have a basic understanding of concepts introduced in earlier units, such as text, tokens, corpora, language analysis, and the challenges involved in processing human language. Familiarity with programming, Artificial Intelligence, or Machine Learning is beneficial but not mandatory. In this unit, we will explore how these fundamental concepts apply to real-world language-processing problems through key NLP tasks.

Keywords

Text Classification, Machine Translation, Text Summarization, Question Answering, Named Entity Recognition, Extractive Summarization, Abstractive Summarization



SGOU

Discussion

The five tasks covered in this unit are closely interconnected. In real-world applications, a single system often uses multiple NLP tasks together. For example, a customer support system may use Text Classification to route a complaint to the appropriate department, Named Entity Recognition to extract information such as the product name and order number, and Question Answering to generate a suitable response.

Learning about each task individually, as you are doing now, is the first step toward understanding how these tasks work together in practical NLP systems. You will explore these integrations in more detail later in the course.

1.3.1 Introduction

Imagine a student in Kollam using a language app to translate Malayalam text into English for a class assignment. In Kannur, a bank employee uses a tool to classify customer emails as complaints or inquiries. A teacher in Palakkad summarizes lengthy articles for classroom discussions. In Thrissur, a government officer relies on a chatbot to automatically answer citizens' questions. Meanwhile, a healthcare worker in Kottayam uses software to identify medicine and disease names in patient records. Although their roles are different, each person depends on specific NLP tasks like Machine Translation, Text Classification, Text Summarization, Question Answering, and Named Entity Recognition**.** This unit covers these key topics.

In Units 1 and 2, you learned what Natural Language Processing (NLP) is and why it is enabling machines to understand human language is challenging. You also explored the fundamental concepts that help machines process and interpret language. NLP combines ideas from linguistics, computer science, and statistics. You learned that human language is largely unstructured and requires appropriate preprocessing before machines can process it effectively.

This unit builds on these concepts by exploring the major tasks NLP systems perform. Many language-based tools that you use daily, such as spam filters, Google Translate, YouTube video summaries, Siri's voice responses, and LinkedIn's entity tagging, rely on one or more of five core NLP tasks: Text Classification, Machine Translation, Text Summarization, Question Answering, and Named Entity Recognition. In this unit, you will learn how each task works, the problems it addresses, and its real-world applications.

These five tasks are different ways machines process and understand language. Text Classification identifies the topic or category of a text. Machine Translation converts information from one language to another. Text Summarization reduces lengthy texts to their essential points. Question Answering retrieves or generates relevant information to answer user queries. Named Entity Recognition identifies entities such as people, locations, organizations, and other important terms in text. By the end of this unit, you will be able to recognize which NLP task supports the language-based applications you use in everyday life.

SGOU

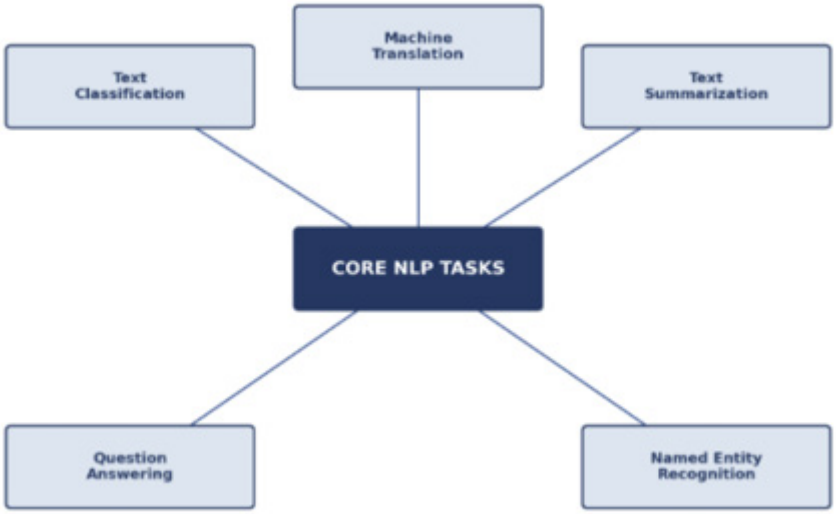


Fig 1.3.1 Core NLP Tasks

1.3.2 Core NLP Tasks

1.3.2.1 Text Classification

When you post a comment on social media, the platform may automatically hide it if it detects offensive content. A text classifier analyzes the comment and decides whether to display it or filter it out.

Text classification is the process of assigning a label or category to a piece of text. It is one of the oldest and most common tasks in NLP because many real-world problems involve determining the category to which a text belongs.

Some common types of text classification include:

- **Sentiment Analysis:** Determining whether a review or comment expresses a positive, negative, or neutral opinion.

  Example*:* Identifying whether a customer liked a product based on their review.

- **Spam Detection:** Determining whether an email or SMS message is unwanted or spam.

  Example*:* Automatically moving suspicious emails to the Spam folder.

- **Topic Labelling**: Categorizing a document or news article into predefined topics such as sports, politics, or technology.

  Example*:* Automatically tagging a news story as "sports."

- **Language Identification**: Detecting the language in which a text is written.

  Example: Distinguishing between Malayalam and Tamil in a multilingual social media post.

SGOU

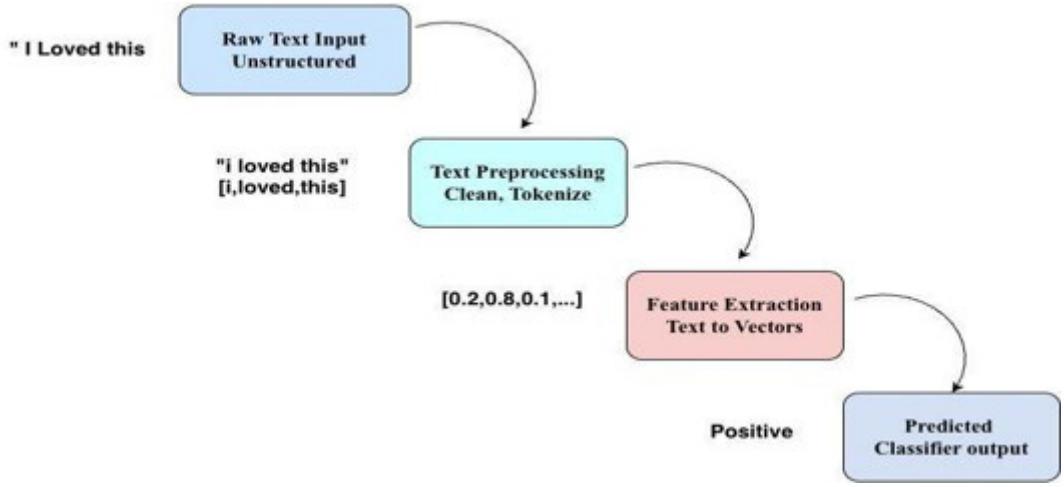


Fig 1.3.2 Text Classification Process

A typical text classification process follows the steps in Figure 1.3.2. First, preprocessing cleans and prepares the raw text. Then, the text is converted into a numerical representation that a machine learning model can understand. This step is known as feature extraction. Finally, a trained classification model predicts the appropriate label.

Later in this course, you will learn more about classification algorithms such as Naive Bayes, Logistic Regression, and Support Vector Machines. For now, remember that text classification mainly involves assigning text documents to one or more predefined categories.

1.3.2.2 Machine Translation

Machine Translation (MT) is the process of using computers to automatically translate text from one language into another while preserving the original meaning. It is one of the most challenging tasks in NLP because languages can differ significantly in vocabulary, grammar, and sentence structure, word order, idiomatic expressions, and cultural context.

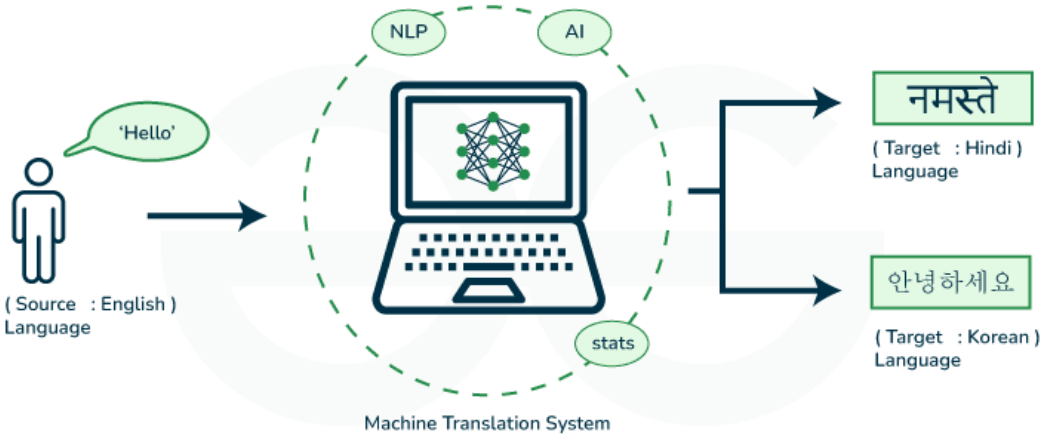


Fig 1.3.3 Machine Translation Process

Why is machine translation so difficult?

Machine Translation is a challenging NLP task because languages differ in their structure, meaning, and usage. Some of the major challenges are:

- **Word order differences:** Malayalam generally follows a Subject–Object–Verb (SOV) word order, whereas English follows a Subject–Verb–Object (SVO) structure. A direct word-by-word translation may therefore produce grammatically incorrect sentences.
- **Ambiguity:** A word may have multiple meanings depending on the context in which it is used. For example, the English word **"bank"** can refer to a financial institution or the side of a river.
- **Idioms and expressions:** Many phrases cannot be translated word-for-word because their actual meanings depend on cultural and contextual understanding. Literal translation may result in incorrect or meaningless output.
- **Low-resource languages:** Languages such as Malayalam have significantly fewer digital text resources available for training compared with widely used languages such as English or French. This limited data availability makes it harder to develop accurate machine translation systems.

Types of Machine Translation

1. Rule - Based Machine Translation (RBMT)

Rule - Based Machine Translation (RBMT)is a translation approach that converts text from one language to another using predefined linguistic rules and bilingual dictionaries.

RBMT relies on the grammatical, syntactic, and semantic rules of both the source and target languages. Since the translation process is controlled by manually created rules, RBMT provides high interpretability, consistency, and predictability.

Example:

Source: "The girl eats an apple.",

Target (Hindi):"लड़की एक सेब खाती है."

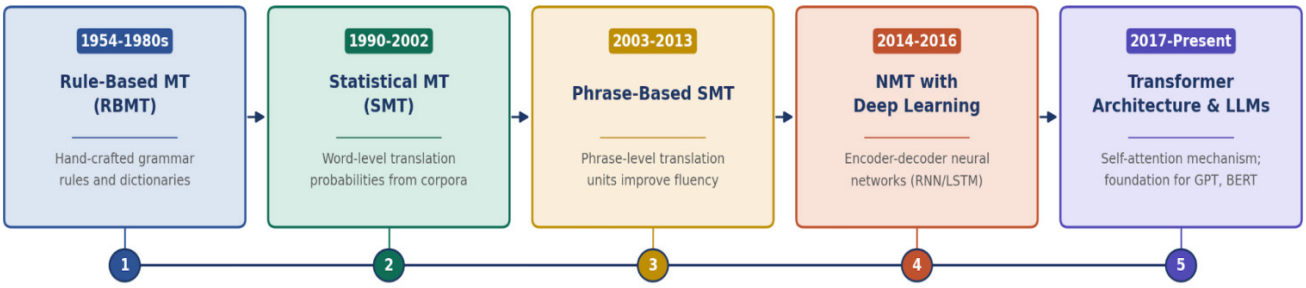


Fig 1.3.4 Evolution of Machine Translation: From Rule-Based Systems to Transformer-Based Models

SGOU

Analysis: The system identifies the grammatical structure of the English sentence. "The girl eats an apple" consists of an article, noun, and verb, article, and noun, following the English Subject–Verb–Object (SVO) structure.

Transfer: This structure is converted according to Hindi grammar, which generally follows the Subject–Object–Verb (SOV) order while maintaining grammatical properties such as gender, number, and case.

Generation:

Hindi generation rules produce the final translated sentence:

- The girl → लड़की
- eats → खाती है
- an apple → एक सेब

Final output: लड़की एक सेब खाती है.

2. Statistical Machine Translation (SMT)

Statistical Machine Translation learns translation patterns automatically from large bilingual datasets instead of depending on manually written linguistic rules. It estimates the probability of possible translations and selects the most likely output based on the learned statistical patterns.

Example:

A statistical system may learn translation probabilities such as:

- girl ↔ लड़की (0.82 probability)
- eats ↔ खाती है (0.75 probability)
- apple ↔ सेब (0.88 probability)

It selects the highest probability translations and generates:

लड़की सेब खाती है

However, word-level SMT may miss grammatical elements such as articles because it treats words independently.

3. Phrase-Based Statistical Machine Translation

Phrase-Based SMT improves upon word-level SMT by translating groups of words together rather than translating individual words separately. It learns phrase correspondences from bilingual data, producing more fluent, context-aware translations.

Example:

Instead of translating individual words, the system learns phrase mappings such as:

"eats an apple" ↔ "एक सेब खाती ह"

SGOU

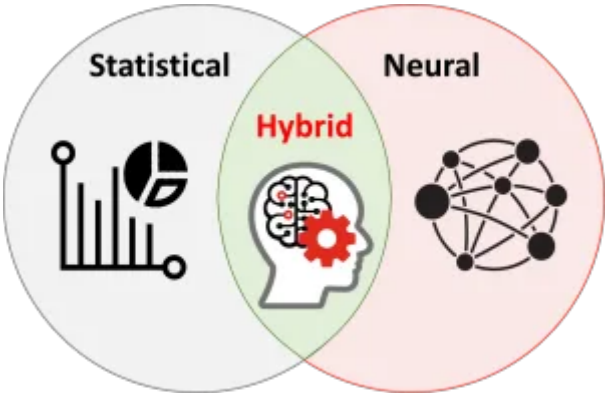


Fig 1.3.5 Hybrid Machine Translation Architecture Combining Statistical and Neural Machine Translation Approaches

The complete translation becomes:

लड़की एक सेब खाती है.

This produces a more natural and grammatically correct output compared with word-based SMT.

4. Neural Machine Translation (NMT) with Deep Learning

Neural Machine Translation uses deep neural networks, typically encoder-decoder architectures, to learn complex relationships between source and target languages. The encoder processes the input sentence into contextual representations, and the decoder generates the translated sentence sequentially.

Unlike rule-based and statistical approaches, NMT learns grammar, word relationships, and context directly from data.

Example:

The model processes the complete sentence:

“The girl eats an apple”

and generates:

लड़की एक सेब खाती है.

The translation is produced as a complete sequence rather than by combining manually defined rules or phrase tables.

5. Transformer Architecture and LLMs

Transformer-based models introduced the self-attention mechanism, which allows the model to consider relationships between all words in a sentence simultaneously. This enables better handling of long-range dependencies and contextual meaning.

Transformers form the foundation of modern large language models (LLMs), including

SGOU

Table 1.3.1: Difference between extractive and abstractive summarization

Extractive Summarization	Abstractive Summarization
Selects important sentences or phrases directly from the original text.	Generates new sentences by rephrasing and interpreting the main ideas of the source text.
Uses only the original text without introducing new wording.	Can generate new words and phrases that may not appear in the original text.
Maintains the exact wording and structure of the selected sentences.	Produces concise, coherent, and human-like summaries
Sentences are "extracted" from the input document.	Sentences are "generated" based on the meaning and context of the input document.
Can provide positional information (such as the start position and length) of the extracted text segments.	Uses contextual information from the source document to generate summary content.
Example: Selects the three most important sentences from a news article.	**Example:** Generates a short paragraph that summarizes the entire article using new wording.
Easier to implement and generally more reliable for preserving factual information.	More challenging to implement but can produce more natural and fluent summaries.

models such as GPT and BERT, which support translation along with many other language tasks.

Example:

When translating:

“The girl eats an apple”

the attention mechanism allows the model to relate:

- “girl” with the appropriate verb form,
- “apple” with the object relationship,
- the entire sentence context during translation.

This contextual understanding helps modern models generate:

लड़की एक सेब खाती है.

SGOU

6. Hybrid Machine Translation

Hybrid machine translation combines two or more translation approaches, such as rule-based, statistical, and neural methods, within a single system. Some hybrid systems may also include human post-editing as a final refinement step.

Hybrid MT aims to combine the strengths of different approaches. Rule-based methods provide grammatical control, statistical methods improve fluency through data-driven learning, and neural methods provide better contextual understanding.

By integrating these approaches, hybrid systems aim to achieve translations that are more accurate, natural, and reliable than individual approaches alone.

1.3.2.3 Text Summarization

When a doctor prepares for morning rounds, there may not be enough time to read every detail from numerous patient records. Instead, the doctor can review concise summaries generated by an NLP system that highlight important updates and changes from overnight. Whether a summary is created by a human or generated automatically by a machine, the main objective is to help users quickly identify the most relevant information.

Figure 1.3.6 illustrates the concept of text summarization, which involves creating a shorter version of a document while retaining the most important information and removing repetitive or less relevant details. Table 1.3.1 shows two main approaches to text summarization: extractive summarization and abstractive summarization.

1.3.2.4 Question Answering

Ask a smart speaker 'What is the capital of Kerala?' and, within a second, a single confident word comes back: 'Thiruvananthapuram.' No document was handed to you, no list of links - just the answer itself. That instant, precise response is the signature of

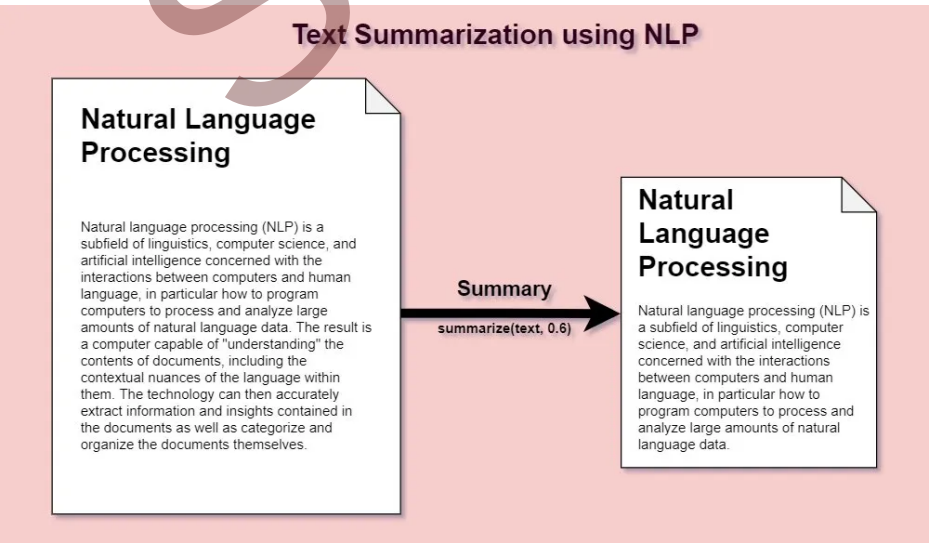


Fig 1.3.6 Text Summarization

SGOU

a working Question Answering system.

Question answering (QA)is an NLP task in which a system provides an answer to a given question. If you have ever asked a virtual assistant such as Alexa, Siri, or Google Assistant about the weather, you have interacted with a question answering system.

There are two common types of question answering tasks:

- **Extractive**: Extracts the answer directly from the given context or document.
- **Abstractive**: Generates a new answer based on the information in the context while correctly addressing the question.

Question Answering Pipeline

A typical Question Answering system involves the following steps:

- **Question Processing:** Analyzing the user's question to determine the type of answer expected, such as a person, place, date, or numerical value.
- **Passage Retrieval:** Searching a large collection of documents to identify relevant passages that are likely to contain the required answer.
- **Answer Extraction** - Identifying the exact span of text within the retrieved passage that provides the answer to the question.

1.3.2.5 Named Entity Recognition (NER)

When LinkedIn automatically detects that the sentence *'I joined Infosys in Thiruvananthapuram*' contains a company name and a city and provides links to their respective pages, it is using Named Entity Recognition (NER). This demonstrates one of the important applications of NER in NLP systems.

Named Entity Recognition (NER) is an NLP task that identifies and extracts entities from a text and assigns them to predefined categories such as person names, organizations, locations, dates, and quantities.

The input to an NER model is generally a sequence of text, and the output consists of the identified named entities along with their corresponding start and end positions in the text.

NER is often considered a sequence labelling problem because the model assigns a label to each word or token in a sequence to identify entity boundaries and categories.

Named Entity Recognition is widely used in applications such as news summarization, information extraction, search engines, recommendation systems, and detecting misinformation.

NER also presents several challenges. Ambiguity is a major issue. For example, *"Mahe"* may refer to a district (a Location entity) or, in another context, a person's name. Some entities may also contain multiple words, such as *"Sree Narayanaguru Open University,"* which represents a single Organization entity rather than four separate entities.

SGOU

Figure 1.3.7 illustrates how NER identifies and classifies entities such as people, organizations, locations, and dates in a text. In later units, you will learn more about NER techniques and their practical implementation.



Fig 1.3.7 Example of NER Recognition

Recap

- Text Classification assigns text to predefined categories. This underpins applications such as spam filtering and sentiment analysis.
- Machine Translation converts text from one language into another while preserving its meaning. It must handle challenges such as different word orders, ambiguous meanings, idiomatic expressions, and limited training data for low-resource languages such as Malayalam.
- Text Summarisation creates a shorter version of a document while preserving the main information. It can be performed by selecting important sentences from the original text (extractive summarization) or by generating new sentences that express the key ideas (abstractive summarization).
- Question Answering systems provide answers to questions expressed in natural language. They typically involve understanding the question, identifying relevant information, and extracting or generating the appropriate answer.
- Named Entity Recognition identifies and labels entities such as people, organizations, locations, and other important terms in text. It is commonly treated as a sequence labelling problem, where each word or token is assigned an appropriate label.
- The five core NLP tasks often work together in real-world applications. In the following units, we will explore each task in greater detail.

SGOU

Objective Type Questions

1. What is the main difference between extractive and abstractive summarization?
2. Named Entity Recognition is best described as which type of NLP task?
3. Which Machine Translation (MT) approach uses predefined grammar rules?
4. Which Machine Translation (MT) approach uses probability-based models for translation?
5. What is the process of creating a shorter version of a document while preserving its important information called?
6. Which type of summarization selects important sentences directly from the original text?
7. Which entity type does "London" represent in Named Entity Recognition?
8. Which NLP task identifies names of people, locations, and organizations in text?
9. Give an example of a Text Classification application.
10. Give an example of a spam detection label used in Text Classification.
11. What type of data is required to train a Supervised Text Classification model?
12. Which stage of Machine Translation introduced the Transformer architecture?

SGOU

Answers to Objective Type Questions

1. Extractive summaries select existing sentences; abstractive summaries generate new ones
2. Sequence labelling
3. Rule-Based MT
4. Statistical MT
5. Summarization
6. Extractive Summarization
7. Location

8. NER
9. Classifying an email as spam or not.
10. ‘Spam’ or ‘not spam’
11. Labeled data
12. Transformer Architecture & LLMs

Assignments

1. List the five core NLP tasks covered in this unit and provide one real-world application for each.
2. Explain the difference between extractive and abstractive summarization with a suitable example.
3. Define Named Entity Recognition (NER). Identify and label all named entities present in a given sentence.
4. Compare Text Classification and Named Entity Recognition in terms of what each task labels (a whole document vs. individual words). Provide one example for each task.
5. List the five stages in the evolution of Machine Translation in chronological order. Briefly explain the core idea behind each stage and provide examples different from those discussed in this text.

Reference

1. NLP4SS, “Session 01,” NLP4SS. [Online]. Available: https://nlp4ss.jeju.ai/en/session01/index.html [Accessed: 6 September 2026].
2. IBM, “Text Classification,” IBM Think. [Online]. Available: https://www.ibm.com/think/topics/text-classification [Accessed: 6 September 2026].
3. Hugging Face, “Question Answering,” *Transformers Documentation*. [Online]. Available: https://huggingface.co/docs/transformers/en/tasks/question_answering [Accessed: 6 September 2026].
4. C. B. Lavanya, H. S. Nagendraswamy, K. N. Manjunath, and M. P. Kumar, “Machine Translation in Natural Language Processing (1948–2025): A Systematic Survey of Methods, Benchmarks, Neural Machine Translation Advances, Challenges and Future Directions,” *SN Computer Science*, vol. 7, no. 6, p. 596, 2026.

SGOU

5. S. Bird, E. Klein, and E. Loper, *Natural Language Processing with Python*. O'Reilly Media, 2009.

6. S. Sharma, M. Diwakar, P. Singh, V. Singh, S. Kadry, and J. Kim, "Machine Translation Systems Based on Classical-Statistical-Deep-Learning Approaches," *Electronics*, vol. 12, no. 7, Art. no. 1716, 2023.

Suggested Reading

1. D. Jurafsky and J. H. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, 2nd ed. Pearson Prentice Hall, 2009.

2. A. Anugu and G. Ramesh, "A Survey on Hybrid Machine Translation," in *E3S Web of Conferences*, vol. 184, p. 01061, 2020.

SGOU

Unit 4

Approaches to NLP

Learning Outcomes

After completing this unit, learner will able to:

- explain what Heuristics-Based NLP is and how it uses hand-crafted rules and word lists to process language
- describe Machine Learning-Based NLP and how it learns patterns from data instead of relying on manually created rules
- explain the differences between supervised, unsupervised, and reinforcement learning methods used in ML-based NLP
- compare the two approaches by considering factors such as scalability, interpretability, and data requirements
- identify which approach is used in a given real-world NLP application
- recognize the limitations of each approach and explain why modern NLP systems usually combine both approaches

Prerequisites

Before starting this unit, take a moment to review what you already know. This unit builds upon the concepts covered in Units 1, 2, and 3. From Unit 1, you should understand what NLP is and why processing human language is challenging for machines. From Unit 2, you should be familiar with key concepts such as corpus, tokenization, and the major challenges in language processing. From Unit 3, you should understand the five main NLP tasks: Text Classification, Machine Translation, Text Summarization, Question Answering, and Named Entity Recognition. This unit explores different approaches for solving these NLP tasks. You should be comfortable with the following concepts:

A machine learning model learns patterns from labelled examples and uses these patterns to make predictions on new data.

- A corpus is a large, organized collection of texts used for training or analyzing language models.
- NLP systems process text through multiple stages, beginning with raw input and producing structured output.



SGOU

Keywords

Heuristic-Based NLP, Lexicon, Machine Learning, Supervised Learning, Unsupervised Learning, Reinforcement Learning, Features, Training Data, Generalization, Preprocessing

Discussion

Choosing an appropriate approach is one of the most important steps when developing an NLP system. It influences the amount of data required, the level of linguistic expertise needed, the system's ability to handle new language patterns, and the interpretability of its decisions. In this section, we will examine each approach in detail and then compare their strengths and limitations.

1.4.1 Introduction - Approaches to NLP

Have you ever wondered how your smartphone understands voice commands, how email filters detect spam, or how translation apps convert one language into another? Although human language is complex and sometimes ambiguous, computers can process it using Natural Language Processing (NLP) techniques.

In the early stages of NLP, computers relied on heuristic-based (rule-based) approaches, where language experts manually created rules and dictionaries to help machines understand text. These approaches worked well for simple and predictable language but struggled with the diversity and complexity of real-world communication.

As digital text became more abundant and computing power increased, Machine Learning-Based NLP transformed the field. Instead of depending on handcrafted rules, computers began learning language patterns from large collections of text. By analyzing examples, machine learning models could identify relationships between words, recognize patterns, and make predictions with greater accuracy and flexibility.

Today, most NLP systems combine linguistic knowledge and machine learning techniques to create intelligent systems that can understand and generate human language. Chatbots, virtual assistants, sentiment analysis systems, and machine translation applications all rely on these methods, making NLP a key component of modern artificial intelligence.

In this unit, you will learn about the two main approaches to NLP: heuristic-based NLP and machine learning-based NLP. You will explore how they work, their strengths and weaknesses, and their real-world applications.

1.4.2 Heuristic -Based NLP

Heuristic-based NLP, or Rule-Based NLP, uses rules, patterns, and language resources created by human experts. In this context, '*heuristic*' refers to a practical guideline based on experience rather than a strict mathematical formula. These guidelines act as intelligent shortcuts that often work effectively for specific language-processing tasks.

SGOU

A heuristics-based NLP system consists of several key components:

- **Rules:** These are if-then statements that connect language patterns to specific outcomes. For example, if a sentence contains words such as 'fantastic' or 'wonderful,' it may be classified as positive.
- **Lexicons**: These are specialized word lists that contain information about words, such as grammatical categories, synonyms, emotional meanings, or whether a word represents a place or a person.
- **Regular Expression**s: These are pattern-matching templates used to identify specific text patterns, such as website URLs, license plate numbers, email addresses, or hashtags.
- **Grammar Rules:** These are guidelines that define how words should be arranged to form grammatically correct sentences in a language.

A heuristics-based NLP system operates through a sequence of predefined steps, as shown in Figure 1.4.1. First, it receives raw text as input. Then, it applies predefined rules, checks lexicons for additional information, identifies specific patterns, and produces the final output. Human language experts design each component of this process in advance, meaning that the system does not learn these steps automatically.



Fig 1.4.1 Heuristic - Based NLP Pipeline

Table 1.4.1: Examples of Sentiment Analysis

Rule Type	Rule	Example
Positive Lexicon	IF word IN [great, excellent, loved, amazing] → +1	'The food was amazing' → Positive
Negative Lexicon	IF word IN [terrible, awful, hate, bad] → -1	'The food was terrible' → Negative
Negation Rule	IF 'not' precedes a positive word → reverse the sentiment	'Not great' → Negative
Intensifier Rule	IF 'very' precedes a sentiment word → score × 1.5	'Very awful' → Strongly Negative

SGOU



Fig 1.4.2 Worked Example: Heuristics-Based Sentiment Analysis

1.4.2.1 Worked Example: Heuristics-Based Sentiment Analysis

Consider a simple sentiment analysis task: classifying customer reviews as positive or negative. A heuristics-based system performs this task using predefined rules, as shown in Figure 1.4.2.

The system applies these rules sequentially to every incoming email, and then determines whether the message is spam or not. The process is fully transparent because each decision can be traced back to a specific rule set created by a human expert.

1.4.2.2 When Heuristics-Based NLP Is Effective?

- Named Entity Recognition (NER) using pattern-based rules, such as identifying an organisation by detecting words that start with a capital letter and end with terms like “Ltd” or “Inc”.
- Extracting information from structured documents like invoices, forms, or legal contracts that follow predictable layouts.
- Spell-checking and grammar correction tools.
- Identifying dates, times, phone numbers, and other structured information using regular expressions.
- Processing low-resource languages when limited training data is available but language expertise is accessible.

1.4.2.3 Limitations of Heuristics-Based NLP

- These systems require significant effort to build and maintain. Every new language, domain or exception requires additional rules, and each rule must be manually created.
- A rule that works well in one situation may fail in another. For example, *“The film was not bad”* expresses a positive sentiment, but a simple rule-based system may incorrectly classify it as negative.
- These systems do not easily adapt across different languages. For example, rules developed for English will not work directly for Malayalam or other languages and must be redesigned.
- They struggle with informal language, slang, spelling variations, and new expressions that are not included in the available dictionaries or rule sets.

SGOU

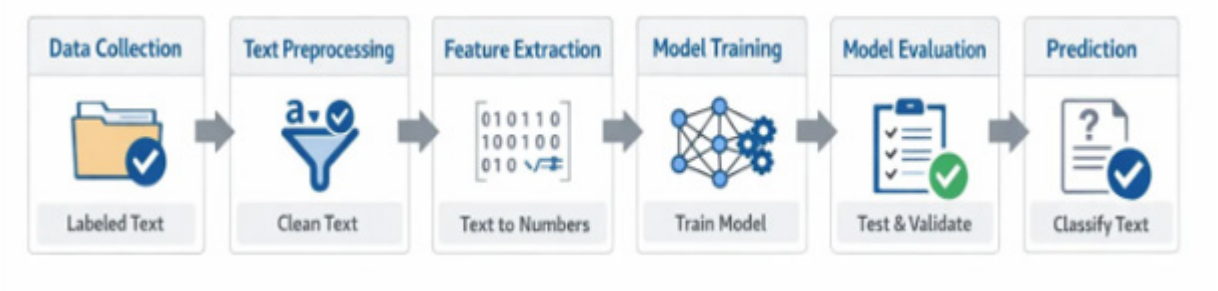


Fig 1.4.3 Machine Learning-Based NLP Pipeline [5]

- They cannot improve automatically over time unless human experts add, modify, or update the rules manually.

1.4.3 Machine Learning-Based NLP

Imagine learning to ride a bike. Most people do not study a manual that explains balance, pedaling, or turning. Instead, they get on the bike, try it out, wobble, sometimes fall, and gradually improve with each attempt. Over time, they develop a sense of how to ride - a skill that may be difficult to explain but easy to demonstrate. Similarly, Machine Learning-based NLP systems improve by training from large amounts of data rather than following a fixed set of manually created rules. Figure 1.4.3 illustrates the Machine Learning-Based NLP pipeline.

1.4.3.1 ML Pipeline Steps in NLP

Machine learning is a key component of Artificial Intelligence. It enables computers to learn from data and identify patterns without requiring explicit step-by-step instructions. While traditional programming relies on predefined rules, machine learning models improve their performance by learning from examples. The quality and quantity of training data strongly influence prediction accuracy. Simply put, machine learning allows systems to learn from experience and improve their performance automatically (Arthur Samuel, 1959).

- **Data Collection**

The first step is to gather labelled text data. This is known as data collection. You can collect text data from various sources.

Examples:

- Emails
- News articles
- Social media posts
- Reviews

SGOU

- **Text Preprocessing**

Raw text cannot be directly used by machine learning models. It must be cleaned and transformed into a suitable format before training.

Typical preprocessing steps include:

1. Tokenization
2. Lowercasing
3. Removing punctuation
4. Stop word removal
5. Stemming
6. Lemmatization

Example:

Original sentence:

"The students are studying in the classroom."

After preprocessing:

"student study classroom"

Feature Extraction

Machine learning models cannot directly understand words. Text must be converted into numerical representations called features or embeddings.

Common feature extraction methods include:

- Bag of Words (BoW)
- TF-IDF
- GloVe
- Word Embeddings
- Contextual Embeddings (e.g., BERT)

Example:

A sentence may be represented as a numerical vector:

[0.2, 0.8, 0.1, ...]

- **Model Training**

Once text is converted into numerical features, a machine learning algorithm is trained. During training, the model learns relationships between input features and output labels.

SGOU

Example:

Input: "NLP is a fantastic subject."

Prediction: Positive

- **Prediction**

The trained model is used to predict outcomes for new, unseen text. After learning from many examples, the model can classify new data.

Example:

Input: "Big data is amazing."

Prediction: Positive

- **Model Evaluation**

Before deploying a model in a real-world application, you must evaluate its performance. Common evaluation metrics include:

- Accuracy
- Precision
- Recall
- F1-score
- ROC-AUC (for binary classification)

1.4.3.2 Types of Machine Learning

There are three broad categories of Machine Learning (ML) commonly used in NLP, as shown in Figure 1.4.4:

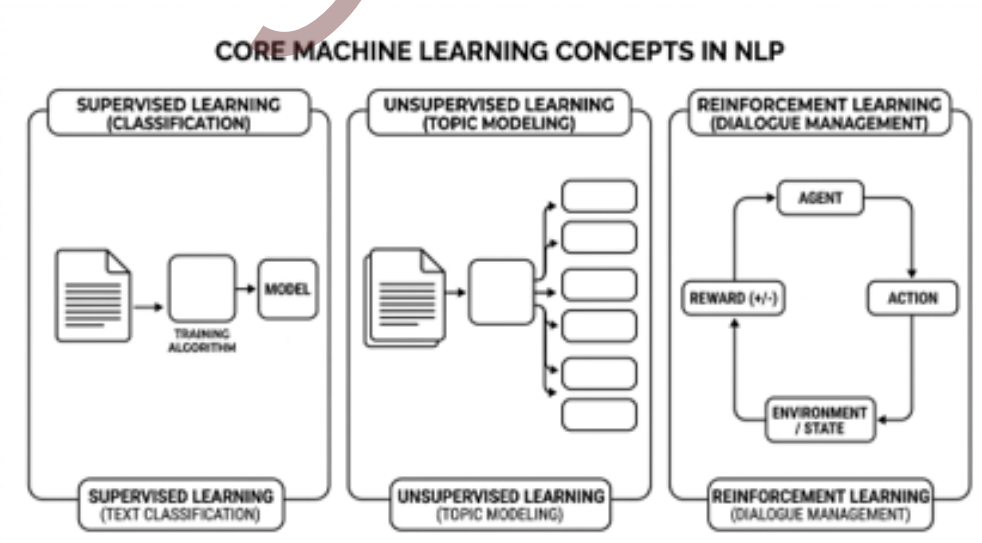


Fig 1.4.4 Types of Machine Learning

SGOU

- **Supervised Learning**

The model learns from data in which each example is associated with the correct output label. By observing many input-output pairs, the model learns patterns and uses them to make predictions on new, unseen data.

For example: Consider teaching a computer to grade student essays. You provide it with hundreds of essays, each labelled as "pass" or "fail" by teachers. The model learns the characteristics of successful essays and can then grade new essays automatically.

Unsupervised Learning

In this approach, the model receives data without predefined labels. It identifies patterns, structures, or groups in the data based on similarities, even when the categories are unknown in advance.

For example:

Imagine having thousands of scientific articles without any labels. The model analyses their content and automatically groups them into categories such as "machine learning," "biology," or "economics" without being told which topics to find. Later, humans can examine these groups and assign appropriate names.

- **Reinforcement Learning**

Reinforcement Learning (RL) learns through interaction and experience. The model takes actions, receives feedback as rewards or penalties, and gradually learns the best strategy to achieve a goal.

For example:

Imagine a robot learning to schedule university classes. Each time it creates a timetable, it receives rewards if there are no conflicts and students are satisfied, but receives penalties for overlapping classes or poor scheduling. After many attempts, it learns to generate better schedules.

1.4.3.3 Common ML Algorithms in NLP

1. Naive Bayes Classifier:

A simple and fast probabilistic algorithm commonly used for text classification tasks such as spam detection and sentiment analysis.

2. Logistic Regression:

A statistical model that predicts the probability of a category, such as positive or negative sentiment. Widely used for binary and multiclass text classification**.**

3. Support Vector Machines (SVM):

SVMs identify the optimal decision boundary that separates different text classes. They are effective for tasks like document classification and email filtering**.**

SGOU

Table 1.4.2: Types of ML

Type of ML	The Model Receives	How It Learns	NLP Applications
Supervised Learning	Labeled text (input + correct output)	Learns to map inputs to correct output labels by minimizing prediction error	Text classification, NER, Machine Translation, Sentiment Analysis
Unsupervised Learning	Unlabeled text (input text only)	Discovers hidden structures and patterns in data without predefined categories	Topic modelling, text clustering, word embeddings (Word2Vec)
Reinforcement Learning	Text + a reward or penalty signal	Learns by trial and error, maximizing cumulative reward	Dialogue systems, chatbots, game-playing agents

4. Decision Trees and Random Forests:

These models split data into branches based on features, such as the presence of specific words. Random Forests combine multiple decision trees to improve performance in classification tasks.

5. K-Nearest Neighbors (KNN):

KNN classifies new text based on the most similar examples in the training data. It is useful for text categorization and recommendation systems.

6. K-Means Clustering:

K-Means is an unsupervised learning algorithm that groups text documents into clusters based on similarity. It is commonly used for topic modelling, document organization, and categorizing large collections of unstructured text documents.

7. Neural Networks:

Neural networks are deep learning models that include simple feedforward networks and advanced architectures such as Recurrent Neural Networks (RNNs) and Long Short-Term Memory networks (LSTMs). They are effective for tasks such as machine translation, text generation, and named entity recognition.

8. Transformers:

Transformers are modern neural network architectures, including models such as BERT, GPT, and RoBERTa, that have revolutionized NLP. They capture context and relationships between words more effectively than previous models and power applications such as chatbots, text summarization, and question answering.

SGOU

1.4.3.4 Advantages of Machine Learning-Based NLP

- **Adaptability across languages and domains:** Machine learning-based NLP systems can be applied to different languages and topics with minimal manual effort. To support a new language, it is often sufficient to provide appropriate training data rather than manually designing new rules.
- **Better handling of language variation and ambiguity:** These systems can handle variations in language and ambiguous expressions effectively because they learn patterns from real-world text data.
- **Continuous improvement:** ML-based NLP systems can improve their performance when provided with additional data or when retrained with updated datasets. Modern models, such as Transformers and Large Language Models (LLMs), achieve human-level or near-human performance on several NLP tasks.
- **Capability to process complex tasks:** Advanced NLP models can perform complex tasks such as translation, text generation, summarization, and question answering. However, training and deploying state-of-the-art models require significant computational resources, memory, and specialized hardware.

1.4.3.5 Limitations of Machine Learning-Based NLP

- Requirement for large amounts of data: Machine learning models often require large quantities of labelled training data, which can be expensive and time-consuming to create. This is particularly challenging for low-resource languages such as Malayalam.
- Limited interpretability: Understanding how complex models make decisions can be difficult, leading to the "black box" problem. Additionally, these models depend heavily on the quality of training data; biases present in the data may be reflected in the model's predictions.
- High computational requirements: Training large-scale models requires substantial computational power, advanced hardware, and significant energy consumption.
- Possibility of incorrect or unreliable outputs: Generative NLP systems may sometimes produce incorrect, misleading, or fabricated information, commonly referred to as hallucinations. This is a major concern in applications that require accuracy and fact verification.
- Lack of true understanding: Although these models can identify complex patterns in language, they do not possess human-like common sense, awareness, or genuine understanding of meaning.

SGOU

Recap

- Heuristics-Based NLP processes language using hand-crafted rules, lexicons, and patterns created by human linguistic experts. It is transparent and effective for structured tasks, but does not scale well and struggles with ambiguity and language variation.
- Machine Learning-Based NLP learns patterns automatically from large datasets, making it scalable and adaptable to linguistic variation. However, it requires substantial labeled data and can be difficult to interpret.
- Within Machine Learning-Based NLP, supervised learning uses labeled data, unsupervised learning discovers hidden structure in unlabeled data, and reinforcement learning learns through interaction from reward signals.
- Common ML algorithms used in NLP include Naive Bayes, Logistic Regression, SVM, Random Forest, Deep Neural Networks, and Transformers.
- The two approaches are complementary rather than competing. Many real-world NLP systems combine rule-based preprocessing with machine learning models to achieve better performance.
- The choice of approach depends on factors such as available data, linguistic expertise, computational resources, and the interpretability requirements of the application.

Objective Type Questions

1. Which NLP approach uses hand-crafted grammar rules and dictionaries to process text?
2. Which type of machine learning in NLP is trained using labelled input-output pairs?
3. Which type of ML is used in dialogue systems and chatbots that learn from user interactions?
4. What does 'generalization' mean in the context of a machine learning model?
5. What type of data is required to train a supervised ML model?
6. What is another name for Heuristics-Based NLP?
7. What information does a lexicon contain in a heuristic-based system?
8. Which approach scales better across multiple languages (Heuristic/ ML based)?

SGOU

9. Name any one metric used to evaluate an ML model's performance
10. Which ML type discovers patterns in data without predefined categories?
11. Name one algorithm used in ML-based NLP.
12. Name any embedding techniques used in NLP.
13. Which type of learning uses reward and penalty signals?
14. Which NLP approach is described as a black box?
15. What is the first step of ML pipeline?

Answers to Objective Type Questions

1. Heuristics- Based NLP
2. Supervised learning NLP
3. Reinforcement Learning
4. Performing well on unseen data beyond the training set
5. Labelled data
6. Rule-Based NLP
7. Words and their linguistic properties
8. ML-Based NLP
9. Accuracy / Precision / Recall / F1 Score
10. Unsupervised Learning
11. Naive Bayes (or SVM / Logistic Regression / Random Forest)
12. Bag of Words (BoW)/ TF-IDF /Glove
13. Reinforcement Learning
14. ML-Based NLP
15. Raw text/ Data collection

SGOU

Assignments

1. List the two major approaches to NLP discussed in this unit and provide one real-world application for each.

2. Compare Heuristic-Based NLP and Machine Learning-Based NLP across any three dimensions of your choice. Provide a specific example to support each comparison point

3. Identify which approach (Heuristic-Based NLP or Machine Learning-Based NLP) you would recommend for developing an NLP system to extract patient names and hospital names from handwritten medical records in Malayalam. Justify your choice with at least two reasons.

4. List five common machine learning algorithms used in NLP that are discussed in this unit, and briefly state what type of NLP task each algorithm is commonly used for.

5. Differentiate between Heuristic-Based NLP and Machine Learning-Based NLP.

Reference

1. D. Jurafsky and J. H. Martin, "Naive Bayes and Sentiment Classification," in *Speech and Language Processing*, 3rd ed. draft. [Online]. Available: https://web.stanford.edu/~jurafsky/slp3/4.pdf

2. IBM, "Types of Machine Learning," IBM Think. [Online]. Available: https://www.ibm.com/think/topics/machine-learning-types

3. NLP4SS, "Lecture 2," NLP4SS. [Online]. Available: https://nlp4ss.jeju.ai/en/session02/lecture2.html

4. Scikit-learn, "Supervised Learning," *Scikit-learn Documentation*. [Online]. Available: https://scikit-learn.org/stable/supervised_learning.html

5. upGrad, "Text Classification in NLP," upGrad. [Online]. Available: https://www.upgrad.com/blog/text-classification-in-nlp/

6. S. Singh and S. Hooda, "A Study of Challenges and Limitations to Applying Machine Learning to Highly Unstructured Data," in *2023 7th International Conference on Computing, Communication, Control and Automation (ICCUBEA)*, pp. 1–6, IEEE, 2023.

SGOU

Suggested Reading

1. D. Jurafsky and J. H. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, 3rd ed., draft. Pearson

SGOU



Pre-processing and Representation Models

SGOU

Unit 1
NLP System Pipeline

Learning Outcomes

After completing this unit, learner will able to:

- describe the overall structure of a data-driven NLP system pipeline
- explain the purpose and significance of each stage in the pipeline
- identify the common sources and methods of data acquisition in NLP
- recognize the challenges associated with acquiring adequate data for an NLP task

Prerequisites

Imagine a chef preparing to cook a complex meal. Before any cooking starts, there is a lot of behind-the-scenes work: finding ingredients, washing vegetables, chopping, and measuring. This part is not glamorous, but it shapes the final dish. If you skip or rush these steps, even the best cooking skills cannot fully save the meal. Building an NLP system is similar. Before a model can classify, translate, or summarize, you need to gather, clean, and prepare raw language data. This early work does not get much attention, but it has a significant impact on how well everything else works.

You should already be familiar with what a corpus is, how tokenization works, and some basic text pre-processing terms. You should also know about the linguistic levels of morphology, syntax, semantics, and pragmatics from earlier lessons. With this background, we can now look at the first and often most underestimated stage of building any real NLP system: acquiring the data itself.

Keywords

NLP Pipeline, Data Acquisition, Web Scraping, Text Cleaning, Pre-processing, Feature Engineering, Model Deployment, Data Annotation



SGOU

Discussion

2.1.1 Introduction

A young journalist gets her first major investigative story. She doesn't start writing right away. Instead, she gathers documents, interviews sources, examines public records, and organizes her notes before writing a single sentence. The final article is just the visible tip of a much longer process, most of which takes place before any writing begins.

Building an NLP system is much the same. The finished product, such as a chatbot, translator, or search engine, is what people see, but it is the result of a long chain of behind-the-scenes work that most users never notice.

This series of steps is called the NLP pipeline. It is a structured sequence of processes through which raw text is transformed into a form a machine learning model can use. Each stage takes the output of the previous stage and refines it further, just as the journalist's research leads to interviews, and the information gathered from those interviews helps shape the final draft.

The first stage in this pipeline, and the main focus of this unit, is data acquisition. Data acquisition involves finding, collecting, and gathering the raw text data an NLP system needs. Just as a journalist's story depends on reliable sources, an NLP system depends heavily on data quality. If the dataset is poorly collected, too limited, or biased, it can negatively affect every stage that follows, regardless of how advanced the later modelling techniques may be.

This unit explains the overall structure of the NLP pipeline and takes a closer look at data acquisition. It discusses methods for gathering text data, common sources of textual data, and the specific challenges encountered at this stage, particularly for languages with limited digital resources.

2.1.2 The NLP System Pipeline

Think about having a cup of filter coffee at breakfast. Before you drink it, the coffee goes through several steps: the beans are roasted and ground, water is boiled, the coffee is filtered, and finally, milk and sugar are added. You only experience the finished drink.

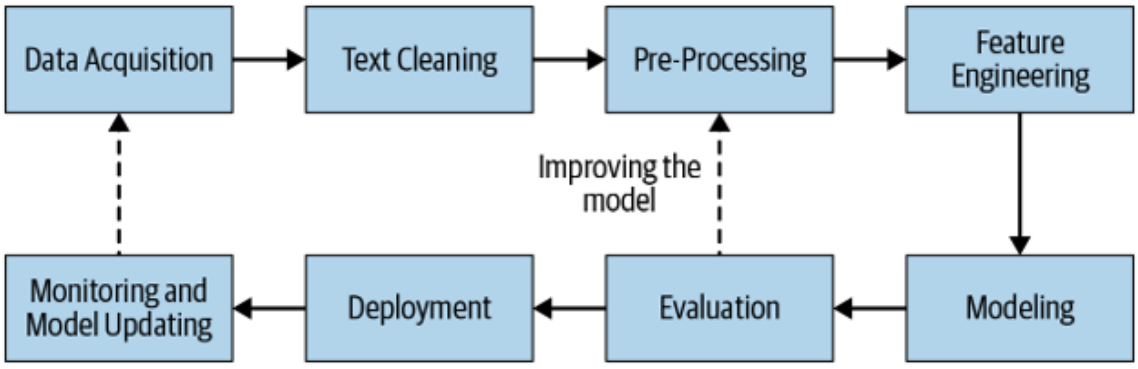


Fig 2.1.1 NLP System Pipeline

SGOU

Similarly, real-world NLP systems involve several behind-the-scenes processes. The chatbot, translator, or search engine that you use is only the final product of a much longer process.

An NLP problem is usually divided into several smaller tasks that are solved step by step. This step-by-step processing of text is called the **NLP pipeline**, as shown in Figure 2.1.1. Each stage takes the output of the previous stage and refines it further, progressively transforming unstructured and often noisy language data into a clean and structured representation that a machine learning model can learn from.

2.1.2.1 Data Acquisition (Data Collection)

Data acquisition is the first step in any NLP project. The goal of this stage is to collect text data that is relevant to the requirements of the task. The effectiveness of the subsequent stages largely depends on the quality and relevance of the data collected.

Data acquisition generally involves one of three situations:

- Data Available Internally
- Data Collection Externally
- Creation of Custom Data

Each approach is discussed in detail below.

Examples of data sources: news websites, social media platforms, books, emails, and speech recordings.

2.1.2.2 Text Cleaning

After collecting the data, the next important step is to clean it. Text cleaning involves preprocessing techniques such as removing noise, correcting errors, and maintaining a consistent format. Clean data is important because noisy or inconsistent data can reduce the accuracy and effectiveness of NLP models.

This stage focuses on removing irrelevant or disruptive elements from the text. Text cleaning typically involves:

- Removing unnecessary URLs and Hyperlinks
- Removing HTML tags
- Removing irrelevant symbols, special characters, unnecessary whitespace and formatting arti-facts
- Removing Emojis and emoticons when they are not relevant to the task
- Removing extraneous words that provide little or no useful information for the task
- Case Normalization: Converting inconsistent capitalization into a uniform form

SGOU

For example:

Suppose we have collected tweets for sentiment analysis.

Original tweet:

"Loved the new update!!  *check it out here:http://xyz.com #excited"*

Cleaned text:

"Loved the new update excited"

Here, the URL, emoji, punctuation marks, and hashtag symbol have been removed.

Many noise-removal operations can be performed using regular expressions (regex). A regular expression is a pattern used to identify, extract, remove or replace specific groups of characters in text.

For example, the pattern

`[^a-zA-Z\s]`

removes characters other than English letters and whitespace, including numbers, punctuation marks, and special characters.

Cleaning the data this way helps ensure relevant information reaches the subsequent stages of the NLP pipeline.

In the following units, these preprocessing techniques will be explored in greater detail, with hands-on examples and practical guidelines for applying them effectively.

2.1.2.3 Text Preprocessing

After cleaning, text must be further processed and structurally normalized to make it suitable for meaningful analysis. Text preprocessing involves several techniques. Some of the commonly used techniques are described below.

- **Sentence segmentation:** Dividing a piece of text into individual sentences.

Example:

"I love NLP. It is fascinating." →

["I love NLP.", "It is fascinating."]

- **Tokenization:** Breaking a sentence into smaller units called tokens, such as words or punctuation marks.

Example:

"I love NLP." →

["I", "love", "NLP", "."]

- **Stopword removal:** Removing frequently occurring words that may carry little information for a particular NLP task.

SGOU

Example:

["this", "is", "a", "useful", "book"] → ["useful", "book"]

- **Stemming/Lemmatization:** Reducing words to their base or root form.

Example:

"loved" → "love"

"loving" → "love"

- **Lowercasing:** Converting all text to lowercase to maintain consistency.

Example:

["I", "love", "NLP", "."] → ["i", "love", "nlp", "."]

- **Part-of-Speech (POS) Tagging:** Assigning a grammatical category, such as noun, verb, adjective, or adverb, to each word in a sentence.

Example:

"NLP is fascinating" → [("NLP", NOUN), ("is", VERB), ("fascinating", ADJECTIVE)]

2.1.2.4 Feature Engineering

Feature engineering is the process of converting preprocessed text into numerical representations that machine learning algorithms can process. Since most machine learning models cannot directly interpret raw text, words and sentences must first be transformed into structured numerical features.

The goal of feature engineering is to provide textual information in a form that has important linguistic characteristics, such as word frequency, semantic meaning, contextual information, and relationships between words. Effective feature representations can improve NLP model performance in tasks such as text classification, sentiment analysis, machine translation, and information retrieval.

Common feature representation techniques include:

- Bag of Words (BoW)
- Term Frequency – Inverse Document Frequency (TF-IDF)
- Word Embeddings, such as Word2Vec, GloVe, FastText
- Contextual Embeddings, such as BERT embeddings
- N-grams, which capture sequences of consecutive words or tokens
- Syntactic Features, such as those obtained through dependency parsing

2.1.2.5 Data Modeling

Data modeling is the stage of the NLP pipeline in which processed and transformed text

SGOU

data is used to train machine learning or deep learning models. At this stage, the model learns patterns, relationships, and linguistic features from the training data to perform a specific NLP task, such as text classification, sentiment analysis, machine translation, or named entity recognition.

The primary objective of data modeling is to develop a model that can accurately predict or generate outputs for new, unseen text. Depending on the application and the nature of the data, you can use different modeling approaches.

Modeling Approaches

1. Machine Learning Models

Traditional machine learning algorithms learn from patterns in numerical features such as Bag of Words (BoW) or TF-IDF.

Example models: Naive Bayes, Logistic Regression, Support Vector Machine (SVM), Decision Tree etc

Applications: Spam detection, Sentiment analysis, and text classification.

2. Deep Learning Models

Deep learning models can learn complex representations and patterns from text, reducing the need for extensive manual feature engineering.

Example models: Recurrent Neural Network (RNN), Long Short-Term Memory (LSTM), Gated Recurrent Unit (GRU)

Applications: Machine Translation, Text generation, and sequence classification.

3. Transformer-Based Models

Transformer models use attention mechanisms to capture relationships and contextual information among words or tokens in a sequence.

Example models: BERT, GPT, T5

Applications: Question Answering, Conversational AI, text summarization, text generation, and machine translation.

2.1.2.6 Evaluation

After developing a model, it is essential to evaluate how well it performs. This process is known as model assessment or model evaluation. At this stage, the model's performance is measured using appropriate evaluation metrics and, when necessary, compared with other models or baseline methods.

Model evaluation can broadly be classified into two types: intrinsic evaluation and extrinsic evaluation. Intrinsic evaluation measures how well the system performs on internal metrics like precision and recall. Extrinsic evaluation looks at real-world impact, such as how much user time is wasted on spam emails.

SGOU

1. Intrinsic Evaluation

Intrinsic evaluation measures the performance of an NLP model on a specific task using predefined datasets and evaluation metrics. It typically compares the model's predictions with a reference or gold-standard dataset.

Intrinsic evaluation helps researchers and developers identify a model's strengths and weaknesses before deploying it in real-world applications.

Some of the measures are illustrated below.

- **Classification Metrics -** Used for tasks like text classification, spam detection, and sentiment analysis, where the output belongs to predefined categories.

Common metrics include:

- Accuracy
- Precision
- Recall
- F1-Score

- **Machine Translation Metrics - BLEU (Bilingual Evaluation Understudy):**

Measures the similarity between a machine-generated translation and one or more human reference translations.

- **Summarization Metrics - ROUGE (Recall-Oriented Understudy for Gisting Evaluation):**

Measures the overlap between a machine-generated summary and one or more human-written reference summaries, usually based on words, phrases, or sequences of tokens.

2. Extrinsic evaluation

Extrinsic evaluation assesses how well an NLP model performs when it is integrated into a real-world application or a larger system. Rather than relying only on benchmark scores, it evaluates whether the model contributes effectively to the application's overall objective.

For example, a machine translation model may be evaluated not only by its BLEU score but also by how effectively its translated customer-support responses help users resolve their problems.

Extrinsic evaluation may therefore consider factors such as task completion, user satisfaction, time saved, error reduction, or overall system effectiveness.

2.1.2.7 Deployment& Monitor

In the final stage of the NLP pipeline, the trained model is deployed so that it can be used in real-world applications. Once trained and evaluated, the model can be integrated into a software system to provide predictions, classifications, recommendations, or

SGOU

generated responses to users.

Monitoring is the process of continuously or periodically checking the performance of a model after deployment. Language usage, user behaviour, and incoming data may change over time, causing the model's performance to decline. Monitoring helps detect such changes early and ensures that the NLP system continues to perform reliably.

Deployment and monitoring are important because they:

- Make the trained model available for practical use.
- Enable real-time or on-demand predictions and responses.
- Allow model performance to be tracked over time.
- Help identify problems caused by changes in incoming data.
- Support the updating or retraining of models when necessary.
- Help maintain the reliability and effectiveness of the NLP system.

Common Deployment Technologies and Platforms

NLP models can be deployed using various cloud platforms, container technologies, orchestration systems, and web services. Common examples include:

- Amazon Web Services (AWS)
- Microsoft Azure
- Google Cloud Platform (GCP)
- Docker
- Kubernetes
- Web services and APIs

SGOU

2.1.3 Data Acquisition

A team of researchers is developing a Natural Language Processing (NLP)-based chatbot for a hospital. The chatbot's primary objective is to answer patients' questions about appointments, symptoms, and available healthcare services. Before the chatbot can effectively process and respond to user queries, it must be trained using examples of human language and communication.

For this purpose, the researchers collect thousands of patient conversations, frequently asked questions (FAQs), medical articles, and healthcare guidelines. These language resources provide examples of real-world communication and help the chatbot learn medical terminology, recognize user intent, and generate appropriate responses.

Data acquisition in Natural Language Processing (NLP) refers to the systematic process of identifying, locating, and collecting raw text data that is relevant and sufficient for a specific NLP task. The primary goal is to create a suitable dataset that serves as the

foundation for model training and development. The quality, quantity, diversity, and relevance of the data influence the accuracy and effectiveness of an NLP application.

2.1.3.1 Data Sources

Language data for NLP applications can generally be acquired in three ways:

1. Using Data Available Internally
2. Collecting Data Externally
3. Creating Custom Data

Data Available Internally

When relevant data is already available within an organization, you can retrieve, prepare, and, when necessary, expand it for use in an NLP application. Common techniques include:

- **Data Engineering:** Data engineering involves extracting data from databases or data ware-houses, files, or other internal sources and preparing it for use in an NLP application.
- **Data Augmentation:** Data augmentation involves increasing the quantity and diversity of the existing data by creating modified versions of the original samples. Techniques may include synonym replacement, word or phrase swapping, back-translation, and controlled noise addition.

Collecting Data Externally

When internally available data is insufficient, additional data can be collected from external sources. Common external sources include the following:

a. Public Datasets and Repositories

Public datasets and repositories provide datasets that can be used for NLP research and applications.

Examples: Wikipedia dumps, Kaggle datasets, and the IMDb movie review dataset.

b. APIs and Data Services

Application Programming Interfaces (APIs) provide programmatic access to data from external platforms and services.

Examples: APIs provided by platforms such as Reddit or X can be used, subject to their access policies and permissions, to collect publicly accessible posts or other textual content.

c. Web Scraping

Extracting article titles, paragraphs, reviews, or other textual information from web pages.

SGOU

Examples: Extracting article titles, paragraphs, reviews, or other textual information from web pages.

You can experiment with Beautiful Soup, one of Python's widely used libraries for parsing HTML documents [5].

d. Multimodal Sources

Linguistic information can also be obtained from non-textual sources such as audio and images.

Audio: Speech recordings can be converted into text using automatic speech recognition (ASR) systems and used for applications such as voice assistants and speech-based NLP systems.

Images: Text contained within images or scanned documents can be extracted using Optical Character Recognition (OCR). Image captions and associated textual metadata may also serve as language data.

e. Document Extraction

Document extraction involves converting the contents of digital or scanned documents into machine-readable text.

Text can be extracted from PDF files, Word documents (.docx), scanned documents, and other file formats using document-parsing libraries. OCR is generally used when the document contains scanned or image-based text.

Creating Custom Data and Handling Limited Data

When little or no suitable data is available for an NLP project, several strategies can be used to create new data, label existing data, or make effective use of limited datasets.

a. Data Generation

Data generation involves creating new textual samples manually, through predefined rules, or with the help of language models.

Example: Generate sample customer queries for a chatbot by writing them manually or using AI-based text generation tools.

b. Transfer Learning

Transfer learning uses a model pre-trained on a large, general-purpose dataset and adapts it to a smaller, task-specific dataset.

Example*:* Start with a pre-trained BERT or GPT-2 model and fine-tune it using a small collection of medical texts for a healthcare chatbot.

c. Active Learning

Active learning allows the model to identify the most informative or uncertain examples for human annotation. This reduces the amount of data that needs manual labelling.

SGOU

Note: **Data annotation** is the process of assigning labels or tags to data so that it can be used for supervised machine learning.

Example: In a sentiment analysis task, the model selects the most uncertain tweets and asks a human annotator to label only those examples.

d. Crowdsourcing

Crowdsourcing is a method of collecting or annotating data by distributing tasks among a large group of people, usually through an online platform.

Example*:* Use a crowdsourcing platform such as Amazon Mechanical Turk to generate or label product reviews.

e. Domain Adaptation

Domain adaptation is the process of adapting a model trained on one domain so that it performs effectively in another related domain. It is particularly useful when only a small amount of labelled data is available in the target domain.

Domain adaptation can be performed using techniques such as fine-tuning, feature alignment, or adversarial training.

Example*:* A model trained on news articles can be adapted to analyze press releases because both domains contain similar forms of formal written language.

2.1.3.2 Data Storage and Organization

After collecting the data, it must be stored and organized so that it can be easily accessed, processed, and managed over time. Choosing an appropriate storage method is important because it affects data accessibility, scalability, processing efficiency, and integration with other tools used in the NLP pipeline.

Common Storage Formats and Systems

- CSV (Comma-Separated Values)

CSV is a widely used format for storing structured data. It represents data in plain text, with values typically separated by commas and records arranged in rows. CSV files are simple, lightweight, and compatible with most data-processing tools.

Best suited for: Labelled datasets with fixed columns, such as a sentiment analysis dataset containing "text" and "label" columns.

- **JSON (JavaScript Object Notation)**

JSON stores information as key-value pairs and can represent nested or hierarchical structures. It is particularly useful for semi-structured data in which different records may contain different fields.

Best suited for: API responses, social media data, and nested textual information.

- **XML (Extensible Markup Language)**

XML uses tags to represent structured and hierarchical information. It has been widely

SGOU

used for linguistic annotation, document representation, and structured NLP corpora.

Best suited for: Annotated corpora and structured linguistic resources

- **Plain Text Files**

Plain text files (.txt) store textual content without a predefined tabular or hierarchical structure. They are useful for preserving raw text before further processing or organization.

Best suited for: Raw text corpora, document collections, and some forms of language-model training data.

- **SQL Databases**

Relational database systems such as MySQL and PostgreSQL organize data into structured tables with defined relationships between them. They support efficient searching, filtering, updating, and retrieval of large datasets using Structured Query Language (SQL).

Best suited for: Large, structured datasets involving multiple related tables, complex queries, and frequent data retrieval.

2.1.3.3 Data Quality Assessment

Data quality assessment involves evaluating collected data before preprocessing and model training. High-quality data improves the accuracy, reliability, and overall performance of Natural Language Processing (NLP) models, whereas poor-quality data may introduce errors, inconsistencies, and bias.

Data quality is commonly assessed across several dimensions, including:

- **Completeness**: The extent to which sufficient and necessary data is available.
- **Accuracy**: The degree to which the data and its associated labels are correct.
- **Consistency**: The uniformity of data representation, formatting, and labelling.
- **Balance**: The extent to which relevant classes or categories are adequately represented.
- **Relevance**: The suitability of the collected data for the intended NLP task.
- **Noise level**: The presence of irrelevant, incorrect, duplicated, or corrupted information.

Assessing these dimensions helps ensure that the dataset is suitable for developing robust, reliable, and effective NLP applications.

2.1.3.4 Ethical and Legal Considerations

Ethical and legal considerations are important during data acquisition, particularly when the collected data contains personal, sensitive, or copyrighted information.

SGOU

Data collected directly from individuals should be obtained with appropriate informed consent and, where necessary, should be anonymized or de-identified to protect privacy. Similarly, text collected from websites, books, publications, or online platforms must comply with applicable copyright, licensing, and terms-of-use requirements.

Privacy regulations, such as the General Data Protection Regulation (GDPR), impose requirements on how personal data is collected, stored, processed, and shared.

Beyond legal compliance, datasets should also be examined for demographic, cultural, social, and linguistic bias. Unrepresentative or biased datasets may lead to NLP systems that perform inconsistently or unfairly across different groups, languages, or communities.

Therefore, ethical data acquisition should aim to ensure privacy, fairness, transparency, representativeness, and responsible use of data.

Recap

- The NLP system pipeline provides a structured approach for processing human language and developing NLP applications.
- The NLP pipeline consists of seven broad stages: data acquisition, text cleaning, pre-processing, feature engineering, modeling, evaluation, and deployment with monitoring.
- Data acquisition is the foundational stage of the NLP pipeline. It involves identifying and collecting relevant language data, and the quality of the collected data can significantly influence model performance.
- Text cleaning removes noise, errors, and irrelevant elements from raw text, while text pre-processing transforms the cleaned text into a form suitable for computational analysis.
- Feature engineering converts textual information into numerical representations that can be processed by machine learning models.
- Data modeling enables NLP systems to learn patterns and relationships from language data and perform tasks such as text classification, sentiment analysis, machine translation, and question answering.
- Model evaluation measures the performance, accuracy, reliability, and effectiveness of trained models before and after deployment.
- Deployment makes trained NLP models available for real-world applications, while monitoring helps ensure that they continue to perform effectively over time.
- Updating or retraining models with appropriate new data can help NLP systems adapt to changes in language usage, user behavior, and incoming data.

SGOU

- NLP data can be obtained from various sources, including books, news articles, social media, customer reviews, public datasets, audio recordings, images, and digital documents.
- Common methods of obtaining data include APIs, web scraping, crowdsourcing, document extraction, data generation, and the use of existing internal or public datasets.
- Proper data storage, organization, and quality assessment help ensure that collected datasets are accessible, consistent, relevant, and suitable for model development.
- Ethical and legal considerations in data acquisition include obtaining informed consent when required, protecting personal information, respecting copyright and licensing requirements, complying with applicable privacy regulations such as the GDPR, and assessing datasets for demographic and linguistic bias.

Objective Type Questions

1. Which stage of the NLP pipeline involves collecting text, speech, or other language data from various sources?
2. Which stage of the NLP pipeline is responsible for measuring the performance of a trained model?
3. Which stage of the NLP pipeline immediately follows Feature Engineering?
4. Name any two feature engineering techniques used in NLP.
5. Which data collection method involves outsourcing data collection or labelling tasks to a large group of people, often through online platforms?
6. What is the process of assigning labels or tags to data for supervised machine learning called?
7. Which process converts text into numerical representations that machine learning algorithms can process?
8. What is the primary purpose of monitoring a deployed NLP model?
9. Name any two formats or storage systems that can be used to store NLP datasets.
10. What are the two main activities associated with the final stage of the NLP pipeline?

SGOU

11. Which pre-processing technique divides a block of text into individual sentences?

12. Which technique is commonly used to extract textual information directly from websites during external data collection?

13. What does regex stand for, and what is its primary use in text cleaning?

14. What are the six key dimensions used to assess data quality in an NLP pipeline?

15. Which type of evaluation measures the performance of an NLP system based on its effectiveness in a real-world task or application?

Answers to Objective Type Questions

1. Data Acquisition
2. Model Evaluation
3. Data Modeling
4. Bag of Words (BoW), Embedding
5. Crowdsourcing
6. Data Annotation
7. Feature Engineering
8. To continuously track model performance and detect performance degradation
9. CSV, JSON, XML, plain text, or SQL databases
10. Deployment and Monitoring.
11. Sentence segmentation
12. Web scraping tools (using tools such as Beautiful Soup or Scrapy)
13. Regular expressions (Regex): A pattern-matching syntax used to identify, remove, or replace specific character sequences in text
14. Completeness, Accuracy, Consistency, Balance, Relevance, and Noise Level
15. Extrinsic evaluation

SGOU

Assignments

1. Explain the importance of data modeling in the NLP pipeline with suitable examples.

2. Scrape customer reviews from a website using Beautiful Soup and prepare a dataset for sentiment analysis.

   (*Hint - Refer to the Beautiful Soup documentation:https://www.crummy.com/software/BeautifulSoup/bs4/doc/)*

3. Draw a neat diagram of the NLP system pipeline and explain the function of each stage with suitable examples.

4. Compare text cleaning and text preprocessing. Explain the differences between them with appropriate examples.

5. Explain the concept of data acquisition in NLP. Discuss the different sources and methods of data acquisition with suitable examples.

Reference

1. S. Bird, E. Klein, and E. Loper, *Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit*. O'Reilly Media, 2009.

2. S. Ghosh and D. Gunning, *Natural Language Processing Fundamentals: Build Intelligent Applications That Can Interpret the Human Language to Deliver Impactful Results*. Packt Publishing, 2019.

3. W. Wagner, "Steven Bird, Ewan Klein and Edward Loper: *Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit*," book review, 2010.

4. S. Vajjala, B. Majumder, A. Gupta, and H. Surana, *Practical Natural Language Processing: A Comprehensive Guide to Building Real-World NLP Systems*. O'Reilly Media, 2020.

5. Beautiful Soup, "Beautiful Soup Documentation." [Online]. Available: https://www.crummy.com/software/BeautifulSoup/bs4/doc/ [Accessed: 6 September 2026].

SGOU

Suggested Reading

1. S. Strassel, C. Christianson, J. McCary, W. Staderman, J. Olive, et al., "Data Acquisition and Linguistic Resources," in *Handbook of Natural Language Processing and Machine Translation: DARPA Global Autonomous Language Exploitation*. New York, NY: Springer, 2011, pp. 1–131.

2. W. McKinney, *Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython*. O'Reilly Media, 2012.

SGOU

Unit 2

Text Preprocessing for NLP

Learning Outcomes

After completing this unit, learner will able to:

- explain the purpose and importance of text pre-processing in the NLP pipeline
- apply core pre-processing techniques including cleaning, tokenization, stemming, lemmatization, and normalization
- implement these techniques using Python and appropriate NLP libraries
- select an appropriate pre-processing technique for a given NLP task
- explain how inadequate or inappropriate pre-processing can affect the performance of downstream NLP models

Prerequisites

Imagine a chef receiving a bag of unwashed, unpeeled, and unevenly cut vegetables and being asked to prepare a refined dish immediately. No matter how skilled the chef is,the quality of the final dish depends greatly on the quality and preparation of the ingredients.

In the same way, text data often comes in a raw and unstructured form, containing noise, inconsistencies, and elements that may not contribute useful information. Before an NLP model can effectively learn from language, the text must be cleaned, standardized, transformed into a suitable format. This process, known as **text pre-processing**, is an important foundation of NLP.

In the previous unit, you learned about the NLP pipeline and saw that text pre-processing is one of its key stages. In this unit, we will examine a major pre-processing technique in detail, including what each technique does, why it is important, how it can be implemented using Python, and what problems may arise if it is skipped or applied incorrectly.

Before studying this unit, learners should have a basic understanding of the NLP pipeline, Python programming, strings and lists in Python, and the basic concepts of text data.



SGOU

Keywords

Text Cleaning, Tokenization, Stemming, Lemmatization, Stop Words, Text Normalization, Regular Expressions (Regex), Subword Tokenization

Discussion

2.2.1 Introduction

Imagine a major airline's customer service team receiving thousands of tweets every day. These messages may include complaints, questions, compliments, and general comments, all expressed in different styles. When engineers build a sentiment classifier to automatically categorize these messages, they may find that the model's performance depends not only on the algorithm but also on the quality of the input text.

Social media text often contains airline-specific hashtags, abbreviations, repeated punctuation marks, emojis, spelling variations, and inconsistent capitalization. Such irregularities can make it difficult for an NLP model to process the text effectively. In such cases, improving text preprocessing can be more beneficial than choosing a more complex algorithm.

This example illustrates an important principle in real-world NLP applications: the quality of an NLP system's output is strongly influenced by the quality of its input. Raw text collected from sources such as social media, customer emails, scanned documents, and web pages is often noisy, inconsistent, and unstructured. Text pre-processing bridges this gap by transforming raw text into a cleaner, more consistent, structured form that NLP models can process effectively.

This unit focuses on five major text pre-processing methods: text cleaning, tokenization, stemming, lemmatization, and normalization. For each technique, the unit explains the basic concept, its importance, and its implementation using Python and commonly used NLP libraries. Practical examples are also included to help learners understand how these techniques are applied in real-world NLP tasks.

2.2.2 Text Extraction and Cleanup

Before applying NLP techniques, the raw text must first be extracted from its source or format and prepared for further processing. **Text extraction and cleanup** involve retrieving textual content from the source, removing unnecessary elements such as markup or metadata, and ensuring that the text is stored using an appropriate character encoding.

Although this stage may involve relatively little linguistic processing, it is often one of the most time-consuming parts of an NLP project. Errors introduced during text extraction and cleanup can affect every subsequent stage of the NLP pipeline.

2.2.2.1 HTML Parsing and Cleanup

Text collected from websites often contains HTML tags and other web-page elements

SGOU

that may not be required for NLP analysis. These elements must be identified and removed so that only the relevant textual content remains.

The Python library **Beautiful Soup** is commonly used for parsing HTML documents. It enables users to navigate the structure of a web page and extract text from specific HTML elements or tags.

So far, we have discussed HTML parsing conceptually. The best way to understand how it works is to apply it to real web data. Let us now write and run a simple web-scraping and HTML-parsing program.

Before beginning, make sure that Python is available in one of the following environments:

- Google Colab
- Jupyter Notebook
- **A Python IDE**, such as PyCharm or Visual Studio Code (VS Code)

Before running the program, install the required libraries.

Required Libraries:

- **Beautiful Soup (beautiful-soup4)** – for parsing HTML and extracting required text.
- **Requests (requests)** – for retrieving the HTML content of a webpage.

Code Snippet 2.2.1demonstrates how to extract book titles from:

http://books.toscrape.com/,

Books to Scrape is a website designed specifically for practicing web scraping, making it suitable for learning and experimenting with basic scraping techniques.

```
from bs4 import BeautifulSoup
import requests

# Books to Scrape - a website built specifically
# for web scraping practice
url = "http://books.toscrape.com/"

headers = {"User-Agent": "Mozilla/5.0"}
response = requests.get(url, headers=headers)
soup = BeautifulSoup(response.text, "html.parser")

# Extract book titles
books = soup.find_all("h3")
print("Book Titles Found:")
for i, book in enumerate(books[:5], 1):
    print(f"{i}. {book.get_text()}")
```

Code Snippet 2.2.1

SGOU

2.2.2.2 Unicode Normalization

Text data, especially from multilingual sources, social media, and web pages, may contain characters that can be represented in different Unicode forms. Although these characters may look identical on the screen, their underlying Unicode representations can differ.

Unicode normalization converts such variations into a consistent representation so that text can be processed and compared reliably. This is particularly important when working with accented characters, multilingual text, special symbols, and characters copied from different sources. For most general NLP applications, NFC is commonly used to maintain a consistent representation of characters.

It is also important to distinguish Unicode normalization from character encoding. Encoding schemes such as UTF-8 determine how Unicode characters are stored as bytes, whereas Unicode normalization determines how equivalent characters are represented internally.

Ignoring Unicode or encoding inconsistencies can lead to problems such as incorrect text comparison, tokenization errors, or unexpected characters in later processing stages.

Code Snippet 2.2.2 demonstrates how Unicode normalization can be performed in Python.

```
text = 'I feel really 😡. GOGOGO!! 💪💪💪  🤣🤣 ĂÆĎĞG'
## encode the strings in bytes
encoded = text.encode("utf-8")
```

Code Snippet 2.2.2

2.2.2.3 Spelling Correction

Think about the last time you quickly typed a message on your phone. You may have made a typo, such as writing “Hllo” instead of “Hello” or “promise” instead of “promise.” Now imagine an NLP system processing thousands of such messages every minute. Spelling errors are common in real-world text, especially in search queries, mobile chats, and social media, where users often type quickly and may not follow standard spelling conventions.

Spelling correction is the process of identifying misspelled words and replacing them with their most likely correct forms. Correcting spelling errors can improve text consistency and may help NLP models interpret the intended meaning more accurately.

```
from textblob import TextBlob

#  Code Snippet 2.2.3  – Spelling Correction using TextBlob
text = "Hllo wrld! I pronise I will not brek the silence again"
corrected = str(TextBlob(text).correct())
```

Code Snippet 2.2.3

SGOU

A simple way to perform spelling correction in Python is to use the TextBlob library. TextBlob provides built-in methods that can suggest corrections for misspelled words based on its language resources.

2.2.2.4 System-Specific Error Correction

In many real-world NLPprojects, text is extracted from sources such as PDF documents, scanned images, or speech recognition systems. Each source can introduce specific types of errors that may not be corrected effectively using standard text-cleaning techniques.

PDF Documents

Text extracted from PDFs using tools like PyPDF or PDFMiner may contain errors caused by complex layouts, broken lines, missing spaces, merged characters, or incorrectly ordered text. In some cases, certain sections may also be omitted during extraction.

Such errors can be particularly problematic for NLP tasks that depend on complete and correctly structured sentences, such as relation extraction, information extraction, and document classification.

Scanned Documents

Scanned documents contain text in image form and therefore require Optical Character Recognition (OCR) to convert the content into machine-readable text. Tools such as **Tesseract OCR**, often accessed through the Python library pytesseract, are commonly used for this purpose.

However, OCR output may contain errors depending on factors such as image resolution, scan quality, font style, document layout, and background noise. These errors can be corrected using spelling-correction tools, rule-based methods, or context-aware language models.

Speech Recognition Output

Text generated by an Automatic Speech Recognition (ASR) system may contain errors caused by background noise, pronunciation differences, accents, dialects, slang, or unfamiliar vocabulary.

Such errors can be reduced using techniques such as spelling correction, contextual language models, vocabulary adaptation, and post-processing rules.

In conclusion, there is no single method that can handle every type of text-extraction error. The appropriate correction technique depends on the source and characteristics of the data. Careful handling of these errors is important because problems introduced during extraction can affect all subsequent stages of the NLP pipeline.

2.2.3 Preprocessing

If text cleaning already removes noise, why is pre-processing still necessary? Text cleaning removes unwanted elements such as HTML tags, URLs, irrelevant symbols, and formatting artifacts. However, even clean text may not yet be in a form suitable for

SGOU

NLP analysis. It may still need to be segmented, tokenized, standardized, or normalized before it can be effectively processed by NLP algorithms. Text pre-processing bridges this gap by transforming cleaned text into a more structured and consistent representation suitable for further analysis and model development. Common pre-processing operations include tokenization, stemming, lemmatization, stop-word removal, and normalization.

It is important to remember that not every pre-processing step is required for every NLP task, nor must the steps always be performed in the same order. The choice and sequence of techniques depend on the characteristics of the dataset, the language being processed, and the requirements of the NLP application.

A good practice is to first examine the data and clearly understand the task requirements. Based on this understanding, an appropriate sequence of pre-processing operations can then be selected.

The following sections introduce some of the commonly used text pre-processing techniques.

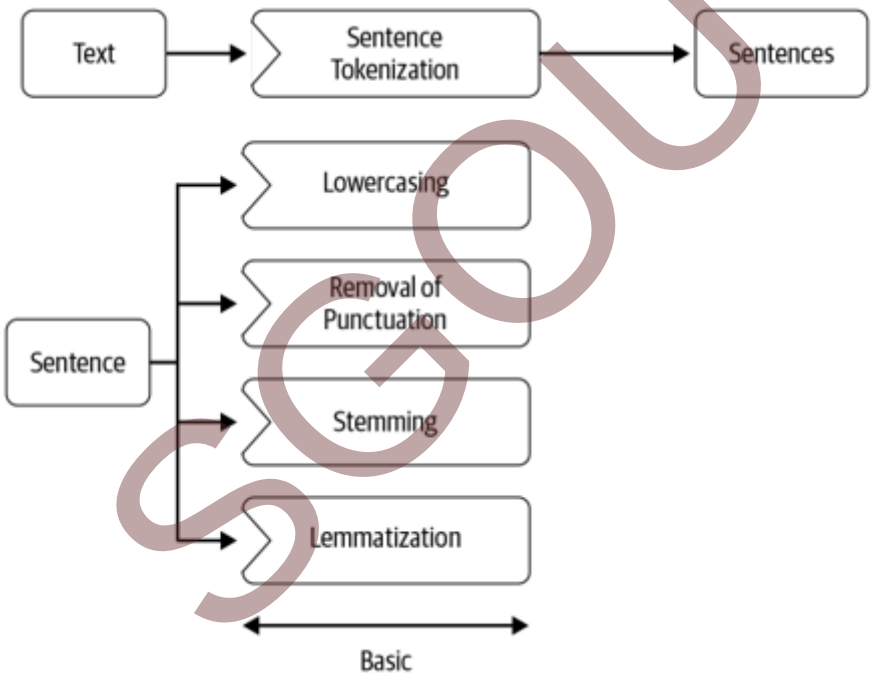


Fig 2.2.1 Common pre-processing steps

2.2.4 Tokenization

A child learning to read does not process an entire paragraph at once. Instead, the child gradually learns to recognize smaller units such as words and sentences before understanding larger structures. Tokenization follows a similar idea: before text can be analyzed computationally, it is often divided into smaller meaningful units.

Tokenization divides text into smaller units called tokens. Depending on the NLP task and the method used, tokens may represent sentences, words, subwords, or characters.

SGOU

These tokens provide structured units that can be processed by NLP algorithms and models.

```
import nltk
nltk.download('punkt')
from nltk.tokenize import word_tokenize, sent_tokenize
from transformers import AutoTokenizer
import spacy

nlp = spacy.load("en_core_web_sm")
text = "Natural Language Processing is fascinating! Let us explore it."

# 1. Word Tokenization – NLTK
print("1. Word Tokenization (NLTK):")
print(word_tokenize(text))

# 2. Sentence Tokenization – NLTK
print("\n2. Sentence Tokenization (NLTK):")
print(sent_tokenize(text))

# 3. Word Tokenization – spaCy
print("\n3. Word Tokenization (spaCy):")
doc = nlp(text)
print([token.text for token in doc])

# 4. Subword Tokenization – BERT
print("\n4. Subword Tokenization (BERT):")
tokenizer = AutoTokenizer.from_pretrained("distilbert-base-uncased")
print(tokenizer.tokenize(text))

# 5. Character Tokenization
print("\n5. Character Tokenization:")
print(list(text.replace(" ", "")))
```

Code Snippet 2.2.4

Code Snippet 2.2.4 demonstrates different tokenization methods in a single Python program, allowing learners to compare their outputs and understand how each method works.

Tokenization is not performed in the same way for every application. Different languages, tasks, and model architectures may require different tokenization strategies. Several types of tokenization are commonly used in NLP, each with its own characteristics and applications. The major types are discussed below.

The examples presented in this section provide an introduction to commonly used tokenization methods. However, tokenization is a broad and continuously evolving area of NLP. Learners who wish to explore the topic further can refer to the official documentation of widely used NLP libraries and frameworks, including:

- NLTK

SGOU

- spaCy
- Stanza
- Hugging Face Transformers

These libraries provide different approaches to tokenization and text pre-processing and support a wide range of languages. Exploring them can help learners understand how tokenization methods vary across languages, applications, and modern NLP models.

Types of Tokenization

1. Character Tokenization

Character tokenization splits text into individual characters. It is the most fine-grained form of tokenization. It can be useful for noisy text, languages with complex word boundaries, and tasks in which character-level patterns are important.

Example:

"NLP" → ['N', 'L', 'P']

2. Word Tokenization

Word tokenization divides text into individual words and, depending on the tokenizer, punctuation marks. It is one of the most intuitive forms of tokenization.

Example:

"Natural Language Processing is fascinating! Let us explore it." →

['Natural', 'Language', 'Processing', 'is', 'fascinating', '!', 'Let', 'us', 'explore', 'it', '.']

3. Sentence Tokenization

Sentence tokenization, also called sentence segmentation, divides a block of text into individual sentences. It is useful for NLP tasks that operate at sentence level.

Example:

"Natural Language Processing is fascinating! Let us explore it." →

['Natural Language Processing is fascinating!', 'Let us explore it.']

4. Subword Tokenization

Subword tokenization divides words into smaller meaningful units that are generally larger than individual characters. It helps NLP models handle rare, unfamiliar, or morphologically complex words efficiently.

Subword tokenization is widely used in modern Transformer-based models such

SGOU

as BERT, GPT, and T5. Different models may use different subword tokenization algorithms, including WordPiece, Byte-Pair Encoding (BPE), and SentencePiece.

Example using WordPiece-style tokeni-zation:

“tokenization” → [‘token’, ‘##ization’]

The prefix ## indicates that “ization” is a continuation of the preceding subword in WordPiece tokenization.

Challenges of Tokenization

1. A period is not always used to indicate the end of a sentence. It may also appear in abbreviations such as **“Dr.”** and **“Ph.D.”**, or in decimal numbers such as **“3.14.”**

2. Contractions like “don’t” can be difficult to tokenize because they may be treated either as a single token or split into multiple tokens, depending on the tokenizer.

3. Hyphenated expressions, such as “state-of-the-art,” may be treated as a single token or split into several smaller tokens.

4. Multiword expressions such as **“New York”** or **“ice cream”** may lose important semantic information if their component words are treated independently.

5. Languages with rich morphology, such as **Malayalam**, can be more challenging to tokenize. A single word may contain several grammatical elements, such as tense, negation, case, or other morphological information. In such cases, subword tokenization may be more effective than simple word-level tokenization.

6. Social media text introduces additional challenges through hashtags, emojis, abbreviations, creative spellings, repeated characters, and informal language, which may not be handled well by standard tokenizers.

2.2.5 Stemming

Stemming is a rule-based process that reduces words to a base or root form, called a stem, by removing suffixes and prefixes. This allows different forms of a word to be treated as related or equivalent during text processing.

Stemming is a heuristic process that applies predefined rules without considering whether the resulting stem is a valid dictionary word. As a result, stemming may sometimes produce forms that are linguistically incomplete or not real words.

Code Snippet 2.2.5 demonstrates different stemming methods in a single program, allowing learners to compare their outputs. Figure 2.2.2 illustrates the results produced

SGOU

by the stemming techniques.

```
from nltk.stem import LancasterStemmer, SnowballStemmer, PorterStemmer

# Create stemmer objects
lancaster = LancasterStemmer()
snowball = SnowballStemmer("english")
porter = PorterStemmer()

# List of words
words = ["running", "happiness", "studies"]

# Apply stemming
lancaster_stems = [lancaster.stem(word) for word in words]
snowball_stems = [snowball.stem(word) for word in words]
porter_stems = [porter.stem(word) for word in words]

# Print results
print("Lancaster Stemmer:", lancaster_stems)
print("Snowball Stemmer :", snowball_stems)
print("Porter Stemmer   :", porter_stems)
```

```
Lancaster Stemmer: ['run', 'happy', 'study']
Snowball Stemmer : ['run', 'happi', 'studi']
Porter Stemmer   : ['run', 'happi', 'studi']
```

Code Snippet 2.2.5



Fig 2.2.2: Visualization of the stemming process

Example:

"running" → "run",

"happiness" → "happi" (*not a real word, but a valid stem)*

SGOU

2.2.5.1 Types of Stemmers

1. Porter Stemmer

The Porter Stemmer, developed by Martin Porter in 1980, is one of the most widely used stemming algorithms for English. It applies a sequence of predefined rules to remove or modify suffixes and reduce words to their stems.

It is simple, efficient, and commonly used in information retrieval and basic NLP applications.

Example:

'agreed'→'agree'

2. Snowball Stemmer

The Snowball Stemmer, also known as the Porter2 Stemmer for English, is an improved and more systematic version of the original Porter stemming algorithm. It supports several languages and provides language-specific stemming rules.

Example:

'happily'→happili'

Snowball is particularly useful when stemming is required for multiple languages.

3. Lancaster Stemmer

The Lancaster Stemmer is a relatively aggressive stemming algorithm. It repeatedly applies stemming rules until no further reduction is possible. Because of its aggressive nature, it may sometimes remove too much information, a problem known as over-stemming.

Example:

'happily'→'happy'

Challenges of Stemming

- **Over - stemming:** Removing too much from a word, causing words with different meanings to be incorrectly reduced to the same stem.
- **Under - stemming:** Removing too little from a word, causing related word forms to remain as separate stems.
- **Non - word stems:** Stemming may produce forms that are not valid dictionary words, which can make interpretation difficult and may affect down-stream tasks that rely on meaningful vocabulary.
- **Language limitations:** Many commonly used stemming algorithms are designed mainly for English and may perform poorly on morphologically rich languages, where words can contain complex grammatical information.

SGOU

2.2.6 Lemmatization

Lemmatization is a linguistically informed process that reduces a word to its dictionary base form, called a lemma. Unlike stemming, which mainly removes prefixes or suffixes using predefined rules, lemmatization considers linguistic information such as the word's part of speech (POS) to determine the appropriate base form.

Lemmatization generally produces valid dictionary words and therefore provides more meaningful results than stemming, although it may require more linguistic information and processing.

Code Snippet 2.2.6 demonstrates lemmatization using NLTK's WordNet-Lemmatizer. WordNet is a lexical database of English that provides information about words and their relationships and is used by the lemmatizer to identify appropriate base forms.

```
import nltk
nltk.download('wordnet')
from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()

# Lemmatizing as verb
print(lemmatizer.lemmatize("running", pos="v"))
# Output: run

# Lemmatizing as adjective
print(lemmatizer.lemmatize("better", pos="a"))
# Output: good

# Lemmatizing as noun
print(lemmatizer.lemmatize("studies", pos="n"))
# Output: study
```

Code Snippet 2.2.6

Example:

"running" → "run"

"better" → "good" *(when the appropriate part of speech is specified)*

"cars" → "car"

2.2.7 Normalization

Even after cleaning and tokenization, stemming, lemmatization, text may still contain inconsistencies that unnecessarily increase vocabulary variation. For example, "NLP," "nlp," and "N.L.P." may refer to the same concept but could be treated as different tokens if such variations are not standardized.

Text normalization is the process of transforming text into a consistent and standardized

SGOU

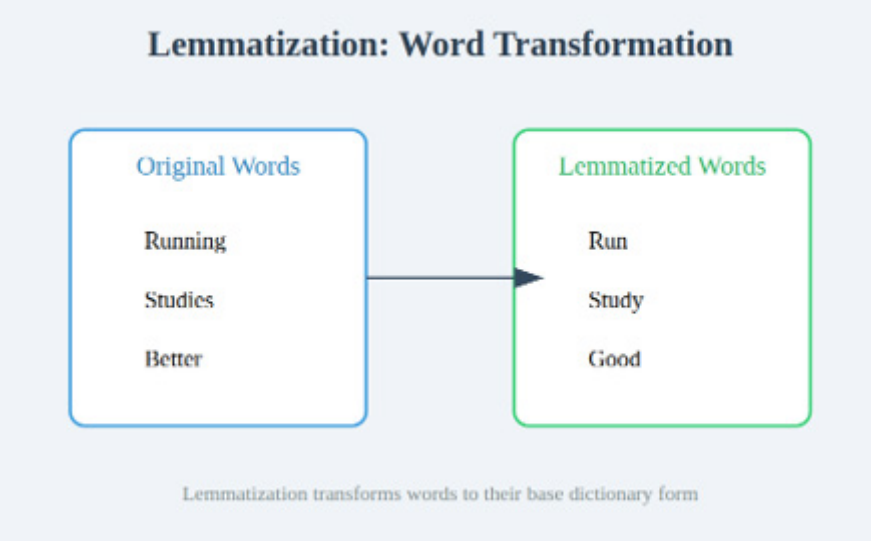


Fig 2.2.3 Visualization of lemmatization and conversion of words to their base dictionary forms

form to reduce unnecessary variation and improve the consistency of textual data. The normalization techniques used depend on the nature of the data and the requirements of the NLP task.

Code Snippet 2.2.7 demonstrates some of the normalization techniques described below.

```python
def normalize_text(text):
    # Unicode normalization
    text = unicodedata.normalize('NFKD', text)
    text = text.encode('ascii', 'ignore').decode('ascii')

    # Expand contractions
    text = contractions.fix(text)

    # Lowercase
    text = text.lower()

    # Spelling correction
    text = str(TextBlob(text).correct())

    # Remove special characters
    text = re.sub(r'[^a-zA-Z\s]', '', text)

    # Remove extra whitespace
    text = re.sub(r'\s+', ' ', text).strip()

    # Tokenize
    tokens = word_tokenize(text)

    # Remove stop words
    tokens = [t for t in tokens if t not in stop_words]

    return tokens
```

Code Snippet 2.2.7

SGOU

2.2.7.1 Lowercasing

Lowercasing converts uppercase characters into lowercase characters to reduce variations caused by capitalization.

Example:

"Natural Language Processing" → "natural language processing"

Lowercasing is commonly used in many NLP applications, although it may not be appropriate when capitalization carries useful information, such as in named entity recognition or when using case-sensitive language models.

2.2.7.2 Expanding Contractions

Contraction expansion converts shortened forms into their full forms to improve consistency.

Example:

"don't" → "do not"

"I'm" → "I am"

2.2.7.3 Spelling Correction

Spelling correction identifies and corrects typographical errors that might otherwise create unnecessary vocabulary variations.

Example:

"recieve" → "receive"

Automatic spelling correction should be applied carefully because it may incorrectly modify names, abbreviations, technical terms, or domain-specific words.

2.2.7.4 Removing Stop Words

Stop-word removal eliminates frequently occurring words that may contribute little information to a particular NLP task.

Example:

["this", "is", "a", "good", "book"] → ["good", "book"]

Stop-word removal is task-dependent and should not be applied automatically in every NLP application.

2.2.7.5 Unicode Normalization

Unicode normalization converts equivalent Unicode character representations into a consistent form. This is especially useful when processing multilingual text or text collected from different digital sources.

SGOU

Common Unicode normalization forms include NFC, NFD, NFKC, and NFKD.

Example:

Input: I can’t believe NLP is sooo fascinating!!

It’s one of the BEST fields in café culture

Output: [‘believe’, ‘nap’, ‘soon’, ‘fascinating’, ‘one’, ‘best’, ‘fields’, ‘cafe’, ‘culture’]

Note that NLP → nap and sooo → soon are unintended spelling corrections produced by the automatic spelling-correction step.

Recap

- Text pre-processing transforms raw and noisy text into a cleaner, more consistent, and structured form suitable for NLP analysis and model development.
- Text cleaning removes unwanted elements such as HTML tags, URLs, irrelevant special characters, emojis, and extra whitespace using tools such as Beautiful Soup and Python’s re module.
- Tokenization divides text into smaller units called tokens, which may be words, subwords, sentences, or characters, depending on the NLP task and tokenizer.
- Word tokenization is a commonly used approach, while subword tokenization is widely used in Transformer-based models and can be useful for morphologically rich languages and rare words.
- Stemming reduces words to stems using rule-based transformations. It is computationally simple but may produce forms that are not valid dictionary words.
- Lemmatization reduces words to their dictionary base forms, called lemmas, using linguistic information such as part of speech. It generally produces more linguistically meaningful forms than stemming.
- Text normalization reduces unnecessary variation in text through techniques such as lowercasing, contraction expansion, spelling correction, and Unicode normalization.
- Stop-word removal eliminates frequently occurring words that may contribute little information to a particular NLP task. Its usefulness depends on the application.

SGOU

- Python libraries such as NLTK and spaCy provide a wide range of tools for tokenization, normalization, stemming, lemmatization, and other text pre-processing tasks..
- A typical pre-processing workflow may include steps such as: clean → normalize → tokenize → remove stop words → stem or lemmatize. However, the exact sequence depends on the dataset, language, NLP task, and model being used.
- The choice of pre-processing techniques should therefore be based on the requirements of the specific NLP application.

Objective Type Questions

1. Which Python library is commonly used to remove HTML tags and extract text from scraped web content?
2. What does OCR stand for in the context of text extraction?
3. Which Python library is used for OCR-based text extraction from scanned images?
4. What is the process of splitting text into smaller units such as words or sentences called?
5. What is the dictionary base form of a word produced by lemmatization called?
6. Which pre-processing technique removes common, low-information words such as “the,” “is,” and “and” from text?
7. Which Python NLP library provides the PorterStemmer, SnowballStemmer, and LancasterStemmer classes?
8. Which stemmer supports multiple languages in addition to English?
9. What is the output of applying the Porter Stemmer to the word “studies”?
10. Which part of speech should be specified when lemmatizing the word “better” to obtain the lemma “good”?
11. What is the main difference between a stem and a lemma?
12. What is the output of word-tokenizing the sentence “NLP is fascinating!”?

SGOU

13. Which tokenization type splits text into individual characters?

14. Which built-in Python module can be used for Unicode normalization?

15. What does stopwords words ('english') return in NLTK?

16. Which Python NLP library provides tokenization and lemmatization as part of a single processing pipeline?

Answers to Objective Type Questions

1. Beautiful Soup
2. Optical Character Recognition
3. Pytesseract
4. Tokenization
5. Lemma
6. Stop word removal
7. NLTK
8. Snowball Stemmer
9. studi
10. Adjective
11. A stem may not be a real word, whereas a lemma is the linguistically valid base form of a word.
12. ['NLP', 'is', 'fascinating', '!']
13. Character tokenization
14. Unicode data
15. A list of common English stop words provided by NLTK
16. spaCy

SGOU

Assignments

1. Explain text cleaning and implement a Python function to remove HTML tags, URLs, special characters, and extra whitespace from a given text sample.

2. Compare word tokenization, subword tokenization, and character tokenization. For each method, provide a code example using NLTK or Hugging Face Transformers and explain a real-world scenario in which it would be appropriate.

3. Differentiate between stemming and lemmatization with suitable examples. Implement both techniques using Python and discuss the trade-offs between computational efficiency and linguistic accuracy.

4. Explain text normalization and implement at least four normalization techniques in Python. Demonstrate the effect of each technique using suitable before-and-after examples.

5. Build a complete text pre-processing pipeline in Python by applying the techniques covered in this unit in an appropriate sequence. Test the pipeline on a sample of noisy social media text and explain the output produced at each stage.

Reference

1. S. Vajjala, B. Majumder, A. Gupta, and H. Surana, *Practical Natural Language Processing: A Comprehensive Guide to Building Real-World NLP Systems*. O'Reilly Media, 2020.

2. S. Ghosh and D. Gunning, *Natural Language Processing Fundamentals: Build Intelligent Applications That Can Interpret the Human Language to Deliver Impactful Results*. Packt Publishing, 2019.

3. S. Bird, E. Klein, and E. Loper, *Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit*. O'Reilly Media, 2009.

4. W. Wagner, "Steven Bird, Ewan Klein and Edward Loper: *Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit*," book review, 2010.

SGOU

Suggested Reading

1. spaCy, “spaCy Usage Documentation.” [Online]. Available: https://spacy.io/usage [Accessed: 6 September 2026].

2. NLTK, “Natural Language Toolkit Documentation.” [Online]. Available: https://www.nltk.org/ [Accessed: 6 September 2026].

3. Great Learning, “Great Learning.” [Online]. Available: https://www.mygreatlearning.com/ [Accessed: 6 September 2026].

SGOU

Unit 3

Text Representation

Learning Outcomes

After completing this unit, learner will able to:

- explain the need for converting textual data into numerical representations for NLP tasks
- describe the Vector Space Model (VSM) and its role in representing textual documents
- distinguish between sparse and dense vector representations
- explain cosine similarity and apply it to measure similarity between text documents
- ientify and compare the major text vectorization approaches used in NLP

Prerequisites

A skilled translator does more than just simply replace words with words from another language. The translator preserves the meaning, intent, tone, and context even when the words themselves change completely. Building an NLP system involves a similar challenge: translating human language into a form that machines can understand.

In the previous units, we explored techniques such as text cleaning, tokenization, stemming, lemmatization, and normalization. However, even after these preprocessing steps, the data still consists primarily of words and tokens rather than numerical values. Most machine learning algorithms cannot directly process raw textual data; they require numerical representations that can be manipulated using mathematical operations.

Before beginning this unit, you should be familiar with the concept of a corpus, the process of tokenization, and the basic text preprocessing techniques discussed in the previous units. You should also understand why raw text must be transformed before it can be used by machine learning algorithms.

With this background, we can now focus on the central question of this unit: How can words and documents be represented numerically, and what mathematical framework allows machines to compare and process these representations?

SGOU



Keywords

Vectors, Vector Space Model (VSM), Sparse Vector, Dense Vector, Euclidean Distance, Count Vectorizer, Term Weighting

Discussion

2.3.1 Introduction

The previous units provided a foundation in NLP by introducing techniques for cleaning, tokenizing, and normalizing raw text so that it can be processed more effectively. However, even after these preprocessing steps, text still consists of words and tokens rather than numerical values, while most machine learning algorithms require numerical input. Therefore, a fundamental task in NLP is converting textual data into numerical representations.

Text representation lets computational systems perform mathematical operations on language, enabling them to measure similarity, identify patterns, and make predictions. A text representation can be as simple as assigning a unique integer to each word or as complex as representing a word or document using a vector containing hundreds of numerical values. Different representation methods provide different levels of simplicity, efficiency, and ability to capture linguistic information.

Before a machine learning model can learn effectively from textual data, words, sentences, or documents must therefore be transformed into numerical vectors that represent their occurrence, importance, context, or semantic characteristics.

In this unit, we introduce the fundamental concepts underlying text representation. We begin with the concept of vectors and examine how they can be used to represent textual data. We then discuss the Vector Space Model (VSM), which provides a mathematical framework for representing and comparing documents, followed by the distinction between sparse and dense vector representations. Measures such as cosine similarity and Euclidean distance are also introduced to explain how similarities and differences between vectors can be quantified.

Specific text vectorization techniques and their implementation will be discussed in detail in Unit 4

2.3.1.1 Why Text Representation Matters

Machine learning algorithms perform mathematical operations such as addition, multiplication, comparison, and distance calculation on numerical data. Raw text, however, consists of characters, words, and sentences that most machine learning algorithms cannot process directly. Therefore, text must first be converted into a numerical form, usually a vector, that represents useful information contained in the original text.

For example, consider a movie review classifier. A review may contain the sentence, *"This movie was fantastic!"* Before a machine learning model can determine whether

SGOU

the review expresses a positive or negative opinion, it must transform the text into numerical features. These features may represent information such as the presence of particular words, their frequency, their importance, or their context.

The choice of text representation directly influences what information a model can use. A very simple representation may fail to capture important distinctions between words such as *"happy"* and *"unhappy."* More advanced representations can preserve additional information about context and relationships between words, allowing models to make more informed predictions.

Therefore, selecting an appropriate text representation is an important step in building an effective NLP system.

2.3.1.2 Vector

Numerical data can be represented using different mathematical structures, such as scalars, vectors, matrices, and tensors.

A scalar is a single numerical value, such as 5, 2.5, or −3. A vector is an ordered one-dimensional collection of numerical values. For example, a vector such as [3,4][3, 4] contains two components. In NLP, vectors are commonly used to represent words, sentences, or documents numerically.

A matrix is a two-dimensional rectangular arrangement of numbers consisting of rows and columns. A tensor is a more general multidimensional array. Scalars, vectors, and matrices can therefore be considered tensors of increasing dimensionality: a scalar is zero-dimensional, a vector is one-dimensional, and a matrix is two-dimensional.

The following Fig 2.3.1 illustrates the basic differences among these mathematical representations.

To relate these concepts to text, consider the hierarchical structure of language:

- A letter is a single unit of an alphabet.
- A word is a sequence of letters or characters.
- A sentence is a sequence of words.
- A paragraph is a sequence of sentences.
- A document may contain one or more paragraphs, sentences, and words.

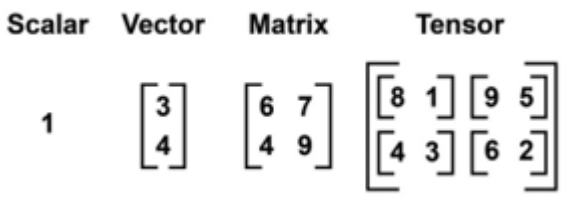


Fig 2.3.1 Illustration of scalar, vector, matrix, and tensor representations

SGOU

In NLP, these textual units can be transformed into numerical vectors so that mathematical and machine learning operations can be applied to them.

Note: ASCII and Unicode provide numerical codes for representing characters in computer systems. However, these encoding schemes primarily identify characters; they do not represent the linguistic meaning, importance, or relationships between words as NLP text-representation techniques are designed to do.

2.3.2 Vector Space Models

For machine learning algorithms to work with text data, text must first be converted into a mathematical form. Throughout this unit, we represent text units, characters, words, phrases, sentences, and documents as vectors of numbers. This approach is known as the Vector Space Model (VSM).

To enable machine learning algorithms to process textual data, text must first be represented in numerical form. Words, sentences, documents, or other textual units can be represented as vectors, where each numerical value describes a particular feature of the text. The mathematical framework used to represent and compare text in this way is known as the Vector Space Model (VSM).

A vector space is an nn-dimensional mathematical space in which each object is represented by a vector containing nn numerical values. In machine learning, each dimension generally corresponds to a feature, while the values associated with those dimensions indicate the characteristics of a particular data item.

In a traditional term-based Vector Space Model, each dimension represents a unique term in the vocabulary. A document is then represented as a vector whose values may indicate the presence, frequency, or importance of those terms. Therefore, documents containing similar term patterns tend to appear closer to one another in the vector space.

Figure 2.3.2 illustrates a simple three-dimensional vector space containing three terms. Document 1, Document 2, and the query are represented as vectors within this space. Their positions and directions depend on the values assigned to the three term dimensions. The similarity between these vectors can then be measured mathematically using measures such as cosine similarity or Euclidean distance.

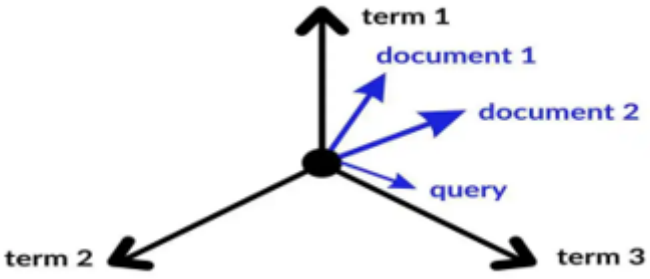


Fig 2.3.2 Three-dimensional Vector Space Model showing document and query vectors represented using three term dimensions.

SGOU

Each dimension corresponds to a unique term, while the documents and queries can be represented as a vector within it. In a traditional term-based Vector Space Model, each dimension corresponds to a unique term in the vocabulary.

Euclidean Distance

Consider two vectors in a three-dimensional vector space:

x = (x1, x2, x3)

and

y = (y1, y2, y3)

The difference between the two vectors is:

y - x = (y1 - x1, y2 - x2, y3 - x3)

The Euclidean distance between the two vectors is the L2-norm of their difference and is calculated as:

d(x, y) = sqrt[(y1 - x1)^2 + (y2 - x2)^2 + (y3 - x3)^2] **(1)**

Euclidean distance measures the straight-line distance between two points in a vector space. A smaller distance indicates the vectors are closer, while a larger distance indicates they are farther apart.

When vectors represent text, vectors that are closer may be considered more similar. However, Euclidean distance can be influenced by factors such as vector magnitude and document length. Therefore, cosine similarity is also commonly used for comparing text vectors.

2.3.2.1 Applying VSM to Text

The Vector Space Model applies the mathematical concept of vector spaces to textual data. Each text unit, such as a word, sentence, or document, is represented as a vector in a high-dimensional space. Each dimension corresponds to a feature of the text. In a traditional term-based VSM, each dimension usually represents a word or term in the vocabulary, and the value along that dimension indicates the presence, frequency, or importance of that term.

The basic idea is that texts with similar content or term usage should have similar vector representations. In contrast, texts with very different content are likely to have less similar vector representations. This allows the problem of comparing textual data to be transformed into a mathematical problem of comparing vectors using measures such as Euclidean distance or cosine similarity.

To understand this intuitively, consider a simple two-dimensional vector space in which one dimension represents the occurrence of the word “cricket” and the other represents the occurrence of the word “bowling.” A document containing both terms frequently would have relatively high values along both dimensions. A document that rarely or

SGOU

never contains these terms would appear in a different region of the vector space.

In real NLP applications, the vocabulary may contain tens of thousands of unique terms. Therefore, a document may be represented by a vector containing tens of thousands of dimensions. Although such high-dimensional spaces cannot be visualized directly, they can be processed mathematically and computationally.

2.3.2.2 Sparse and Dense Representations

One of the important distinctions in text representation is between sparse and dense vectors. These are two different ways of representing textual information numerically.

1. Sparse Representation

A sparse vector is a vector in which most of the values are zero. Sparse representations commonly arise when each dimension of a vector corresponds to a word or term in a large vocabulary.

Table 2.3.1 Comparison of Sparse and Dense representations

Feature	Sparse Vector	Dense Vector
Dimensionality	Usually high; often related to vocabulary size	Usually lower than vocabulary-based sparse vectors
Values	Most values are zero	Most or all values are non-zero
Representation	Individual dimensions often correspond to specific terms	Information is distributed across dimensions
Semantic information	Limited in basic count-based methods	Can capture semantic and syntactic relationships
Examples	Bag of Words, TF-IDF	Word2Vec, GloVe
Interpretability	Generally high because dimensions can correspond to terms	Generally lower because individual dimensions are difficult to interpret
Storage	Can be large in dimensionality, but sparse storage can be efficient	Compact because all information is represented using fewer dimensions
Similarity	Mainly reflects overlap in terms or their weights	Can identify similarity between semantically related words
Vocabulary mismatch	More sensitive to different words expressing similar meanings	Can reduce vocabulary mismatch by representing related words similarly

SGOU

For example, suppose a vocabulary contains 50,000 unique words and a particular document contains only 200 distinct words. The document may be represented by a 50,000-dimensional vector in which only about 200 positions contain non-zero values, while the remaining 49,800 positions contain zeros.

Sparse representations are generally simple and interpretable because individual dimensions can often be associated directly with specific words or terms. They are also straightforward to construct and use with many traditional machine learning algorithms.

However, sparse vectors may have very high dimensionality when the vocabulary is large. Although efficient sparse storage techniques can reduce memory requirements by storing only non-zero values, processing very high-dimensional representations can still be computationally demanding. In addition, basic sparse representations such as binary, count, and TF-IDF vectors generally capture word occurrence or importance rather than the semantic relationships between words.

2. Dense Representations

Dense representations use vectors in which most or all positions contain non-zero values. Unlike traditional sparse representations, where each dimension often corresponds to a specific word in the vocabulary, dense representations encode information across fewer dimensions.

Dense representations can help reduce vocabulary mismatch. Words with similar meanings or usage patterns can have similar vector representations even when they are different words. Thus, semantically related words tend to be located closer to each other in the vector space.

A dense vector is generally more compact and lower-dimensional than a traditional vocabulary-based sparse vector. Classical word embedding methods such as Word2Vec and GloVe commonly use vectors with dimensions such as 50, 100, 200, or 300. Modern embedding methods may use considerably higher-dimensional vectors.

In dense representations, information is distributed across vector dimensions rather than assigning a separate dimension to each word. Consequently, individual dimensions usually lack an easily interpretable meaning.

Methods such as Word2Vec and GloVe learn dense word representations from large text corpora. Words that occur in similar linguistic contexts tend to obtain similar vector representations. For example, the words “car” and “automobile” may have similar vectors because they frequently occur in similar contexts, even if they do not frequently occur next to each other.

2.3.2.3 Cosine Similarity

Once texts are represented as vectors, we need a way to measure their vector representations. One of the most commonly used measures in the Vector Space Model is cosine similarity. Cosine similarity measures the cosine of the angle between two vectors. Rather than focusing on the magnitude or length of the vectors, it primarily compares their directions in the vector space.

SGOU

Given two vectors, A and B, each containing *n* components, cosine similarity is calculated as:

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\mathbf{A}\,||_2 \mathbf{B}\,||_2} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} \mathrm{A}_i^2}\sqrt{\sum_{i=1}^{n} \mathrm{B}_i^2}} \quad \textbf{(2)}$$

where A_i and B_i represent the ith components of vectors A and B, respectively.

The numerator represents the dot product of the two vectors, while the denominator is the product of their magnitudes.

Geometrically, cosine similarity can take values from -1 to +1:

- A cosine similarity of 1 indicates that the two vectors point in the same direction and have maximum cosine similarity.
- A cosine similarity of 0 indicates that the vectors are perpendicular, or orthogonal, to each other.
- A cosine similarity of -1 indicates that the vectors point in exactly opposite directions.
- For commonly used non-negative text representations, such as Bag of Words and TF-IDF, cosine similarity generally ranges from 0 to 1 because the vector components are non-negative.

Cosine similarity is particularly useful for comparing text documents because it focuses on the orientation of the vectors rather than their overall magnitudes. Therefore, two documents with similar patterns of term usage can obtain a high cosine similarity even when one document is longer than the other. This makes cosine similarity particularly suitable for comparing documents of different lengths.

Euclidean distance, introduced in Equ. 2.3.1, can also be used to measure the distance between text vectors. While Euclidean distance measures how far apart two vectors are, cosine similarity measures how closely their directions align.

2.3.3 Basic Vectorization Approaches

Once you understand the Vector Space Model, the next step is to determine how to convert textual information into numerical vectors. This process is known as vectorization.

Vectorization transforms words, sentences, or documents into numerical representations that machine learning algorithms can process. Different vectorization methods represent different characteristics of text. Some methods indicate whether a word is present in a document, some record how frequently a word occurs, while others assign weights according to the importance of a word within a document or collection of documents.

SGOU

These numerical representations enable computers to perform NLP tasks such as text classification, information retrieval, sentiment analysis, document comparison, and recommendation.

This section introduces the basic approaches to text vectorization and provides the foundation for more advanced text representation methods discussed in the next unit.

Basic vectorization approaches can be broadly classified as:

- Binary-Based Representation
- Frequency - Based Representation
- Term - Weighting Representation

These approaches differ mainly in how numerical values are assigned to the terms in a document.

2.3.3.1 Basic Sparse Representation Techniques

Basic sparse representation techniques represent text using the words or terms that occur in a document. These methods generally focus on characteristics such as word presence, word frequency, or term importance rather than directly modelling the semantic relationships between words.

Because each dimension commonly corresponds to a term in the vocabulary, the resulting vectors can be very high-dimensional. At the same time, most values are usually zero because an individual document contains only a small portion of the words in the vocabulary. Such representations are therefore referred to as sparse representations.

These techniques are simple, interpretable, and widely used in traditional NLP and machine learning applications. However, basic sparse representations have limited ability to capture word meaning, context, and semantic relationships between different words.

1. Binary Representation

Binary representation is one of the simplest methods of converting text into numerical vectors. Each term in the vocabulary is assigned a value of 1 if it occurs at least once in a document and 0 if it does not occur.

For example, if a word occurs several times in a document, its value remains 1. Therefore, binary representation records only a term's presence or absence and does not consider how often it occurs.

This approach is computationally simple and can be useful when the presence of particular keywords is more important than their frequency. However, it loses information about repeated occurrences and the relative importance of different terms.

2. Frequency-based Representation

Frequency-based representation considers how frequently a term occurs in a document. One common approach is to calculate the relative frequency of a term by dividing the

SGOU

number of times the term occurs by the total number of terms in the document.

Relative Frequency = Number of occurrences of a term / Total number of terms in the document

By using relative frequencies instead of only presence or absence, this representation provides more information about the importance of a term within a document. Normalization also helps reduce the effect of differences in document length.

However, frequently occurring words are not necessarily the most informative words. Common terms may receive high frequency values even though they contribute little to distinguishing one document from another.

3. Count Vectorizer

Count-based representation records how many times each vocabulary term occurs in a document. Each document is represented as a vector of term counts.

For example, if a particular word occurs five times in a document, the corresponding position in the vector receives the value 5. Unlike binary representation, Count Vectorization therefore preserves information about repeated occurrences.

Count-based vectors form a common implementation of the Bag of Words representation. In Python, the Count Vectorizer class provided by the scikit-learn library can be used to automatically construct a vocabulary and generate document-term count vectors.

Although Count Vectorization is simple and interpretable, it treats all terms by their occurrence counts and does not consider whether a term is common across the entire document collection.

4. Term Weighting

Term weighting assigns different numerical weights to terms according to their statistical importance in a document or collection of documents. Instead of treating every occurrence equally, a weighting scheme can assign lower weights to terms that appear frequently across many documents and higher weights to more distinctive terms.

A widely used term-weighting method is Term Frequency-Inverse Document Frequency (TF-IDF). TF-IDF combines information about how frequently a term occurs in a particular document with information about how common or rare that term is across the entire corpus.

As a result, terms that occur frequently in a particular document but are relatively uncommon across the corpus generally receive higher weights, whereas terms that occur in many documents receive lower weights.

Term weighting can therefore provide a more informative representation than raw term counts for tasks such as document retrieval, text classification, and document similarity.

2.3.3.2 Dense Vector Representation Techniques

Dense vector representation techniques represent words, sentences, or documents

SGOU

using relatively compact vectors in which most or all dimensions contain numerical values. Unlike traditional sparse representations, where individual dimensions often correspond directly to vocabulary terms, dense representations distribute information across multiple dimensions.

Vector-based techniques extend the idea of direct text representation by assigning numerical values (weights) to words based on their importance within a document or across a collection of documents (corpus). Rather than simply recording whether a word appears or counting its occurrences, these techniques aim to represent text more informatively.

The primary objective of vector-based techniques is to improve text representation by reducing the limitations of simple frequency-based methods and enabling computers to compare documents more effectively. These representations serve as the basis for many machine learning algorithms used in NLP.

Examples of vector-based techniques include:

- Term Frequency–Inverse Document Frequency (TF–IDF)
- Word Embeddings
- Document Embeddings (Doc2Vec)

These techniques will be explored in detail in the subsequent unit.

Recap

- Text representation transforms words, sentences, or documents into numerical forms that can be processed by machine learning algorithms for tasks such as classification, clustering, and sentiment analysis.
- Vector Space Model (VSM) represents text as vectors in a multidimensional space, allowing mathematical comparison between texts.
- In a term-based VSM, each dimension represents a vocabulary term, while its value indicates the term's presence, frequency, or importance.
- The L2-norm represents the length of a vector, while Euclidean distance measures the straight-line distance between two vectors.
- Cosine similarity measures similarity based on the angle between two vectors. It is useful for comparing documents of different lengths. Its value ranges from -1 to +1, although for common non-negative text representations such as BoW and TF-IDF, it usually ranges from 0 to 1.
- Sparse vectors have mostly zero values and are generally simple and interpretable, while dense vectors contain values in most dimensions and can better capture semantic and syntactic relationships.

SGOU

- Vocabulary mismatch occurs when different words express similar meanings. Traditional sparse representations may fail to capture such relationships because they depend heavily on exact word matching.
- Binary representation records whether a term is present or absent using values such as 1 and 0.
- Frequency-based representation represents terms based on how often they occur in a document, sometimes using normalized frequencies to account for document length.
- Count Vectorization represents documents using the raw counts of vocabulary terms and is commonly used to create Bag of Words (BoW) representations.
- Term-weighting methods assign importance to terms. TF-IDF considers both the frequency of a term in a document and how common or rare it is across the corpus.
- Sparse vectorization methods are simple and interpretable but have limited ability to capture semantic relationships, which motivates the use of dense representations such as word embeddings.

Objective Type Questions

1. In a traditional term-based Vector Space Model, what does each dimension typically represent?
2. Which distance measure calculates the straight-line distance between two vectors in a vector space?
3. What does the L2-norm of a vector represent?
4. What does a cosine similarity value of 0 indicate geometrically?
5. Why is cosine similarity commonly used for comparing text documents?
6. Which representation assigns a value of 1 when a term is present in a document and 0 when it is absent?
7. In TF-IDF, what happens to the importance of a term that appears in a large number of documents?
8. What information does binary representation record about a term?
9. What does Count Vectorization record for each vocabulary term in a document?

SGOU

10. Which representation expresses term occurrence relative to the total number of terms in a document?

11. What type of values dominates a sparse text vector?

12. Which basic vectorization approach ignores the number of times a term occurs after its first occurrence?

13. If two document vectors point in similar directions, what can generally be inferred about their cosine similarity?

14. What is a vector called when most of its values are zero?

15. Which Python machine learning library provides the CountVectorizer class?

Answers to Objective Type Questions

1. A word or feature in the vocabulary.
2. Euclidean distance
3. The length or magnitude of the vector
4. The vectors are orthogonal (perpendicular)
5. Because it measures direction rather than magnitude
6. One-Hot Encoding
7. Zero
8. Presence or absence of the word
9. The exact (raw) count of occurrences
10. Frequency-based representation
11. Zero values
12. Binary (Boolean) representation
13. Similar documents
14. Sparse
15. Scikit-learn

SGOU

Assignments

1. Compare the four basic vectorization approaches discussed in this unit: Binary Representation, Frequency-Based Representation, Count Vectorization, and Term Weighting. Highlight their working principles, advantages, limitations, and suitable applications.

2. Distinguish between sparse and dense vector representations in NLP.

3. Explain the Vector Space Model as a framework for text representation in NLP. In your answer, describe how documents are represented as vectors, what each dimension represents in a traditional term-based VSM, and how similarity between documents can be measured using Euclidean distance and cosine similarity. Explain why cosine similarity is commonly preferred for text comparison and illustrate your answer with a suitable example.

4. Describe basic vectorization techniques with examples.

Reference

1. S. Vajjala, B. Majumder, A. Gupta, and H. Surana, *Practical Natural Language Processing: A Comprehensive Guide to Building Real-World NLP Systems*. O'Reilly Media, 2020.

2. D. Jurafsky and J. H. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*.

3. Scikit-learn, "Data Transforms." [Online]. Available: https://scikit-learn.org/stable/data_transforms.html [Accessed: 6 September 2026].

4. C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. Cambridge University Press, 2008.

Suggested Reading

1. G. Salton, A. Wong, and C. S. Yang, "A Vector Space Model for Automatic Indexing," *Communications of the ACM*, vol. 18, no. 11, pp. 613–620, 1975.

SGOU

Unit 4

Text Vectorization

Learning Outcomes

After completing this unit, learner will able to:

- explain the purpose of text vectorization and the reasons discrete text must be converted into numerical form
- explain the principles and characteristics of major text vectorization techniques.
- construct One-Hot Encoding, Bag of Words, and Bag of N-Grams representations for a given text corpus
- compute and interpret Term Frequency–Inverse Document Frequency (TF-IDF) scores for terms in a document collection
- distinguish between sparse vectorization methods and distributed representations
- explain how Word2Vec and Doc2Vec generate dense vector representations of words and documents

SGOU

Prerequisites

Imagine a search engine trying to determine whether the query "bank loan interest" is more closely related to a financial news article or to an article about a river bank. Although both contain the word "bank," their meanings differ greatly. For a computer, however, text initially consists only of characters and words. Most machine learning algorithms require numerical input. This leads to the central question of this unit: how can text be transformed into numerical representations that a machine can process, compare, and learn from?

In the previous unit, we introduced the Vector Space Model and examined the idea of representing text as numerical vectors. Before beginning this unit, you should therefore be familiar with basic text pre-processing concepts such as tokenization, stemming, lemmatization, and normalization, as discussed in Unit 2. You should also understand the concepts of vocabulary, corpus, vectors, sparse and dense representations, and basic similarity measures introduced in Unit 3.



A basic understanding of sets, matrices, counting, and ratios will also be helpful when working with the mathematical concepts presented in this unit.

With this background, you are ready to explore how words and documents can be transformed into numerical vectors using techniques such as One-Hot Encoding, Bag of Words, Bag of N-Grams, TF-IDF, Word2Vec, and Doc2Vec.

Keywords

One-Hot Encoding, Bag of Words (BoW), N-Grams, TF-IDF, Word Embeddings, Word2Vec, Doc2Vec, PV-DM, PV-DBOW

Discussion

> "If it is true that there is always more than one way of construing a text, it is not true that all interpretations are equal."
>
> Paul Ricoeur

2.4.1 Introduction - Text Vectorization

Whether the NLP task involves classifying emails, searching documents, analyzing sentiment, or building conversational systems, textual data must ultimately be represented in a numerical form that computational models can process. Most machine learning algorithms cannot operate directly on words and sentences. Therefore, text must be transformed into numerical features or vectors before mathematical and machine learning operations can be applied.

Text vectorization is the process of converting textual data into numerical vectors while preserving useful information from the original text. Depending on the technique used, the resulting representation may indicate which words are present, how frequently they occur, the order in which they appear, their statistical importance, or relationships among words and documents.

There is no single vectorization technique that is suitable for every NLP task. Different approaches offer different trade-offs in simplicity, dimensionality, computational efficiency, interpretability, and the amount of linguistic information they capture.

The techniques discussed in this unit can be broadly grouped into two categories. The first category consists of sparse lexical representation techniques, including One-Hot Encoding, Bag of Words, Bag of N-Grams, and TF-IDF. These techniques represent text using vocabulary-based features and generally produce high-dimensional sparse vectors. Depending on the method, the vector values may represent word presence, occurrence counts, n-gram frequencies, or term importance.

The second category consists of distributed or dense representations, such as Word2Vec

SGOU

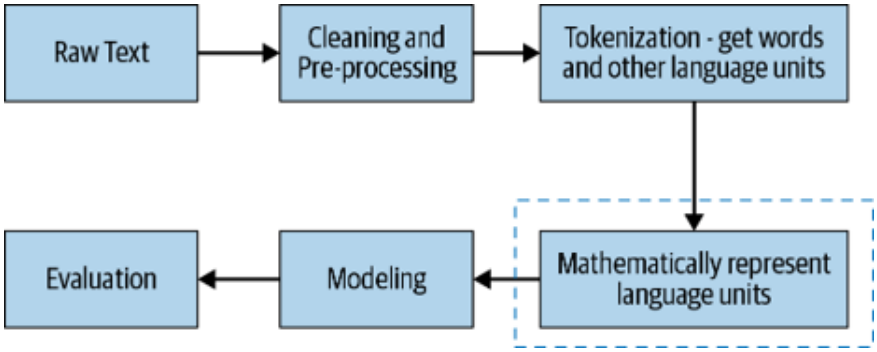


Fig 2.4.1 Scope of text vectorization within the NLP pipeline

and Doc2Vec. Instead of assigning an independent vocabulary dimension to every term, these methods learn compact numerical vectors from patterns of word usage in a corpus. Words or documents with similar usage patterns can therefore obtain similar vector representations.

The quality of a text representation can significantly affect a machine learning model's performance. A representation that preserves task-relevant information can help the model identify useful patterns, whereas an unsuitable representation may discard important information.

This unit examines major methods of text vectorization, beginning with simple vocabulary-based representations and progressing to distributed representations of words and documents. Within the broader NLP processing pipeline, the scope of this unit is illustrated by the dotted box in Figure 2.4.1.

2.4.2 Sparse Lexical Representations

2.4.2.1 One Hot Encoding

One-Hot Encoding is one of the simplest methods for representing discrete text units numerically. The first step is to construct a vocabulary, which is the set of unique tokens or words occurring in the corpus.

If the vocabulary contains V unique words, each word is represented by a vector of length V. All positions in the vector are set to 0 except the position corresponding to that word, which is set to 1.

Consider the sentence:

"the cat sat on the mat"

The unique words form the vocabulary:

{the, cat, sat, on, mat}

Thus, the vocabulary size is V = 5.

If the vocabulary positions are assigned as:

the = 1, cat = 2, sat = 3, on = 4, mat = 5

SGOU

Table 2.4.1: One-Hot Encoding for a five-word vocabulary

Word	the	cat	sat	on	mat
the	1	0	0	0	0
cat	0	1	0	0	0
sat	0	0	1	0	0
on	0	0	0	1	0
mat	0	0	0	0	1

then the word "cat" is represented as:

[0, 1, 0, 0, 0]

and the word "mat" is represented as:

[0, 0, 0, 0, 1]

Each word is therefore represented by a unique binary vector. Since different one-hot vectors have no overlapping non-zero positions, they are orthogonal to one another. Consequently, one-hot encoding does not express semantic similarity between words.

One-hot encoding is widely used for representing categorical data and is supported by many machine learning libraries. In NLP, however, it becomes inefficient when the vocabulary is very large.

Note: One-hot encoding is primarily used for categorical variables rather than continuous numerical variables.

A categorical variable takes values from a fixed set of categories, such as city, colour, or product type.

A continuous variable can take numerical values over a continuous range, such as temperature, height, or weight.

Pros and Cons of One-Hot Encoding

One-hot encoding is simple to understand and easy to implement. However, it has several limitations:

- **No representation of similarity:** Each word is treated as an independent category. For example, "cat" and "kitten" are represented as different from each other as much as "cat" and "economics," even though "cat" and "kitten" are semantically related.
- **High dimensionality:** The length of each vector is equal to the vocabulary size. A vocabulary containing 100,000 words therefore produces 100,000-dimensional vectors, with almost all values equal to zero. Although

SGOU

sparse storage methods can reduce memory requirements, the representation remains very high-dimensional.

- **No fixed size for sentences or documents:** One-hot encoding only represents single words. It doesn't give a fixed-length vector for a whole sentence or document, which most machine learning models require.

2.4.2.2 Bag of Words

The Bag-of-Words model represents an entire document as a numerical vector based on the words it contains. Unlike one-hot encoding, which represents individual words, BoW represents a document by recording the occurrence or frequency of vocabulary terms.

The basic idea is to construct a vocabulary containing all unique words in the corpus. Each document is then represented by a vector whose length is equal to the vocabulary size. Each position in the vector corresponds to a particular vocabulary word, and the value at that position indicates how many times the word occurs in the document.

The term "Bag of Words" reflects that the method considers which words occur and how often, while ignoring grammar and word order.

Consider the sentences:

- S1 = "the cat sat on the mat"
- S2 = "the dog chased the cat"
- S3 = "the cat and the dog are friends"

Together, these sentences form a vocabulary of 10 unique words:

{the, cat, sat, on, mat, dog, chased, and, are, friends}

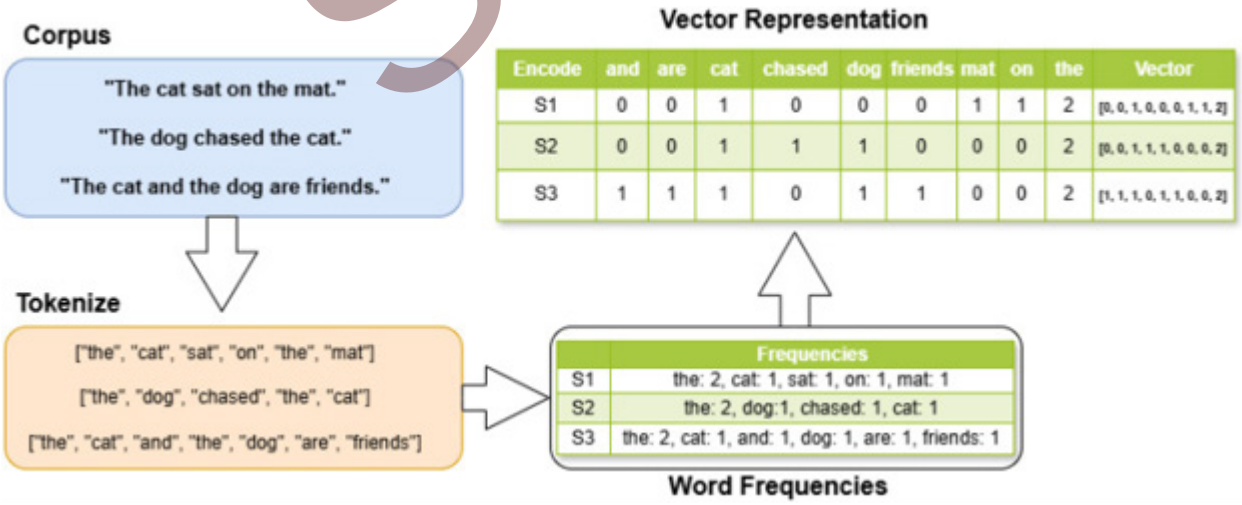


Fig 2.4.2 Illustration of the Bag-of-Words text representation process

SGOU

For example, the word “the” occurs twice in S1, twice in S2, and twice in S3. The word “cat” occurs once in each sentence. Words that do not occur in a particular sentence are assigned a count of zero.

- Each sentence can therefore be represented as a fixed-length vector of word counts. Figure 2.4.2 illustrates tokenizing sentences, counting each vocabulary word, and constructing the corresponding Bag-of-Words vectors.
- BoW is widely used in applications such as spam detection, document classification, sentiment analysis, and information retrieval. It is simple, interpretable, and relatively easy to implement. However, because it ignores word order and contextual relationships, it cannot distinguish sentences that contain similar words arranged differently.

2.4.2.3 Bag of N-Grams

The text representation methods discussed so far largely treat individual words as independent units and therefore preserve little or no information about word order. The Bag of N-Grams approach addresses this limitation by representing text using sequences of N consecutive words or tokens. Each such sequence is called an n-gram.

The value of N determines the number of consecutive tokens included in each n-gram. The most common types are:

- **Unigram (N = 1):** A single word or token
- **Bigram (N = 2):** A sequence of two consecutive words or tokens
- **Trigram (N = 3):** A sequence of three consecutive words or tokens

Consider the sentence:

“Natural Language Processing is fascinating”

The corresponding n-grams are:

- **Unigrams:**

[“Natural”, “Language”, “Processing”, “is”, “fascinating”]

- **Bigrams:**

[“Natural Language”, “Language Processing”, “Processing is”, “is fascinating”]

- **Trigrams:**

[“Natural Language Processing”, “Language Processing is”, “Processing is fascinating”]

Figure 2.4.3 illustrates the formation of unigrams, bigrams, and trigrams from a sentence.

To construct a Bag-of-N-Grams representation, create a vocabulary from all unique n-grams in the corpus. Each document is then represented by a vector whose length

SGOU

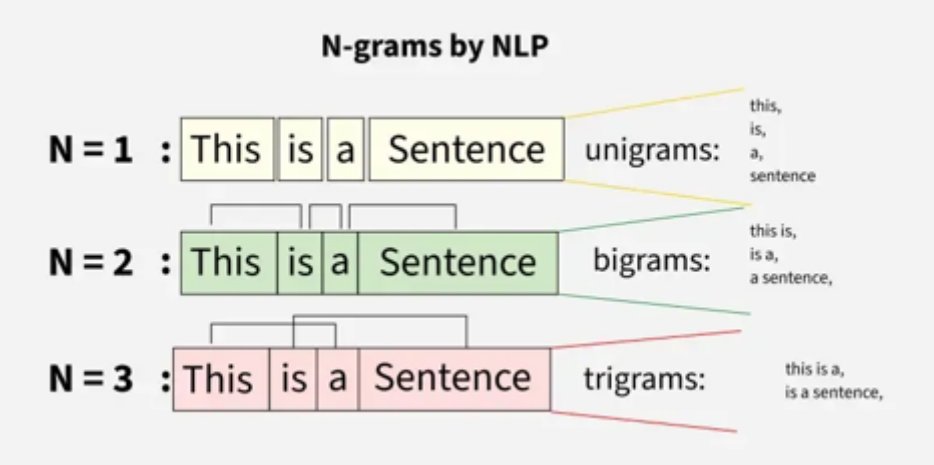


Fig 2.4.3 Illustration of unigram, bigram, and trigram formation

is equal to the size of the n-gram vocabulary. The value at each position indicates the occurrence count or weight of the corresponding n-gram, while n-grams that do not occur in the document receive a value of zero.

Compared with a unigram-based Bag-of-Words representation, Bag of N-Grams can preserve some local word-order and phrase information. For example, the bigram "machine learning" is represented as a single feature, allowing the model to distinguish it from isolated occurrences of the words "machine" and "learning."

However, as N increases, the number of possible n-grams grows rapidly. This results in higher-dimensional and increasingly sparse vectors. Bag of N-Grams also remains dependent on vocabulary overlap and therefore does not inherently capture semantic relationships between different words or phrases. It may also encounter the **out-of-vocabulary (OOV)** problem when previously unseen n-grams occur in new text.

2.4.2.4 TF-IDF (Term Frequency-Inverse Document Frequency)

TF-IDF (Term Frequency–Inverse Docu-ment Frequency) is a statistical weighting technique widely used in Natural Language Processing and information retrieval. It assigns a numerical weight to a term based on two factors: how often the term occurs in a particular document and how rare or common it is across the entire corpus.

TF-IDF is used in areas like information retrieval (e.g., search engines), text classification (e.g., sentiment analysis), keyword extraction, document clustering, and recommendation systems

Unlike a simple Bag-of-Words representation, which uses raw word counts, TF-IDF reduces the influence of terms that occur in many documents and gives relatively greater importance to terms that are more distinctive within a document.

The TF-IDF score of a term is calculated as:

Thus, TF-IDF score = TF * IDF **(1)**

SGOU

Term Frequency (TF)

Term Frequency (TF) measures how often a term occurs in a document. Since documents may have different lengths, raw term counts can be normalized by dividing the number of occurrences of a term by the total number of terms in the document.

$$\mathrm{TF}\left(t, d\right) = \frac{(\text{Number of occurrences of term } t \text{ in document } d)}{(\text{Total number of terms in the document } d)} \quad \textbf{(2)}$$

where:

t = term

d = document

Inverse Document Frequency (IDF)

Inverse Document Frequency (IDF) measures how informative a term is across the corpus. Terms that occur in many documents receive lower IDF values, while terms that occur in fewer documents receive higher IDF values.

Using the classical unsmoothed formulation:

$$\mathrm{IDF}\left(t\right) = \log_e \frac{(\text{Total number of documents in the corpus})}{(\text{Number of documents with term } t \text{ in them })} \quad \textbf{(3)}$$

where:

N = Total number of documents in the corpus

DF(t) = Number of documents containing term t

The logarithm may be computed using different bases depending on the implementation. In the example below, base-10 logarithm is used.

Therefore:

TF-IDF(t, d) = TF(t, d) × IDF(t)

TF-IDF combines the local importance of a term within a document with its global importance across the corpus.

Figure 2.4.4 illustrates the complete TF-IDF vectorization process for a small corpus containing three sentences:

sen1 = “this is good”

sen2 = “this is bad”

sen3 = “awesome this is awesome”

The three sentences are passed through the TF-IDF vectorization process, which

SGOU

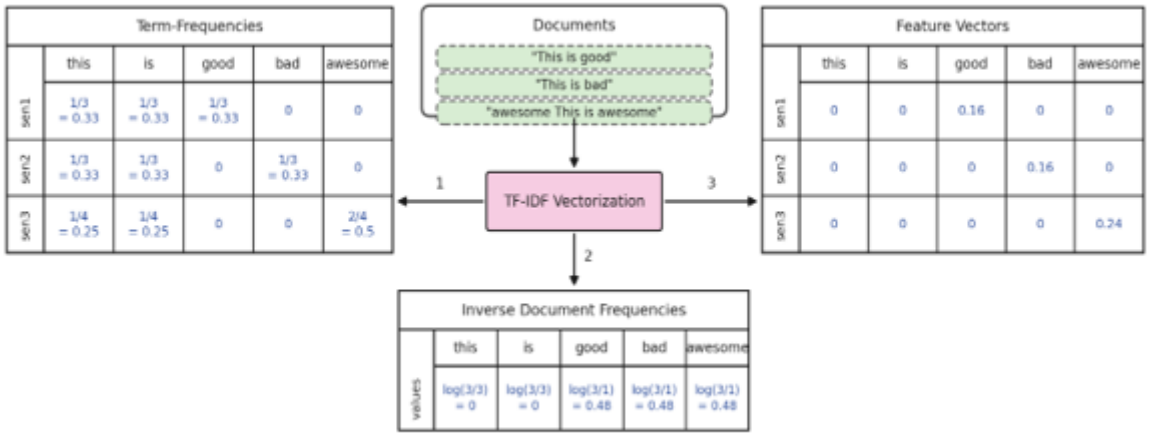


Fig 2.4.4: TF-IDF vectorization process showing the computation of term frequency, inverse document frequency, and final TF-IDF feature vectors

produces term-frequency values, inverse document-frequency values, and final TF-IDF feature vectors.

Output 1: Term Frequency

The Term Frequency table shows the normalized frequency of each vocabulary term in every sentence. The vocabulary is:

{this, is, good, bad, awesome}

In sen1, which contains three words, each occurring word appears once. Therefore:

TF(this) = 1/3 = 0.33

TF(is) = 1/3 = 0.33

TF(good) = 1/3 = 0.33

Similarly, in sen3, the word “awesome” occurs twice in a four-word sentence:

TF(awesome) = 2/4 = 0.50

The words “this” and “is” each occur once:

TF(this) = 1/4 = 0.25

TF(is) = 1/4 = 0.25

Terms that do not occur in a sentence receive a TF value of zero.

Output 2: Inverse Document Frequency

The IDF value is calculated once for each vocabulary term across the entire corpus.

The words “this” and “is” appear in all three documents. Therefore, using the classical unsmoothed IDF formula:

SGOU

IDF(this) = log10(3/3) = 0

IDF(is) = log10(3/3) = 0

The words “good”, “bad”, and “awesome” each occur in only one document:

IDF = log10(3/1) ≈ 0.48

Thus, terms that occur in every document receive no discriminative weight under this particular IDF formulation, whereas rarer terms receive higher IDF values.

Output 3: TF-IDF Feature Vectors

The final TF-IDF value is obtained by multiplying the TF value of each term by its corresponding IDF value.

For sen1:

TF-IDF(good) = 0.33 × 0.48 ≈ 0.16

For sen2:

TF-IDF(bad) = 0.33 × 0.48 ≈ 0.16

For sen3:

TF-IDF(awesome) = 0.50 × 0.48 = 0.24

Because “this” and “is” have an IDF value of zero in this example, their TF-IDF values are also zero.

This example demonstrates the main idea of TF-IDF: terms that occur frequently within a particular document but are less common across the corpus receive relatively higher weights, while terms that occur across many documents receive lower weights.

2.4.3 Distributed Representations

Earlier approaches to word representation in NLP, known as local or discrete representations, assign each word a unique vector position. A common example is the one-hot representation, where each word is represented by a vector containing a single non-zero value. Since different words have non-zero values at different positions, their vectors are orthogonal, resulting in a cosine similarity of zero. Such representations are unable to capture semantic relationships between words.

Consider the sentence:

“*He poured the steaming chai into a clay cup and handed it to the traveller*,”

A person who is unfamiliar with the word **“chai”** can infer its meaning from the surrounding context. The nearby words such as **“poured,” “steaming,”** and **“cup”** suggest that chai is a hot, drinkable substance. This demonstrates how humans often understand new words by using contextual clues rather than relying only on dictionary definitions.

SGOU

Distributed representations, also known as **word embeddings**, address this limitation. Instead of assigning each word to a single independent position, embeddings represent words as dense vectors distributed across multiple dimensions. The meaning of a word is captured by the combined pattern of values across all dimensions rather than by any single value.

They are called "distributed" representations because the information about a word's meaning is distributed across the entire vector. For example, the meaning of "chai" is not represented by one particular dimension; rather, it emerges from the overall relationship between its vector and the vectors of related words.

Word embeddings typically use tens to hundreds of dimensions (commonly 50–300 dimensions in traditional embeddings), with each dimension containing a continuous numerical value. Compared with high-dimensional sparse representations such as one-hot vectors, embeddings provide compact representations that allow models to measure semantic similarity between words.

Word2Vec, introduced by Tomas Mikolov and his team at Google in 2013, played a major role in popularizing word embeddings. Word2Vec is based on distributional similarity and the distributional hypothesis: words that occur in similar contexts tend to have similar meanings.

Rather than using manually defined meanings, Word2Vec learns word representations by analyzing the surrounding words in large text collections. If two words frequently appear in similar contexts, their vector representations are likely to be similar.

The Distributional Hypothesis

The concept of learning word meaning from context is based on the distributional hypothesis, a linguistic principle stating that words appearing in similar contexts tend to have similar meanings.

Linguist Zellig Harris (1954) expressed this idea as:

"A word is known by the company it keeps."

Consider the sentences:

- *"I drank a cup of tea."*
- *"I drank a cup of coffee."*

There is no explicit statement that tea and coffee are related. However, both words appear in similar positions and are surrounded by similar words such as **"**drank**"** and **"**cup**."** When this pattern occurs repeatedly across large text collections, an NLP model learns that these words have similar contextual behaviour.

Therefore, Word2Vec places tea and coffee closer together in the embedding space, not because the model understands beverages like humans do, but because these words show similar usage patterns in language.

SGOU

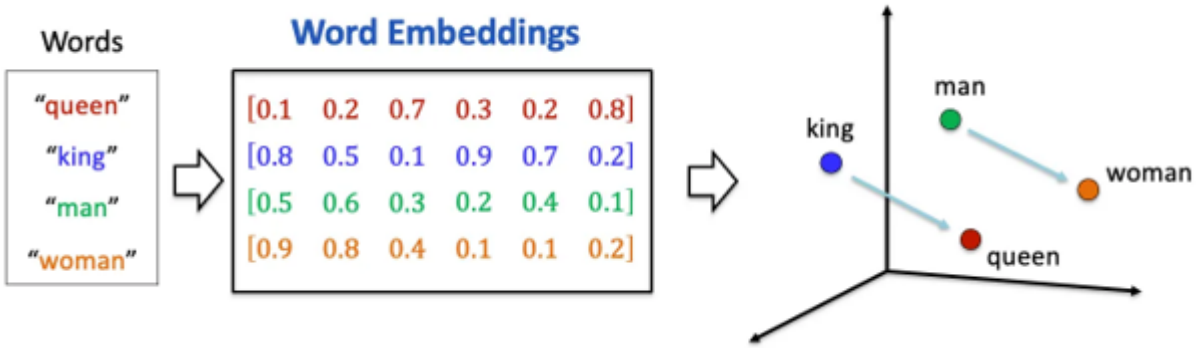


Fig 2.4.5 Word embedding representation showing words mapped to dense vectors and positioned in a continuous vector space

2.4.3.1 Word Embeddings

Word embeddings (also called word vectors) represent dense numerical vectors in a continuous vector space, where words with similar usage patterns are positioned closer together. A **dense vector** contains meaningful numerical values across most of its dimensions, unlike sparse representations such as one-hot encoding, where most values are zero.

These vectors are learned automatically from large collections of text by analyzing the contexts in which words appear. As a result, word embeddings capture relationships between words, including semantic similarity, syntactic patterns, and contextual associations.

Word embeddings have significantly improved the performance of many NLP applications, including text classification, sentiment analysis, machine translation, information retrieval, and question answering.

1. Word2Vec

Word2Vec is a machine learning technique that has been around since 2013, courtesy of Tomas Mikolov and his data science team at Google [6].

Word2Vec is a neural network-based technique for learning word embeddings, introduced by Tomas Mikolov and his team at Google in 2013. It learns dense vector representations of words by analyzing the context in which words appear in large text collections.

Word2Vec is based on the distributional hypothesis: words that occur in similar contexts tend to have similar meanings. During training, Word2Vec learns numerical representations that capture semantic and syntactic relationships between words.

Word2Vec can be trained using two different architectures. These architectures perform opposite but complementary tasks:

- Continuous Bag-of-Words (CBOW)
- Skip-gram

SGOU

a. CBOW

The CBOW architecture predicts the target (centre) word using the surrounding context words. For example, given the sentence:

"The cat ___ on the mat"

The model learns to predict the missing word "sat" from the surrounding context.

As illustrated in Figure 2.4.6, the architecture consists of three stages: Input, Projection, and Output. The context words w(t−2), w(t−1), w(t+1), and w(t+2) are represented as one-hot vectors and passed to the projection layer. These vectors are combined to form a single hidden representation.

Since the context words are combined without considering their order, this representation is referred to as a "bag-of-words" representation. The output layer then uses this combined representation to predict the target word w(t).

The weights between the input and projection layers form a V×N matrix, where V is the vocabulary size and N is the embedding dimension. During training, this matrix learns the numerical representations of words. After training, the output layer is discarded, and the learned weights of the input-to-projection matrix are used as the word embeddings.

CBOW is computationally efficient and trains faster than Skip-gram. It generally performs well for frequently occurring words and large datasets.

b. Skip-Gram

The Skip-gram architecture performs the reverse task of CBOW. Instead of predicting

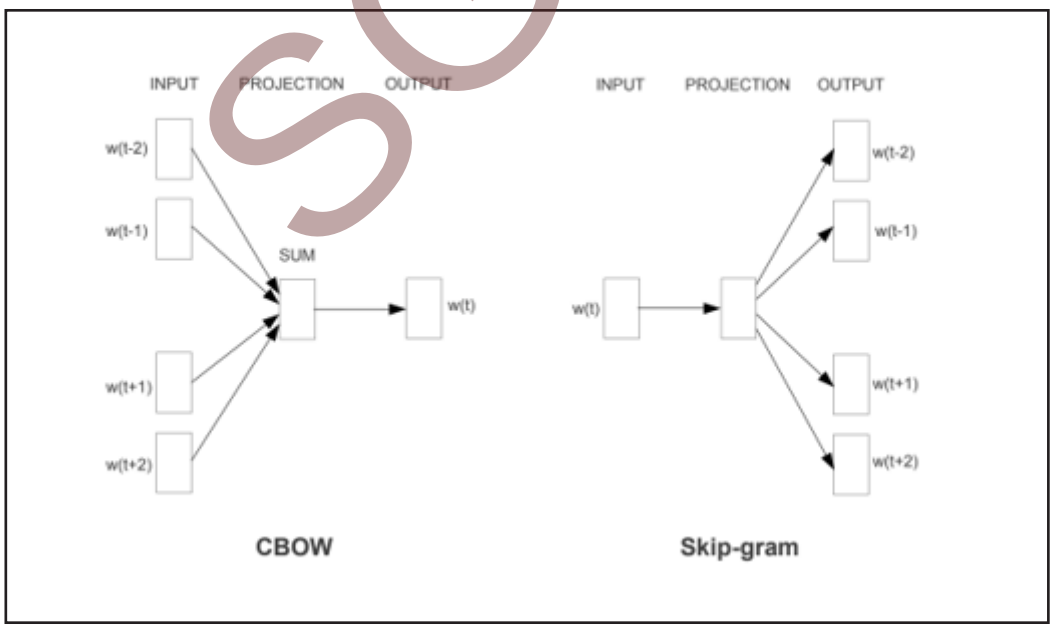


Fig 2.4.6: Word2Vec architectures: The CBOW architecture predicts the target word using its surrounding context words, whereas the Skip-gram architecture predicts surrounding context words using the target word.

SGOU

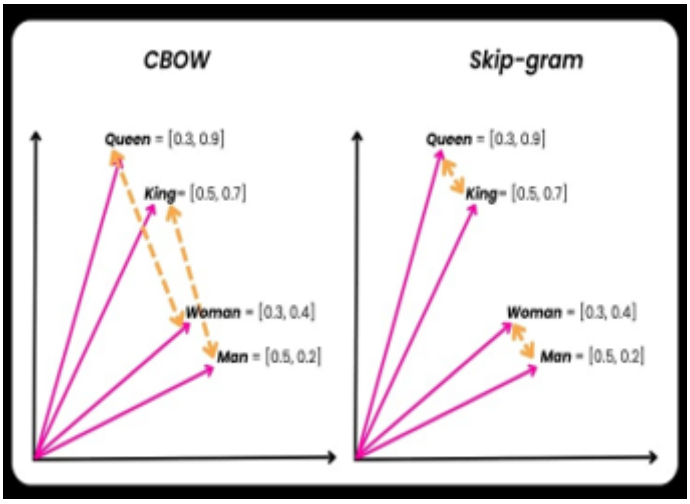


Fig 2.4.7.1: CBOW and Skip-gram word embedding representations showing semantic relationships between words

the target word from its surrounding context words, it takes the target word as input and predicts the words that appear around it.

For example, given the target word "sat" in the sentence:

"The cat sat on the mat"

The model attempts to predict the surrounding context words such as "the", "cat", "on", and "the".

As illustrated in Figure 2.4.6, the Skip-gram architecture consists of the same three stages: Input, Projection, and Output. The target word w(t) is represented as a single one-hot vector and provided as input to the projection layer. The output layer then predicts the surrounding context words w(t−2), w(t−1), w(t+1), and w(t+2).

Since only one target word is provided as input, the projection layer does not require any pooling or combination of multiple input vectors.

Similar to CBOW, the weights between the input layer and the projection layer form a V× embedding matrix, where V is the vocabulary size and N is the embedding dimension. After training, the learned weights of this matrix are used as word embeddings.

Because each occurrence of a word can generate multiple training examples by pairing the target word with its surrounding context words, Skip-gram can learn richer representations from limited text data. It generally performs better for rare words and smaller datasets, but it requires more computational time compared with CBOW.

2.4.3.2 Document Embeddings

1. Doc2Vec

Word embeddings, discussed in the previous section, generate a vector representation for each word. However, many NLP applications require a single representation for larger text units such as phrases, sentences, paragraphs, or complete documents. Doc2Vec

SGOU

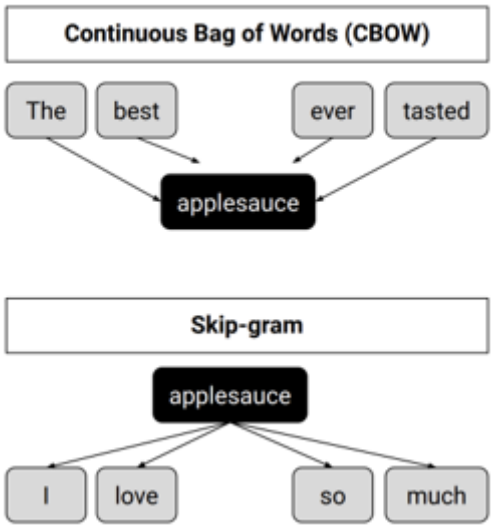


Fig 2.4.7.2: Working principle of CBOW and Skip-gram architectures in Word2Vec: CBOW predicts a target word from context words, while Skip-gram predicts context words from a target word

addresses this limitation by learning a fixed-length vector representation for an entire document while considering the context of the words within that document.

Doc2Vec, also known as Paragraph Vector, is an extension of Word2Vec that learns dense vector representations of documents rather than only individual words. It was introduced by Quoc Le and Tomas Mikolov in 2014 and is implemented in libraries such as Gensim.

The main idea behind Doc2Vec is similar to Word2Vec. It uses a shallow neural network architecture but introduces an additional input called a document vector or paragraph vector. This vector is unique to each vector and word vectors work together to predict words within the document. After training, the learned document vector provides a numerical representation of the complete text. While word vectors are shared across the entire corpus, each document has its own trainable vector that captures the overall characteristics of that document. During training, the document vector and word vectors work together to predict words within the document. After training, the learned document vector provides a numerical representation of the complete text. This allows text of different lengths, including phrases, sentences, paragraphs, and full documents, to be represented as fixed-length dense vectors. These representations can then be used for NLP tasks such as document classification, sentiment analysis, information retrieval, and document similarity analysis.

Structure of the Model

Figure 2.4.8 illustrates the working principle of the Doc2Vec model. The inputs to the model consist of word vectors and a document ID vector. Each word is initially represented as a one-hot vector with dimension 1×V, where V represents the vocabulary size. Similarly, the document ID is represented as a one-hot vector with dimension 1×C,

SGOU

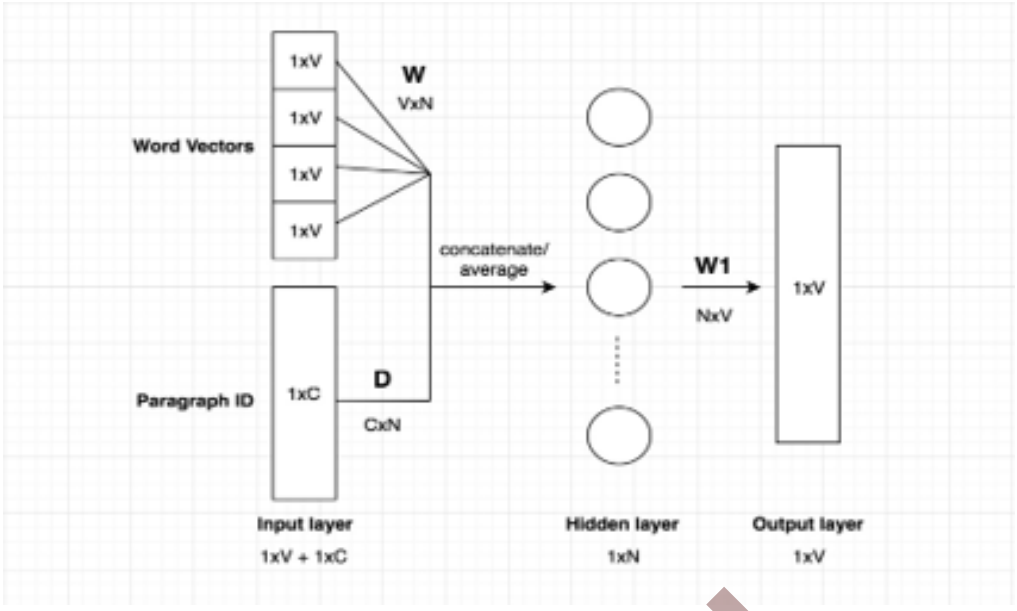


Fig 2.4.8: Doc2Vec (Paragraph Vector–Distributed Memory) architecture showing the combination of word vectors and document vectors for learning document representations.

where C represents the total number of documents in the corpus.

The hidden layer contains two weight matrices: matrix W, associated with word representations, has dimensions V×N, and matrix D, associated with document representations, has dimensions C×N, where N is the selected embedding dimension.

During training, both word and document weight matrices are updated simultaneously. After training, the learned rows of matrix D represent the document embeddings, providing dense vector representations of individual documents.

Doc2Vec Architectures

Doc2Vec learns document representations using two main architectures:

- Paragraph Vector–Distributed Memory (PV-DM)
- Paragraph Vector–Distributed Bag of Words (PV-DBOW)

Paragraph Vector–Distributed Memory (PV-DM)

PV-DM is an extension of the CBOW architecture used in Word2Vec. Instead of predicting a target word only from its surrounding context words, PV-DM predicts the target word using both the context word vectors and a document (paragraph) vector**.**

During training, a sliding window moves through the document to capture the local context of each target word. At each position, the document vector is combined with the surrounding context word vectors. These representations are merged at the projection layer using either concatenation or averaging, and the resulting representation is used to predict the target word.

SGOU

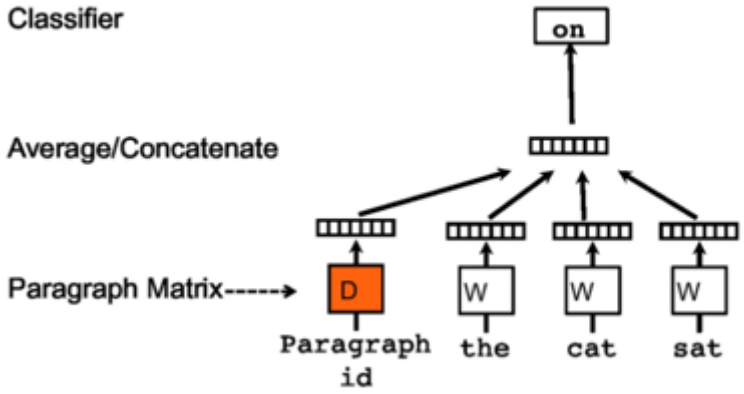


Fig 2.4.9: Paragraph Vector–Distributed Memory (PV-DM) architecture

The document vector acts as a memory component because it is shared across all prediction tasks within the same document. By repeatedly participating in word prediction, the document vector learns information that represents the overall characteristics and theme of the document. During training, the model updates both the word vectors and document vectors based on the prediction error.

For example (Figure 2.4.9), consider the sentence:

"The cat sat on the mat."

If the model is trained to predict the target word "on", it uses:

- The paragraph vector (D) represents the entire document.
- The context word vectors (W) of surrounding words, such as "the," "cat," and "sat."
- These vectors are combined through averaging or concatenation to generate a representation used to predict the target word.

Paragraph Vector–Distributed Bag of Words (PV-DBOW)

PV-DBOW is inspired by the Skip-gram architecture of Word2Vec. Unlike PV-DM, it does not use context words as input. Instead, the model takes only the document vector (paragraph vector) as input and learns to predict words that appear within that document.

As illustrated in Figure 2.4.10, the document vector is used to predict a set of words randomly sampled from the document during training. Through this process, the document vector learns information that represents the overall content and characteristics of the document.

Since PV-DBOW does not require context word vectors as input, its architecture is simpler and generally requires less computational memory than PV-DM. However, because it does not explicitly model local word order or neighbouring word relationships, it may capture fewer contextual details compared with PV-DM.

SGOU

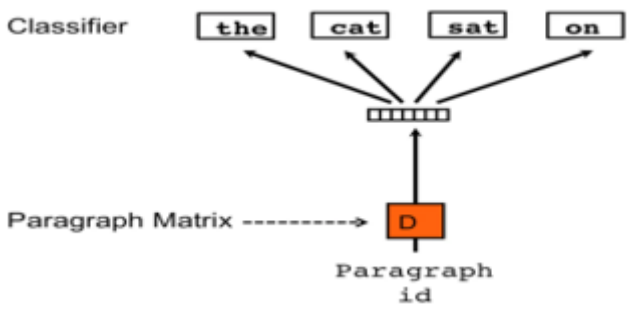


Fig 2.4.10: Paragraph Vector–Distributed Bag of Words

Despite this limitation, PV-DBOW is effective for learning document representations and is often combined with PV-DM to obtain stronger document embeddings.

Figure 2.4.10: The PV-DBOW model learns a document vector by predicting words from the document vector without using surrounding context words. Le and Mikolov [2] reported that combining PV-DM and PV-DBOW architectures generally produces better document representations.

What Happens When a New Document Arrives?

When a new, unseen document is introduced, Doc2Vec does not retrain the entire model. Instead, the learned word vectors and model parameters are kept fixed, and only a new document vector is learned for the incoming document.

During inference, the new document is passed through the trained model, and its document vector is adjusted iteratively to best predict the words in that document. Once optimized, this newly learned vector is placed in the same embedding space as the vectors of the training documents.

This capability lets Doc2Vec represent documents of different lengths as fixed-size vectors, making it useful for applications such as document classification, document clustering, plagiarism detection, recommendation systems, and semantic search.

SGOU

Recap

- Text vectorization converts words, sentences, or documents into numerical vectors that machine learning algorithms can process.
- One-Hot Encoding represents each vocabulary word as a sparse binary vector. However, it cannot capture semantic relationships or similarities between words.
- Bag of Words (BoW) represents a document using the frequency of vocabulary words while ignoring word order and contextual relationships.

- Bag of N-Grams represents text using sequences of n consecutive words or tokens, capturing limited local context that is missing in the Bag-of-Words approach.
- TF-IDF (Term Frequency–Inverse Document Frequency) assigns weights to terms based on their frequency within a document and their importance across the entire corpus. It reduces the influence of common terms and gives higher importance to more distinctive terms.
- Word embeddings represent words as dense vectors that capture semantic and syntactic relationships, providing more meaningful representations than sparse vector techniques.
- Word2Vec has two main architectures:
  - Continuous Bag-of-Words (CBOW), which predicts a target word using its surrounding context words.
  - Skip-gram, which predicts surrounding context words using a target word.
- Doc2Vec, also known as Paragraph Vector, is an extension of Word2Vec that learns vector representations for complete documents rather than only individual words.
- Doc2Vec can represent text of different lengths, including sentences, paragraphs, and documents, as fixed-length dense vectors.
- The two main Doc2Vec architectures are:
  - PV-DM (Paragraph Vector–Distributed Memory), which uses both context words and the document vector to predict a target word.
  - PV-DBOW (Paragraph Vector–Distributed Bag of Words), which predicts words from the document vector without using surrounding context words.

SGOU

Objective Type Questions

1. What is the process of converting text into numerical form called?
2. Which model represents documents as vectors in a multi-dimensional space?
3. Which similarity measure compares the angle between two vectors?
4. Which Doc2Vec model predicts a target word using both the document vector and surrounding context words?

5. Which Doc2Vec model predicts words using only the document vector?
6. What does TF stand for in TF-IDF?
7. What does IDF stand for in TF-IDF?
8. Which technique represents sequences of consecutive words?
9. Which representation captures the semantic relationships between words?
10. Which model learns vector representations for individual words?
11. Which model learns vector representations for entire documents?
12. Which text representation technique produces sparse vectors?
13. Which types of vectors are generated by Word2Vec?
14. What does CBOW stand for?
15. Which technique converts words into dense vector representations?

Answers to Objective Type Questions

1. Text Vectorization
2. Doc2Vec (Document Embeddings / Paragraph Vector)
3. Cosine Similarity
4. PV-DM
5. PV-DBOW
6. Term Frequency
7. Inverse Document Frequency
8. N-Grams representation (Bag of N-Grams)
9. Word Embeddings
10. Word2Vec
11. Doc2Vec
12. One-Hot Encoding

SGOU

13. Dense Vectors

14. Continuous Bag of Words

15. Word Embedding techniques (such as Word2Vec, GloVe, FastText)

Assignments

1. Construct the One-Hot Encoding, Bag of Words (BoW), and TF-IDF representations for a small corpus of your choice. Compare the resulting vector representations and discuss the strengths and limitations of each technique.

2. Explain, with suitable examples, how Bag of N-Grams differs from Bag of Words. Discuss why N-Grams are more effective in capturing local context, phrases, and sentiment information in text.

3. Describe the concept of Term Frequency–Inverse Document Frequency (TF-IDF). Explain how TF-IDF assigns importance to terms and why frequently occurring words across a corpus receive lower weights compared with distinctive words.

4. Explain the Continuous Bag of Words (CBOW) and Skip-gram architectures of Word2Vec. Compare their working principles, advantages, and applications.

5. Describe the Doc2Vec model and explain how it extends Word2Vec to represent entire documents. Compare the two Doc2Vec architectures, PV-DM and PV-DBOW, highlighting their working principles, differences, and applications.

SGOU

Reference

1. S. Vajjala, B. Majumder, A. Gupta, and H. Surana, *Practical Natural Language Processing: A Comprehensive Guide to Building Real-World NLP Systems*. O'Reilly Media, 2020.

2. D. Jurafsky and J. H. Martin, *Speech and Language Processing*. Harlow: Pearson Education, 2014.

3. T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient Estimation of Word Representations in Vector Space," *arXiv preprint arXiv:1301.3781*, 2013.

4. Q. Le and T. Mikolov, "Distributed Representations of Sentences and Documents," in *Proceedings of the 31st International Conference on Machine Learning (ICML)*, pp. 1188–1196, 2014.

5. J. Pennington, R. Socher, and C. D. Manning, "GloVe: Global Vectors for Word Representation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1532–1543, 2014.

Suggested Reading

1. S. Bird, E. Klein, and E. Loper, *Natural Language Processing with Python*. O'Reilly Media, 2009.

2. Q. Le and T. Mikolov, "Distributed Representations of Sentences and Documents," in *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 1188–1196, PMLR, 2014.

3. M. Y. Khan, A. Qayoom, M. S. Nizami, M. S. Siddiqui, S. Wasi, and S. M. K. U. R. Raazi, "Automated Prediction of Good Dictionary Examples (GDEX): A Comprehensive Experiment with Distant Supervision, Machine Learning, and Word Embedding-Based Deep Learning Techniques," *Complexity*, vol. 2021, Art. no. 2553199, 2021.

SGOU



Classification and Information Extraction

SGOU

Unit 1
Text Classification

Learning Outcomes

After completing this unit, learner will able to:

- explain the concept of Text Classification and its role in Natural Language Processing (NLP)
- distinguish between binary, multiclass, and multilabel classification
- describe the major real-world applications of text classification
- compare rule-based, machine learning-based, and deep learning-based approaches to text classification

Prerequisites

NLP pipeline and the preprocessing phases required to prepare raw text for analysis were also covered in Block 1. These include cleaning, tokenization, stemming, lemmatization, and normalization. The block also introduced methods for converting processed text into numerical representations, including One-Hot Encoding, Bag of Words (BoW), Bag of N-Grams (BoN), and TF-IDF, as well as distributed representations such as word embeddings and Doc2Vec. These representations are essential for text classification because classification algorithms operate on numerical feature vectors rather than raw text. Learners should understand how a document is transformed into a numerical representation before proceeding with this unit.

Basic machine learning concepts will also be helpful, although this unit does not require advanced knowledge. Learners should understand the idea of supervised learning, where models learn patterns from examples labelled with the correct outputs. A basic understanding of training data, testing data, and the importance of evaluating models on unseen data will make the classification methods discussed in this unit easier to understand.

Basic mathematical concepts such as probability and vectors will be useful in later sections of this block but are not mandatory prerequisites for this unit.

Finally, learners should be familiar with general NLP terminology introduced earlier in the course, including corpus, document, token, vocabulary, and feature. These terms are used throughout this unit without additional explanation.

SGOU



Keywords

Corpus, Label, Feature, Training Set, Test Set, Classifier, Supervised Learning, Unsupervised Learning

Discussion

> Organizing is what you do before you do something, so that when you do it, it is not all mixed up.
>
> *A.A. Milne*

3.1.1 Introduction - Text Classification

Most of us check our email several times a day without considering the processes that occur before a message appears in our inbox. Between the sender and the recipient's mailbox, each incoming message is automatically analyzed by a program and assigned to one of two categories: genuine mail (inbox) or spam. This automatic sorting is one of the most common real-world examples of text classification, a fundamental task in Natural Language Processing (NLP). That automatic sorting is one of the most common real-world examples of text classification, a fundamental task in Natural Language Processing (NLP).

Text classification assigns one or more predefined categories to a given piece of text. In the spam filtering example shown in Fig 3.1.1, the possible categories are spam and not spam, and each incoming message is assigned to one of these classes.

Although spam detection appears to be a simple task, the same underlying problem occurs in many large-scale applications. Examples include routing customer complaints to appropriate departments, categorizing social media posts by topic, identifying the language of a message, and determining whether a product review expresses a positive or negative sentiment.

The wide range of applications across domains such as social media, e-commerce, healthcare, law, and marketing highlights an important characteristic of text

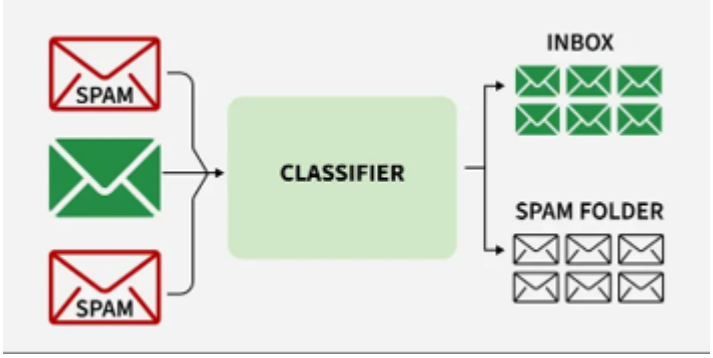


Fig 3.1.1 E-mail classification (spam or not spam)

SGOU

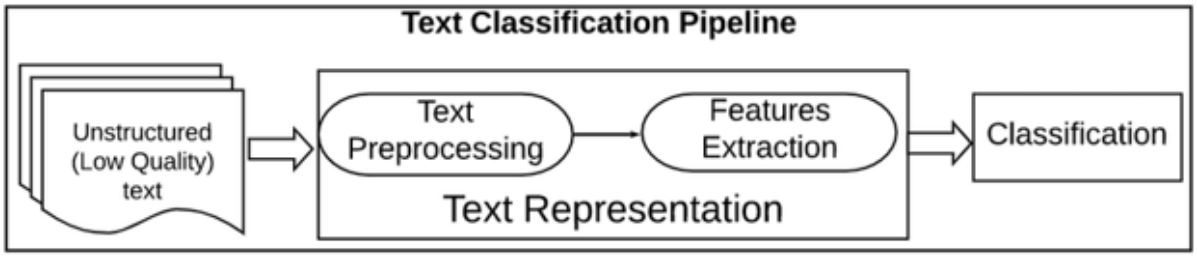


Fig 3.1.2 Text Classification Pipeline

classification. Although the purpose and categories may vary across different domains, the fundamental problem remains the same. The system receives a text input and assigns an appropriate category label.

3.1.1.1 What is Text Classification?

Classification is a fundamental task in language processing and artificial intelligence. It is used in applications where large volumes of text must be organized automatically.

Text classification is a type of classification problem in which the input is text, and the objective is to assign it to one or more predefined categories, also called classes. The input text can have different lengths, including a character, word, sentence, paragraph, or complete document.

A text classification pipeline (Fig 3.1.2) transforms raw text into a predicted category label through a sequence of processing steps. It begins with unstructured text, which may contain noise, spelling errors, or irrelevant information. The text is first cleaned and preprocessed using techniques such as tokenization, stop-word removal, stemming or lemmatization. The processed text is then converted into numerical representations using feature extraction methods such as Bag of Words (BoW), TF-IDF, or word embeddings.

These numerical features are provided to a classification model, which learns patterns from the data and predicts the appropriate class label. Examples of predicted labels include spam/not spam, positive/negative sentiment, or specific news categories.

3.1.2 Types of Text Classification

A common text classification scenario involves categorizing customer reviews of a product into different classes, such as positive, negative, and neutral. The primary objective is to train a model using a set of labelled examples so that it can accurately assign the appropriate category to new, unseen customer reviews. However, text classification does not always involve assigning a single category, and the number of possible categories may vary by application.

Text classification is therefore not a single task but a family of related tasks, classified based on the number of possible categories and the number of labels that can be assigned to each text instance. The three main types of text classification are binary classification, multiclass classification, and multilabel classification, as illustrated in Fig 3.1.3.

SGOU

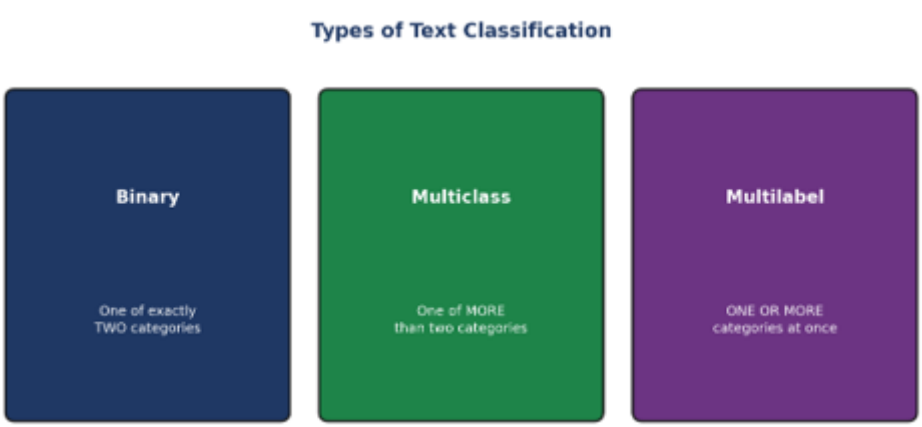


Fig 3.1.3 Types of Text Classification

3.1.2.1 Binary Classification

Binary classification is the simplest form of text classification, with exactly two possible categories, and each text instance is assigned to one of them.

The email spam filtering example (Fig 3.1.4) represents a binary classification problem because each message is classified as either spam or not spam. Similarly, a sentiment analysis system that categorizes a product review as positive or negative performs binary classification.

In binary classification, the model decides between two possible outcomes for each input text.

3.1.2.2 Multi-Class Classification

In multiclass classification, there are more than two possible categories, and each text instance is assigned to exactly one category. Unlike binary classification, where the model chooses between two classes, multiclass classification handles multiple mutually exclusive categories.

For example (Fig 3.1.5), a news article classification system may categorize articles into classes such as Politics, Sports, Business, Science, and Entertainment. Each article receives only one label, even if its content discusses multiple topics. The classifier selects the category that best represents the primary topic of the text.

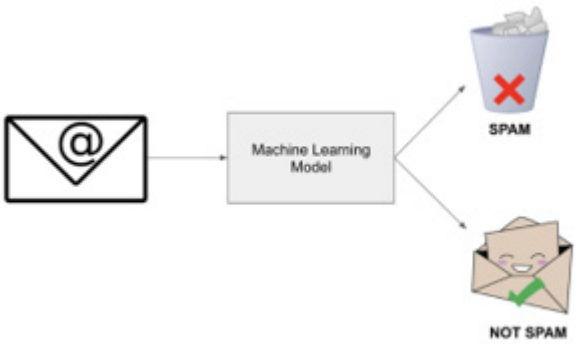


Fig 3.1.4 Email Spam filtering as an example of binary text classification

SGOU

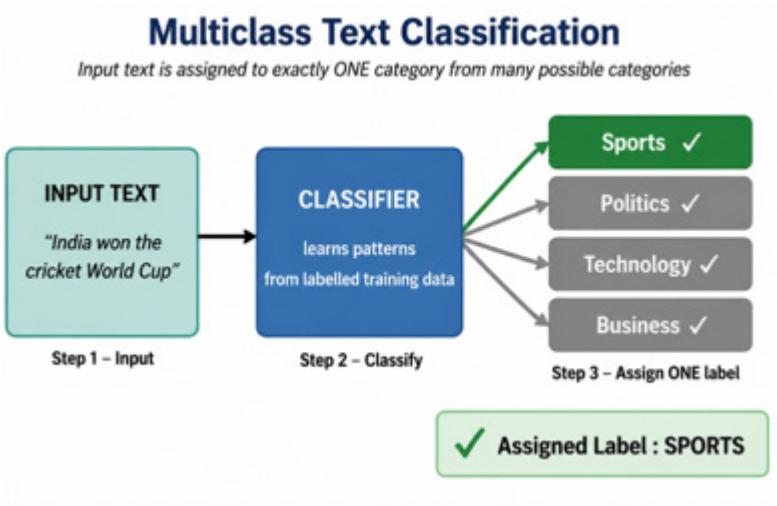


Fig 3.1.5 Multiclass text classification

3.1.2.3 Multi Label Classification

A Multi-label classification problem involves multiple possible class labels where a single text instance can belong to one or more classes simultaneously. Unlike multiclass classification, where each instance is assigned to exactly one category, the classes in multilabel classification are not mutually exclusive.

For example, a research paper may belong to both Machine Learning and Natural Language Processing categories. Similarly, a news article may discuss multiple topics and can be assigned labels such as Politics, Sports, and Finance simultaneously (Fig 3.1.6).

In multilabel classification, the classifier evaluates each category independently and determines whether a label applies to the given text.

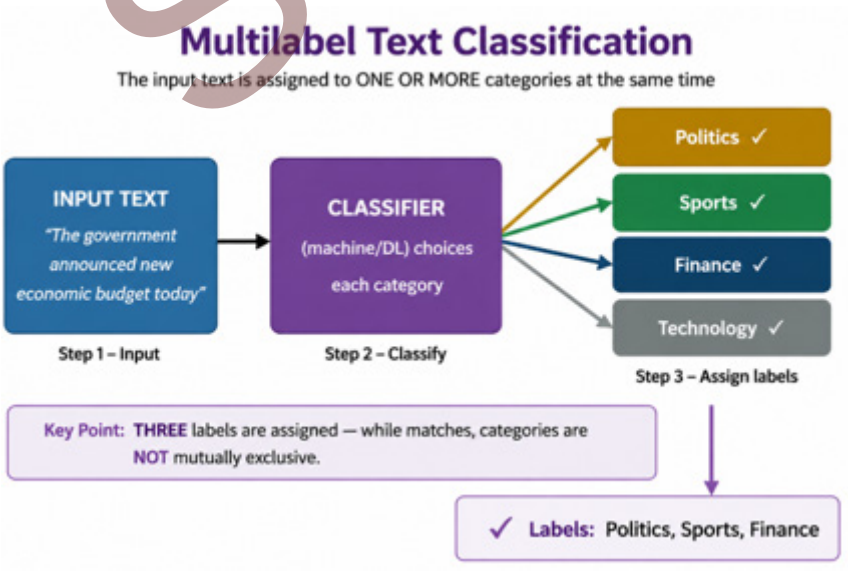


Fig 3.1.6 Multilabel text classification

SGOU

Table 3.1.1 Comparison of Multiclass and Multilabel Classification

Aspect	Multiclass Classification	Multilabel Classification
Number of possible classes	More than two	More than two
Labels assigned per text	Exactly one label	One or more labels
Relationship between classes	Mutually exclusive	Not mutually exclusive
Decision process	Selects one class among many	Independently predicts each label
Example	News categorization (Sports/ Politics/Business)	Topic tagging of articles
Output format	Single class label	Multiple class labels

Fig 3.1.6 Multilabel text classification showing a single text instance assigned to multiple categories simultaneously.

Comparison of Multiclass and Multi-label Classification

Although both multiclass and multilabel classification involve multiple possible categories, the key difference lies in how labels are assigned to each text instance.

The major differences are summarized in table 3.1.1.

3.1.3 Methods of Text Classification

Text classification involves teaching computers to automatically assign prede-fined categories to text documents. Since machine learning models cannot directly process raw text, the text is first converted into numerical representations using techniques such as Bag of Words, TF-IDF, or word embeddings.

The classification model then learns patterns from these numerical representations by identifying relationships between words, their frequencies, contextual information, and other linguistic features. Based on the patterns learned from labelled training data, the model predicts the appropriate category for new, unseen text.

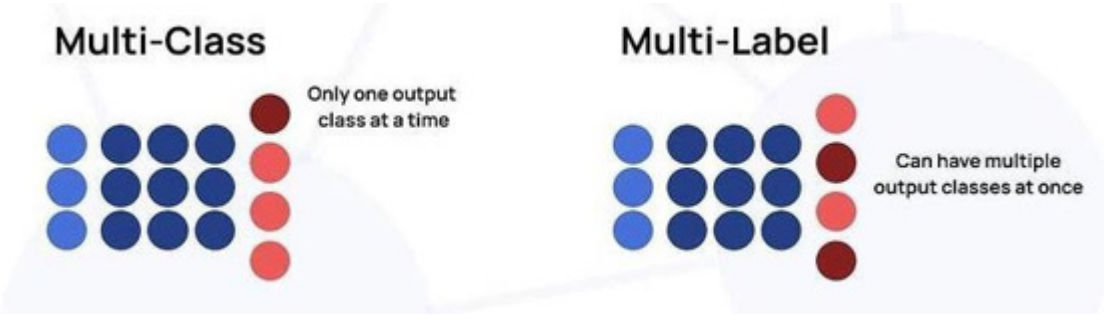


Fig 3.1.7 Comparison of multiclass and multilabel classification

SGOU

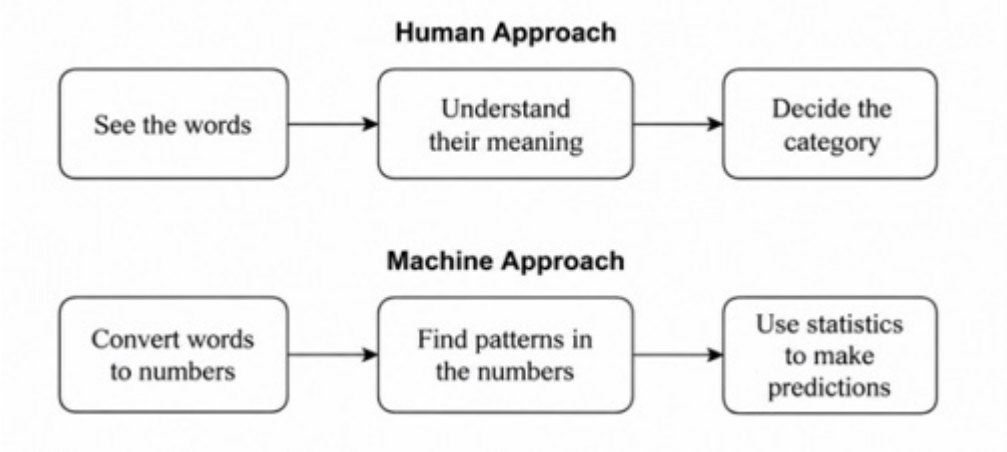


Fig 3.1.8 Comparison between human understanding and machine-based text classification process

Figure 3.1.8 illustrates the difference between human and machine approaches to text classification. Humans can directly understand the meaning of words and their context, whereas computers must first convert text into numerical representations, analyze statistical patterns, and use learned models to determine the most likely category.

Researchers have developed various approaches for text classification, ranging from traditional rule-based techniques to advanced deep learning methods. These approaches differ in how they identify patterns in text and make classification decisions. The three major approaches are:

- Rule-Based Methods
- Machine Learning-Based Methods
- Deep Learning-Based Methods

3.1.3.1 Rule-Based Methods

Rule-based methods are among the earliest approaches used for text classification. These systems assign categories to text by applying manually defined logical conditions, commonly expressed as if–then rules.

As illustrated in Fig 3.1.9, the rule-based text classification process involves four main stages: rule creation, rule application, rule processing, and rule refinement. Rules are initially created from domain knowledge, applied to incoming text, evaluated for

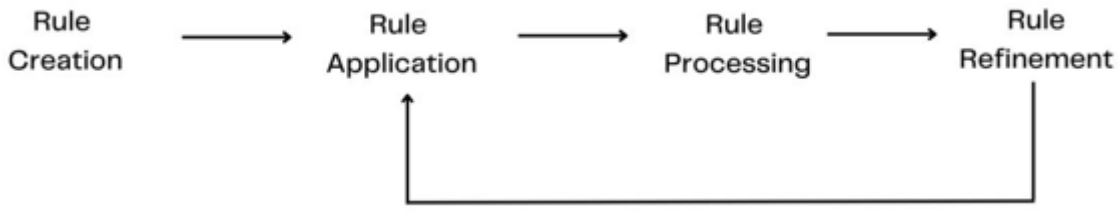


Fig 3.1.9 Rule-based text classification process

SGOU

performance, and refined through feedback to improve classification accuracy.

Common tools and libraries that support rule-based text processing include spaCy, NLTK, and regular expression libraries.

Rule-Based Classification Techniques

- **Keywords or Phrases:** Specific words or combinations of words are used as indicators for a particular category.

  Example: Emails containing terms such as *"win," "lottery,"* or *"prize"* may be classified as spam.

- **Regular Expressions:** Pattern-matching techniques are used to identify specific text patterns or formats.

  Example: Regular expressions can detect email addresses, phone numbers, dates, or other structured information within text.

- **Linguistic Features:** Linguistic characteristics such as syntax, part-of-speech patterns, and grammatical structures can be used to define classification rules.

  Example: A sentence beginning with *"Dear"* may indicate a formal letter or email.

Advantages

- Rule-based systems are simple to understand and provide high interpretability because the decisions are based on explicitly defined rules.
- They perform well in domains where language patterns are fixed and predictable, such as legal documents, medical reports, and technical docum-ents.
- They do not require large labelled datasets for training.

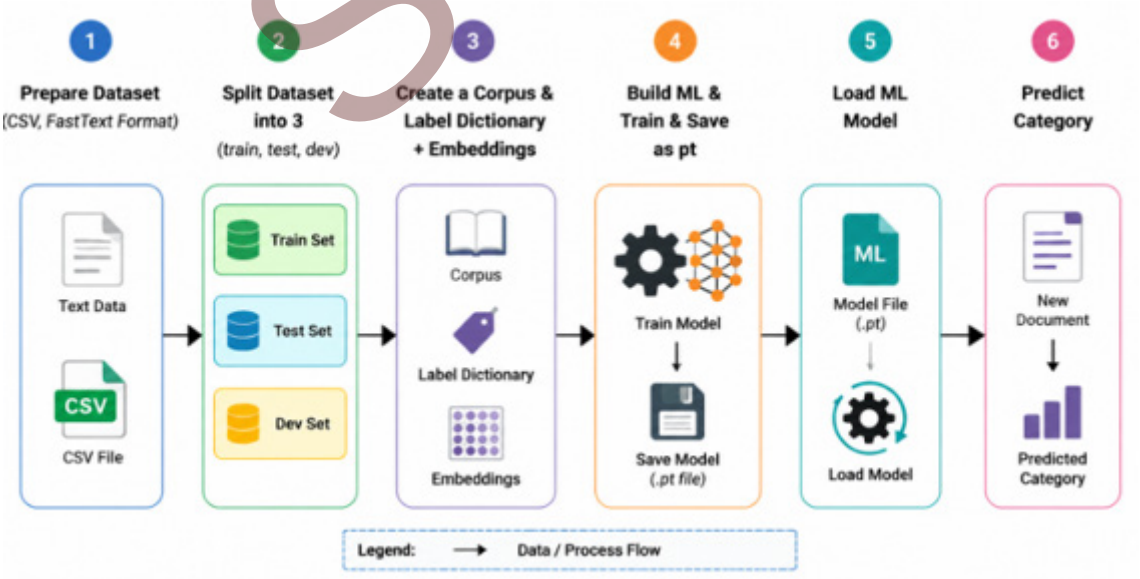


Fig 3.1.10 Working of a machine learning-based text classification system

SGOU

Limitations

- Developing rule-based systems requires significant manual effort because domain experts must define and maintain each rule.
- Rules are difficult to maintain because language evolves, requiring frequent manual updates.
- Rule-based systems often have limited generalization capability, as rules developed for one domain may not perform effectively in another domain.
- They may fail to handle complex language variations, such as sarcasm, ambiguity, and changing expressions.

3.1.3.2 Machine Learning-Based Methods

Machine learning approaches use statistical and computational models to classify text by learning patterns from data. Unlike rule-based methods, where rules are manually defined, machine learning models automatically learn important features from examples. These models are typically trained using labelled datasets and optimized to improve classification performance.

Figure 3.1.10 illustrates the typical pipe-line of a machine learning-based text classification system, starting from dataset preparation to model prediction. The major steps involved are described below:

- The pipeline begins with a labelled dataset (corpus), which contains text samples along with their corresponding class labels. The dataset is divided into training, development (validation), and test sets as shown in Figure 3.1.11.

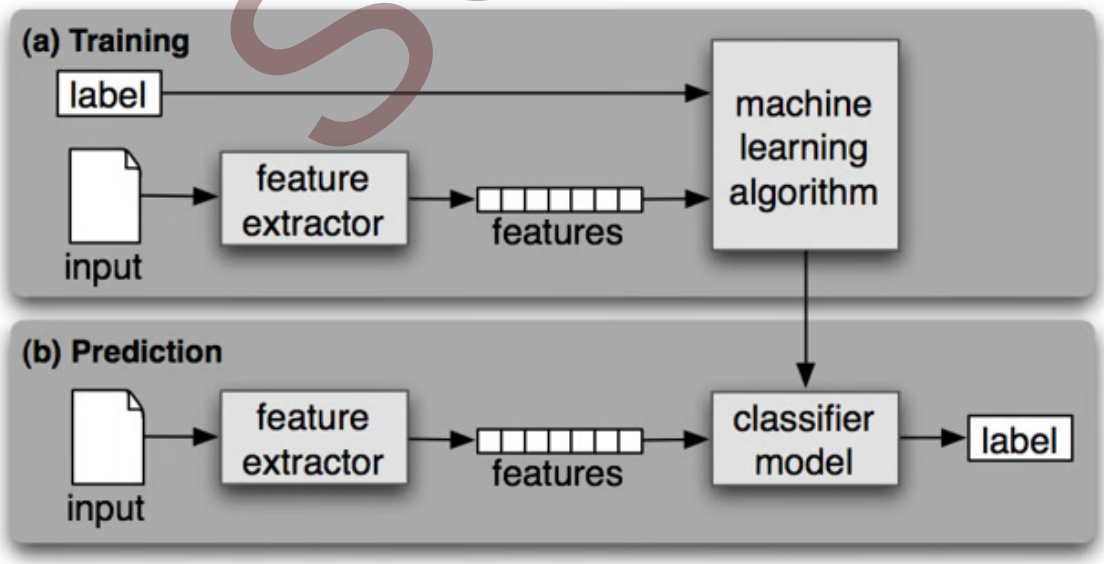


Fig 3.1.11 Supervised text classification framework

SGOU

- The text data is then cleaned and preprocessed using techniques such as tokenization, normalization, and removal of unwanted information. The processed text is converted into numerical representations using feature extraction methods such as Bag of Words, TF-IDF, or word embeddings (Block 2).

- The machine learning model is trained using the training set to learn patterns from the extracted features. The model performance is monitored and optimized using the development/validation set, while the final performance is measured using the independent test set.

- After evaluation, the trained model is saved. When a new document is received, the saved model is loaded, processes the input text, and predicts the appropriate category.

Figure 3.1.12 illustrates the organization of corpus data for training supervised classifiers. The corpus is divided into separate subsets, including a training set, development set, and test set. The development set is used for model tuning and parameter selection, while the test set is reserved for final performance evaluation.

Types of Machine Learning-Based Classification

Machine learning-based text classification approaches are generally divided into two categories:

1. Supervised Classification
2. Unsupervised Classification

1. Supervised Classification

A classifier is called a supervised classifier when it is trained using a labelled corpus, where each input text is associated with its correct category label. The model learns the relationship between input features and their corresponding labels during the training phase and uses this knowledge to classify new, unseen text.

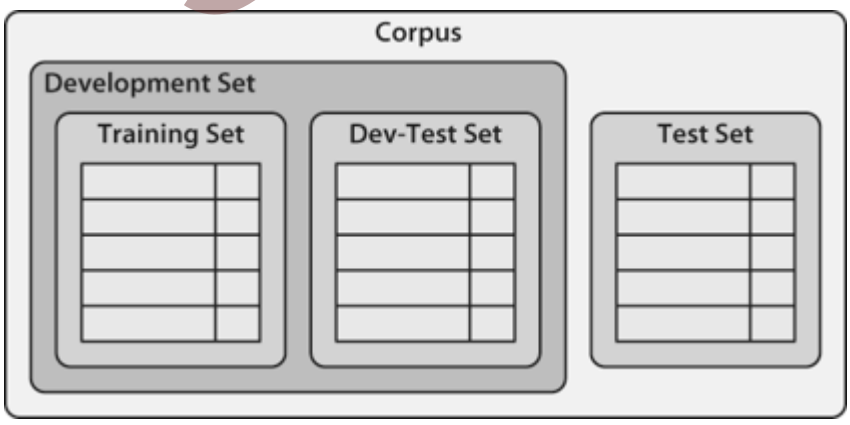


Fig 3.1.12 Organization of Corpus data into training, development, and test sets for supervised classification

SGOU

The framework of supervised classification is illustrated in Fig 3.1.11.

(a) Training Phase

- A feature extractor converts each input text into a numerical feature representation.
- These feature sets capture important characteristics of the input text that are useful for classification.
- The extracted feature sets and their corresponding labels are provided to a machine learning algorithm to train a classification model.

(b) Prediction Phase

- During prediction, the same feature extraction process is applied to unseen text inputs.
- The extracted features are provided to the trained model, which analyzes the learned patterns and generates the predicted class label.

In supervised text classification, the corpus is generally divided into different subsets:

- **Training Set:** Used to train the classification model.
- **Development (Validation) Set:** Used for model tuning and parameter selection.
- **Test Set:** Used for final evaluation of the trained model on unseen data.

Common algorithms used for supervised text classification include:

- Logistic Regression
- Decision Trees
- Random Forests
- Support Vector Machines (SVM)
- K-Nearest Neighbors (KNN)

2. Unsupervised Classification

Unsupervised machine learning is an approach in which models learn from unlabeled data without predefined class information. Instead of learning from examples with known outputs, the algorithm automatically identifies hidden structures, patterns, and relationships within the data.

In text classification, unsupervised methods analyze the similarity between documents and group related documents into clusters. These techniques can discover common topics, organize large collections of documents, and reveal meaningful patterns without requiring manual annotation.

SGOU

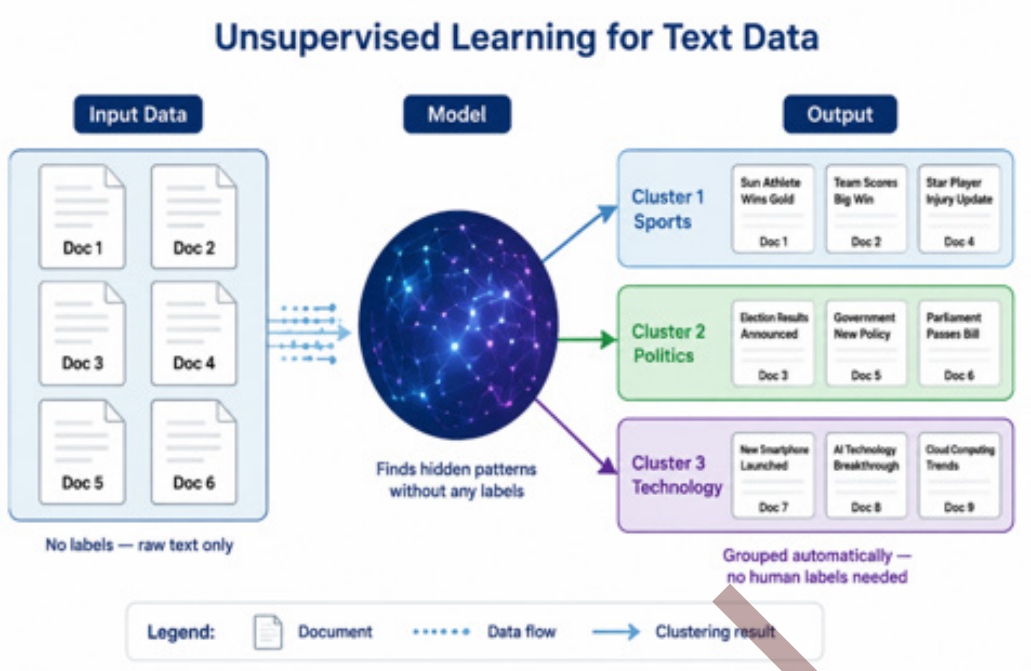


Fig 3.1.13 Unsupervised classification of raw text documents showing automatic grouping of documents into thematic clusters

Figure 3.1.13 illustrates the process of unsupervised classification, where raw text documents without labels are provided as input. The model analyzes the content and automatically groups similar documents into different thematic clusters, such as Sports, Politics, and Technology. This approach is particularly useful for organizing large volumes of unstructured text where labelled datasets are unavailable.

Common algorithms used for unsupervised text analysis and classification include:

- K-Means Clustering
- Hierarchical Clustering
- DBSCAN (Density-Based Spatial Clustering of Applications with Noise)
- Principal Component Analysis (PCA)

3.1.3.3 Deep Learning-Based Methods

Deep learning has significantly advanced text classification by enabling models to learn complex language patterns and automatically capture contextual relationships. Unlike traditional machine learning approaches, which rely on manually engineered features, deep learning models can automatically learn and extract meaningful representations directly from raw text data.

Important deep learning architectures used for text classification include Recurrent Neural Networks (RNNs), Long Short-Term Memory networks (LSTMs), Convolutional Neural Networks (CNNs), and transformer-based models such as BERT, GPT, and RoBERTa.

SGOU

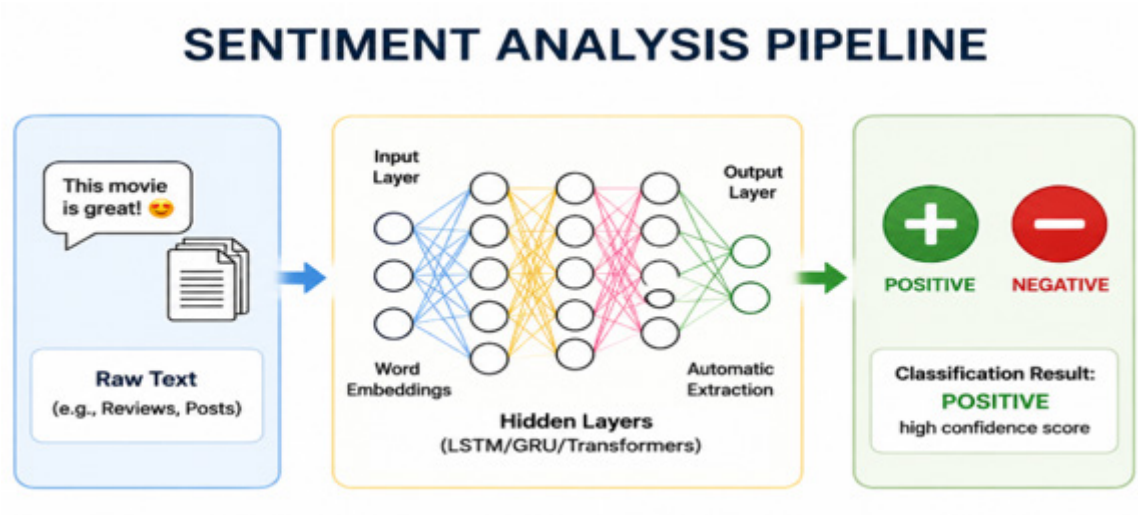


Fig 3.1.14 Deep learning-based text classification pipeline showing automatic feature extraction through neural network layers and generation of classification output

Figure 3.1.14 illustrates the deep learning-based text classification pipeline, where raw text is processed through a neural network architecture. The model automatically learns hierarchical features through hidden layers and generates the final classification output.

Figure 3.1.14: This diagram illustrates the deep learning-based text classification pipeline, showing how raw input text is processed through a neural network architecture with hidden layers to automatically extract features and generate a classification output.

How does it work?

Consider the sentence:

"The bank was flooded after the storm."

A traditional machine learning model treats "bank" as a fixed feature and may associate it with the financial meaning because it appears so often in the training data. It does not consider the surrounding context unless explicitly provided through feature engineering.

An LSTM, however, processes the sentence sequentially and maintains information from previously observed words. By the time it reaches the word "bank", it has already processed words such as "flooded" and "storm". Based on this contextual information, the model can infer that "bank" refers to the side of a river rather than a financial institution.

A transformer-based model such as BERT goes further by analyzing all words in a sentence simultaneously using self-attention mechanisms. It creates a contextual representation of each word based on its relationship with other words in the sentence. Therefore, the word "bank" receives different representations in:

"The bank was flooded after the storm."

and

SGOU

"I opened a savings account at the bank."

This ability to understand context allows transformer-based models to perform well on tasks that require semantic understanding, such as sentiment analysis, question answering, and document classification.

Advantages

- Deep learning models automatically earn features from raw text, reducing the need for manual feature engineering.
- These models can capture complex patterns, semantic relationships, and contextual information in language.
- Deep learning approaches achieve high performance when trained on large-scale text data-sets.
- Transformer-based models can effectively handle long-range dependencies and contextual variations in language.

Limitations

- Deep learning models generally require large amounts of labelled training data to achieve high accuracy.
- Training and deploying these models require significant computational resources, such as GPUs, increasing hardware and energy costs.
- Deep learning models often behave as black-box systems, making it difficult to interpret how specific classification decisions are made.
- Large models may have higher memory requirements and longer inference times compared with traditional machine learning models.

3.1.4 Applications

Since any task that assigns one or more labels to a piece of text can be considered a text classification problem, its applications extend across a wide range of domains.

Text classification enables automatic organization, filtering, and analysis of large volumes of textual data. Figure 3.1.15 illustrates some of the most common real-world applications of text classification.

1. Sentiment Analysis

Sentiment analysis determines whether a piece of text expresses a positive, negative, or neutral opinion. E-commerce platforms use sentiment analysis to automatically classify customer reviews, while companies analyse social media feedback to understand public opinions about their products and services.

2. Spam Detection

Spam detection identifies and filters unwanted or potentially harmful messages from

SGOU

legitimate communications by analysing message content and patterns. Email service providers train classifiers using large collections of spam and non-spam examples, allowing new messages to be automatically classified before reaching users' inboxes.

3. Topic Labelling

Topic labelling automatically assigns one or more subject categories to documents, making large collections of information easier to organise and retrieve. News agencies and digital platforms use this approach to assign categories such as Politics, Sports, Business, and Technology to articles.

4. Language Identification

Language identification is a multiclass classification task that determines the language of a given text from a predefined set of possible languages. Social media platforms, translation systems, and customer support applications use language identification to select appropriate processing methods for incoming text.

5. News Categorization

News categorisation automatically assigns news articles to predefined sections such as Technology, Finance, Health, or Sports. Large news platforms process thousands of articles daily and use classifiers to organise content efficiently and improve information retrieval.

6. Product Classification

Product classification assigns appropriate categories to product listings based on their titles, descriptions, and other available information. E-commerce platforms use this

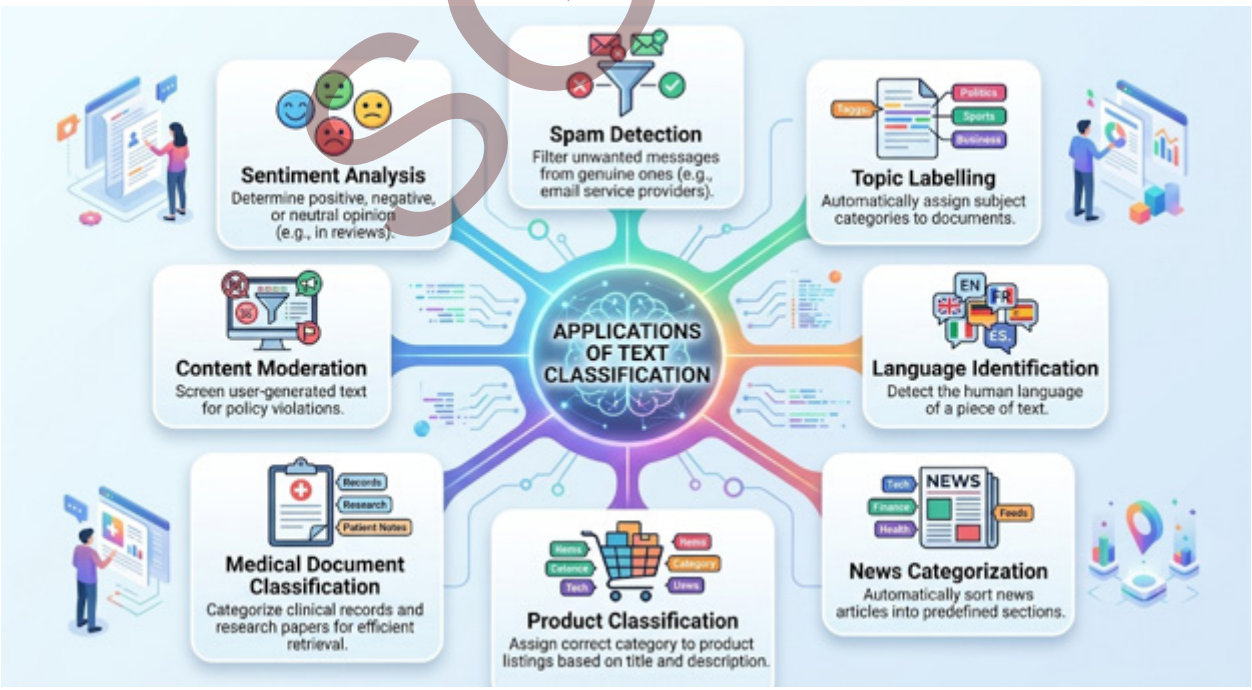


Fig 3.1.15 Applications of text classification in different real-world domains

SGOU

technique to organise products into suitable categories, improve search results, and provide better recommendations.

7. Medical Document Classification

Medical document classification categorises healthcare-related documents such as patient records, clinical reports, and research articles into predefined groups. Hospitals and healthcare systems use this approach to improve document organisation, retrieval, and clinical information management.

8. Content Moderation

Content moderation automatically analyses user-generated text to identify content that violates platform guidelines, such as hate speech, misinformation, or harmful content. Social media platforms use text classifiers as an initial screening mechanism to identify potentially problematic posts for further human review.

Recap

- Text classification is the task of assigning text documents to predefined categories or labels.
- Text classification is one of the most widely used tasks in NLP, with applications across domains such as social media, healthcare, law, e-commerce, and marketing.
- A text classification pipeline begins with raw text, applies preprocessing techniques, converts the text into numerical representations, and feeds the extracted features into a classification model to predict the appropriate label.
- Binary classification assigns a text instance to exactly one of two categories, such as spam or not spam.
- Multiclass classification assigns a text instance to exactly one category from more than two possible categories.
- Multilabel classification allows a single text instance to be assigned to multiple categories simultaneously.
- Rule-based methods classify text using manually written if-then rules and do not require any labeled training data.
- Machine learning-based methods learn classification patterns automatically from labelled training data.
- Supervised learning trains a model using labelled examples and uses the learned patterns to classify unseen text.

SGOU

- **Unsupervised learning** discovers hidden patterns and groups similar documents without predefined class labels.
- **Deep learning models** automatically learn features from raw text and capture complex patterns, semantic relationships, and contextual information.
- Common **supervised machine learning algorithms** include Logistic Regression, Naive Bayes, Decision Trees, Random Forests, and Support Vector Machines (SVM).
- Common **unsupervised learning algorithms** include K-Means Clustering, Hierarchical Clustering, DBSCAN, and Principal Component Analysis (PCA).
- Common **deep learning architectures** used for text classification include RNNs, LSTMs, CNNs, and transformer-based models such as BERT and GPT.

Objective Type Questions

1. What is the task of assigning one or more predefined categories to a given piece of text called?
2. Name the three main types of text classification.
3. In binary classification, how many categories can a text be assigned to?
4. What type of classification assigns exactly one label from more than two possible categories?
5. What type of classification allows a text instance to belong to more than one category simultaneously?
6. What is the first step in a text classification pipeline?
7. Name any two feature extraction techniques used in text classification.
8. What type of machine learning uses labelled training data?
9. What type of machine learning discovers patterns without predefined labels?
10. Name any two supervised machine learning algorithms used in text classification.
11. Name any two unsupervised machine learning algorithms used in text classification.

SGOU

12. What does a rule-based system use to classify text?

13. Name any two deep learning architectures used in text classification.

14. What is the term used for the process of manually assigning category labels to text data?

15. What is the main limitation of deep learning models in terms of computational requirements?

Answers to Objective Type Questions

1. Text classification

2. Binary classification, Multiclass classification, and Multilabel classification

3. Exactly two categories

4. Multiclass classification

5. Multilabel classification

6. Text preprocessing (cleaning and standardising the raw text)

7. Bag of Words (BoW), TF-IDF, and word embeddings

8. Supervised learning

9. Unsupervised learning

10. Logistic Regression and Support Vector Machines (SVM)

11. K-Means Clustering and Principal Component Analysis (PCA)

12. Manually defined if–then rules based on keywords, regular expressions, or linguistic features

13. RNN, LSTM, BERT, GPT, and RoBERTa

14. Annotation

15. They require powerful computational resources, such as GPUs, for training and deployment, which increases computational cost.

SGOU

Assignments

1. What is text classification? Explain the concept with a suitable real-world example from any application domain.

2. Distinguish between binary, multiclass, and multilabel classification. Provide one example of each and explain how they differ from one another.

3. What is the difference between supervised and unsupervised text classification? Give one real-world example of each approach.

4. Compare rule-based, machine learning-based, and deep learning-based approaches to text classification. Discuss the advantages and limitations of each.

5. Explain the major categories of machine learning-based text classification in detail, with suitable examples.

6. Explain the major applications of text classification in real-world domains.

Reference

1. Vajjala, S., Majumder, B., Gupta, A., & Surana, H. (2020). *Practical natural language processing: a comprehensive guide to building real-world NLP systems*. O'Reilly Media.

2. Jurafsky, D., & Martin, J. H. Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition.

Suggested Reading

1. Bird, S., Klein, E., & Loper, E. (2009). Natural Language Processing with Python. O'Reilly Media,

2. Manning, C. D. (2008). *Introduction to information retrieval*. Syngress Publishing,.

3. Kowsari, K., Jafari Meimandi, K., Heidarysafa, M., Mendu, S., Barnes, L., & Brown, D. (2019). Text classification algorithms: A survey. *Information*, *10*(4), 150.

4. Khan, M. Y., Qayoom, A., Nizami, M. S., Siddiqui, M. S., Wasi, S., & Raazi, S. M. K. U. R. (2021). Automated prediction of good dictionary examples (GDEX): A comprehensive experiment with distant supervision, machine learning, and word embedding-based deep learning techniques. *Complexity*, *2021*(1), 2553199.

SGOU

Unit 2

Designing Text Classifiers

Learning Outcomes

After completing this unit, learner will be able to:

- describe the complete pipeline involved in building a text classification system
- explain the role of each stage in the pipeline, from data collection to deployment
- design appropriate features for a text classification task
- explain the Naive Bayes algorithm and how it uses probability for text classification, with a worked example
- evaluate a classifier using performance metrics such as precision, recall, and F1-score

Prerequisites

This unit builds upon the concepts introduced in Unit 1, where you learned about text classification, its three main types, and the three broad approaches: rule-based, machine learning-based, and deep learning-based methods. Ensure that you have a clear understanding of these fundamental concepts before proceeding, as this unit focuses on the practical steps involved in developing a working text classifier.

You should also be familiar with the text preprocessing techniques discussed in Block II, such as tokenization, stop-word removal, stemming, and lemmatization, along with vectorization methods, particularly the Bag-of-Words and TF-IDF approaches. These techniques form essential components of the text classification pipeline discussed in this unit.

Additionally, a basic understanding of probability concepts is required, including interpreting the probability of an event (for example, a probability value of 0.6) and performing multiplication of probabilities. These concepts are essential for understanding the Naïve Bayes algorithm discussed in the later sections.

SGOU

Keywords

Feature, POS, Accuracy, Precision, Recall, F1-Score, Annotation

Discussion

3.2.1 Introduction - Text Classifiers

In Unit 1, we explored the question: *what is text classification?* In this unit, we address the next important question: *How do we actually build* a text classifier?

There is a significant difference between understanding that a spam filter exists and knowing how to develop one. A functional text classifier is not simply a single algorithm that can be activated instantly. Instead, it is a complete system with multiple stages, each requiring careful design and implementation. An issue in one stage can affect the performance of subsequent stages.

The process of transforming raw text data into a functional, ready-to-use classifier is known as the text classification pipeline. Understanding this pipeline forms the foundation for designing and implementing effective text classification systems.

Figure 3.2.1 illustrates the various stages involved in building a text classifier. Developing a text classifier can be compared to building a house. A house cannot be built by starting with the roof; similarly, a text classifier must be developed through a systematic sequence of steps. Skipping or improperly executing any stage may introduce errors that become difficult and expensive to correct in later stages.

This unit explains the complete text classification pipeline in detail and introduces one of the most fundamental and widely studied algorithms in text classification: Naïve Bayes. It is introduced in almost every NLP textbook, not because it is the most advanced classifier, but because of its simplicity, efficiency, and interpretability. Understanding the working principle of Naïve Bayes provides the intuition required to understand more complex classification algorithms discussed later in this block.

3.2.2 Pipeline - Text Classification

This section introduces the common Natural Language Processing (NLP) pipe-line and the essential Python libraries required to construct a text classification system. The

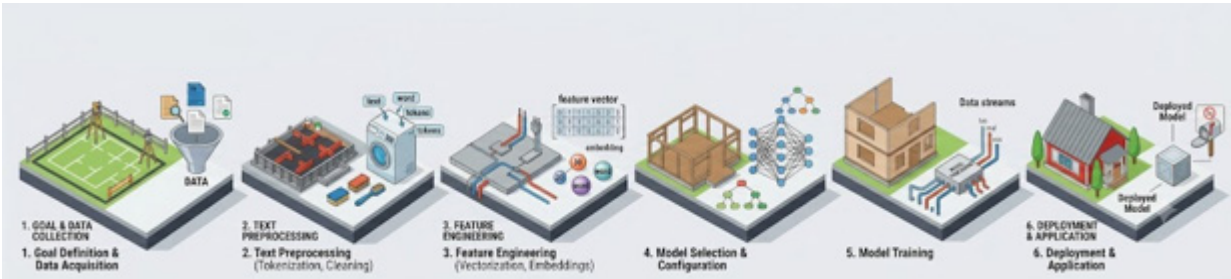


Fig 3.2.1 Building a Text Classification Pipeline – The House Metaphor

SGOU

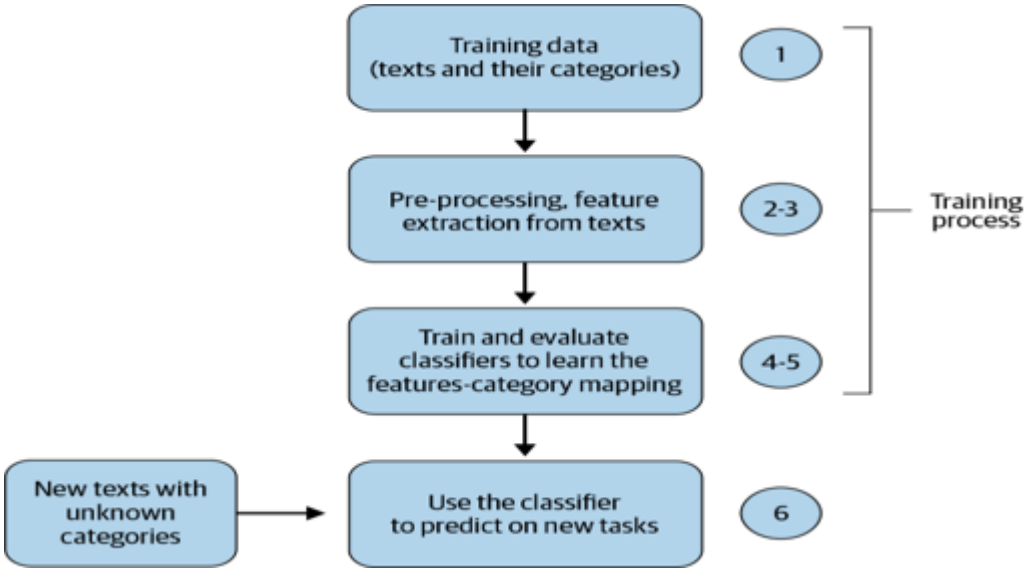


Fig 3.2.2 Flowchart of a text classification pipeline [1]

following steps are typically involved in developing a text classification model (Fig. 3.2.2):

1. Collect or create a labeled data-set suitable for the task.
2. Obtain a suitable labeled dataset relevant to the target classification task.
3. Split the dataset and define evaluation criteria
4. Divide the dataset into two parts (training and testing sets) or three parts (training, validation/development, and testing sets). Select appropriate evaluation metrics for measuring model performance.
5. *Transform raw text into feature representations:* Convert the preprocessed text data into numerical feature vectors that can be processed by machine learning algorithms.
6. *Train the classifier:* Train a classification model using the extracted feature vectors and their corresponding labels from the training dataset.
7. *Evaluate model performance:* Use the selected evaluation metrics from Step 2 to assess the performance of the trained model on the test dataset.
8. *Deploy and monitor the model:* Deploy the trained model for real-world applications and continuously monitor its performance to ensure reliability and effectiveness.

3.2.2.1 Step1 - Data Collection and Labelling

Data collection and labelling in Natural Language Processing (NLP) are fundamental

SGOU



Fig 3.2.3 Labelling Text Data - Sentiment Analysis

steps that involve gathering raw text data and assigning meaningful labels to create ground-truth training data.

For example, Fig 3.2.3 illustrates a sentiment classification task in which customer reviews are labelled as positive, negative, or neutral. These reviews may be collected from various sources, such as news articles, social media platforms, product reviews, and customer feedback.

All text classification systems begin with data. After collecting the text data, each sample must be assigned an appropriate category label. This process, known as annotation, is generally performed by human annotators or supported by annotation tools such as Label Studio and Prodigy. These labels act as the target outputs during model training, allowing the classifier to learn the patterns associated with each category.

After annotation, the dataset is divided into three subsets (as illustrated in Block 3, Unit 1, Fig 3.1.12):

- Training set: Approximately 80% of the dataset is used to train the model by learning patterns from the input text and corresponding labels.
- Development set (Dev set) / Validation set: Used during model development to tune hyperparameters, compare different models, and identify potential issues such as over-fitting.
- Test set: A separate and unseen portion of the dataset used to evaluate the final model performance and measure its ability to generalize to new data.

3.2.2.2 Step 2 - Text Preprocessing

Raw text is often noisy and unstructured. It may contain punctuation marks, inconsistent capitalization, spelling errors, special characters, and words that do not contribute meaningful information for classification. Before extracting features, the text must be cleaned and transformed into a consistent format through preprocessing.

The preprocessing steps covered in Block II - Unit 2 are directly applicable to this stage. These techniques include:

- **Text cleaning:** Removing unnecessary characters, punctuation, and unwanted symbols from the text.

SGOU

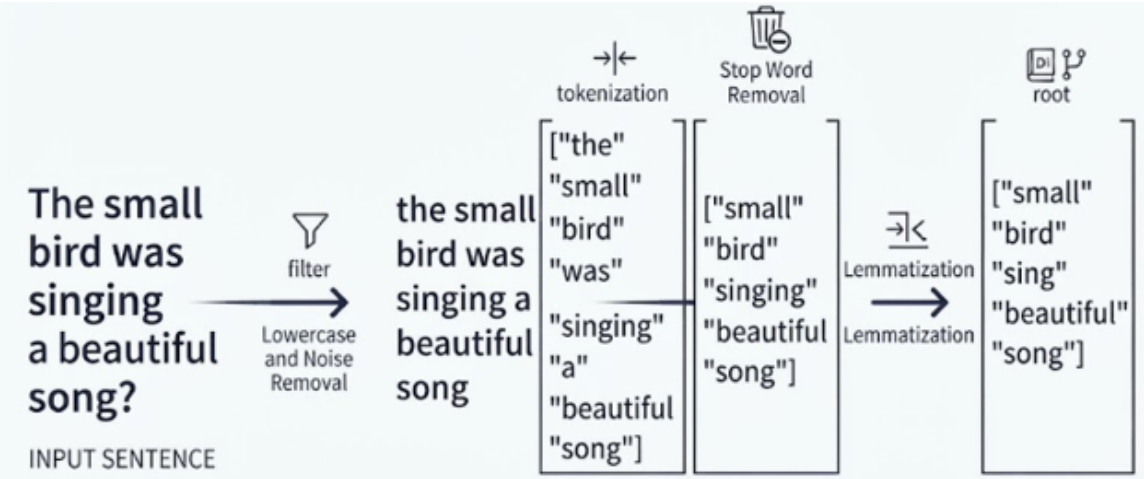


Fig 3.2.4 Text preprocessing Pipeline

- **Tokenization:** Splitting text into smaller units called tokens, such as words or subwords.
- **Case normalization:** Converting text into a standard case (usually lowercase) to avoid treating the same word with different capitalization as separate terms.
- **Stop-word removal:** Eliminating frequently occurring words that provide limited information for classification.
- **Stemming and lemmatization:** Reducing words to their base or root forms to improve consistency.

Proper preprocessing reduces noise and improves the quality of the extracted features, thereby helping the classifier learn meaningful patterns from the text data.

3.2.2.3 Step 3 - Feature Extraction

A machine learning model cannot directly process raw words or text. Feature extraction is the process of converting preprocessed text into numerical representations that can be understood and processed by machine learning algorithms.

The quality of extracted features plays an important role in determining the performance of a text classification system. Different techniques can represent text numerically, depending on the task requirements.

- Bag-of-Words (BoW)
- TF-IDF (Term Frequency–Inverse Document Frequency)
- Word2Vec

These techniques transform text into numerical feature vectors, which are then used to train classification models.

SGOU

```
from sklearn.feature_extraction.text import CountVectorizer
texts = [
    "the film was amazing",
    "terrible film hated it",
    "loved the acting great film"
]
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(texts)

print("Vocabulary:", vectorizer.get_feature_names_out())
print("Feature Matrix:\n", X.toarray())
```

```
Vocabulary: ['acting' 'amazing' 'film' 'great' 'hated' 'it' 'loved' 'terrible' 'the'
 'was']
Feature Matrix:
 [[0 1 1 0 0 0 0 0 1 1]
 [0 0 1 0 1 1 0 1 0 0]
 [1 0 1 1 0 0 1 0 1 0]]
```

Code snippet 3.2.1 Count Vectorization

Code snippet 3.2.1 illustrates the process of Count Vectorization, where text documents are transformed into a numerical feature matrix using the Bag-of-Words (BoW) approach. Each document is represented as a vector, with each element indicating the frequency (count) of a particular word in the document. This numerical representation enables machine learning algorithms to process and classify textual data effectively.

For further details and a deeper understanding, learners are encouraged to refer to the suggested reading [1].

3.2.2.4 Step 4 - Model Training

After feature extraction, the generated feature vectors and their corresponding labels are provided as input to a classification algorithm. During training, the algorithm learns the statistical relationships between the extracted features and their associated class labels.

Examples of classification models include: Logistic Regression, Support Vector Machine (SVM), Decision Trees, and Naïve Bayes (the primary focus of this unit), which uses probability-based approaches for classification.

Model training is performed exclusively using the training set. The model must not access the development (validation) set or the test set during this phase. Exposure to these datasets during training can cause data leakage and lead to an overly optimistic estimate of the model's performance on unseen data.

3.2.2.5 Step 5 - Model Evaluation

After training, the performance of the model is evaluated using the development (validation) set during the model development phase and finally measured on the test set using unseen data. Evaluation metrics are used to quantify how effectively the classifier predicts the correct class labels.

The commonly used evaluation metrics for text classification include:

SGOU

- **Accuracy** - The proportion of all classifications that were correct, whether positive or negative. It is mathematically defined as: Measures the proportion of correctly classified samples among all predictions, including both positive and negative classes. It is mathematically defined as:

Accuracy = (True Positives + True Negatives) / (True Positives + True Negatives + False Positives + False Negatives).

- **Precision** - Measures the proportion of samples predicted as positive that are actually positive. It indicates the reliability of positive predictions.

  Precision = True Positives / (True Positives + False Positives)

- **Recall -** Measures the proportion of actual positive samples that are correctly identified by the model. It indicates the model's ability to detect positive instances.

  Recall = True Positives / (True Positives + False Negatives)

- **F1-score** - Combines precision and recall into a single performance measure using their harmonic mean.

  F1 = 2 × (Precision × Recall) / (Precision + Recall)

Note:

True Positive (TP): The model predicts the sample as positive, and the actual class label is positive.

True Negative (TN): The model predicts the sample as negative, and the actual class label is negative.

False Positive (FP): The model predicts the sample as positive, but the actual class label is negative.

False Negative (FN): The model predicts the sample as negative, but the actual class label is positive.

These four values form the basis for calculating various evaluation metrics. A detailed discussion of these concepts, along with the confusion matrix that provides a visual representation of TP, TN, FP, and FN values, will be covered in the upcoming units.

If the model does not achieve the desired performance level, the classification pipeline must be revisited. Improvements may involve better data cleaning, redesigning the feature extraction approach, tuning the model parameters, or selecting an alternative classification algorithm. This iterative process is represented by the feedback arrow from the evaluation stage to the earlier stages in Figure 3.2.2**.**

3.2.2.6 Step 6 - Deployment

Once the model achieves satisfactory performance on the test set, it is deployed into a real-world application environment, where it can classify new and unseen text data in real time.

SGOU

However, deployment is not the final stage of the text classification pipeline. In practical applications, the characteristics of language and user behaviour continuously evolve. For example, spam messages may adopt new patterns and vocabulary, while new expressions and slang may emerge on social media platforms. These changes can reduce the effectiveness of the deployed model over time.

Therefore, the model must be continuously monitored and periodically retrained using updated data to maintain its accuracy, reliability, and performance in real-world scenarios.

3.2.3 Naive Bayes Classifier

Naive Bayes is a machine learning classification algorithm that predicts the category of a data instance using probability-based reasoning. It is based on Bayes' theorem and assumes that all features are conditionally independent of each other given the class label.

Despite this simplifying assumption, Naïve Bayes performs well in many real-world applications, particularly in text classification tasks, because of its simplicity, efficiency, and ability to handle high-dimensional data.

Examples of applications include:

- Spam filtering
- Document classification
- Sentiment analysis

Bayes' Theorem

Bayes' theorem describes the relationship between conditional probabilities and is given by:

$$P(y|X) = \frac{P(X|y) \, . \, P(y)}{P(X)}$$

where to return to classification,

- P(C|X) – Posterior probability of class C given the observed features X
- P(X|C) – Likelihood probability of observing features X given class C
- P(C) – Prior probability of class C
- P(X) – Probability of observing the features X

In text classification, Naïve Bayes uses this equation to calculate the probability of a document belonging to each possible class and assigns the document to the class with the highest probability.

SGOU

Terminology

Consider a classification problem (Table 3.2.1) that predicts whether a person will play golf based on weather conditions.

Dataset sample:

The dataset can be represented using two main components:

Feature Matrix (X):

The feature matrix contains the input attributes used by the classifier for prediction.

Response Vector (y):

The response vector contains the output class labels corresponding to each data sample.

For the given dataset:

The feature vector (X) consists of the input attributes:

X={Outlook, Temperature, Humidity, Windy}

The class variable **(y)** [response vector] represents the target output:

y = Play Golfy

where the possible class labels are:

y ∈ {Yes, No}

Example:

For a sample data instance:

Table 3.2.1 Weather-based Golf Play Dataset [4]

Outlook	**Temp**	**Humidity**	**Windy**	**Play Golf**
Rainy	Hot	High	FALSE	No
Rainy	Hot	High	TRUE	No
Overcast	Hot	High	FALSE	Yes
Sunny	Mild	High	FALSE	Yes
Sunny	Cool	Normal	FALSE	Yes
Sunny	Cool	Normal	TRUE	No
Overcast	Cool	Normal	TRUE	YES

SGOU

X = (Rainy, Hot, High, False)

the corresponding class label is:

y = No

This indicates that, for the given weather conditions, the person does not play golf.

Applying Bayes' Theorem

For the above classification problem, Bayes' theorem can be applied as:

$$P(y|X) = \frac{P(X|y) \cdot P(y)}{P(X)}$$

where:

- **y** is the class variable.
- X is the feature vector containing nn attributes:

$X = (x_1, x_2, x_3, ..., x_n)$

The term:

P(y|X) represents the probability of a class label **y** given the observed feature vector **X**.

For example:

P(No| Rainy, Hot, High, False)

represents the probability that a person does not play golf given that:

- Outlook = Rainy
- Temperature = Hot
- Humidity = High
- Windy = False

Naïve Bayes Assumption

The fundamental assumption of the Naïve Bayes classifier is that all features are conditionally independent given the class label.

In other words, once the class is known, the presence of one feature does not influence the probability of another feature.

For the golf dataset, this assumption means:

- The temperature being Hot is assumed to be independent of the humidity value.

SGOU

- The outlook being Rainy is assumed to not influence the wind condition.

Therefore, the joint probability can be simplified as:

$P(X \mid y) = P(x_1 \mid y) \times P(x_2 \mid y) \times ... \times P(x_n \mid y)$

This simplification makes Naïve Bayes computationally efficient, especially for high-dimensional text classification problems.

Note: Naïve Bayes does not assume that all features have equal importance. Each feature contributes according to its probability distribution with respect to the class label.

3.2.4 Naive Bayes for Sentiment Classification

The best way to understand Naive Bayes is to follow it through a complete example in Table 3.2.2. We will classify movie reviews as positive or negative. This section studies its application through a complete sentiment classification example. In this example, we classify movie reviews into two categories: positive and negative.

Consider the training corpus shown in Table 3.2.2, which contains five labelled movie reviews.

The objective is to classify the new unseen review:

“predictable with no fun”

Step 1 - Calculate Prior Probabilities

The prior probability represents the probability of each class occurring in the training dataset.

Total number of documents = 5

- Negative reviews = 3
- Positive reviews = 2

Table 3.2.2 Training Dataset for Sentiment Classification

Review	Label
"just plain boring"	NEGATIVE
"entirely predictable and lacks energy"	NEGATIVE
"no surprises and very few interesting moments"	NEGATIVE
"very powerful"	POSITIVE
"the most fun film of the summer"	POSITIVE

SGOU

Therefore,

P(NEGATIVE) =3/5 =0.6

P(POSITIVE) = 2/5 =0.4

Step 2 - Count the Words in Each Class

The words appearing in the training documents are counted separately for each class. Stop words may be removed during preprocessing; however, sentiment-related words such as "no" should be retained because they influence the meaning of the sentence.

The total number of words in each class is calculated to estimate the likelihood probabilities.

Step 3 - Calculate Word Likelihoods

For Naïve Bayes with Laplace smoothing:

$$P(word|class) = \frac{Count(word, class) + 1}{Total\ words\ in\ class + Vocabulary\ size}$$

The likelihood values for the words in the test review are calculated for both classes.

where:

- Count(word,class)Count(word,class) = frequency of the word in that class
- |V| = vocabulary size

For this example:

- Vocabulary size, |V|=20
- Total words in NEGATIVE class = 10
- Total words in POSITIVE class = 6

Step 4 - Calculate the Class Scores

The posterior score for each class is calculated by multiplying the prior probability with the likelihoods of the words:

Score(class) = P(class) × P($word_1$ | class) × P($word_2$ | class)...

For example:

Multiply the prior by the word likelihoods for each class:

Score(NEGATIVE)=P(NEGATIVE)×P(predictable|NEGATIVE)×P(fun|NEGATIVE)

$= 0.6 \times 0.066667 \times 0.033333 \approx 0.00133333$

Score (POSITIVE) = P(POSITIVE) × P(predictable|POSITIVE) × P(fun|POSITIVE)

$= 0.4 \times 0.038 \times 0.077 \approx 0.0011704$

SGOU

Table 3.2.3 Word Likelihoods for each word in the test review

Word	P(word | NEGATIVE)	P(word | POSITIVE)
predictable	Calculated using Laplace smoothing	Calculated using Laplace smoothing
fun	Calculated using Laplace smoothing	Calculated using Laplace smoothing
Word	**P(word | NEGATIVE)**	**P(word | POSITIVE)**
predictable	(1+1) / (10+20) = 2/30 ≈ 0.066667	(0+1) / (6+20) = 1/26 ≈ 0.038
fun	(0+1) / (10+20) = 1/30 ≈ 0.033333	(1+1) / (6+20) = 2/26 ≈ 0.077

Step 5 - Assign the final Label

The class with the higher probability score is selected as the predicted label.

Since:

Score(NEGATIVE) > Score(POSITIVE)

the classifier assigns the label:

NEGATIVE

to the review:

“predictable with no fun”

3.2.5 Code Implementation - Sentiment Analysis

The following example demonstrates the step-by-step implementation of a sentiment analysis system using Python. This example provides a simplified illustration of the complete workflow using a small dataset, including text representation and classification using a machine learning approach.

For real-world applications involving large-scale text corpora, additional preprocessing techniques, feature engineering methods, and comprehensive evaluation strategies may be required to improve the reliability and performance of the model.

1. Import the libraries

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score
```

SGOU

2. Create sample dataset

```
# Small movie review dataset
reviews = [
    "the film was amazing and brilliant",
    "excellent acting and great story",
    "loved every moment of this film",
    "wonderful and very powerful performance",
    "the best film I have seen this year",
    "terrible film hated every moment",
    "boring and completely predictable",
    "worst film ever very disappointing",
    "poor acting and no interesting moments",
    "entirely lacks energy and very dull",
]

labels = [
    "positive", "positive", "positive", "positive", "positive",
    "negative", "negative", "negative", "negative", "negative",
]
```

3. Split the dataset

```
# Split into training (80%) and test (20%) sets
X_train, X_test, y_train, y_test = train_test_split(
    reviews, labels, test_size=0.2, random_state=42
)
```

4. Vectorization

```
# Convert text to numerical features using TF-IDF
vectorizer = TfidfVectorizer(stop_words="english")

# Learn vocabulary from training data and transform
X_train_vec = vectorizer.fit_transform(X_train)

# Transform test data using the same vocabulary
X_test_vec = vectorizer.transform(X_test)
```

5. Train the model

```
# Train a Multinomial Naïve Bayes classifier
model = MultinomialNB()
model.fit(X_train_vec, y_train)
```

6. Model performance

```
# Predict labels for the test set
y_pred = model.predict(X_test_vec)

# Calculate evaluation metrics
accuracy  = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

To execute this code, open a Python environment such as Google Colab, Visual Studio Code (VS Code), or Jupyter Notebook, and install the required libraries before running the program. Detailed information about the functions, classes, and parameters used in this implementation is available in the official scikit-learn documentation: https://scikit-learn.org/stable/

SGOU

Recap

- The text classification pipeline consists of six stages: data collection and labelling, preprocessing, feature extraction, model training, evaluation, and deployment.
- The collected dataset is generally divided into three subsets: training set, development (validation) set, and test set.
- Annotation is the process of assigning appropriate category labels to raw text data. It can be performed by human experts or supported by annotation tools such as Label Studio and Prodigy.
- Text preprocessing prepares raw text for analysis by performing operations such as lowercasing, punctuation removal, tokenization, stop-word removal, and stemming or lemmatization.
- Features are numerical representations of text that serve as inputs to classification algorithms. Common feature representation techniques include Bag-of-Words, TF-IDF, and word embeddings.
- Part-of-Speech (POS) tagging is the process of identifying the grammatical role of each word in a sentence and assigning an appropriate POS label.
- Model evaluation measures classifier performance using metrics such as accuracy, precision, recall, and F1-score.
- If the model does not achieve satisfactory performance, earlier stages of the pipeline must be revisited and improved.
- Deployment is not the final stage of the process. The model should be continuously monitored and periodically retrained using updated data as language patterns evolve in the target domain.
- Naïve Bayes is a probabilistic classifier that assigns a document to the class with the highest posterior probability based on the words present in the document.
- F1-score is the harmonic mean of precision and recall, given by: F1 = 2 × (Precision × Recall) / (Precision + Recall).

SGOU

Objective Type Questions

1. What is the first stage of the text classification pipeline?
2. How many stages are present in a text classification pipeline have?
3. Name two feature extraction methods used in text classification.

4. Write the formula for the F1-score.

5. Name one annotation tool used for labelling text data.

6. Write the formula for accuracy.

7. What type of classifier is Naive Bayes ?

8. In the worked example, which label was assigned to the review "predictable with no fun"?

9. Name the three subsets into which a labelled dataset is divided before model training.

10. What does POS tagging identify in a sentence?

11. Name one real-world application of Naive Bayes classification.

12. What happens if a model does not achieve the required performance level during evaluation?

13. What does P(y|X) represented in Bayes' theorem?

14. Into how many subsets is the dataset divided in Naive Bayes example?

15. What does X represent in the golf dataset example?

16. What is the term used for the process of assigning labels to text data?

Answers to Objective Type Questions

1. Data collection.

2. Six

3. Bag-of-Words and TF-IDF.

4. F1 = 2 × (Precision × Recall) / (Precision + Recall).

5. Label Studio

6. Accuracy = (True Positives + True Negatives) / (True Positives + True Negatives + False Positives + False Negatives).

7. A probabilistic classifier based on Bayes' theorem.

SGOU

8. NEGATIVE

9. Training set, development set, and test set.

10. The grammatical role of each word in a sentence.

11. Spam filtering

12. he classification pipeline is revisited, and earlier stages such as preprocessing, feature extraction, or model selection may be improved. The probability of class y given the feature vector X .

13. The probability of class yy given the feature vector X

14. Two components: the feature matrix and the response vector.

15. The feature vector containing the input attributes (Outlook, Temperature, Humidity, and Windy).

16. Annotation.

Assignments

1. Describe the six stages of the text classification pipeline. Explain what happens at each stage and why the order matters.

2. What is Naive Bayes? Explain how it uses probability to classify text and why it is considered one of the most important algorithms for text classification.

3. Write a Python pipeline that preprocesses a set of text reviews using NLTK (tokenization, stop word removal, stemming) and then applies each step of the preprocessing pipeline described in Section 3.2.2.

4. How Naive Bayes performs in sentiment analysis. Illustrate the process with a suitable example.

References

1. S. Vajjala, B. Majumder, A. Gupta, and H. Surana, *Practical Natural Language Processing: A Comprehensive Guide to Building Real-World NLP Systems*. O'Reilly Media, 2020.

SGOU

2. D. Jurafsky and J. H. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, 2023.

3. Siino, I. Tinnirello, and M. La Cascia, "From Foundations to GPT in Text Classification: A Comprehensive Survey on Current Approaches and Future Trends," *Foundations and Trends in Information Retrieval*, vol. 19, no. 5, pp. 557–711, 2025.

4. T. M. Mitchell, *Machine Learning*. McGraw-Hill, 1997.

Suggested Reading

1. Scikit-learn, "Feature Extraction." [Online]. Available: https://scikit-learn.org/stable/modules/feature_extraction.html [Accessed: 6 September 2026].

SGOU

Unit 3

Text Classification Algorithms

Learning Outcomes

After completing this unit, learner will be able to:

- explain how the Naive Bayes estimates its parameters from training data
- apply techniques to optimize Naive Bayes for sentiment analysis
- explain how Logistic Regression classifies text using the sigmoid function and feature weights
- implement Logistic Regression for sentiment classification using Python
- describe how a Support Vector Machine (SVM) identifies the optimal decision boundary
- compare Naive Bayes, Logistic Regression, and SVM and select AN appropriate algorithm for a given text classification task

Prerequisites

This unit builds upon the concepts introduced in Unit 2, where Naïve Bayes was introduced and applied to sentiment classification using a movie review dataset. Before proceeding, ensure that you have a clear understanding of prior probabilities, likelihood estimation, and the Naïve Bayes classification process discussed in Unit 2.

You should also be familiar with the text classification pipeline and feature extraction techniques covered in the previous unit, as these concepts form the foundation for the algorithms discussed in this unit.

No prior knowledge of Logistic Regression or Support Vector Machines (SVMs) is required. A basic understanding of probability concepts and familiarity with Python programming will be sufficient to follow the explanations and code implementations presented in this unit.

SGOU



Keywords

Support Vectors, Kernel, Feature Vector, Margin, Hyperplane, Gradient descent

Discussion

3.3.1 Introduction

In Unit 2, you learned that the Naïve Bayes classifier classifies a new review by combining the prior probability of a class with the likelihood probabilities of the words present in the review. However, the process of estimating these probabilities from training data and techniques for improving the classifier's performance for sentiment analysis were not discussed in detail.

This unit addresses both of these aspects in detail. You will also learn about two advanced classification algorithms: Logistic Regression and Support Vector Machines (SVMs). These approaches overcome some of the limitations of Naïve Bayes by using different learning strategies for decision-making.

By the end of this unit, you will understand three important algorithms for text classification, Naïve Bayes, Logistic Regression, and Support Vector Machines and learn how to select an appropriate algorithm for a given text classification task.

3.3.2 Naive Bayes Classifier -Training

In Unit 2, you learned how the Naïve Bayes classifier classifies a new document by calculating probabilities from labelled training data. This section explains how these probabilities are estimated during the training process.

Training a Naïve Bayes classifier involves estimating two important sets of probability values from the labelled training dataset:

- Prior probability of each class – the probability of each class occurring in the training data.
- Likelihood probability of each word given a class – the probability of observing a particular word within a specific class.

Once these probabilities are calculated, classification becomes straightforward. For a new document, the classifier multiplies the prior probability of each class with the likelihood probabilities of the words present in the document. It assigns the document to the class with the highest probability score.

The detailed working of the Naïve Bayes classifier was introduced in Unit 2 of Block 3.

3.3.2.1 Estimating the Prior Probability

The prior probability of a class represents the proportion of training documents that belong to that class. It indicates the initial likelihood of a class occurring before considering the words present in a new document.

SGOU

The prior probability is calculated as:

$$P(Class) = \frac{Number\ of\ documents\ in\ the\ class}{Total\ number\ of\ documents}$$

Example:

Consider a training corpus containing:

- Positive reviews = 60
- Negative reviews = 40
- Total reviews = 100

The prior probabilities are:

$$P(\text{Positive}) = \frac{60}{100} = 0.60$$

$$P(\text{Negative}) = \frac{40}{100} = 0.40$$

A training corpus containing a larger number of positive reviews will result in a higher prior probability for the positive class. This means that, before considering the actual words in a new review, the classifier already considers the positive class more likely based on the distribution of the training data.

3.3.2.2 Estimating Word Likelihoods

The likelihood probability of a word given a class represents how frequently a particular word appears in documents belonging to that class. It indicates how strongly a word is associated with a specific category.

It is calculated by counting the number of times a word appears in documents of a particular class and dividing it by the total number of words present in that class:

$$P(\text{word}|\text{class}) = \frac{Count\ of\ word\ in\ class}{Total\ number\ of\ \ words\ in\ class}$$

For example, if the word **"**excellent**"** appears frequently in positive reviews, then:

P(excellent|Positive)

will have a higher value, indicating that the word is strongly associated with the positive class.

However, a problem occurs when a word does not appear in a particular class. In such cases, the probability becomes zero, causing the entire classification probability

SGOU

Label	Review
NEGATIVE	just plain boring
NEGATIVE	entirely predictable and lacks energy
NEGATIVE	no surprises and very few laughs
POSITIVE	very powerful
POSITIVE	the most fun film of the summer
TEST	*"predictable with no fun"*

Fig 3.3.1 Training Corpus [1]

Table 3.3.1 Word Counts in Each class

Word	NEG count	POS count
predictable	1	0
no	1	0
fun	0	1
plain	1	0
boring	1	0
entirely	1	0
lacks	1	0
energy	1	0
surprises	1	0
laughs	1	0
powerful	0	1
film	0	1
summer	0	1
very	1	1

Total NEG words = 14 Total POS words = 9
Vocabulary |V| = 20 (unique words across all classes)

SGOU

to become zero. To overcome this issue, Laplace smoothing (add-one smoothing) is commonly applied:

$$P(word \mid class) = \frac{Count\ (word,\ class) + 1}{Total\ words\ in\ class + \ |V|}$$

where |V| represents the total vocabulary size.

These likelihood values, along with the prior probabilities, are used by the Naïve Bayes classifier to calculate the probability of each class for a new document.

3.3.2.3 Worked Example

Figure 3.3.1 illustrates how a Naïve Bayes classifier works using a small training corpus

of movie reviews. In this example, sentiment analysis uses two classes: Positive and Negative. A small set of training documents and a new test document are considered to demonstrate how the classifier predicts the sentiment category.

The training corpus consists of simplified movie reviews, and the objective is to classify the unseen review:

S=”predictable with no fun”

Step 1: Calculate Prior Probabilities

The prior probability of each class is calculated from the number of documents belonging to each class.

Total documents = 5

- Negative documents = 3
- Positive documents = 2

Therefore,

P(NEGATIVE) = $\frac{3}{5}$= 0.60

P(POSITIVE) = $\frac{2}{5}$ = 0.40

Step 2: Remove Unknown Words

The test sentence is:

S=”predictable with no fun”

The word “with” does not appear in the training vocabulary. Therefore, it is removed.

Remaining words:

predictable, no, fun

Step 3: Calculate Word Likelihoods

Using Laplace smoothing:

$$P(word|class) = \frac{Count(word, class) + 1}{Total\ words\ in\ class + |V|}$$

where:

- Total negative words = 14
- Total positive words = 9
- Vocabulary size, |V|=20

SGOU

For NEGATIVE class

Denominator:

$$14 + 20 = 34$$

$$P(\text{predictable} \mid \text{NEGATIVE}) = \frac{1+1}{34} = 0.0588$$

$$P(\text{“no”} \mid \text{NEGATIVE}) = \frac{1+1}{34} = 0.0588$$

$$P(\text{“fun”} \mid \text{NEGATIVE}) = \frac{0+1}{34} = 0.0294117$$

For POSITIVE class

Denominator:

$$9 + 20 = 29$$

$$P(\text{“predictable”} \mid \text{POSITIVE}) = \frac{0+1}{29} = 0.03448$$

$$P(\text{“no”} \mid \text{POSITIVE}) = \frac{0+1}{29} = 0.03448$$

$$P(\text{“fun”} \mid \text{POSITIVE}) = \frac{1+1}{29} = 0.06896$$

Step 4: Calculate Class Scores

The posterior score is calculated as:

$$\text{Score(class)} = P(\text{class}) \times P(\text{word}_1|\text{class}) \times P(\text{word}_2|\text{class}) \times P(\text{word}_3|\text{class})$$

For the sentence S = ”predictable with no fun”, after removing the word ‘with’, the chosen class is therefore computed as follows:

Negative Class Score Score(NEGATIVE)

: P(NEGATIVE) x P(word|NEGATIVE)

= 0.60 x 0.0588 x 0.0588 x 0.0294117

= 0.00006101 = 6.101×10^{-5}

Positive Class Score Score(POSITIVE)

: P(POSITIVE) x P(word|POSITIVE)

= 0.40 x 0.03448 x 0.03448 x 0.06896

= 0.00003279 = 3.279×10^{-5}

SGOU

Step 5: Assign the Predicted Label

The classifier compares the two scores:

Score(NEGATIVE)

$$= 6.101 \times 10^{-5} Score(POSITIVE)$$

$$= 3.279 \times 10^{-5}$$

Since,

$$6.101 \times 10^{-5} > 3.279 \times 10^{-5}$$

the Naïve Bayes classifier predicts:

$$NEGATIVE$$

Therefore, the sentiment of the review:

"predictable with no fun"

is classified as NEGATIVE.

3.3.3 Optimizing for Sentiment Analysis

The basic Naïve Bayes model introduced in Unit 2 provides a strong baseline for sentiment classification. However, sentiment analysis involves several linguistic challenges, such as negation, varying word importance, and context-dependent meanings, which require specific optimization strategies.

Several practical techniques can improve the performance of the Naïve Bayes classifier for sentiment analysis. These optimizations focus on improving text representation, handling linguistic patterns, and refining the probability estimation process to achieve more accurate classification results.

3.3.3.1 Binary Naive Bayes - Clipping Word Counts

In the standard Naive Bayes approach, each word is counted according to the number of times it appears in a document. However, for sentiment classification tasks, it is often more useful to consider whether a word appears in a document rather than how many times it is repeated.

For example, if a reviewer uses the word **"amazing"** three times in a review, it does not necessarily indicate that the sentiment is three times more positive than a review where the word appears only once. To address this issue, Binary Naïve Bayes represents word occurrence using binary values, converting each word count in a document to either 1 (present) or 0 (absent) before training.

The modification is simple: for each training document, repeated occurrences of the same word are ignored, and only the presence or absence of a word is considered. These binary word representations are then combined to form the class-level document representation. Similarly, duplicate word occurrences in test documents are ignored

SGOU

Table 3.3.2 Comparison of Standard Word Counts and Binary Word Counts After Per-Document Binarization

Word	Count (POS)	Count (NEG)	Binary Count (POS)	Binary Count (NEG)
and	2	0	1	0
boxing	0	1	0	1
film	1	0	1	0
great	3	1	2	1
no	0	1	0	1
pathetic	0	1	0	1
plot	1	1	1	1
satire	1	0	1	0
scenes	1	2	1	2
twists	1	1	1	1
worst	0	1	0	1

before classification. The remaining steps of the Naïve Bayes algorithm remain unchanged.

Table 3.3.1 illustrates the difference between standard word counts and binary word counts using four short movie reviews. The Counts column represents the raw frequency of words across all documents belonging to each class. The Binary Count column represents the frequencies obtained after applying per-document binarization, where duplicate occurrences within the same document are removed before counting. However, the same word can still contribute multiple times if it appears in different documents of the same class.

It is important to note that binarization does not convert every word count in a class to 1. For example, the word **"great"** may have a binary count of 2 in the positive class if it appears once in each of two different positive documents. Per-document binarization removes repeated occurrences within an individual document but does not remove occurrences across different documents.

3.3.3.2 Negation Handling

Consider the difference between the sentences *"I really like this movie"* and *"I didn't like this movie."* The addition of the word **"didn't"** completely changes the sentiment of the word **"like"**. Similarly, a negative word can sometimes reverse the meaning of a phrase. For example, *"don't dismiss this film"* and *"don't let us get bored"* convey positive sentiments even though they contain negative words.

SGOU

A simple and effective approach for handling negation during text preprocessing is to add a NOT_ prefix to every word that follows a negation word, such as *not, n't, no,* or *never*, until the next punctuation mark is encountered.

Example:

The sentence:

"I didn't like this movie, but I recommend it"

is transformed into:

"I didn't NOT_like NOT_this NOT_movie, but I recommend it"

The prefixed terms, such as NOT_like and NOT_movie, preserve the effect of negation and provide additional information to the classifier. During training, these modified terms help the model learn that certain word combinations are associated with negative sentiment. Similarly, expressions such as NOT_bored and NOT_dismiss may acquire positive sentiment associations because they represent phrases where negation changes the original meaning.

This simple rule-based approach improves sentiment classification performance in many cases and does not require additional linguistic resources or complex language models.

3.3.3.3 Using Sentiment Lexicons

When labelled training data is limited or does not adequately represent the target domain, the standard approach of learning from individual word occurrences may not generalize effectively.

In such situations, sentiment lexicons provide an alternative source of sentiment information. A sentiment lexicon is a pre-annotated collection of words associated with sentiment polarity, where words are labelled as positive or negative based on their emotional meaning.

Several widely used sentiment lexicons are available, including VADER (Valence Aware Dictionary and Sentiment Reasoner) and NRC Emotion Lexicon. Examples of sentiment words include:

- Positive words: admirable, beautiful, confident, dazzling, ecstatic, glee, great
- Negative words: awful, bad, catastrophe, cheat, envious, foul, harsh, hate

A common approach to incorporating lexicon information into a Naïve Bayes classifier is to add binary features rather than treating every sentiment word as an individual feature.

The additional features can be defined as:

- Feature 1: The document contains one or more words from the positive sentiment lexicon.

SGOU

- Feature 2: The document contains one or more words from the negative sentiment lexicon.

All occurrences of words from the positive lexicon contribute to Feature 1, regardless of the specific word that appears. Similarly, words from the negative lexicon contribute to Feature 2. This approach creates a small number of dense and reliable sentiment-related features instead of thousands of sparse individual word features.

When the training dataset is large and representative of the target domain, using individual word features generally provides better performance. However, when training data is limited, or the target domain differs significantly from the training domain, lexicon-based features often generalize better because they capture sentiment patterns that remain consistent across different domains.

3.3.4 Logistic Regression for Text Classification

The goal of a classification system is to take a single input, called an observation, extract useful features from it, and assign it to one of a set of discrete classes.

For text classification, the input document is represented as a feature vector:

$x = [x_1, x_2, x_3, \ldots, x_n]$

where each x_i represents a feature value, such as the presence or frequency of a word in the document.

Logistic Regression assigns a weight to each feature (for example, each word in the vocabulary). These weights indicate how strongly each feature contributes to the classification decision.

- Words that frequently appear in positive documents receive large positive weights.
- Words that frequently appear in negative documents receive large negative weights.
- Words that appear with similar frequency in both classes receive weights close to zero.

We call the input x**,** and the output comes from a fixed set of output classes $Y = \{y_1, y_2, \ldots, y_m\}$**.**

For sentiment analysis**,**

- The input x is a review or other piece of text
- The output set Y might be {positive, negative} or simply {0, 1}.

For example**:** For language identification,

- The input is a text whose language is unknown
- The output set Y is the full set of possible languages.

SGOU

Each feature value is multiplied by its corresponding weight w_i, the results are summed, and a bias term b is added:

- $z = w \cdot x + b$ **(3.3.1)**

  $= w_1x_1 + w_2x_2 + \ldots + w_nx_n + b$

- W_i - Represents how important that feature is to the classification decision.
- Each weight w_i represents how important that feature is to the classification decision. A positive weight provides evidence that the document belongs to the positive class. A negative weight provides evidence that it belongs to the negative class.

The model calculates a weighted sum of all input features and passes the result through the sigmoid function to produce a probability value between 0 and 1 (illustrated in Figure 3.3.2).

The input is represented as **x**, and the output belongs to a fixed set of classes:

$$Y = \{y_1, y_2, \ldots, y_m\}$$

Examples:

For sentiment analysis:

- The input x is a review or any other text document.
- The output set Y may be:

  Y = {Positive, Negative}

  or

  Y = {0,1}

where 0 and 1 represent the two sentiment classes.

For language identification:

- The input is a text document whose language is unknown.
- The output set Y consists of all possible language categories.

Feature Weights and Linear Combination

Each feature value is multiplied by its corresponding weight w_i The weighted features are summed, and a bias term b is added:

$z = w \cdot x + b$ **(3.3.2)**

Expanding:

SGOU

$$z = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + b$$

where:

- x_i represents the value of the i^{th} feature.
- w_i represents the importance of the i^{th} feature in the classification decision.
- b represents the bias term.

The weight assigned to each feature determines its contribution to the final prediction:

- A positive weight indicates that the feature provides evidence towards the positive class.
- A negative weight indicates that the feature provides evidence towards the negative class.
- A weight close to zero indicates that the feature has little influence on the classification decision

Sigmoid Function

In Equation (3.3.3), the value of **z** can range from $-\infty$ to $+\infty$. Since probability values must lie within the range of 0 to 1, the linear output z cannot be directly interpreted as a probability. Therefore, z is passed through the sigmoid function to transform it into a probability value (Figure 3.3.3).

The sigmoid function is defined as:

$$\sigma(z) = \frac{1}{(1 + e^{-z})} \qquad \textbf{(3.3.3)}$$

The sigmoid function maps any real-valued input into the range (0, 1), making it suitable for binary classification problems.

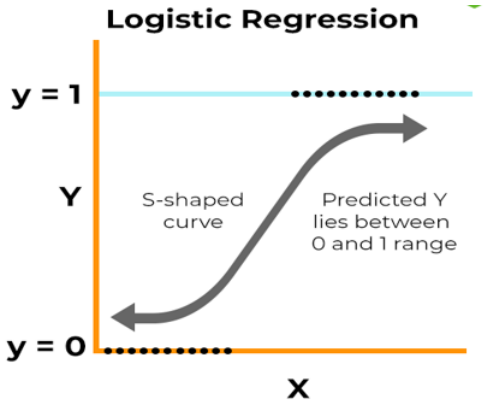


Fig 3.3.2 Sigmoid Function in Logistic Regression for Binary Classification

SGOU

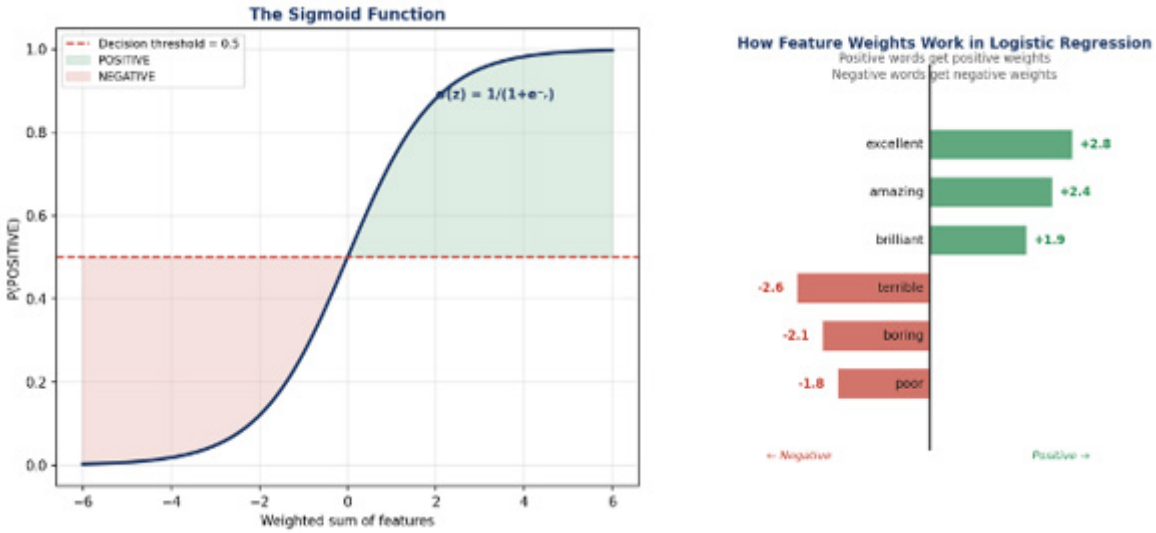


Fig 3.3.3 Conversion of Weighted Features into Sentiment Probabilities Using Logistic Regression

For sentiment classification:

$$P(x) = \sigma(w \cdot x + b)$$

represents the probability that the input document xx belongs to the positive class.

The probability of the negative class is calculated as:

$$P(x) = 1 - \sigma(w \cdot x + b)$$

Thus, Logistic Regression converts the weighted feature combination into class probabilities, and the class with the higher probability is selected as the predicted output.

Decision Rule

A decision boundary of 0.5 is commonly applied in Logistic Regression for binary classification.

- If: $P(x) > 0.5$

the document is assigned to the *positive class*.

- Otherwise, the document is assigned to the *negative class*.

The threshold value of 0.5 indicates that the model chooses the positive class when the probability of belonging to that class is greater than 50%.

Note:

- The weights w and bias b are not manually assigned. They are learned automatically from the labelled training data during the training process.
- Logistic Regression training involves two important components:

SGOU

1. Loss function

2. Optimization algorithm, such as gradient descent

Loss Function

A loss function measures the difference between the model's predicted probability $\hat{y}$ and the actual class label y. For binary Logistic Regression, the commonly used loss function is binary cross-entropy loss:

$$L(\hat{y}, y) = -[y \backslash log(\hat{y}) + (1 - y) \backslash log(1 - \hat{y})] \quad \textbf{(3.3.4)}$$

where:

- y is the actual class label (0 or 1).
- $\hat{y}$ is the predicted probability produced by the Logistic Regression model.
- L($\hat{y}$,y) represents the loss for a single training example.

A smaller loss indicates that the predicted probability is closer to the actual class label, whereas a larger loss indicates a poor prediction.

Optimization Algorithm

An optimization algorithm adjusts the model parameters, such as weights and bias, to minimize the loss function and improve prediction accuracy.

Gradient Descent

Gradient descent is an iterative optimization algorithm used to learn the weights of a Logistic Regression model. It starts with initial (often random) weight values and gradually updates them in the direction that reduces the loss.

The learning process involves the following steps:

1. Make predictions

The model predicts the class probabilities for each training document using the current weights and bias.

2. Calculate the error

The predicted probabilities are compared with the actual class labels, and the loss value is calculated.

3. Update the parameters

The weights and bias are adjusted slightly in the direction that minimizes the loss.

4. Repeat the process

These steps are repeated over multiple iterations until the model parameters converge and the loss is minimized.

SGOU

3.3.5 Support Vector Machine for Text classification

Consider two groups of points scattered on a sheet of paper, with red squares representing negative reviews and green circles representing positive reviews. The task is to draw a boundary that separates the two groups. There are infinitely many possible boundaries that can separate the two classes. The question is: Which boundary should be selected?

A Support Vector Machine (SVM) answers this question using a clear principle: it selects the decision boundary that is as far as possible from the nearest data points of the two classes. The distance between the decision boundary and the nearest data points is related to the margin. SVM finds the boundary, called a hyperplane in higher-dimensional feature spaces, that maximizes this margin.

The training points that lie closest to the decision boundary and determine its position are called support vectors.

Figure 3.3.4 illustrates this concept for sentiment classification, where each point represents a review and the two classes are separated by a decision boundary.

The SVM finds the hyperplane that maximizes the margin between the positive and negative classes. The circled points in the figure represent the support vectors, which play a critical role in determining the position of the decision boundary.

3.3.5.1 Linear SVM for Text Classification

When text is represented as TF-IDF vectors, each document becomes a high-dimensional numerical feature vector. Although the resulting data is not necessarily perfectly linearly separable, Linear SVM is well suited to text classification because it can learn an effective linear decision boundary in this high-dimensional feature space.

For example, consider a TF-IDF representation of movie reviews. A positive film review may have relatively high TF-IDF values for words such as *brilliant*, *excellent*, and *powerful*, and relatively low values for words such as *boring* and *terrible*. A negative

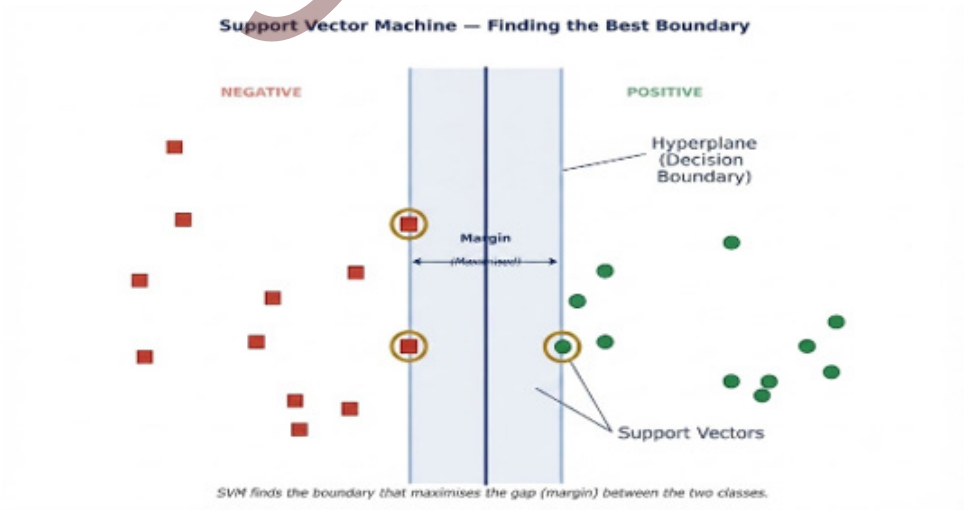


Fig 3.3.4 SVM boundary separation - Linearly separable

SGOU

review may exhibit the opposite pattern. These differences in feature values allow a Linear SVM to learn a hyperplane that separates the two sentiment classes as effectively as possible while maximizing the margin.

Linear SVM is also relatively robust to noise when appropriate regularization is used. Training examples that are far from the decision boundary generally have little or no influence on the position of the maximum-margin hyperplane. However, mislabeled or noisy examples that are close to the boundary can still influence the model, particularly when they become support vectors.

This makes Linear SVM a strong choice for text classification, especially when the feature space is high-dimensional and sparse, as is commonly the case with Bag-of-Words and TF-IDF representations.

3.3.5.2 Non-Linear SVM - The Kernel Trick

Sometimes, the data may be non-linearly separable. Figure 3.3.5 shows an example in which positive (Class A) and negative (Class B) data points are arranged in such a way that no single straight-line decision boundary can effectively separate the two classes. In such cases, SVM can use a technique called the kernel trick to handle non-linear patterns.

A kernel function is a mathematical function that measures the similarity between pairs of data points. The kernel trick enables SVM to implicitly represent the data in a higher-dimensional feature space, where a linear decision boundary may be able to separate the classes more effectively. This avoids explicitly transforming the data into the higher-dimensional space.

The commonly used kernel functions include:

- **Linear Kernel:** Computes the dot product between two feature vectors. It corresponds to a standard Linear SVM and is particularly effective for **high-dimensional and sparse text data**, such as Bag-of-Words and TF-IDF representations.

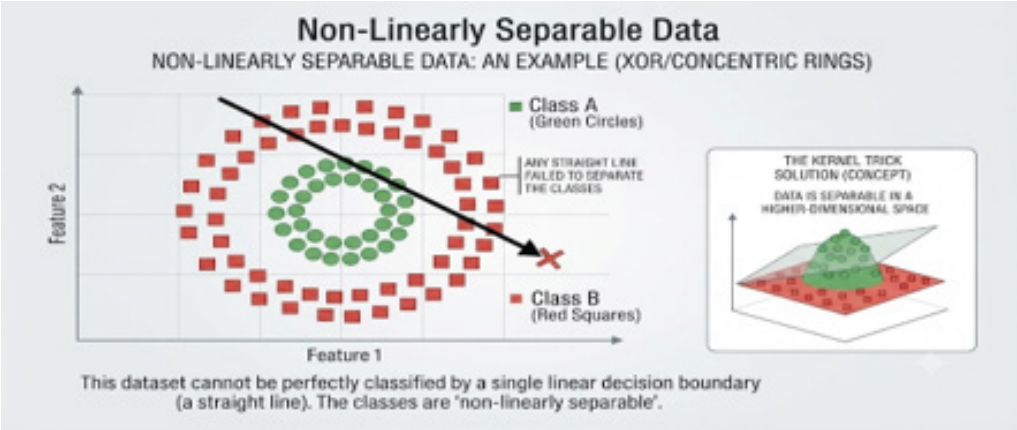


Fig 3.3.5 SVM Classification of Non-Linearly Separable Data Using the Kernel Trick

SGOU

- **Polynomial Kernel:** Captures relationships and interactions among features by considering polynomial combinations of the input features. It can be useful when interactions between features are important for classification.

- **RBF Kernel (Radial Basis Function):** Measures similarity between data points based on their distance in the feature space. It can model complex, non-linear decision boundaries and is particularly useful for datasets where a linear boundary is insufficient.

3.3.6 Which is the right choice?

After studying all three classifiers, it is important to understand not only how each algorithm works but also when each classifier is appropriate.

1. When to use Naive Bayes

Naïve Bayes is a suitable choice when:

- Fast training and prediction are more important than achieving the highest possible accuracy.

- The dataset is small or moderately sized, particularly when a simple probabilistic classifier is required.

- The data contains high-dimensional features, such as Bag-of-Words or TF-IDF representations.

- The features approximately satisfy the conditional independence assumption of Naïve Bayes.

- A simple and computationally efficient baseline is required.

Naïve Bayes trains quickly, handles high-dimensional text data efficiently, and often performs well in applications such as spam detection, document classification, and short-text classification.

2. When to use SVM

SVM is particularly suitable when:

- The feature space is high-dimensional, as commonly occurs in text classification.

- There are many features relative to the number of training examples.

- High classification accuracy is important.

- A Linear SVM can be used with sparse representations such as TF-IDF.

Linear SVM often performs very well on large text classification tasks because it can learn an effective decision boundary in a high-dimensional feature space.

SGOU

One limitation of the standard SVM output is that it does not directly provide probability estimates. It primarily produces a class label or decision score. If probability estimates are required, additional probability calibration can be applied.

3. When to use Logistic Regression

Logistic Regression is widely used for text classification tasks and is particularly useful when:

- Probability estimates are required along with class predictions.
- The feature space is high-dimensional and sparse.
- An interpretable linear classification model is desired.
- Feature weights need to be examined to understand their contribution to the prediction.

Logistic Regression often performs as well as or better than Naïve Bayes on medium- to large-sized datasets because it learns feature weights jointly by optimizing a classification loss function, rather than estimating each feature's probability independently.

It also provides probability estimates for the predicted classes. The learned weight associated with each word indicates the direction and strength of its contribution to the classification decision:

- A positive weight supports the positive class.
- A negative weight supports the negative class.
- A weight close to zero indicates little influence on the decision.

Logistic Regression is widely used in practical applications, including sentiment analysis, document classification, and other binary and multiclass text classification tasks.

SGOU

Recap

- Training a Naive Bayes classifier involves estimating two sets of probabilities: the prior probability of each class and the likelihood of each word given each class.
- Laplace smoothing adds one to each word count before calculating the likelihood, ensuring that an unseen word does not result in a zero probability.
- Log probabilities are used instead of raw probabilities to avoid numerical underflow when multiplying many small probability values.
- Binary Naïve Bayes converts word counts within each document to binary values (0 or 1), because the presence of a word can be more informative than its frequency for some sentiment classification tasks.

- Negation handling adds a NOT_ prefix to words following a negation term until the next punctuation mark, allowing the model to distinguish between expressions such as *good* and *not good*.
- Logistic Regression extracts features from the input, multiplies each feature by its learned weight, adds a bias term, and passes the resulting weighted sum through the sigmoid function to produce a probability.
- The sigmoid function maps any real-valued input to a value between 0 and 1, making it suitable for probability estimation in binary classification.
- A decision threshold of 0.5 is commonly used in binary Logistic Regression. If P(POSITIVE|x)>0.5, the document is assigned to the positive class; otherwise, it is assigned to the negative class.
- Logistic Regression learns its weights and bias through an iterative optimization process, such as gradient descent, which minimizes the cross-entropy loss on the training data.
- A Support Vector Machine finds the hyperplane that maximises the margin between the two classes in the feature space.
- A kernel function implicitly maps the data into a higher-dimensional space where a straight boundary can be found.
- A Support Vector Machine (SVM) finds a hyperplane that maximizes the margin between the two classes in the feature space.
- A kernel function enables SVM to handle non-linear relationships by implicitly representing the data in a higher-dimensional feature space, where a linear decision boundary may be found.
- Support vectors are the training samples closest to the decision boundary and play a critical role in determining the position of the maximum-margin hyperplane.

SGOU

Objective Type Questions

1. What are the two sets of probabilities estimated during Naive Bayes training?
2. Give one example of a word that should be retained rather than removed during sentiment preprocessing.
3. What does the sigmoid function map the weighted sum to in Logistic Regression?
4. Write the formula for the sigmoid function.

5. What is the decision threshold commonly used in binary Logistic Regression??
6. What loss function is used to train Logistic Regression?
7. Name the three commonly used kernel functions in SVM.
8. Which algorithm is commonly used to optimize the weights in Logistic Regression?
9. What type of classification problem does Logistic Regression primarily solve?
10. What is the name of the constant term added to the weighted sum in Logistic Regression?
11. What is the numerical representation of text used as input to Logistic Regression called?
12. What is the distance between the decision boundary (hyperplane) and the nearest data points related to?
13. What are the training points closest to the decision boundary called?
14. Which kernel is most commonly used for text classification?
15. Which technique enables SVM to handle non-linear data by implicitly representing it in a higher-dimensional space?

Answers to Objective Type Questions

1. The prior probability of each class, P(c), and the likelihood probability of each word given each class, P(w|c).
2. Not, no, or never.
3. A value between 0 and 1.
4. $\sigma(z) = 1 / (1 + e^{-z})$
5. A threshold of 0.5
6. Binary cross-entropy (cross-entropy) loss
7. Linear kernel, Polynomial kernel, and RBF (Radial Basis Function) kernel.

SGOU

8. Gradient Descent
9. Binary classification.
10. Bias
11. Feature Vector
12. Margin
13. Support Vectors
14. Linear kernel.
15. Kernel Trick

Assignments

1. Explain how Naive Bayes estimates its parameters from training data. Why are the prior probabilities and word likelihoods necessary for making the model practically usable?
2. Describe the complete flow of text classification using Logistic Regression. Starting with a raw text document, explain each stage of the classification process.
3. Explain the core idea behind Support Vector Machines. How does an SVM determine the optimal decision boundary for classification?
4. Compare Naive Bayes, Logistic Regression, and SVM for text classification.
5. Describe the optimization techniques used to improve Naïve Bayes performance for sentiment analysis. Explain techniques such as binary word counts, negation handling, and sentiment lexicons.

SGOU

Reference

1. Jurafsky, D. (2025) Speech and Language Processing/Daniel Jurafsky & James H. *New International ed. Harlow: Pearson Education, Harlow*.

2. https://nlp.stanford.edu/IR-book/pdf/15svm.pdf

3. Scikit-learn developers. (2024). Naive Bayes, Logistic Regression, Support Vector Machines. scikit-learn documentation. Retrieved from https://scikit-learn.org/stable/

4. Vajjala, S., Majumder, B., Gupta, A., & Surana, H. (2020). *Practical natural language processing: a comprehensive guide to building real-world NLP systems*. O'Reilly Media.

Suggested Reading

1. Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning*, 20(3), 273–297.

2. Manning, C. D., Raghavan, P., & Schütze, H. (2008). *Introduction to Information Retrieval.* Cambridge University Press. Chapter 15.

SGOU

Unit 4
Information Extraction (IE)

Learning Outcomes

After completing this unit, learner will be able to:

- explain the concept of Information Extraction (IE) and give examples of its main real-world applications
- describe the typical steps involved in an Information Extraction pipeline and explain the purpose of each step
- define Named Entity Recognition (NER) and identify common types of named entities, with suitable examples
- explain how to evaluate a Named Entity Recognition system using precision, recall, and F1-score
- demonstrate how to build a simple Named Entity Recognition system in Python using commonly used NLP libraries

Prerequisites

This unit builds upon the classification concepts introduced in Units 1 to 3. By this stage, you should understand how to assign labels to text and be familiar with the evaluation metrics precision, recall, and F1-score, which were introduced in Unit 2 and used again in Unit 3. In this unit, these same metrics are applied to named entities rather than entire documents.

You should also be familiar with the preprocessing techniques discussed in Block II, particularly tokenization and Part-of-Speech (POS) tagging, as these techniques can contribute to the information extraction process. Basic knowledge of Python will help you follow the code examples; however, you can understand the main concepts without executing the code.

This unit introduces a different perspective on text processing. While Units 1 to 3 primarily focused on classifying entire documents into predefined categories, this unit focuses on examining the content within documents to identify and extract specific pieces of information from sentences, such as who did what, to whom, where, and when.

SGOU

Keywords

IE, Structured Data, Unstructured Data, NER, Precision, Recall, F1-score

Discussion

3.4.1 Introduction to Information Extraction

If you pick up an Indian business newspaper on a weekday morning, you will find hundreds of fact-filled sentences. For example, a company may announce the appointment of a new chief executive, a state government may announce an investment, or a pharmaceutical company may receive approval for a new drug. Each sentence contains valuable information, but it is expressed in natural language, making it difficult for a database or computer system to search, organize, and process directly.

Consider the following sentence from a business newspaper, illustrated in Figure 3.4.1:

"Infosys announced that it will invest ₹2,000 crore in Bengaluru, Karnataka, creating 5,000 jobs by March 2026."

This single sentence contains several important pieces of information, including the organization, investment amount, location, number of jobs, and time period.

Information Extraction (IE) techniques can automatically identify such information in unstructured text and convert it into a structured representation that computer systems can store, search, and analyze. Illustrated in Fig 3.4.2

This ability to transform useful information from unstructured text into structured data is the central objective of Information Extraction.

How does it differ from classification?



Fig 3.4.1 Information Extraction – Converting Unstructured Text into Structured Information

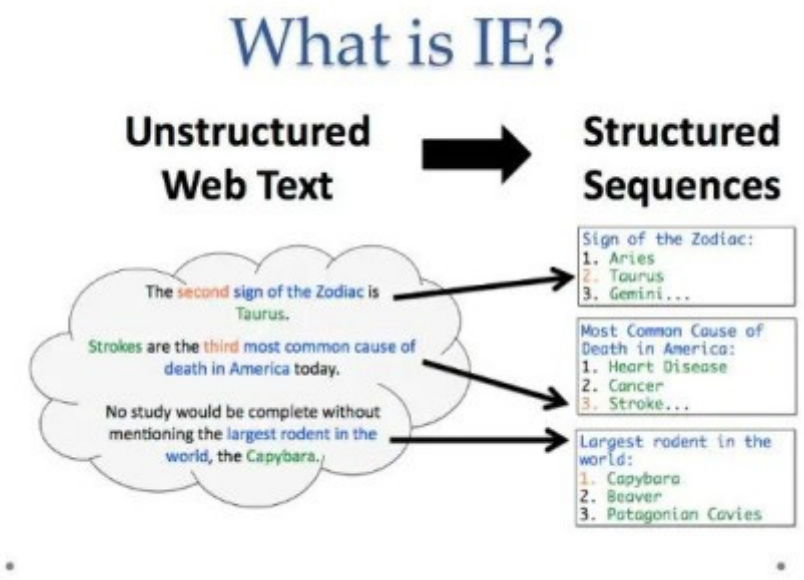


Fig 3.4.2 Information Extraction (IE) transforms unstructured web text into structured information sequences

Information Extraction differs fundamentally from the text classification tasks studied in the previous three units.

- Text classification assigns a single label to an entire document. For example, a review may be classified as *positive*, or an email may be classified as *spam*.
- Information Extraction (IE) examines the content within a document and identifies the individual facts and entities it contains. These facts are then represented as structured information rather than as a single category label.

The output of an IE system is therefore not simply a class label. Instead, it consists of a set of structured records or extracted facts that can be stored in a database, indexed, searched, and queried like other structured data.

For example, from the sentence:

"Infosys announced that it will invest ₹2,000 crore in Bengaluru, Karnataka, creating 5,000 jobs by March 2026."

an IE system might produce a structured record such as:

{Organization: Infosys, Investment: ₹2,000 crore, Location: Bengaluru, Karnataka, Jo bs: 5,000, Deadline: March 2026}

Thus, classification answers the question "Which category does this document belong to?", whereas information extraction answers questions such as "Who, what, where, when, and how much?"

Recall the distinction between structured and unstructured data discussed in the previous units. Figure 3.4.3 illustrates how Information Extraction transforms unstructured

SGOU

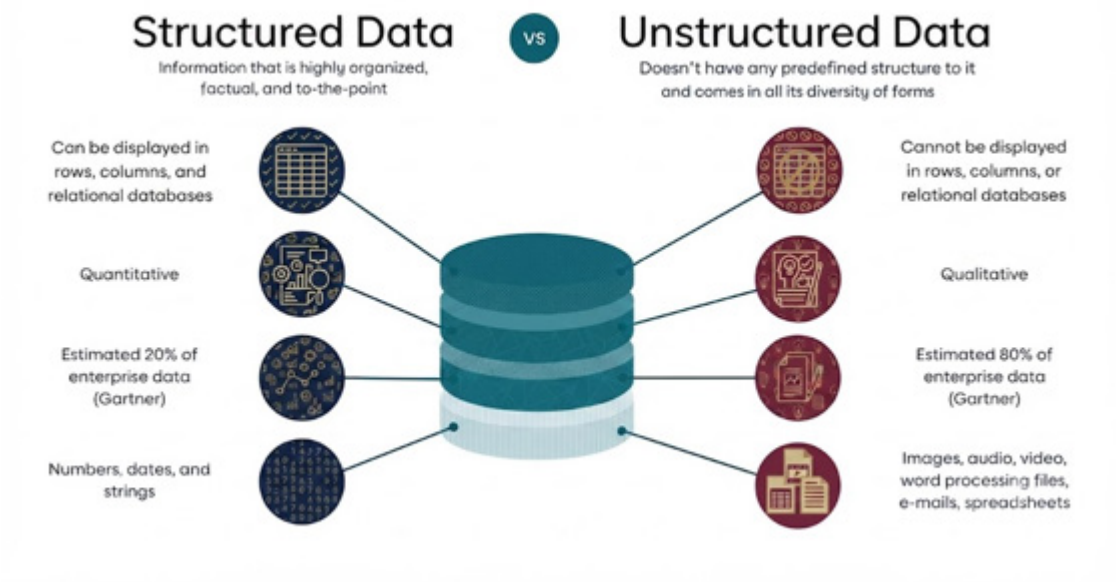


Fig 3.4.3 Structured Data vs. Unstructured Data

text into structured information by identifying and organizing relevant entities and relationships.

3.4.2 Applications of Information Extraction

Information Extraction (IE) is one of the important areas of Natural Language Processing (NLP) because a significant amount of the world's knowledge is contained in unstructured text.

IE techniques can automatically identify and extract useful information from such text and convert it into a structured form that can be stored, searched, and analyzed.

The following examples illustrate the wide range of applications of Information Extraction and the types of knowledge that can be automatically extracted from unstructured text.

3.4.2.1 Financial and Business Intelligence

Financial analysts often need to identify facts that are available only in news articles written in natural language. Information Extraction (IE) helps by converting information from such articles into structured records that can be searched, compared, and combined with historical financial data.

For example, consider the sentence:

"IndiGo said Friday it has increased fares by ₹400 per round trip, and SpiceJet immediately matched the move."

An Information Extraction system could extract relevant information such as:

{Lead airline: IndiGo, Amount: ₹400, Date: Friday, Follower: SpiceJet}

SGOU

This structured information can then be used for financial analysis, market monitoring, competitor analysis, and tracking business events.

3.4.2.2 Healthcare and Clinical NLP

What is considered a named entity depends on the specific task and domain. In biomedical and clinical texts, important entities may include genes, proteins, drugs, diseases, symptoms, medical procedures, and body parts, rather than only people or places.

Healthcare NLP systems often use specialized biomedical resources such as the Unified Medical Language System (UMLS). UMLS integrates information from multiple biomedical vocabularies and provides standardized concepts and relationships that help NLP systems recognize and interpret medical information. These resources can help identify relationships between medical entities, such as a drug being used to treat a disease.

For example, consider the clinical note:

"Metformin 500 mg prescribed for Type 2 Diabetes."

A clinical information extraction system can identify Metformin as a drug, Type 2 Diabetes as a disease, and 500 mg as the dosage. It can also extract the relationship that Metformin is prescribed to treat Type 2 Diabetes.

The extracted information can then be stored in a structured form for applications such as clinical decision support, medical record analysis, drug–disease relationship extraction, and biomedical research.

3.4.2.3 Legal Document Analysis

Legal documents are often lengthy, use formal and specialized language, and are available in large quantities. Information Extraction (IE) helps by identifying important information such as parties, dates, obligations, legal provisions, and citations from contracts and court judgments, and converting them into searchable and comparable structured records.

For example, from a court judgment, an Information Extraction system can extract details such as:

- Court: Supreme Court of India
- Date: 12 January 2024
- Petitioner: State of Kerala
- Section cited: Section 302 of the Indian Penal Code (IPC)

The extracted information can be used for legal research, case retrieval, document summarization, and automated analysis of legal documents.

SGOU

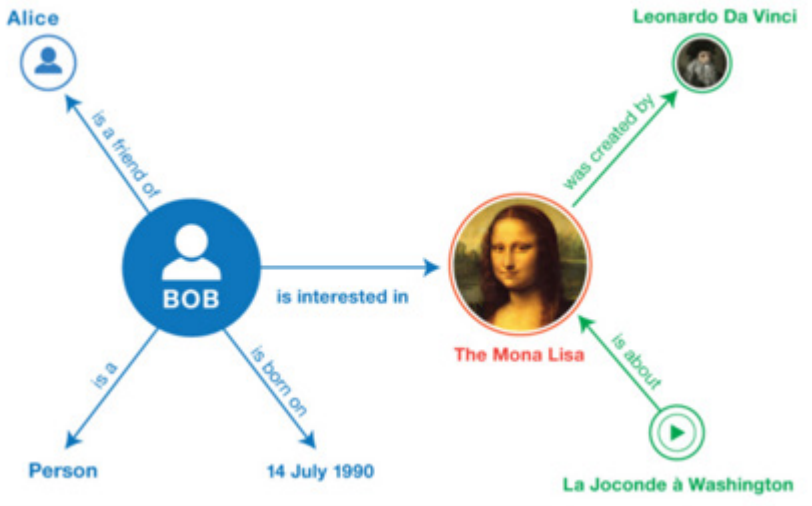


Fig 3.4.4 Example of a knowledge graph

3.4.2.4 Search Engines and Knowledge Graphs

A knowledge graph is a data structure that represents information as a network of entities and their relationships. It can be represented as a directed, labelled graph, where entities are represented as nodes and the relationships between them are represented as labelled edges (Fig 3.4.4).

Modern search engines use knowledge graphs to understand the meaning and relationships associated with a user's query and provide more accurate, relevant, and structured search results. A knowledge graph connects entities through semantic relationships and supports applications such as semantic search, question answering, and recommendation systems.

For example, the node Bob has several relationships: Bob is a friend of Alice, Bob is a Person, Bob was born on 14 July 1990, and Bob is interested in The Mona Lisa. Furthermore, The Mona Lisa is linked to Leonardo da Vinci through the relationship it was created by, and La Joconde à Washington is linked to The Mona Lisa through the relationship is about.

The example can be represented using triples such as:

- (Bob, is a friend of, Alice)
- (Bob, is a, Person)
- (Bob, was born on, 14 July 1990)
- (Bob, is interested in, The Mona Lisa)
- (The Mona Lisa, was created by, Leonardo da Vinci)
- (La Joconde à Washington, is about, The Mona Lisa)

SGOU

This illustrates how Information Extraction can identify entities and relationships from text and represent them in a structured knowledge graph.

3.4.2.5 Customer Experience and Sentiment Monitoring

In sentiment analysis, it is often necessary to determine a consumer's sentiment toward specific entities or product features, rather than simply determining whether a review is positive or negative overall. Extracting entities or features first enables sentiment to be associated with each one individually, providing businesses with a more detailed and actionable understanding of customer feedback.

For example, consider the following review:

Text: "The laptop boots up instantly, but the trackpad feels cheap and clicks too loudly."

The extracted information can be represented as:

Feature	*Sentiment*
Boot Speed	Positive
Trackpad Build	Negative

A standard document-level sentiment classifier might label the entire review as mixed sentiment, but this provides limited insight into the specific aspects of the product that customers like or dislike. By extracting each feature separately and associating it with its corresponding sentiment, the manufacturer can identify that boot speed is a strength, while the trackpad is a weakness.

This information can directly support product improvement, customer experience analysis, and decision-making.

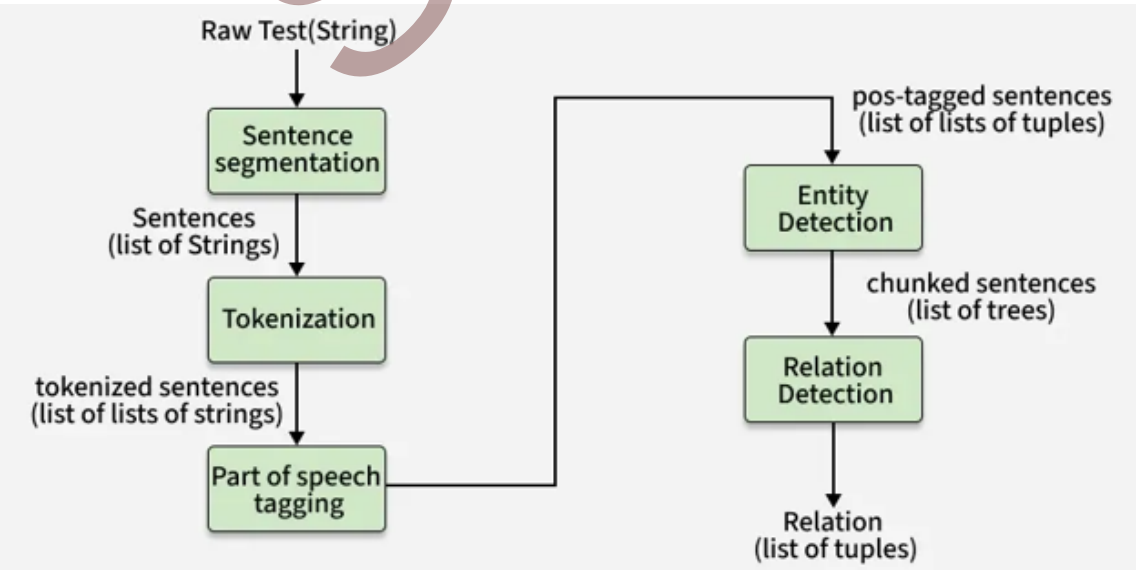


Fig 3.4.5 Simple Pipeline Architecture for an Information Extraction System

SGOU

3.4.3 The General Pipeline for Information Extraction

Figure 3.4.5 shows the architecture of a simple Information Extraction (IE) system. Information extraction does not occur in a single step. Raw text passes through several processing stages, with the output of each stage serving as input to the next. This staged design has an important consequence: errors at an earlier stage can propagate to subsequent stages. For example, if sentence segmentation incorrectly divides a sentence, the following stages may operate on incorrect or incomplete input.

To understand the pipeline concretely, we will trace the following sentence through all six stages:

"Infosys was founded by N. R. Narayana Murthy in Pune, Maharashtra in 1981."

3.4.3.1 Stage 1 : Sentence Segmentation

The first stage divides a document into individual sentences. Although this may seem as simple as splitting the text at every full stop, it is more challenging in practice. Full stops can occur in abbreviations such as *Dr.*, *Ltd.*, and *N. R.*, in decimal numbers such as *₹2.5 crore*, and in ellipses.

In the example sentence, the name "N. R. Narayana Murthy" contains full stops that do not indicate the end of a sentence. Therefore, a sentence segmentation system must distinguish between full stops that mark sentence boundaries and those that occur within abbreviations or other expressions.

Modern sentence segmentation systems may use rule-based methods, statistical models, or machine learning techniques to identify sentence boundaries. These systems consider contextual information, such as surrounding words, punctuation patterns, and known abbreviations, to determine whether a punctuation mark ends a sentence.

Output

The complete text is identified as one sentence and preserved as a single unit, despite containing full stops within the person's initials.

3.4.3.2 Stage 2: Tokenization

Each sentence is divided into individual tokens, which are the basic units used for further text processing. Tokens may include words, numbers, and punctuation marks. A tokenization system must handle different forms of text, such as contractions, hyphenated compounds, abbreviations, and special characters, consistently.

For the sentence:

"Infosys was founded by N. R. Narayana Murthy in Pune, Maharashtra in 1981."

the tokenization output may be:

Output:

['Infosys', 'was', 'founded', 'by', 'N.', 'R.', 'Narayana',

'Murthy', 'in', 'Pune', ',', 'Maharashtra', 'in', '1981', '.']

SGOU

Tokenization is an important stage in the IE pipeline because subsequent processes, such as Part-of-Speech tagging and Named Entity Recognition, typically operate on individual tokens.

3.4.3.3 Stage 3: Part-of-Speech Tagging

In this stage, each token is assigned a Part-of-Speech (POS) tag that represents its grammatical category, such as noun, verb, adjective, adverb, pronoun, or preposition. The POS tag assigned to a token depends on its context and grammatical role within the sentence.

For example, consider the sentence:

"Infosys was founded by N. R. Narayana Murthy in Pune, Maharashtra in 1981."

A POS tagger such as spaCy may assign tags to the tokens as follows:

Output sample:

TOKEN	POS TAG
Infosys	PROPN
was	AUX
founded	VERB
by	ADP
N.	PROPN
R.	PROPN
Narayana	PROPN
Murthy	PROPN
in	ADP
Pune	PROPN
, ,	PUNCT
Maharashtra	PROPN
in	ADP
1981	NUM
.	PUNCT

SGOU

Here:

- PROPN – Proper noun
- AUX – Auxiliary verb
- VERB – Verb
- ADP – Adposition (preposition)
- PUNCT – Punctuation
- NUM – Numeral

POS tagging provides useful grammatical information that can assist subsequent IE tasks, particularly Named Entity Recognition and relationship extraction.

3.4.3.4 Stage 4: Named Entity Recognition

The fourth stage identifies tokens or spans of tokens that represent named entities and assigns an appropriate entity type to each one. Common entity types include PERSON, ORGANIZATION (ORG), LOCATION (LOC), and DATE.

This is a central component of Information Extraction and is discussed in detail in Section 3.4.4.

For the example sentence:

"Infosys was founded by N. R. Narayana Murthy in Pune, Maharashtra in 1981."

Output sample:

- Infosys → ORG

Table 3.4.1 Common Named Entity Types and Examples

Type	Code	Examples
Person	PER	[Not specified in source]
Organisation	ORG	Infosys, ISRO, Reserve Bank of India
Location	LOC	Western Ghats, Deccan Plateau
Geopolitical Entity (GPE)	GPE	India, Kerala, Bengaluru
Date	DATE	1981, March 2026, last Diwali
Time	TIME	9:30 am, midnight
Money	MONEY	₹2,000 crore, $5 million
Percent	PERCENT	26 percent, 7.5%
Miscellaneous (MISC)	MISC	Padma Bhushan, Kumbh Mela

SGOU

- N. R. Narayana Murthy → PERSON
- Pune → LOC
- Maharashtra → LOC
- 1981 → DATE

3.4.3.5 Stage 5: Relation Extraction

Relation Extraction identifies the semantic relationships between entities identified in the previous stage. Knowing that Infosys is an organization and N. R. Narayana Murthy is a person is useful, but the more valuable information is the relationship that connects these entities.

For example, in the sentence:

"Infosys was founded by N. R. Narayana Murthy in Pune, Maharashtra in 1981."

The phrase "was founded by" indicates a FOUNDED_BY relationship between the two entities.

Output sample:

(Infosys, FOUNDED_

BY, N. R. Narayana Murthy)

Output sample: The phrase "was founded by" signals a FOUNDED_BY relationship.

Relation extraction can be performed using hand-written rules or patterns, supervised machine learning methods, or neural network-based models.

3.4.3.6 Stage 6: Output

The final stage converts the extracted entities and relationships into a machine-readable structured format, such as JSON, RDF triples, or database records. This structured representation makes the extracted information easier to store, search, query, integrate, and analyze.

For example, the information extracted from the sentence:

"Infosys was founded by N. R. Narayana Murthy in Pune, Maharashtra in 1981."

can be represented as:

Output Example:

```
{
 "organization": "Infosys",
 "founder": "N. R. Narayana Murthy",
 "location": "Pune, Maharashtra",
```

SGOU

"founded_year": 1981

}

3.4.4 Named Entity Recognition (NER)

Among the stages in the Information Extraction pipeline, Named Entity Recognition (NER) is one of the most extensively studied and widely implemented tasks. Its output directly supports downstream applications such as relation extraction, question answering, information retrieval, and knowledge graph construction. Errors in NER can propagate to subsequent stages and affect the accuracy of downstream information extraction tasks.

A named entity is a word or group of words that refers to a specific real-world entity, concept, or value belonging to a predefined category, such as a person, organization, location, date, time, monetary value, or percentage (Table 3.4.1).

Named Entity Recognition (NER) is the process of automatically identifying named entities in text and assigning each entity to an appropriate predefined category.

For example, consider the sentence:

"Infosys was founded by N. R. Narayana Murthy in Pune, Maharashtra in 1981."

An NER system may identify:

- Infosys → ORG
- N. R. Narayana Murthy → PER
- Pune → LOC
- Maharashtra → GPE
- 1981 → DATE

The NER task can be divided into two main subtasks:

- Identifying the boundaries of an entity – determining which words or tokens form the complete entity.
- Identifying the type of the entity – assigning the appropriate category, such as PERSON, ORGANIZATION, LOCATION, or DATE.

Both subtasks must be performed correctly for the extracted information to be useful. For example, in the sentence *"Infosys was founded by N. R. Narayana Murthy,"* an NER system must correctly identify "N. R. Narayana Murthy" as the complete entity and classify it as PERSON.

A worked example of NER extraction and entity-relation diagram construction is provided in Section 3.4.4.1.

SGOU

The following code demonstrates how to perform NER using spaCy, a popular Python library for Natural Language Processing. Try running the code to explore the capabilities of spaCy and understand how named entities are identified and classified.

3.4.4.1 Worked Example: Code implementation

```
import spacy
from spacy.tokens import Doc
from spacy.matcher import Matcher
from spacy import displacy


nlp = spacy.load("en_core_web_sm")

def information_extraction(doc):
    matcher = Matcher(nlp.vocab)

    svo_pattern = [
        {"DEP": "nsubj"},
        {"DEP": "aux", "OP": "?"},
        {"POS": "VERB"},
        {"DEP": "det", "OP": "?"},
        {"DEP": {"IN": ["dobj", "attr", "pobj"]}}
    ]

    matcher.add("SVO_PATTERN", [svo_pattern])
    matches = matcher(doc)

    relations = []
    for _, start, end in matches:
        span = doc[start:end]
        subj = span[0].text
        verb = span[2].lemma_
        obj = span[-1].text
        relations.append((subj, verb, obj))

    return relations
```

```
Doc.set_extension("relations", getter=information_extraction, force=True)

text = "Infosys was founded by N. R. Narayana Murthy in Pune, Maharashtra in 1981."
doc = nlp(text)
print("Named Entities:")
for ent in doc.ents:
    print(f"{ent.text} --> {ent.label_}")
```

```
print("\nExtracted Relations (SVO):")
for rel in doc._.relations:
    print(rel)
```

OUTPUT

Named Entities:

Infosys --> ORG

SGOU

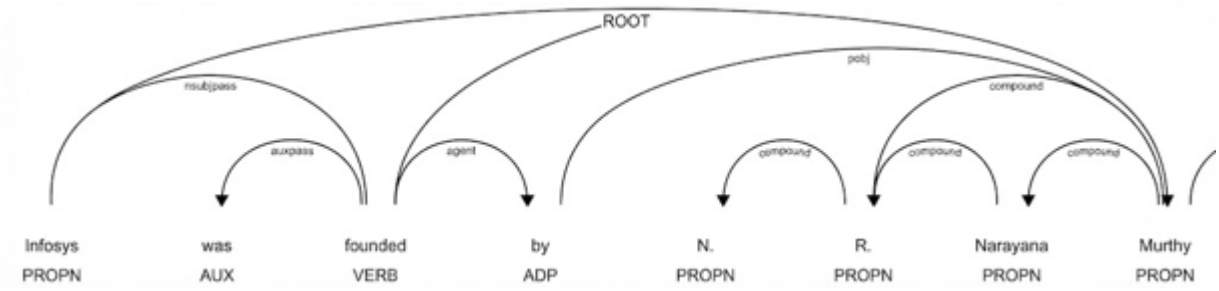


Fig 3.4.6 Dependency Parse and Named Entity Recognition Output for the Sample Sentence

N. R. Narayana Murthy --> PERSON

Pune --> GPE

Maharashtra --> ORG

1981 --> DATE

Extracted Relations (Sample Output)

The NER system identifies the following entities in the sentence:

Infosys → ORG

N. R. Narayana Murthy → PERSON

Pune → GPE

Maharashtra → GPE

1981 → DATE

Note: The label Maharashtra → ORG shown in your text is incorrect. Maharashtra is a GPE (Geopolitical Entity), not an organization.

SGOU

3.4.5 Ambiguity in Named Entity Recognition

If NER were simply a matter of looking up names in a dictionary, the problem would have been solved decades ago. The main challenge is that the same word or sequence of words can refer to different entities or entity types depending on the surrounding context. This section examines some of the major sources of ambiguity in Named Entity Recognition.

Ambiguity occurs when a word or phrase has multiple possible interpretations depending on the context in which it appears. NER systems may find it difficult to assign the correct entity type when contextual information is insufficient or when the same expression can refer to different types of entities.

Ambiguity in NER can be broadly categorized into two types:

- **Lexical ambiguity:** Occurs when the same word or phrase can have different

meanings or refer to different entity types. For example, "Amazon" can refer to a company (Amazon announced a new service) or a geographical region (The Amazon is one of the world's largest rainforest regions).

- **Contextual ambiguity:** Occurs when the correct interpretation of an entity depends strongly on the surrounding words or sentence context. For example, "Washington" can refer to a person (Washington led the army) or a location (Washington is the capital of the United States).

Therefore, effective NER systems must use contextual information, rather than relying only on dictionaries or lists of known entity names, to determine the correct entity boundaries and types.

3.4.5.1 Type / Class Ambiguity

Type ambiguity occurs when the same name or sequence of words can refer to different types of entities depending on the context in which it appears. It is one of the common sources of errors in Named Entity Recognition (NER).

Consider the name "Gandhi" in the following sentences:

- "Gandhi led the Salt March in 1930." → PERSON *(Mahatma Gandhi)*
- "Gandhinagar is the capital of Gujarat." → GPE *(the capital city of Gujarat)*

3.4.5.2 Boundary Ambiguity

Even when the type of an entity is clear, determining the exact beginning and end of the entity span can be challenging. This is known as boundary ambiguity and is particularly common with long organization names, person names, and entities that appear next to titles or other descriptive expressions.

Consider the sentence:

"State Bank of India Chairman Dinesh Khara addressed the meeting."

The correct entity spans should be:

- [State Bank of India] → ORG
- [Dinesh Khara] → PERSON

3.4.5.3 Domain - Specific Entities

NER models trained on general-purpose datasets, such as CoNLL-2003, may perform poorly when applied to specialized domains such as medicine, law, or finance. This is because specialized domains contain entity types, terminology, and expressions that may not be represented in the training data of general-purpose NER models.

For example, consider the following clinical sentence:

"The patient was prescribed metformin for controlling Type 2 diabetes."

A domain-specific clinical NER system may identify:

SGOU

- metformin → MEDICATION
- Type 2 diabetes → DIAGNOSIS

NER models may face difficulties when processing languages with complex grammatical structures, rich morphology, limited capitalization conventions, or substantial variation in word forms.

Example 1: Capitalization Variations

Capitalization can provide useful clues for identifying named entities in languages such as English. However, capitalization information may be lost in lowercase, noisy, or informally written text.

Sentence:

"steve jobs was the co-founder of apple inc."

Challenge: A model that relies heavily on capitalization may have difficulty identifying "steve jobs" as a PERSON and "apple inc." as an ORGANIZATION (ORG).

Example 2: Morphological Variations

In languages with rich morphology, the form of a word may change depending on its grammatical role, such as case, number, gender, or syntactic function. As a result, the same person's or location's name may appear in different forms in different sentences.

For example, languages such as Finnish, German, Russian, and many Indian languages exhibit various types of morphological variation. NER models trained primarily on English may struggle to recognize these variations because English has relatively limited morphological variation compared with such languages.

Therefore, NER systems designed for multilingual or morphologically rich languages often require language-specific resources, multilingual training data, or appropriate morphological and contextual features.

SGOU

3.4.6 NER as Sequence Labeling

Named Entity Recognition (NER) is typically formulated as a sequence labelling problem in which each token in a sentence is assigned a label indicating whether it belongs to a named entity and, if so, identifying the type of entity. Rather than classifying the entire sentence as a single unit, the model assigns a label to each token while preserving the original order of the tokens.

This approach enables the NER system to identify both entity boundaries and entity types.

The BIO (Beginning, Inside, Outside) tagging scheme, illustrated in Figure 3.4.7, is commonly used for sequence labelling in NER.

- B (Beginning): Indicates the first token of a named entity.
- I (Inside): Indicates a token that continues an already identified named entity.

- O (Outside): Indicates a token that does not belong to any named entity.

The entity type is usually appended to the B and I tags. For example:

- B-PER → Beginning of a Person entity
- I-PER → Inside/continuation of a Person entity
- B-ORG → Beginning of an Organization entity
- B-LOC → Beginning of a Location entity

For the sentence shown in Figure 3.4.7:

"Sundar Pichai became CEO of Google in Chennai in 2015."

The tokens "Sundar Pichai" are labelled B-PER, I-PER, indicating one person entity. "Google" is labelled B-ORG, indicating an organization, while "Chennai" is labelled B-LOC, indicating a location.

3.4.7 Evaluation of Named Entity Recognition

After developing an NER system, the next question is how accurately it identifies named entities. Evaluation requires a gold-standard version of the test text in which human annotators have correctly marked the entities and their corresponding types. The predictions produced by the NER system are then compared with this gold standard, and the correct and incorrect predictions are counted.

An important point distinguishes NER evaluation from text classification evaluation. In text classification, each document receives a single label, making the comparison relatively straightforward. In NER, an entity prediction can be incorrect in two main

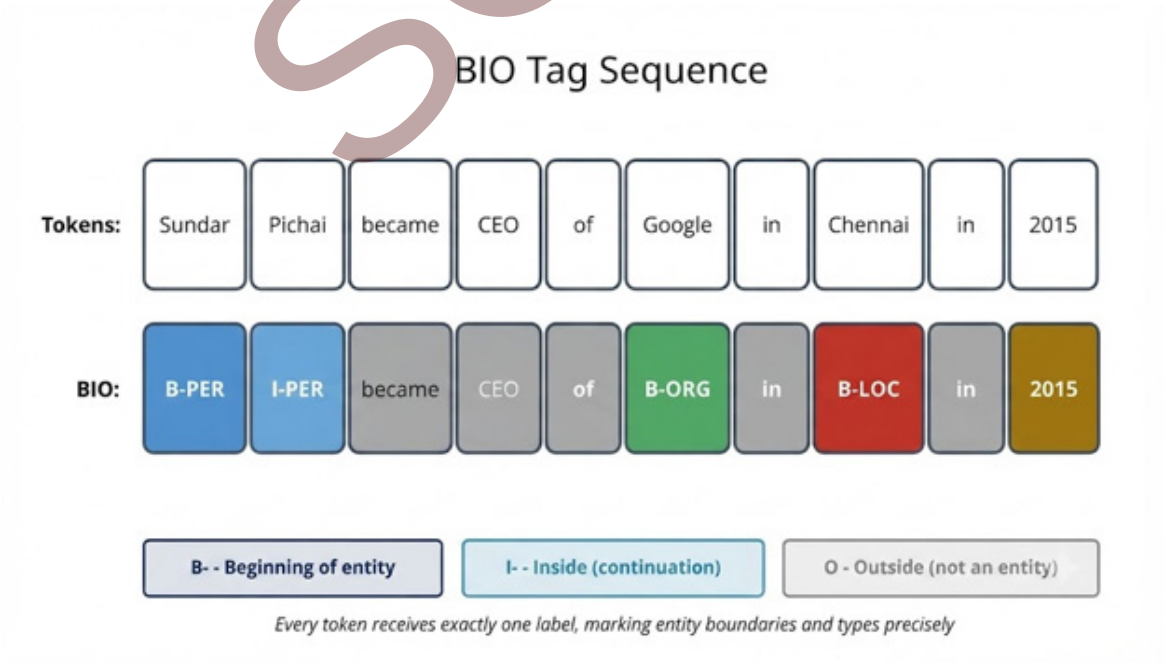


Fig 3.4.7 Example of BIO Tag Sequence for Named Entity Recognition

SGOU

ways: the entity type may be wrong, or the entity boundary may be incorrect. Both types of errors affect the evaluation.

For example, suppose the gold-standard annotation is:

[State Bank of India] → ORG

If the system predicts:

[State Bank of India] → PERSON

the boundary is correct, but the entity type is incorrect.

If the system predicts:

[State Bank of India Chairman] → ORG

The entity type is correct, but the boundary is incorrect.

In strict entity-level evaluation, a prediction is considered correct only when both the entity span and the entity type exactly match the gold-standard annotation.

Evaluation Metrics

1. Precision

Precision measures the proportion of entities predicted by the NER system that are actually correct.

$$\text{Precision} = \frac{True\ Positives}{True\ Positives\ + False\ Positives}$$

2. Recall

Recall measures the proportion of the actual entities in the gold-standard data that are correctly identified by the NER system.

High recall indicates that the model identifies most relevant entities and produces fewer false-negative errors. High precision indicates that the system produces fewer false-positive entity predictions.

$$\text{Recall} = \frac{True\ Positives}{True\ Positives\ + False\ Negatives}$$

3. F1-Score

The F1-score is the harmonic mean of Precision and Recall, providing a single score that balances both measures. It is particularly useful when both correctly identifying entities and finding as many actual entities as possible are important. A high F1-score indicates a good balance between precision and recall, while a low F1-score indicates that the system may have problems with one or both measures.

$$\text{F1- Score} = 2 \times \frac{\text{Precision x Recall}}{\text{Precision} + \text{Recall}}$$

SGOU

The Evaluation outcomes

Example Test sentence

"Apple Inc. is planning to open a new office in San Francisco in March 2025"

Gold standard

The human-annotated gold standard contains:

[Apple Inc. / ORGANIZATION], [San Francisco/ LOCATION], [March 2025 / DATE]

System output

Entity	System Output (Gold)	Outcome
Apple Inc.	ORGANIZATION	✓ True Positive
San Francisco	LOCATION	✓ True Positive
office	LOCATION	✗ False Positive
March 2025	Not detected	✗ False Negative

Here, Apple Inc. and San Francisco are correctly identified with the correct entity types, so they are counted as True Positives (TP = 2). The incorrectly identified entity office is a False Positive (FP = 1). The gold-standard entity March 2025, which the system failed to identify, is a False Negative (FN = 1).

Evaluation Metric

1. Precision = TP/(TP + FP) = 2 / (2 + 1) = 0.67

2. Recall = TP / (TP + FN) = 2 / (2 + 1) = 0.67

F1-Score = 2 x (Precision x Recall / Precision + Recall)

= 2 x (0.67 x 0.67 / 0.67 + 0.67)

= 0.67

Therefore,

$Recall = 67\%$

3. F1-Score

$$F1 - Score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

$$= 2 \times \frac{0.67 \times 0.67}{0.67 + 0.67} \approx 0.67$$

Therefore,

F1-Score ≈ 67%

SGOU

Recap

- Information Extraction (IE) automatically extracts useful, structured information from unstructured text.
- Unlike text classification, which assigns a label to an entire document, IE identifies entities, attributes, and relationships within a document.
- The output of an IE system is structured information that can be stored, searched, queried, and analysed.
- Information Extraction is widely used in finance, healthcare, legal analysis, search engines, knowledge graphs, and customer feedback analysis.
- Knowledge graphs represent information as entities and their relationships and support applications such as semantic search and question answering.
- A typical IE pipeline consists of sentence segmentation, tokenization, Part-of-Speech tagging, Named Entity Recognition, relation extraction, and structured output generation.
- Sentence segmentation identifies sentence boundaries within a document.
- Tokenization divides text into individual tokens such as words, numbers, and punctuation marks.
- Part-of-Speech (POS) tagging assigns grammatical categories to tokens based on their context and grammatical role.
- Named Entity Recognition (NER) identifies entities and classifies them into predefined categories such as Person, Organization, Location, and Date.
- Relation extraction identifies semantic relationships between the entities identified by the NER system.
- Structured output generation converts the extracted entities and relationships into machine-readable formats such as JSON, RDF triples, or database records.
- Ambiguity in NER includes type/class ambiguity, boundary ambiguity, domain-specific entities, and language and morphological variations.
- Sequence labelling enables NER systems to identify both entity boundaries and entity types, commonly using the BIO tagging scheme.
- Precision, Recall, and F1-score are commonly used to evaluate the performance of NER systems.
- Accurate Information Extraction improves downstream NLP applications such as question answering, knowledge graph construction, information retrieval, and automated document analysis.

SGOU

Objective Type Questions

1. Which NLP task extracts structured facts from unstructured text?
2. Which technique identifies names of people, organizations, and locations?
3. Which stage of the IE pipeline divides a document into individual sentences?
4. Which process splits text into individual words or tokens?
5. Which graph represents entities and their relationships?
6. Which evaluation metric measures the proportion of predicted entities that are correct?
7. Which evaluation metric measures the proportion of actual entities that are correctly identified?
8. Which metric combines Precision and Recall into a single value?
9. Which format is commonly used to store extracted information in a machine-readable form?
10. Which labelling approach assigns one label to each token in a sentence?
11. Which type of data contains free-form content such as emails and news articles?
12. Which named entity type is assigned to “1981” in NER?
13. Which NLP task identifies semantic relationships between extracted entities?
14. Which stage of the IE pipeline identifies named entities in a sentence?
15. Which stage of the IE pipeline converts extracted information into a machine-readable format?

SGOU

Answers to Objective Type Questions

1. Information Extraction
2. NER
3. Sentence Segmentation
4. Tokenization

5. Knowledge Graph
6. Precision
7. Recall
8. F1-Score
9. JSON
10. Sequence Labeling
11. Unstructured data
12. Date
13. Relation Extraction
14. Named Entity Recognition (NER)
15. Structured Output Generation / Structuring and Storing

Assignments

1. Explain the Information Extraction pipeline with the help of a suitable example.
2. Describe the role of Named Entity Recognition (NER) in Information Extraction. Illustrate your answer with an example.
3. Discuss the different types of ambiguity in Named Entity Recognition and explain how they affect the performance of NER systems.
4. Explain the concept of sequence labeling in Named Entity Recognition. Why is sequence labelling important for identifying entity boundaries and entity types?
5. Describe the evaluation metrics used for Named Entity Recognition. Calculate the Precision, Recall, and F1-score for a given example and interpret the results.

SGOU

Reference

1. S. Vajjala, B. Majumder, A. Gupta, and H. Surana, *Practical Natural Language Processing: A Comprehensive Guide to Building Real-World NLP Systems*. O'Reilly Media, 2020.

2. D. Jurafsky and J. H. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*.

3. S. Singh, "Natural Language Processing for Information Extraction," *arXiv preprint arXiv:1807.02383*, 2018.

Suggested Reading

1. S. Bird, E. Klein, and E. Loper, *Natural Language Processing with Python*. O'Reilly Media, 2009.

SGOU



Data Integration, Data Quality Assessment

SGOU

Unit 1

Relation Detection and Classification

Learning Outcomes

After completing this unit, learner will able to:

- explain what a relation is in the context of Information Extraction and represent relations as triples
- describe the standard types of relations used in relation extraction and the ontologies that define them
- describe the supervised learning approach to relation extraction, including the training process and classification algorithm
- design appropriate features for a supervised relation classifier, including word-based, entity-based, and syntactic features

Prerequisites

In many real-world applications, large amounts of information are available as unstructured text, such as news articles, business reports, medical records, and online documents. For example, in the sentence *"Narayana Murthy founded Infosys in Pune in 1981,"* a Named Entity Recognition (NER) system can identify *Narayana Murthy* as a person, *Infosys* as an organisation, *Pune* as a location, and *1981* as a date. However, merely identifying these entities does not explain how they are connected. Relation Extraction is therefore important because it identifies meaningful relationships between entities and represents the information in a structured form. Such structured relationships can be used in knowledge graphs, question-answering systems, information retrieval, semantic search, and biomedical text mining.

Studying this unit helps learners understand how to automatically extract meaningful facts and relationships from natural language text. Learners will benefit from understanding relation detection, relation classification, supervised learning approaches, text annotation, feature design, and neural-based relation extraction. The unit also helps learners understand how features such as lexical, entity, dependency, and syntactic information contribute to relation identification, while neural approaches can learn useful representations directly from text. These concepts provide learners with a foundation for developing and understanding NLP systems that transform unstructured textual information into structured knowledge, supporting applications such as knowledge graph construction, question answering, information retrieval, and domain-specific text analysis.



SGOU

Keywords

Relation, Relation Extraction, Relation Detection, Relation Classification, Triple, RDF, Entity Pair, Ontology, ACE, UMLS, Wikipedia Infobox, Feature Vector

Discussion

4.1.1 Introduction to Entity & Relation

In Block III Unit 4 you learned how a named **entity** recogniser reads a sentence and identifies the people, organisations, locations, and dates it mentions. That is genuinely useful, but on its own it produces only a list of names. It tells us that certain entities appear in a document; it does not tell us how they are connected.

In Block III, Unit 4, you learned how a Named Entity Recognition (NER) system reads a sentence and identifies the people, organizations, locations, and dates it contains. This is useful, but NER alone produces a list of identified entities primarily. It tells us which entities appear in a document, but it does not tell us how those entities are related to one another.

Consider the sentence:

"Narayana Murthy founded Infosys in Pune in 1981."

A Named Entity Recognition system may identify four entities:

- Narayana Murthy → PERSON
- Infosys → ORG
- Pune → GPE
- 1981 → DATE

However, this information is incomplete in an important way. It does not indicate that Narayana Murthy founded Infosys, rather than working for it. Similarly, it does not indicate that Pune is the location associated with the founding of Infosys, rather than the place where Narayana Murthy was born.

The entities have been correctly identified, but the relationships between them and the facts expressed by those relationships are missing.

Extracting these facts and the connections between entities is the task of Relation Extraction (RE). Relation Extraction builds upon the output of NER by identifying the semantic relationship between pairs of entities.

For example, from the sentence above, a relation extraction system may produce:

$$(Infosys)$$

SGOU

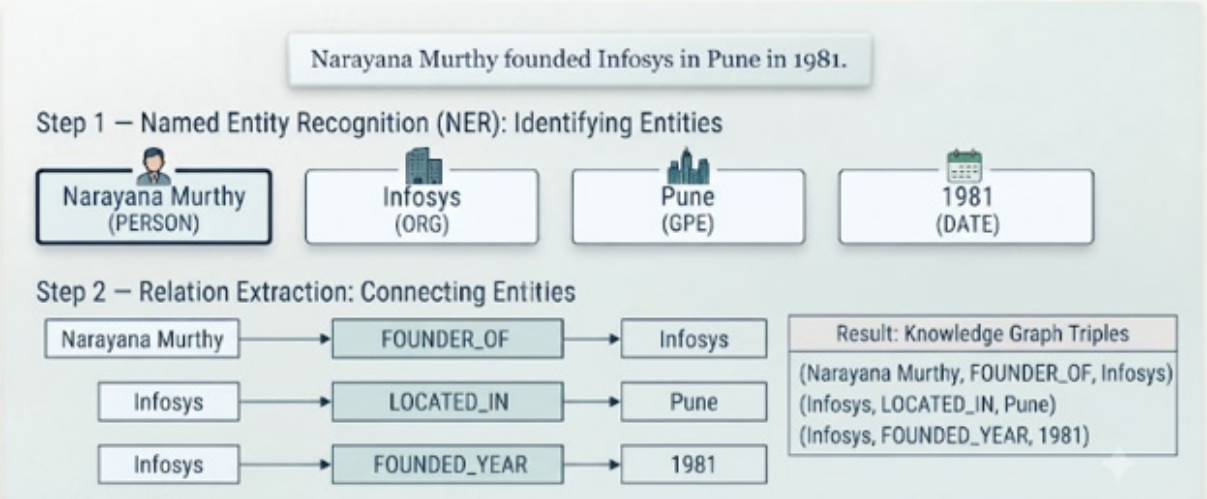


Fig 4.1.1 From Named Entities to Relations

and:

$(Pune)$

The extracted relations can then be represented as structured triples and used in applications such as knowledge graph construction, question answering, and information retrieval.

Figure 4.1.1 illustrates how Relation Extraction builds directly upon the output of Named Entity Recognition.

Relation Extraction takes the entities identified by NER and determines the semantic relationships between them, producing structured triples.

This unit examines Relation Extraction in detail, with particular emphasis on supervised learning approaches, which this unit focuses on. The next unit in this block will cover approaches that require less labelled training data.

4.1.2 Relation

A relation describes how two or more entities are connected. In Information Extraction, a relation represents a semantic connection between two entities mentioned in a text.

Relations are conventionally represented as an ordered triple of the form:

$(entity_2)$

Example

From the sentence,

"Narayana Murthy founded Infosys in Pune in 1981."

the following relations can be extracted:

SGOU

$$(Infosys)(Pune)(1981)$$

These relations are illustrated in Figure 4.1.1 (Step 2).

Note : A relation triple is often described as a **Subject–Predicate–Object (S–P–O)** expression:

- **Subject:** the first entity
- **Predicate:** the relation connecting the entities
- **Object:** the second entity

Why Order Matters

Relations are generally directional, meaning that the order of the entities in a relation is important. Reversing the two entities can change the meaning of the relation or produce an incorrect statement.

For example, the triple:

$$(Infosys)$$

is correct, whereas:

$$(Narayana\ Murthy)$$

is incorrect because Infosys did not found Narayana Murthy.

- A Relation Extraction system must determine not only whether two entities are related, but also which entity plays which role in the relation. This is known as argument ordering.
- For example, in the sentence:

  "Wipro acquired Capco in 2021."

the correct relation is:

$$(Capco)$$

Here, Wipro is the entity acting, while Capco is the entity affected by the action.

Reversing the arguments would produce an incorrect statement:

$$(Wipro)$$

Importantly, the order of words in a sentence does not always determine the order of arguments in the relation. Consider the passive-voice sentence:

"Capco was acquired by Wipro in 2021."

Although Capco appears before Wipro in the sentence, the correct relation remains:

SGOU

$(Capco)$

Thus, a Relation Extraction system must understand the semantic roles of the entities, rather than simply following their order of appearance in the sentence.

4.1.3 Standard Relation Types

The types of relations extracted by an Information Extraction system depend on the purpose and domain of the application.

For example, a system designed to process business news may need to identify relations such as *ACQUIRED, FOUNDED_BY,* or *WORKS_FOR*, whereas a system designed for clinical records may focus on relations such as *TREATS, CAUSES,* or *DIAGNOSED_WITH*.

To support consistent representation of information, experts have developed standard sets of relations, often organized into ontologies, for different domains. An ontology defines the concepts, entity types, and relationships that are relevant to a particular domain.

4.1.3.1 ACE Relation Types

The ACE (Automatic Content Extraction) evaluations defined a standard set of 7 broad relation types with 17 relation subtypes for general-purpose information extraction. These relation types describe common semantic relationships between entities.

4.1.3.2 Domain-Specific Relations - UMLS

The Unified Medical Language System (UMLS), developed by the U.S. National Library of Medicine (NLM), provides a framework for representing biomedical and clinical concepts and their relationships. The UMLS Semantic Network defines 134 semantic types and 54 semantic relations that describe relationships between biomedical concepts.

These domain-specific relations are useful for extracting meaningful information from medical and clinical texts, where relationships between diseases, drugs, anatomical structures, procedures, and physiological functions are particularly important.

Table 4.1.1: Examples of ACE Relation Types

Relation	Entity Types	Example
Physical-Located	PER–GPE	He was in Tennessee
Part-Whole-Subsidiary	ORG–ORG	XYZ, the parent company of ABC
Person-Social-Family	PER–PER	Yoko's husband John
Org-AFF-Founder	PER–ORG	Steve Jobs, co-founder of Apple

SGOU

Table 4.1.2: Examples of UMLS Semantic Relations

Entity	Relation	Entity
Injury	disrupts	Physiological Function
Bodily Location	location-of	Biologic Function
Anatomical Structure	part-of	Organism
Pharmacologic Substance	causes	Pathological Function
Pharmacologic Substance	treats	Pathologic Function

Example

Consider the sentence:

"Doppler echocardiography can be used to diagnose left anterior descending artery stenosis in patients with type 2 diabetes."

A relation extraction system can identify a relationship between the medical procedure and the disease:

Doppler Echocardiography — DIAGNOSES → Left Anterior Descending Artery Stenosis

4.1.3.3 Relations from Wikipedia Infoboxes

Wikipedia provides a large source of structured relations through its infoboxes, which are structured tables associated with many Wikipedia articles.

For example, the infobox of a university may contain structured facts such as:

- State: California
- President: Mark Tessier-Lavigne

These facts can be represented as relations such as LOCATED_IN and PRESIDENT_OF.

Such structured facts can also be represented using RDF (Resource Description Framework) triples. An RDF triple consists of three components:

$$(Object)$$

The subject represents the entity being described, the predicate represents the relationship, and the object represents the entity or value associated with that relationship.

Example

SGOU

Subject	Predicate	Object
Golden Gate Park	Location	San Francisco

This can be interpreted as:

Golden Gate Park is located in San Francisco.

Thus, Wikipedia infoboxes provide a useful source of structured facts that can be represented as RDF triples and used in knowledge graphs, semantic search, and information retrieval systems.

4.1.3.4 Ontological Relations - WordNet

WordNet is a lexical database that organizes English words and concepts into groups of synonyms called synsets and provides several semantic relationships between them. One of its important relationships is the is-a (hypernym–hyponym) relationship, which represents a hierarchical relationship between a specific concept and a more general concept.

For example:

Giraffe → is-a → ruminant → is-a → ungulate → is-a → mammal → is-a → vertebrate

Here, giraffe is a more specific concept, while vertebrate is a more general concept. Each is-a relationship connects a concept to a broader class.

WordNet also represents instance-of relationships, which connect a specific individual or instance to a general class.

4.1.4 Relation Detection and Relation Classification

Relation extraction is a key component of modern Natural Language Processing (NLP) applications. It supports information extraction, knowledge graph construction, question answering, biomedical text mining, and semantic search by converting information from unstructured text into structured knowledge.

The task of identifying relations in text is commonly divided into two stages:

- Relation Detection – determines whether a relation exists between a candidate pair of entities.
- Relation Classification – determines the specific type of relation between the entities when a relation exists.

4.1.4.1 Step 1 - Relation Detection

Relation detection is a binary classification task. For each candidate pair of entities, the system determines whether a meaningful semantic relation exists between them.

SGOU

The output is:

- **YES** – a relation exists between the entity pair.
- **NO** – no relevant relation exists between the entity pair.

4.1.4.2 Step 2 - Relation Classification

Relation classification is a multiclass classification task applied to the entity pairs for which a relation has been detected. Given a related entity pair, the system assigns the specific relation type from a predefined set of relations.

4.1.5 Supervised Learning for Relation Extraction & Classification

The supervised approach reframes relation extraction as the classification problem you already know from Block III. Instead of writing rules by hand, we collect a **corpus** in which relations have been annotated by human experts, extract features from each annotated entity pair, and train a classifier to predict the relation label. Figure 4.1.2 illustrates the end-to-end architecture of a supervised learning model used for relation extraction and classification**,** breaking down both the training and prediction phases.

4.1.5.1 Text Annotation Pipeline

Before training a supervised relation extraction model, a labelled dataset must be prepared. This process is known as text annotation. During annotation, raw text is progressively enriched with linguistic and semantic information, such as sentence

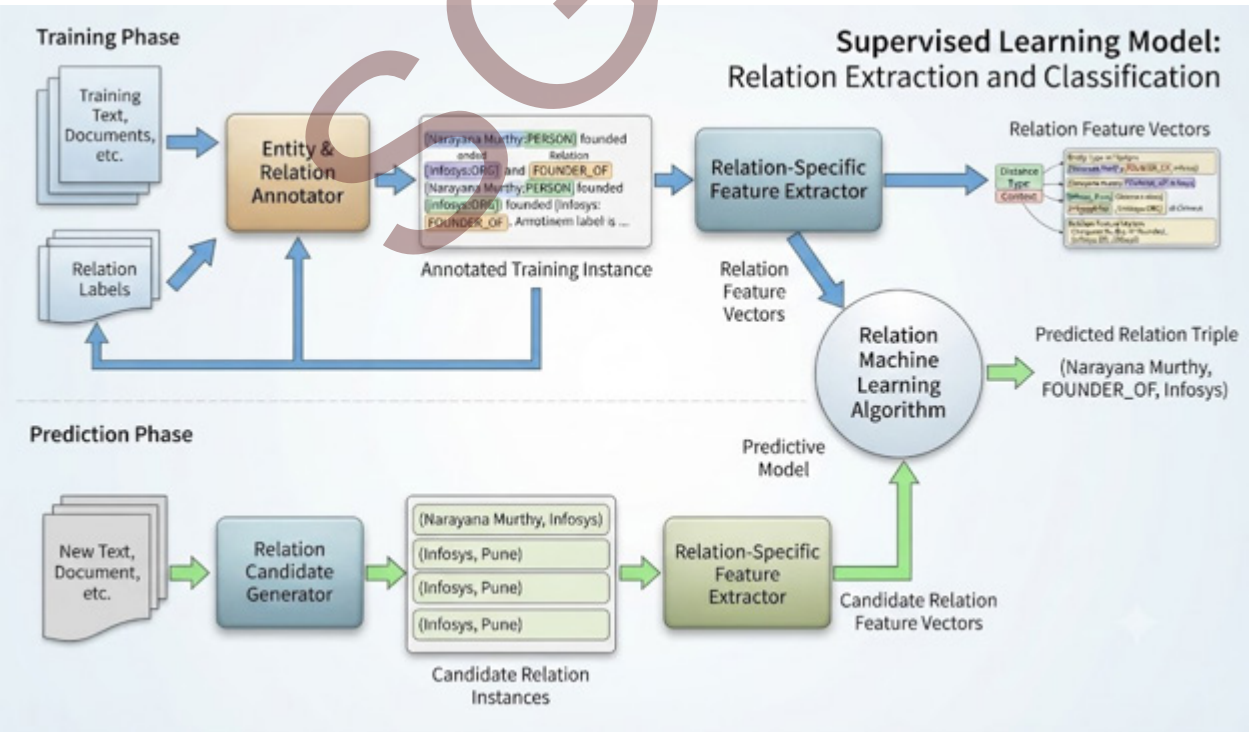


Fig 4.1.2: Supervised Learning Model for Relation Extraction and Classification

SGOU

boundaries, tokens, Part-of-Speech (POS) tags, named entities, and relation labels. Feature extraction, which is required for supervised learning, is covered in detail in Block III.

Figure 4.1.3 illustrates how raw text is processed through a sequence of annotation components. The raw text is first provided to an annotation object, after which a series of annotators add different types of linguistic and semantic information. The resulting annotation contains the information produced by these components and can be represented or exported in formats such as XML or plain text.

The annotation process can therefore be viewed as a sequence of stages:

Raw Text→Sentence Segmentation→Tokenization→POS Tagging→NER→Relation Labelling

The resulting annotated corpus contains the information required to construct training instances for supervised relation extraction models. These labelled instances are then used to train and evaluate the machine learning classifier.

4.1.5.2 The Supervised Algorithm

The supervised approach to relation extraction consists of three main steps during the training phase, followed by three corresponding steps during the testing (prediction) phase. The same feature representation learned during training must be used during prediction.

1. Training Phase

Step 1: - Obtain an annotated corpus.

A collection of documents is prepared in which human annotators identify named

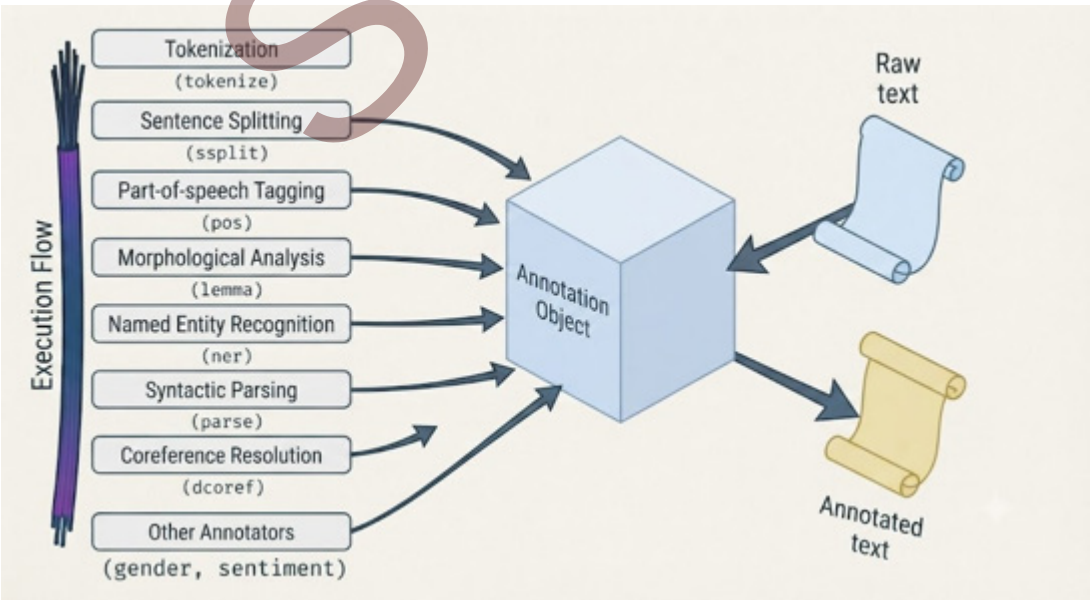


Fig 4.1.3: Overall System Architecture for Text Annotation [4]

SGOU

entities and assign the correct relation label to each related entity pair.

For example:

"Narayana Murthy founded Infosys."

The annotated information may be represented as:

$$(Infosys)$$

Creating a high-quality annotated corpus is often time-consuming and expensive, as it requires manual annotation by trained annotators.

Step 2: - Extract features

For each annotated entity pair, extract a set of relevant features and represent them as a feature vector.

The choice of features is one of the most important design decisions in a supervised relation extraction system. Features may include word features, entity features, and syntactic features.

Step 3: - Train a classifier

The extracted feature vectors and their corresponding relation labels are used to train a supervised machine learning classifier.

Classifiers introduced in Block III, such as:

1. Naïve Bayes
2. Logistic Regression
3. Support Vector Machines (SVMs)

can be used for relation classification. In practice, SVMs and Logistic Regression are often effective for relation extraction because the feature space can be large and sparse.

Testing Phase/prediction Phase

Step 1: - Identify Candidate Pairs

A new document is processed through Named Entity Recognition (NER), and pairs of entities within a sentence are generated as candidates.

Step 2: - Extract the same Features

Exactly the same feature extraction procedure is applied to each candidate pair. Using a different feature set at test time would make the trained model incompatible with the input representation.

Step 3:- Predict

The trained classifier assigns a relation label to each candidate pair, or NO_RELATION where appropriate.

SGOU

4.1.6 Feature Design for Supervised Relation Extraction

The performance of a supervised relation extractor depends heavily on the quality of the features it uses, in addition to the choice of classifier. Several types of features can be used, but four main families commonly used in practice are described below.

4.1.6.1 Lexical Features

Lexical features use the specific words or tokens in a sentence. They help capture the immediate linguistic context surrounding the entities.

Examples:

- Words between the two entities
- Words before and after the entities
- Entity names
- N-grams (unigrams, bigrams, trigrams)

Sentence: *"Microsoft acquired GitHub in 2018."*

Words between the entities: *acquired*

This word strongly indicates an ACQUIRED relation.

4.1.6.2 Entity Features

Entity features describe the characteristics and properties of the identified entities.

Common entity features include:

- Entity type (Person, Organization, Location)
- Number of entities
- Order of entities
- Distance between entities

Sentence: *"Satya Nadella works for Microsoft."*

Entity	*Type*
Satya Nadella	Person
Microsoft	Organization

4.1.6.3 Dependency Features

Dependency parsing identifies the grammatical relationships between words in a sentence. These features are especially helpful when entities are far apart or when the sentence has a complex structure.

SGOU

Sentence: *"Amazon acquired Whole Foods in 2017."*

The dependency parser identifies:

- Subject → Amazon
- Verb → acquired
- Object → Whole Foods

The dependency path helps determine that Amazon is the acquirer and Whole Foods is the acquired organization.

4.1.6.4 Syntactic Features

Syntactic features represent the grammatical structure of a sentence, using parse trees or phrase structures.

Examples include:

- Noun phrases
- Verb phrases
- Parse tree paths
- Constituency structures

These features help the model understand grammatical relationships between entities that may not be clear from the order of the words alone.

4.1.7 Supervised Approaches: Advantages & Limitations

Advantages of Supervised Approaches

- Supervised classifiers can often achieve higher recall than rule-based methods because they learn from a wide range of examples rather than relying on a fixed set of manually defined rules.
- They can identify subtle statistical patterns and feature combinations that may be difficult for humans to capture through manually written rules.
- They use well - understood and reliable machine learning algorithms such as Naïve Bayes, Logistic Regression, and Support Vector Machines (SVMs).
- When feature-based models are used, they are relatively interpretable because the feature weights and their contribution to the prediction can be examined.

Limitations of Supervised Approaches

- They require a sufficiently large and accurately annotated corpus, which can be expensive and time-consuming to produce.
- They may perform poorly when applied to a domain that differs significantly from the training domain.

SGOU

- They are generally limited to the relation types represented in the training data. Supporting a new relation type usually requires additional annotation and retraining.
- Traditional feature-based approaches may require linguistic expertise and careful feature engineering to achieve good performance.

4.1.8 Neural-Based Approach to Supervised Relation Extraction

In traditional feature-based approaches, experts must manually decide which information in the text matters for identifying a relation. Neural approaches reduce the need for manual feature engineering by learning useful representations and patterns directly from the text.

4.1.8.1 How Neural Relation Extraction Works

Neural relation extraction generally involves the following key steps:

1. Input: The model receives a sentence along with the two target entities whose relationship needs to be identified.
2. Word Embeddings: Each word or token is represented as a numerical vector that captures its semantic and contextual information. Methods such as Word2Vec or contextual models such as BERT can be used.
3. Neural Encoder: The word representations are processed by a neural network, such as an LSTM, CNN, or Transformer, to capture the context and relationships between words and entities.
4. Feature Representation: The encoder produces a vector representation that captures the information relevant to the relationship between the two target entities.
5. Classification: The resulting representation is passed to a classification layer, which predicts the most likely relation label from the predefined set of relation types.

4.1.8.2 Entity Marking

To help the model focus on the target entities involved in a relation, special markers can be placed around them in the sentence before processing. This technique, called entity marking, helps the model distinguish the entities whose relationship needs to be determined.

For example:

[E1] Narayana Murthy [/E1] founded [E2] Infosys [/E2] in Pune in 1981.

Here, [E1] and [/E1] mark the first target entity, while [E2] and [/E2] mark the second target entity.

The model can then use the contextual information surrounding these marked entities

SGOU

to construct a representation for relation classification.

4.1.8.3 A Worked Example

For example, consider the sentence:

"Satya Nadella is the CEO of Microsoft."

A neural relation extraction model can learn to predict the relation CEO_OF between "Satya Nadella" and "Microsoft" from labelled training examples, without requiring an explicitly written rule for the phrase *"is the CEO of"*.

Entity 1	Satya Nadella (PERSON)
Entity 2	Microsoft (ORG)
Predicted relation	CEO_OF
Output triple	(Satya Nadella, CEO_OF, Microsoft)

The model does not need to be explicitly told that the phrase *"is the CEO of"* indicates this relation. Instead, it learns useful patterns and representations from the training examples and stores this information in its learned parameters. It then uses these learned patterns to predict the relation for new sentences.

4.1.8.4 Advantages and Limitations

Advantages

Learns useful features and representations automatically from text.

- Captures contextual information and semantic relationships effectively.
- Can achieve higher accuracy than traditional feature-based methods for many relation extraction tasks.
- Can use pre-trained models such as BERT to leverage knowledge learned from large text corpora and reduce the amount of task-specific training data required.

Limitations:

- Requires lots of labelled training data
- Requires significant computing resources (GPUs)
- Results depend on data quality
- Models are difficult to interpret

SGOU

Recap

- Relation Extraction identifies semantic relationships between entities mentioned in text and represents them as structured triples of the form ($Entity_1$, Relation, $Entity_2$).
- Named Entity Recognition (NER) identifies and classifies entities, while Relation Extraction determines how those entities are semantically connected.
- Relation direction (argument ordering) is important because reversing the entities can change the meaning of the extracted fact. The order of words in a sentence may change in passive constructions, but the semantic roles of the entities remain the same.
- Different application domains use different relation inventories and ontologies, such as ACE for general-purpose relations, UMLS for biomedical relations, Wikipedia Infoboxes for structured facts, and WordNet for semantic relationships and hierarchies.
- Relation Detection is a binary classification task that determines whether a meaningful relation exists between a candidate entity pair.
- Relation Classification is a multiclass classification task that assigns a specific relation label, such as FOUNDER_OF, CEO_OF, or LOCATED_IN, to an entity pair identified as being related.
- Supervised learning requires a labelled dataset in which entity pairs are associated with their correct relation labels. The model learns patterns from these examples and uses them to predict relations in new sentences.
- In a supervised relation extraction approach, an annotated corpus is used to extract features and train classifiers such as Naïve Bayes, Logistic Regression, and Support Vector Machines (SVMs).
- Effective supervised relation extraction depends on carefully designed lexical, entity, dependency, and syntactic features, which capture contextual, grammatical, and structural information from text.
- Supervised approaches can achieve good performance when sufficient high-quality training data and appropriate features are available. However, they require annotated datasets, may have limited adaptability across domains, and can require significant feature engineering.
- Relation Extraction is a key component of modern NLP applications, supporting knowledge graph construction, question answering, biomedical text mining, semantic search, and information extraction by converting information from unstructured text into structured knowledge.

SGOU

- Neural-based supervised relation extraction uses deep learning models such as CNNs, LSTMs, GRUs, and Transformers such as BERT to automatically learn useful representations from text and classify the relationship between two entities.

Objective Type Questions

1. What is the semantic connection between two entities called?
2. In which format are relations usually represented?
3. Which NLP task identifies named entities in a sentence?
4. What is the first stage of relation extraction?
5. What is the second stage of relation extraction?
6. Which ontology defines general-purpose relation types?
7. Which ontology is widely used for biomedical relation extraction?
8. Which lexical resource provides hypernym (is-a) relations?
9. What is the relation between a specific entity and its class in WordNet called?
10. What type of classification is relation detection?
11. What type of classification is relation classification?
12. Which framework represents knowledge as Subject–Predicate–Object triples?
13. Which learning approach uses labelled training data?
14. Which feature type uses words between two entities?
15. Which feature type identifies grammatical links between words?

SGOU

Answers to Objective Type Questions

1. Relation
2. Triple

3. Named Entity Recognition (NER)
4. Relation Detection
5. Relation Classification
6. ACE
7. UMLS
8. WordNet
9. Instance-of
10. Binary Classification
11. Multiclass Classification
12. RDF
13. Supervised Learning
14. Lexical Feature
15. Dependency Feature

Assignments

1. Explain the concept of Relation Extraction and describe how it differs from Named Entity Recognition (NER).
2. Discuss the standard relation types used in Relation Extraction with suitable examples from ACE, UMLS, Wikipedia Infoboxes, and WordNet.
3. Explain the supervised learning approach to Relation Extraction, describing both the training and testing phases.
4. Describe the different feature types used in supervised Relation Extraction, including lexical, entity, dependency, and syntactic features, with suitable examples.
5. Discuss the advantages and limitations of supervised approaches to Relation Extraction.

SGOU

6. Explain the neural-based supervised approach to Relation Extraction. Describe its working process with a suitable example, and discuss the advantages and limitations of this approach.

Reference

1. S. Vajjala, B. Majumder, A. Gupta, and H. Surana, *Practical Natural Language Processing: A Comprehensive Guide to Building Real-World NLP Systems*. O'Reilly Media, 2020.

2. D. Jurafsky and J. H. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*.

3. C. D. Manning, P. Raghavan, and H. Schütze, "XML Retrieval," in *Introduction to Information Retrieval*. Cambridge University Press, 2008.

4. C. D. Manning, M. Surdeanu, J. Bauer, J. R. Finkel, S. Bethard, and D. McClosky, "The Stanford CoreNLP Natural Language Processing Toolkit," in *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pp. 55–60, 2014.

Suggested Reading

1. S. Bird, E. Klein, and E. Loper, *Natural Language Processing with Python*. O'Reilly Media, 2009.

2. M. Song, W. C. Kim, D. Lee, G. E. Heo, and K. Y. Kang, "PKDE4J: Entity and Relation Extraction for Public Knowledge Discovery," *Journal of Biomedical Informatics*, vol. 57, pp. 320–332, 2015.

SGOU

Unit 2

Lightly Supervised Approaches to Relation Analysis

Learning Outcomes

After completing this unit, learner will able to:

- differentiate supervised, unsupervised, semi-supervised, distant supervision, and weak supervision approaches to relation analysis
- explain the bootstrapping process used in semi-supervised relation extraction
- describe semantic drift in bootstrapping and explain the role of confidence measures in controlling it
- explain how distant supervision uses existing knowledge bases or databases to automatically generate labelled training data
- describe weak supervision and explain the sources and effects of noise in weakly labelled data
- explain the principles of Open Information Extraction and the working of the ReVerb algorithm
- compare lightly supervised approaches to relation extraction and select an appropriate approach for a given application

SGOU

Prerequisites

In real-world Natural Language Processing (NLP) applications, extracting meaningful relationships between entities from large collections of text is essential for building knowledge graphs, information retrieval systems, question-answering systems, and other intelligent applications. For example, from the sentence *"Sundar Pichai is the CEO of Google,"* an NLP system should be able to identify the entities *Sundar Pichai* and *Google* and determine the relationship *CEO of* between them. Understanding how such relationships can be automatically identified provides a foundation for studying advanced approaches to relation extraction. The approaches discussed in this unit are particularly useful for processing large-scale textual data and for extracting relationships when manually labelled training data are limited or when the system needs to work across different domains.



Learners are expected to understand the definition of a relation, the representation of relations as triples, and the distinction between relation detection and relation classification. They should also be familiar with the major feature families used in supervised relation extraction. In addition, learners should recall the two important limitations of supervised relation extraction discussed in Unit 1: the requirement for costly manually annotated corpora and the limited ability of supervised models to generalize across different domains. Understanding these limitations will help learners appreciate the need for the alternative relation extraction approaches introduced in this unit. Basic familiarity with Python programming will also be helpful for understanding and following the code examples presented in the unit. Studying these approaches helps learners understand how relations can be extracted more efficiently from large amounts of text, reduce dependence on manually annotated datasets, improve the applicability of relation extraction across different domains, and develop a stronger foundation for building practical NLP applications such as knowledge graphs, information retrieval, and question-answering systems.

Key words

Bootstrapping, Seed Tuple, Seed Pattern, Semantic Drift, Confidence Score, Knowledge Base, Canonicalization, Semi-Supervised Learning, Distant Supervision, Weak Supervision, Open Information Extraction, ReVerb, Noisy Labels

Discussion

4.2.1 Introduction

Unit 1 showed that supervised relation extraction can perform well when sufficient manually annotated training data is available. The major challenge, however, is obtaining such data. Relation extraction approaches differ considerably in the amount and type of supervision they require. Unit 1 introduced hand-written patterns and supervised approaches. This unit examines methods that reduce or eliminate the need for manual annotation, including ***semi-supervised learning, distant supervision, weak supervision, and unsupervised relation extraction***.

Consider the task of building a supervised relation extractor for Indian business news. A team of expert annotators may need to read thousands of articles, identify named entities, and examine pairs of entities to determine whether a relation exists and, if so, which relation is expressed. Even for a corpus containing ten thousand sentences, this process can require considerable time and expert effort. Furthermore, a model trained on business news may not perform as well on clinical notes or legal judgments because vocabulary, writing style, and sentence structure differ across domains.

To overcome these limitations, researchers have developed approaches that require much less manually labelled data. Lightly supervised approaches obtain supervision indirectly - for example, from a small number of seed examples, an existing knowledge

SGOU

base, automatically generated labels, or heuristic rules. Unsupervised approaches go one step further and attempt to discover relational information without labelled training examples.

The major relation extraction approaches can therefore be broadly organized according to the amount and source of supervision they use, as illustrated in Figure 4.2.2:

- Hand-Written Pattern-Based Methods
- Supervised Methods
- Semi-Supervised Methods
- Distant Supervision
- Weak Supervision
- Unsupervised Methods

4.2.2 Supervised versus Unsupervised Relation Analysis

4.2.2.1 Supervised Relation Analysis

In supervised relation analysis, a fixed set of entity types and relation types is defined in advance. A training corpus is manually annotated with these relations, and the annotated examples are used to train a classifier that predicts relations in previously unseen text.

Supervised approaches can achieve high accuracy when sufficient high-quality labelled data is available. However, they have two major limitations.

First, manual annotation is labour-intensive and expensive. Annotators must examine

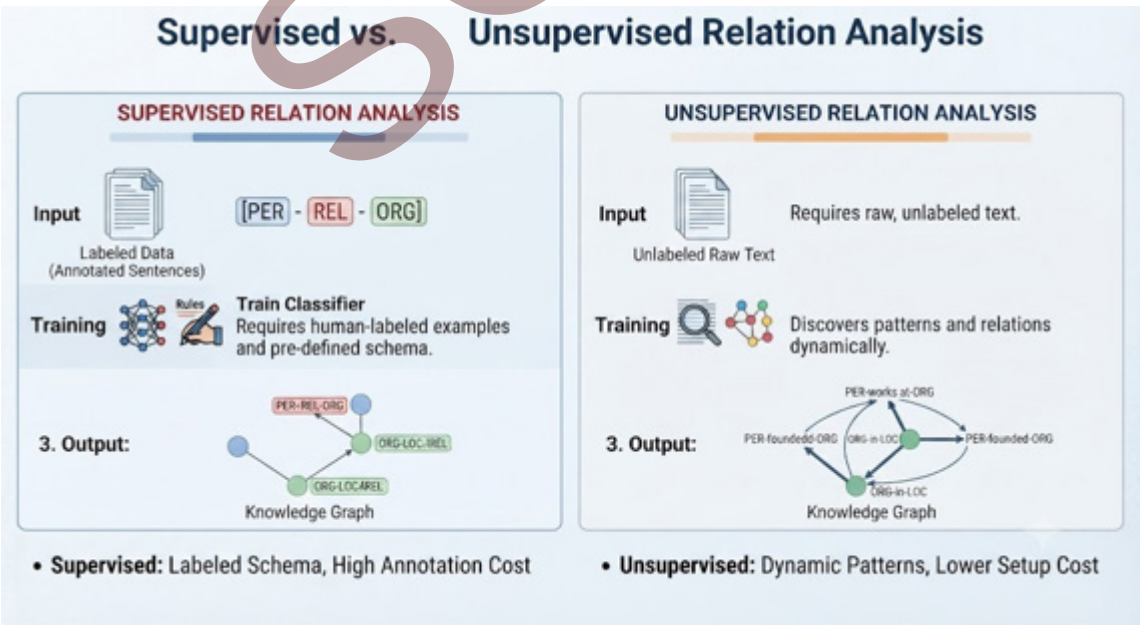


Fig 4.2.1: Comparison of supervised and unsupervised relation analysis in terms of input data, training process, and relation extraction output.

SGOU

large numbers of sentences, identify relevant entity pairs, and assign the appropriate relation labels. Consequently, supervised relation extraction can be difficult to scale to large datasets or new domains.

Second, introducing a new relation type is costly. Since the model learns only the relation categories represented in the training data, extending the system to additional entity types or relations generally requires new annotated examples and retraining of the model.

These limitations motivate relation extraction methods that require less manually labelled data.

4.2.2.2 Unsupervised Relation Analysis

In unsupervised relation analysis, the system does not rely on manually labelled training examples and may operate without a predefined set of relation categories. Instead, it attempts to discover relational patterns directly from raw text.

One important schema-free approach is Open Information Extraction (Open IE). Unlike traditional supervised relation extraction, Open IE does not restrict extraction to a fixed inventory of relations. Instead, it identifies relational expressions directly from sentences, often in the form of verb or relational phrases.

For example, from the sentence:

"IndiGo operates a hub in Hyderabad."

an Open IE system may produce the triple:

(IndiGo, operates a hub in, Hyderabad)

Here, "operates a hub in" is extracted directly from the sentence rather than being mapped to a predefined relation label such as HUB_AT.

This version also prepares the learner better for your later section on ReVerb, because ReVerb is specifically an Open IE system, rather than simply an "unsupervised classifier."

4.2.2.3 The Middle Ground

Lightly supervised approaches, which form the main focus of this unit, occupy an intermediate position between fully supervised and unsupervised relation extraction methods. They reduce the need for large manually annotated corpora by obtaining supervision from limited or indirect sources.

Semi-supervised methods use a small amount of labelled data together with a much larger collection of unlabelled text. A common semi-supervised technique is bootstrapping, in which a small set of manually selected seed examples is used to discover additional relation instances and patterns from unlabelled text.

Distant supervision uses an existing knowledge base or database to automatically generate training labels. Relations stored in the knowledge base are aligned with

SGOU

sentences containing the corresponding entities, thereby reducing the need for manual sentence-level annotation.

Weak supervision uses approximate, incomplete, or noisy labels obtained from indirect sources such as heuristic rules, external knowledge bases, pattern-matching rules, or labelling functions. Since these sources may not always produce correct labels, weakly supervised training data is generally noisier than manually annotated data.

Overall, these approaches substantially reduce the time, cost, and effort required for manual annotation while still providing useful supervision for relation extraction.

4.2.3 Semi-Supervised Relation Extraction via Bootstrapping

Supervised machine learning assumes that we have a large amount of labelled data. Semi-supervised relation extraction addresses this limitation by learning from a small set of labelled examples, called seed examples, together with a large collection of unlabelled text. These seed examples bootstrap the learning process.

4.2.3.1 What Is Bootstrapping?

Bootstrapping is a semi-supervised learning technique that begins with a limited number of manually labelled seed examples. These examples guide the system in identifying new patterns and extracting additional relation instances from an unlabelled corpus.

A seed example refers to an initial manually labelled entity pair that exemplifies the target relation.

For example, consider the relation Works_For:

Person	Organisation
Satya Nadella	Microsoft
Sundar Pichai	Google
Tim Cook	Apple

These examples enable the system to learn expressions such as:

“X is the CEO of Y”

“X works at Y”

“Y appointed X”

The learned patterns are then used to identify new entity pairs, such as:

Andy Jassy → Amazon

Bootstrapping proceeds by taking the entities in a seed pair, finding sentences that contain both entities, and generalising the context around them into patterns. These patterns are then applied to the corpus to discover new relation tuples. High-confidence

SGOU

newly extracted tuples are added to the seed set, and the process is repeated until no useful new relations are found.

Figure 4.2.2 illustrates the complete cycle.

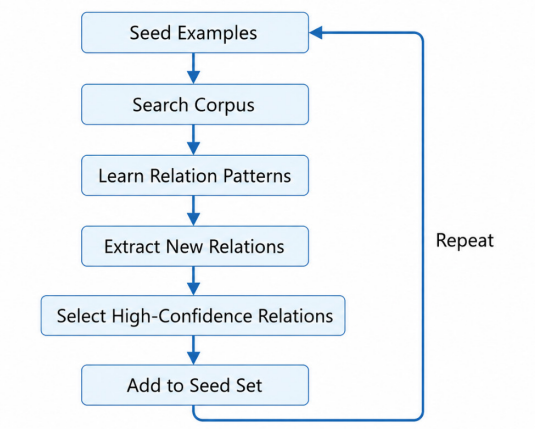


Fig 4.2.2: Bootstrapping cycle for semi-supervised relation extraction

Figure 4.2.2: Bootstrapping cycle for semi-supervised relation extraction: seed tuples produce patterns, patterns produce new tuples, and high-confidence tuples become seeds for the next iteration.

4.2.3.2 The Bootstrapping Algorithm [1]

tuples ← gather a set of seed tuples that have relation R

repeat

sentences ← find sentences that contain entities in tuples

patterns ← generalise the context between and around the entities

newtuples ← use patterns to search for additional relation tuples

newtuples ← retain new tuples with high confidence

tuples ← tuples + newtuples

until no useful new tuples are found

return tuples

4.2.3.3 Worked Example

Suppose we need to build a list of airline hub locations. Consider a scenario in which only one fact is initially known: IndiGo has a hub at Hyderabad. This single seed tuple is then used to search a news corpus for additional sentences that mention both entities.

The following examples may be identified:

“IndiGo, which uses Hyderabad as a hub, cancelled twelve flights today.”

SGOU

"All flights from IndiGo's hub at Hyderabad were delayed on Monday."

"A spokesman at Hyderabad, a main hub for IndiGo, confirmed the delays."

From these three sentences, by analyzing the words between the entity mentions, the words preceding the first mention, the word following the second, and the named entity types, three general patterns can be extracted:

/ [ORG], which uses [LOC] as a hub /

/ [ORG]'s hub at [LOC] /

/ [LOC], a main hub for [ORG] /

These patterns are subsequently applied to the broader corpus, enabling the identification of sentences about airlines not previously encountered by the system.

For example, from the sentence "Air India, which uses Delhi as a hub, added new international routes," the first pattern extracts the new tuple (Air India, Delhi). This tuple is then added to the seed set, initiating the next iteration.

4.2.3.4 Semantic Drift

Bootstrapping has a potential failure mode. Since each iteration builds on the output of the previous one, a single error can propagate and compound over time. This is known as **semantic drift**: an incorrect pattern introduces erroneous tuples, which in turn generate further incorrect patterns, causing the meaning of the extracted relation to gradually drift away from the intended target.

For example:

Consider the sentence:

"Kochi has a ferry hub at Vypin."

If this sentence is incorrectly accepted as a positive example of the Airline–Hub relation, the system may extract the tuple (Kochi, Vypin), even though Kochi is a city, not an airline. Patterns learned from this incorrect tuple may then begin matching ferry terminals, bus depots, and shipping ports. After several iterations, the system may extract a much broader relation than originally intended, thereby contaminating the extracted database.

4.2.3.5 Confidence Values

To prevent drift, bootstrapping systems assign confidence values to both patterns and tuples, and accept only those above a threshold.

Confidence in a pattern balances two factors: how well the pattern agrees with the tuples already trusted and how productive the pattern is in the corpus. Given a document collection D, a current set of trusted tuples T, and a proposed pattern p, we define:

- $hits_p$ **:** The number of tuples in T that pattern p matches in D.

SGOU

- ***finds*** $_p$ **:** The total number of tuples that the pattern $p$extracts from D.

These are combined using the **RlogF confidence score**:

$$Conf_{RlogF}(p) = \frac{hits_p}{finds_p} \times log(finds_p)$$

The first term rewards accuracy, indicating the proportion of tuples found by the pattern that are already trusted. The second term rewards productivity, giving greater importance to patterns that extract more tuples. The logarithm limits the influence of extremely productive patterns.

Combining Evidence from Multiple Patterns

A new tuple may be supported by several patterns. Their evidence can be combined using the noisy-or technique. It assumes that a tuple is false only if all of its supporting patterns are incorrect and that the pattern failures are independent.

If $Conf(p)$is a bounded confidence value between 0 and 1, the probability that pattern p is incorrect is:

$1 - Conf(p)$

Therefore, the confidence of a tuple $t$supported by several patterns is:

$$Conf(t) = 1 - \prod_{p \square\square\square\square\square\square\square\square\square\square\square\square t} (1 - Conf(\text{p}))$$

Setting conservative thresholds for accepting new patterns and tuples helps prevent the system from drifting away from the target relation.

4.2.4 Distant Supervision Methods

Distant supervision combines some of the advantages of bootstrapping with supervised learning. Instead of relying on only a small number of seed examples, distant supervision uses a large knowledge base or database to obtain many automatically labelled examples. It then extracts noisy pattern features from these examples and combines them to train a supervised classifier.

4.2.4.1 How Distant Supervision Works

Supervised relation extraction requires large manually annotated datasets, which are expensive and time-consuming to create. Distant supervision addresses this limitation by automatically generating training examples from existing knowledge bases.

Rather than relying on a limited set of manually created seed examples, distant supervision utilizes existing knowledge bases such as Wikidata and DBpedia, which contain thousands of known relation instances. The system subsequently searches extensive collections of unlabelled text for sentences that include both entities from a

SGOU

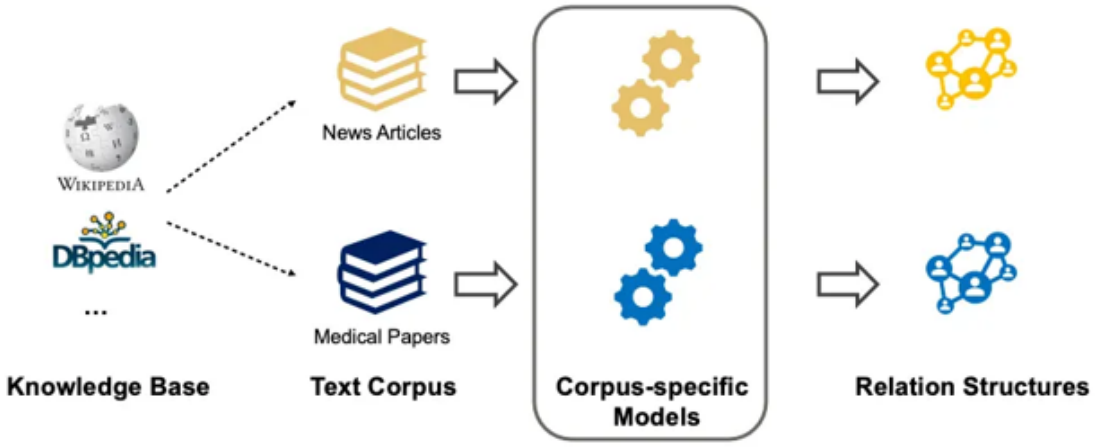


Fig 4.2.3: General schema for distantly supervised relation extraction using a knowledge base and unlabelled text corpus

known relation tuple. These sentences are automatically designated as positive training examples and serve to train a relation extraction model.

4.2.4.2 The Distant Supervision Algorithm [1]

function DISTANT_SUPERVISION (Database D, Text T) returns classifier C

observations ← empty set

for each relation R

for each tuple (e1, e2) with relation R in D

sentences ← sentences in T that contain e1 and e2

f ← features extracted from sentences

observations ← observations + (e1, e2, f, R)

C ← train supervised classifier on observations

return C

4.2.4.3 A Worked Example

Consider the database tuple **(Narayana Murthy, FOUNDER_OF, Infosys)**. Searching a corpus of Indian business news for sentences containing both entities might return:

"Narayana Murthy founded Infosys with six engineers."

"Infosys was started by Narayana Murthy in 1981."

"Narayana Murthy, the Infosys founder, spoke at the event."

Although these three sentences use different wording, they all express the same

SGOU

FOUNDER_OF relation. The classifier can therefore learn multiple linguistic patterns for the same relation, making it more flexible than manually written extraction rules.

4.2.4.4 Rich Conjunctive Features

Because distant supervision produces very large training sets, it can make use of **rich conjunctive features** that combine several individual features. Thousands of patterns may be extracted by combining entity types with intervening words or dependency paths.

Pattern	Example
PER founded ORG	Narayana Murthy founded Infosys
ORG started by PER	Infosys was started by Narayana Murthy
PER, the ORG founder	Narayana Murthy, the Infosys founder

A single feature may combine several conditions at once, such as: entity 1 is an ORG, entity 2 is a PERSON, the next word is *"said"*, and the dependency path follows a particular sequence. Such a specific feature may be too rare to learn reliably from a small annotated corpus, but with hundreds of thousands of training instances, it can be learned more effectively.

Advantages

- Requires much less manual annotation.
- Can automatically generate very large training datasets.
- Supports learning multiple relation types simultaneously.
- Produces richer feature representations because of the large amount of training data.

Disadvantages

- Training data may contain noisy labels because not every sentence containing both entities expresses the intended relation.
- Reliable negative examples are difficult to obtain.
- Performance depends on the completeness and quality of the knowledge base.
- Some parameter tuning may still be required for different domains.

4.2.5 Weak Supervision Techniques

Weak supervision refers to training on data that is labelled imperfectly, incompletely, or indirectly rather than through careful human annotation. Distant supervision is itself

SGOU

a form of weak supervision; the labels are derived from a database rather than assigned by a human reading each sentence, so some of the automatically generated labels may be incorrect.

The idea predates the term distant supervision. Craven and Kumlien (1999) [4] used the term weakly labelled data for a similar approach applied to biomedical text, several years before Mintz et al. introduced the term distant supervision.

4.2.5.1 Why are the Labels noisy?

Distant supervision makes a strong assumption, sometimes called the distant supervision assumption: if two entities are related in the database, then any sentence containing both entities expresses that relation.

This assumption is frequently incorrect. Consider the database fact (Narayana Murthy, FOUNDER_OF, Infosys) and the sentence:

"Narayana Murthy stepped down from the Infosys board in 2014."

The sentence contains both entities but expresses a different relation, namely stepping down from the board, rather than founding. Distant supervision may nevertheless label it as a positive FOUNDER_OF example, thereby introducing a false training instance. In a large corpus, many such mislabelled instances may be generated.

4.2.5.2 How Do Systems Cope with Noise?

Because individual labels may be unreliable, weakly supervised systems can benefit from large amounts of training data. The reasoning is statistical. A feature that genuinely indicates the FOUNDER_OF relation, such as the word *"founded"* appearing between the entities, may occur frequently in correctly labelled sentences. A spurious feature arising from a mislabelled sentence, such as *"stepped down from"*, may occur less consistently and may also appear in sentences associated with other relations.

The classifier assigns weights to features according to how reliably they predict each relation across the training data. Reliable features tend to receive higher weights, whereas inconsistent or noisy features receive lower weights. The noise is therefore not eliminated, but its influence can be reduced.

This illustrates an important trade-off in weak supervision: a large quantity of imperfectly labelled data can sometimes compensate for lower label quality, provided that the noise is not excessive or systematic.

4.2.5.3 Sources of Weak Supervision

Distant supervision from a knowledge base is one of the most common forms of weak supervision, but several other sources of weak labels are also used in practice:

- **Heuristic rules:** A simple rule such as *"if the sentence contains 'founded' between a PERSON and an ORG, label it FOUNDER_OF"* can generate many labels quickly, although some may be incorrect.
- **Existing structured resources:** Wikipedia infoboxes, product catalogues,

SGOU

and company registries can provide entity pairs with known rela-tions.

- **Crowdsourced annotation:** Non-expert annotators can produce labels faster and more cheaply than domain experts, although the resulting labels may contain more errors.
- **Cross - lingual projection:** Labels from a well-resourced language can be projected onto a low-resource language using parallel or aligned text.

4.2.6 Unsupervised Relation Extraction

The goal of unsupervised relation extrac-tion is to extract relations from text when no labelled training data and no predefined list of relations are available. One important approach to this task is Open Information Extraction (Open IE).

The defining feature of Open IE is that the relations are not selected from a fixed inventory of labels but are extracted directly as phrases from the text, often as verb-based expressions. Rather than deciding whether a sentence expresses a predefined relation such as FOUNDER_OF or HQ_IN, the system extracts the relation phrase that connects the two entities.

4.2.6.1 The ReVerb Algorithm

The **ReVerb system** extracts a relation from a sentence **s** in four steps:

- **Step 1:** Run a part-of-speech tagger and entity chunker over s.
- **Step 2:** For each verb in s, find the longest sequence of words w that begins with a verb and satisfies the required syntactic and lexical constraints. Merge adjacent matches where appropriate.
- **Step 3:** For each phrase w, find the nearest noun phrase x to the left that is not a relative pronoun, wh-word, or existential "there." Then find the nearest noun phrase y to the right.
- **Step 4:** Assign a confidence score c to the relation

  r = (x, w, y) using a confidence classifier, and return the extracted relation.

4.2.6.2 Syntactic and Lexical Constraints

A relation is accepted only if it satisfies both syntactic and lexical constraints. The syntactic constraint ensures that the relation phrase begins with a verb and may also contain nouns and other permitted word classes.

$$V \mid VP \mid VW^*P$$

where:

- V = verb particle? adverb?
- W = noun | adjective | adverb | pronoun | determiner
- P = preposition | particle | infinitive marker

SGOU

The lexical constraint removes very rare or overly specific relation phrases. ReVerb applied the extraction algorithm offline to approximately 500 million web sentences, normalized the extracted relation phrases, and retained relations that occurred with at least 20 distinct argument pairs. This resulted in a dictionary containing approximately 1.7 million normalized relation phrases.

4.2.6.3 Worked Example

Consider the sentence:

"IndiGo has a hub in Hyderabad, which is the headquarters of InterGlobe Aviation."

The system identifies two relation phrases: "has a hub in" and "is the headquarters of." Shorter candidates such as "has" and "is" may also satisfy the verb-initial condition, but the longest valid relation phrases are preferred.

Step 3 then finds IndiGo to the left and Hyderabad to the right of *"has a hub in."* For *"is the headquarters of,"* the system skips the relative pronoun "which" and selects Hyderabad as the left argument and InterGlobe Aviation as the right argument.

The final output is:

r_1: <IndiGo, has a hub in, Hyderabad>

r_2: <Hyderabad, is the headquarters of, InterGlobe Aviation>

4.2.6.4 Advantages and Limitations

Advantage

- A major advantage of unsupervised relation extraction is its ability to handle a large number of relations without requiring them to be specified in advance.
- A single Open IE system can extract facts from different domains, such as airlines, medicine, sports, and politics, because it is not restricted to a predefined relation inventory.

Limitations

- A major limitation is the need to map the large number of extracted relation strings into a canonical form before they can be stored or compared effectively.
- Phrases such as *"has a hub in," "operates a hub at,"* and *"maintains a hub in"* may express the same relation but appear as different relation strings.
- Determining that such expressions are equivalent is known as canonicalisation, and this is itself a difficult problem.
- Many Open IE methods focus mainly on relations expressed through verbs and may therefore miss relations expressed nominally. For example, the phrase *"the Infosys founder Narayana Murthy"* expresses a founding relation without using a verb and may be missed by a verb-centred extractor.

SGOU

Recap

- Supervised relation extraction requires a large manually annotated training corpus and predefined relation types.
- Semi-supervised bootstrapping begins with a small set of seed tuples and iteratively discovers new relation instances from unlabelled text.
- Seed tuples are manually labelled examples that guide the bootstrapping process.
- Semantic drift occurs when incorrect patterns or tuples are repeatedly learned, causing the extracted relation to deviate from its intended meaning.
- Confidence values help reduce semantic drift by selecting only reliable patterns and relation tuples.
- Distant supervision automatically generates training data by matching facts from a knowledge base with unlabelled text.
- Weak supervision learns from noisy, incomplete, or imperfect labels generated using sources such as heuristic rules, knowledge bases, or crowdsourced annotations.
- Open Information Extraction (Open IE) extracts relations directly from text without relying on predefined relation types.
- ReVerb is an Open IE system that extracts verb-based relation phrases between entity pairs.
- Lightly supervised and unsupervised approaches reduce the need for manual annotation and support relation extraction from large collections of text.

SGOU

Objective Type Questions

1. What is the primary limitation of supervised relation extraction?
2. What is the initial labelled entity pair used in bootstrapping called?
3. What phenomenon causes bootstrapping to gradually extract incorrect relations?
4. What is used to reduce semantic drift in bootstrapping?
5. What type of learning does bootstrapping belong to?
6. What external resource is primarily used in distant supervision?

7. Name one knowledge base used in distant supervision.

8. What type of training labels are generated automatically in distant supervision?

9. Which type of supervision uses imperfect or noisy labels?

10. Name one source of weak supervision.

11. What does Open IE stand for?

12. Which Open IE system discussed in this unit extracts verb-based relation phrases?

13. What usually forms the relation phrase in Open IE?

14. What process maps equivalent relation phrases into a common representation?

15. Which approach discovers relations without predefined relation types?

Answers to Objective Type Questions

1. Manual annotation

2. Seed tuple

3. Semantic drift

4. Confidence values

5. Semi-supervised learning

6. Knowledge base

7. Wikidata and DBpedia

8. Automatically generated training labels

9. Weak supervision

10. Heuristic rules *(or Knowledge base, Crowdsourcing, Cross-lingual projection)*

11. Open Information Extraction

12. ReVerb

SGOU

13. Verb phrase

14. Canonicalization

15. Open Information Extraction (Open IE)

Assignments

1. Differentiate supervised and unsupervised relation extraction with suitable examples.

2. Explain the bootstrapping approach for semi-supervised relation extraction using a simple example.

3. Describe how distant supervision automatically generates training data. Illustrate your answer.

4. What is weak supervision? Explain why weakly labelled data is considered noisy.

5. What is Open Information Extraction (Open IE)? Explain how the ReVerb algorithm extracts relations from text with a suitable example.

Reference

1. D. Jurafsky and J. H. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*.

2. Z. Li, "Representation Learning for Weakly Supervised Relation Extraction," *arXiv preprint arXiv:2105.00815*, 2021.

3. S. Vajjala, B. Majumder, A. Gupta, and H. Surana, *Practical Natural Language Processing: A Comprehensive Guide to Building Real-World NLP Systems*. O'Reilly Media, 2020.

4. M. Craven and J. Kumlien, "Constructing Biological Knowledge Bases by Extracting Information from Text Sources," in *Proceedings of the 7th International Conference on Intelligent Systems for Molecular Biology (ISMB)*, pp. 77–86, 1999.

SGOU

Suggested Reading

1. S. Bird, E. Klein, and E. Loper, *Natural Language Processing with Python*. O'Reilly Media, 2009.

2. C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. Cambridge University Press, 2008.

SGOU

Unit 3
Evaluation of Relation Analysis Systems

Learning Outcomes

After completing this unit, learner will able to:

- explain the need for evaluating relation extraction systems and the challenges involved in their evaluation
- describe major benchmark datasets used for relation extraction and the distinguishing characteristics of each
- identify the possible evaluation cases in relation extraction and explain the double-penalty effect
- compute precision, recall, and F1-score for a supervised relation extraction system
- compare micro-averaging and macro-averaging and explain when each is appropriate
- describe commonly used evaluation standards and protocols for relation extraction systems

SGOU

Prerequisites

This unit concludes the discussion of relation analysis introduced in Units 1 and 2. Learners should have a basic understanding of the concept of a relation, the representation of relations as triples, and the major approaches to relation extraction, including supervised learning, bootstrapping, distant supervision, weak supervision, and Open Information Extraction (Open IE). Learners should also recall the concepts of precision, recall, and F1-score introduced in Block III for evaluating text classifiers. In this unit, these measures are applied to relation tuples, where evaluation can be more complex because a text may contain several relations and Open IE does not use a fixed set of relation types. A basic understanding of Python will also help learners follow the code examples. No new mathematical concepts are introduced beyond those already studied.



Studying this topic matters because developing a relation extraction system is not enough; you also need to determine how accurately and effectively the system extracts relations. Evaluation helps learners compare relation extraction approaches, identify correct and incorrect extractions, measure performance improvements, and understand the strengths and limitations of different methods. This knowledge is relevant to practical NLP applications such as knowledge graphs, information retrieval, knowledge bases, and intelligent information-processing systems. Through this unit, learners will understand benchmark datasets, evaluation standards, precision, recall, F1-score, micro- and macro-averaging, and methods for evaluating lightly supervised and Open IE systems. These concepts help learners interpret evaluation results and understand whether a relation extraction system performs reliably for a particular application.

Key Concepts

Gold Standard, Benchmark Dataset, Evaluation Protocol, True Positive, False Positive, False Negative, Precision, Recall, F1-Score, Micro-Averaging, Macro-Averaging, Double-Penalty Effect, Relation Tuple

Discussion

4.3.1 Introduction

Units 1 and 2 outlined multiple approaches to constructing a relation extraction system. Each approach has its own strengths and limitations. Supervised learning generally achieves high accuracy, distant supervision reduces the need for manual annotation, and Open IE can discover previously unseen relations. Determining which approach performs best for a given task requires systematic evaluation.

Evaluation serves three important purposes. First, it provides an objective basis for comparing different relation extraction systems to determine which performs more effectively. Second, it measures the impact of modifications or improvements made to a system, allowing researchers to determine whether performance has improved or deteriorated. Third, it supports reliable and reproducible research by providing standard measures and procedures for reporting system performance. Without systematic evaluation, claims about a system's performance cannot be independently assessed and therefore remain unsubstantiated.

Evaluating relation extraction systems is more challenging than evaluating classification systems because of fundamental differences in task structure. In classification, each test document is typically assigned a predefined label, which simplifies error counting. In contrast, relation extraction may involve a variable number of relations per document; systems may identify relations that are absent from the gold-standard annotation, and in Open IE, extracted relations are not restricted to a predefined relation inventory. Table 4.3.1 illustrates the distinctions among these paradigms.

This unit introduces the benchmark datasets, evaluation standards, and performance

SGOU

Table 4.3.1: Comparison of classification, closed relation extraction, and Open Information Extraction

Classification	Closed Relation Extraction	Open IE
Usually, one label per instance	Usually, one label per instance	Usually, one label per instance
Fixed class set	Fixed class set	No predefined relation types
Label matching	Entity and relation matching	Argument and relation-phrase matching
Evaluation against class labels	Evaluation against gold relation triples	Evaluation may require flexible or semantic matching

measures commonly used to assess relation extraction systems, enabling fair and reproducible comparison of different approaches. Benchmark datasets are particularly important because results from different corpora or experimental settings cannot be compared reliably.

Examples:

Classification

Sentence: “The movie was excellent.”

Prediction: Positive

Gold: Positive - ✓Correct

Relation Extraction

Sentence : “Narayana Murthy founded Infosys.”

Prediction: (Narayana Murthy,

FOUNDER_OF, Infosys)

Gold : (Narayana Murthy, FOUNDER_OF, Infosys) - ✓ Correct

Open IE

Sentence : “Narayana Murthy founded Infosys.”

Prediction: (Narayana Murthy, established, Infosys)

Gold : (Narayana Murthy, founded, Infosys),

The two extractions may overlap in meaning but differ in their relation and argument boundaries, making evaluation more difficult than simple label matching.

SGOU

4.3.2 Benchmark Datasets and Standards

A benchmark dataset consists of text in which relationships are annotated by human experts according to a documented annotation scheme. These datasets are then made publicly available for training and evaluating relation extraction systems.

Benchmark datasets are fundamental to natural language processing (NLP) research because they provide a standardized basis for performance evaluation. When different research groups evaluate their systems using the same benchmark dataset and the same evaluation metrics, such as precision, recall, and F1-score, the resulting comparisons are fairer and more meaningful. Without benchmark datasets, differences in training data, test data, and evaluation procedures would make objective comparison difficult.

Relation Extraction (RE) is an NLP task that aims to identify and classify relationships between entities mentioned in text. In a typical relation extraction pipeline, Named Entity Recognition (NER) first identifies the entities, after which the system performs relation identification and relation classification.

Several previously proposed relation extraction datasets are discussed in [4]. The following sections trace the major benchmark datasets used for relation extraction, from early benchmarks introduced in the 2000s to more recent document-level datasets.

4.3.2.1 Automatic Content Extraction (ACE)

The Automatic Content Extraction (ACE) programme ran from 2000 to 2008 and shifted the focus of information extraction from fixed templates toward the direct identification of entities and relations in text. ACE defined 17 relation types, grouped into broader categories such as PHYSICAL, PART-WHOLE, PERSON-SOCIAL, and ORG-AFFILIATION.

ACE datasets were drawn largely from newswire and related text sources. A key feature of ACE was the annotation of both entity mentions and the relations between them within the same corpus. For many years, ACE served as an important benchmark for evaluating supervised relation extraction systems.

4.3.2.2 DocRED

Earlier relation extraction benchmarks mainly focused on relations expressed within a single sentence. DocRED, introduced in 2019, extends relation extraction to the document level. A relation may involve entities mentioned in different sentences, and identifying it may require combining information from multiple parts of a document.

DocRED is a large-scale document-level relation extraction dataset constructed from Wikipedia and Wikidata. It contains both human-annotated data and additional data generated through distant supervision. The dataset covers 96 relation types, including relations such as educated_at and spouse.

The original DocRED dataset contains 5,053 documents, 132,375 entity mentions, and 56,354 relational facts.

SGOU

4.3.2.3 TACRED

TACRED, released in 2017, is a large-scale relation extraction dataset created using annotations collected through Amazon Mechanical Turk. It was derived from the Text Analysis Conference (TAC) Knowledge Base Population evaluations and focuses mainly on relations involving people, organizations, and locations. TACRED contains 106,264 examples covering 41 relation types, in addition to a nonrelation class.

TACRED is notable for two important design characteristics. First, it contains a large proportion of negative examples, in which an entity pair does not express any of the predefined relations. This introduces substantial class imbalance and requires relation extraction systems to distinguish true relations from unrelated entity pairs.

Second, its relation types are derived from knowledge base population tasks, meaning many extracted relations correspond to factual information suitable for storage in a structured knowledge base.

4.3.2.4 SciERC

SciERC is a dataset designed for constructing scientific knowledge graphs. It provides multi-task annotations for scientific entities, relations, and coreference clusters.

The dataset contains 500 scientific abstracts collected from 12 AI conference and workshop proceedings spanning four artificial intelligence research communities. The abstracts were obtained from the Semantic Scholar Corpus [5].

SciERC is particularly useful for evaluating relation extraction in the scientific domain, where systems must identify specialized entities and the relationships between them.

4.3.2.5 SemEval-2010 Task8

SemEval-2010 Task 8 is a benchmark dataset [8] for relation extraction that focuses on the multi-way classification of semantic relations between pairs of nominals. The dataset contains 10,717 annotated examples covering nine semantic relation types, together with an OTHER class.

The relations include CAUSE-EFFECT, COMPONENT-WHOLE, CONTENT-CONTAINER, and others. Unlike benchmarks that mainly represent factual relations between named entities, SemEval-2010 Task 8 focuses on semantic relationships between nominal expressions.

Because of its clearly defined relation categories and standardized evaluation setting, SemEval-2010 Task 8 became a widely used benchmark for evaluating relation classification models.

4.3.2.6 What Makes a Good Benchmark

Not every annotated corpus is suitable as a benchmark. Four properties are particularly important.

- **A documented annotation scheme:** Annotators should be provided with clear guidelines specifying what counts as each relation. Without clear guidelines, different annotators may interpret the task differently, resulting

SGOU

in inconsistent labels.

- **High inter-annotator agreement:** If trained annotators frequently disagree when independently annotating the same text, this may indicate ambiguity in the task or annotation guidelines. Inter-annotator agreement is therefore often reported as an indicator of annotation quality.

- **A held-out test set:** The test set should be kept separate from model training and parameter tuning so that reported results reflect performance on unseen data rather than adaptation to the test examples.

- **Public availability:** A benchmark is most useful for comparison when it is accessible to other researchers. Datasets with restrictive access conditions may be used less widely.

4.3.2.7 Resources for Relation Extraction in Indian Languages

The benchmark datasets discussed above are predominantly in English. Relation extraction for Indian languages remains comparatively under-resourced, and developing suitable annotated datasets is still an important research area.

Some useful resources include:

- **AI4AI4Bharat resources:** AI4Bharat and related research groups have developed datasets and language resources for Hindi, Malayalam, Tamil, and several other Indian languages. These resources support a range of NLP tasks and can facilitate the development of relation extraction systems.

- **IndicCorpV2:** A large multilingual corpus covering 22 Indian languages. Although it is not a relation extraction benchmark, it provides large-scale textual data that can support pretraining and other NLP research.

- **WikiAnn:** A multilingual data-set widely used for Named Entity Recognition (NER). Since entity recognition is often an important preliminary step in relation extraction, WikiAnn can support the development and evaluation of entity-processing components.

- Compared with English, dedicated relation extraction benchmarks for Indian languages are still limited. Developing high-quality, carefully annotated relation extraction datasets for these languages would therefore make a valuable contribution to the field.

4.3.3 Evaluation Methodologies

Supervised relation extraction systems are typically evaluated using test sets containing human-annotated gold-standard relations. Their performance is measured using precision, recall, and F1-score.

This is the most straightforward evaluation setting because the gold standard specifies the correct relation information for the annotated entity pairs in the test set. The system predictions can therefore be directly compared with the gold-standard relations.

SGOU

Table 4.3.2 True positive, false positive, and false negative outcomes in relation extraction evaluation

Outcome	Definition	Example
True Positive (TP)	The system correctly predicts a relation that exists in the gold standard	(Narayana Murthy, FOUNDER_OF, Infosys)
False Positive (FP)	The system predicts a relation that does not match the corresponding relation in the gold standard.	(Infosys, LOCATED_IN, Bengaluru)
False Negative (FN)	A relation exists in the gold standard, but the system fails to predict it correctly.	(Infosys, HQ_IN, Bengaluru)

4.3.3.1 The Evaluation Outcomes

Each relation predicted by the system, and each relation present in the gold standard, can be classified into one of three evaluation categories. Table 4.3.2 illustrates these outcomes using a concrete example.

Test Sentence: "Narayana Murthy founded Infosys, which is headquartered in Bengaluru."

This example is particularly effective because predicting LOCATED_IN instead of the correct HQ_IN creates both an FP for LOCATED_IN and an FN for HQ_IN. This can lead naturally into your next discussion of the double-penalty effect.

For relation extraction, True Negatives (TN) are often not emphasized because the number of possible entity pairs that do not express a relation can be extremely large. Therefore, evaluation commonly focuses on True Positives (TP), False Positives (FP), and False Negatives (FN).

4.3.3.2 Evaluation Metrics

The metrics are the familiar ones introduced in Block III, but here they are computed over relation tuples rather than document labels.

Table 4.3.3: Confusion matrix for relation extraction evaluation

	Gold Relation Exists	No Gold Relation
System Predicts a Relation	True Positive (TP)	False Positive (FP)
System Does Not Predict a Relation	False Negative (FN)	True Negative (TN)

SGOU

1. Precision

Precision measures the correctness of the relations extracted by the system. It answers the question:

Out of all the relations predicted by the system, how many are actually correct?

$$Precision = \frac{TP}{TP + FP}$$

$2 \times (0.750 \times 0.692)/(0.750 + 0.692) \approx 0.720$

where TP denotes True Positives and FP denotes False Positives.

2. Recall

Recall measures how many of the actual relations in the dataset were successfully identified by the system. It answers the question:

Out of all correct relations present in the dataset, how many did the system find?

$$Recall = \frac{TP}{TP + FN}$$

3. F-measure (F1-score)

A system may achieve high precision by extracting only a few highly confident relations, but it may miss many valid relations. Another system may achieve high recall by extracting many relations but may also include many incorrect predictions.

The F1-score combines precision and recall into a single value. It is calculated as the harmonic mean of precision and recall:

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

Example: Suppose a supervised system is evaluated on a test corpus and produces the following counts:

TP = 45, FP = 15, FN = 20

- Precision = $45/(45 + 15) = 45/60 = 0.750$
- Recall = $45/(45 + 20) = 45/65 = 0.692$
- F1 = $2 \times (0.750 \times 0.692)/(0.750 + 0.692) \approx 0.720$

A precision of 0.750 means that three-quarters of the relations predicted by the system were correct. A recall of 0.692 means that the system identified about seven out of every ten relations actually present in the dataset. The F1-score of 0.720 provides a balanced summary of both precision and recall.

SGOU

4.3.4.4 Code Example

```
from sklearn.metrics import classification_report
from sklearn.metrics import precision_recall_fscore_support

# Gold standard
y_true = ['FOUNDER_OF','FOUNDER_OF','FOUNDER_OF','FOUNDER_OF',
          'CEO_OF','CEO_OF','CEO_OF',
          'LOCATED_IN','LOCATED_IN',
          'NO_RELATION','NO_RELATION']

# system predicted
y_pred = ['FOUNDER_OF','FOUNDER_OF','FOUNDER_OF','CEO_OF',
          'CEO_OF','CEO_OF','NO_RELATION',
          'LOCATED_IN','NO_RELATION',
          'NO_RELATION','LOCATED_IN']

# Exclude NO_RELATION from the score
labels = ['FOUNDER_OF','CEO_OF','LOCATED_IN']

print(classification_report(y_true, y_pred,
                            labels=labels, zero_division=0))

mi = precision_recall_fscore_support(y_true, y_pred, labels=labels,
                                     average='micro', zero_division=0)
ma = precision_recall_fscore_support(y_true, y_pred, labels=labels,
                                     average='macro', zero_division=0)
```

Output

```
                precision    recall  f1-score   support

   FOUNDER_OF       1.00      0.75      0.86         4
       CEO_OF       0.67      0.67      0.67         3
   LOCATED_IN       0.50      0.50      0.50         2

    micro avg       0.75      0.67      0.71         9
    macro avg       0.72      0.64      0.67         9
 weighted avg       0.78      0.67      0.71         9

MICRO   P=0.750  R=0.667  F1=0.706
MACRO   P=0.722  R=0.639  F1=0.675
```

4.3.4 Micro and Macro Averaging

Relation extraction commonly involves multiple relation types, so overall system performance must summarize results across all relation classes. This raises a question that the single-relation example above does not address: if a system performs well on some relation types and poorly on others, how should its overall performance be reported?

Two common approaches are micro-averaging and macro-averaging, and they can produce noticeably different results.

4.3.4.1 Micro-Averaging

Micro-averaging combines the true positives, false positives, and false negatives across all relation types and then computes a single precision, recall, and F1-score from these pooled totals.

Because micro-averaging uses the total number of instances, it gives greater weight to

SGOU

frequent relation types. Relations that occur many times contribute more to the final score, while rare relations have less influence.

$$Precision_{micro} = \frac{\sum TP}{\sum TP + \sum FP}$$

$$Recall_{micro} = \frac{\sum TP}{\sum TP + \sum FN}$$

4.3.4.2 Macro-Averaging

Macro-averaging computes precision, recall, and F1 separately for each relation type, and then takes the plain arithmetic mean of those per-relation scores.

Because each relation type contributes equally to the final average regardless of how frequently it occurs, macro-averaging treats all relation types as equally important. Therefore, poor performance on a rare relation can affect the macro-average as much as poor performance on a frequent relation.

$$Precision_{macro} = \frac{1}{N} \sum Precision_i$$

4.3.4.3 Choosing Between Micro- and Macro-Averaging

Neither averaging method is inherently better than the other; they answer different evaluation questions.

Use the micro-average when the focus is on overall performance across all relation instances in the dataset. Frequent relation types therefore have greater influence on the final score.

Use the macro-average when each relation type is considered equally important, regardless of how many examples of that relation occur in the dataset.

It is often useful to report both micro- and macro-averaged scores, particularly when the relation classes are imbalanced.

Two details are worth noting. NO_RELATION is excluded from the label list because it represents the negative class, and including it could inflate the overall score when unrelated entity pairs are much more frequent than actual relations. The evaluation therefore focuses on the predefined relation types.

The macro-average is lower than the micro-average because the poorly performing LOCATED_IN relation has only two gold instances but contributes equally to the macro-average. In this example, each of the three relation types contributes one-third of the macro score, regardless of its frequency.

4.3.5 Evaluating Lightly Supervised Systems

Lightly supervised and unsupervised relation extraction systems can be more difficult to evaluate than fully supervised systems. Although semi-supervised and distant-

SGOU

supervision systems may still be evaluated using labelled test sets, they are often applied to very large text collections where exhaustive gold-standard annotation is unavailable. Open IE presents an additional challenge because it can extract relation phrases that are not drawn from a predefined relation inventory.

Two common evaluation methodologies address these challenges.

4.3.5.1 Held-Out Evaluation

Held-out evaluation is commonly used for distant supervision. A portion of the knowledge base is withheld during training, and the system is then evaluated to determine whether it can rediscover those hidden facts from text.

For example, if 1,000 FOUNDER_OF facts are held back and the system extracts 700 of them, this gives an automatic estimate of how many held-out facts the system can recover without requiring additional human annotation.

A major limitation is that knowledge bases are incomplete. A genuinely correct relation may not appear in the knowledge base and may therefore be counted as an error. For example, a system may correctly extract a fact about a small Kerala startup that has not yet been added to Wikidata.

Thus, held-out evaluation may underestimate precision when the reference knowledge base is incomplete.

4.3.5.2 Manual Evaluation

An alternative approach is to use human judgement. A random sample of the extracted relations is selected, and human annotators examine each relation and determine whether it is correct or incorrect.

The estimated precision is:

$$\hat{P} = \frac{Number\ of\ correct\ tuples\ in\ the\ sample}{Total\ number\ of\ tuples\ in\ the\ sample}$$

This evaluation focuses on the unique relation tuples stored in the final database rather than on individual mentions in the text. For example, the important question is whether the system correctly discovered that IndiGo has a hub at Hyderabad; the number of times this fact was extracted is not important.

4.3.5.3 Why Recall Cannot Be Measured

Manual evaluation can estimate precision, but it cannot reliably measure recall in very large text collections. Recall requires knowing the total number of correct relations present in the corpus.

For a small annotated dataset containing a few thousand sentences, this total is known because human annotators have identified all relevant relations. However, for a collection containing hundreds of millions of web pages, the total number of relations is unknown. Determining it would require manually examining the entire collection.

Therefore, the denominator needed to compute recall is unavailable, making recall

SGOU

Table 4.3.4: Estimated precision at increasing output sizes

Output size	Sample drawn	Judged correct	Estimated precision
Top 1,000	100	94	0.94
Top 10,000	100	81	0.81
Top 100,000	100	62	0.62
Top 1,000,000	100	38	0.38

difficult to measure directly in such large-scale extraction settings.

4.3.5.4 Precision at Different Output Sizes

One technique provides partial information about system coverage without directly measuring recall. If the system ranks its extracted relations by confidence, we can estimate precision at different output sizes by sampling and manually evaluating results at each level.

The shape of the curve is informative even though absolute recall remains unknown. This system is highly reliable for its top thousand extractions but degrades steadily, giving a practitioner the information needed to choose a sensible cutoff.

4.3.6 Common Pitfalls in Evaluation: Key Considerations

When evaluating relation extraction systems, several common mistakes should be avoided:

- **Testing on the training data:** Evaluating a system on the same data used for training produces artificially high results that do not reflect performance on unseen data. Keep the test set separate from training and model development. In addition, accuracy may be misleading when most entity pairs do not express any relation.
- **Ignoring class imbalance:** Rare relation types may have poor performance even when the overall F1-score appears high.
- **Comparing across different datasets:** Datasets may differ in size, relation inventory, annotation scheme, and difficulty. Therefore, performance comparisons are most meaningful when systems are evaluated on the same bench-mark.
- **Ignoring inter - annotator agreement:** Low agreement between annotators may indicate ambiguity in the annotation scheme or difficulty in determining the correct relation labels.
- **Ignoring error analysis:** Examining false positives and false negatives helps identify specific weaknesses in a relation extraction system.
- **Overfitting to the benchmark dataset:** A system that performs well on a single benchmark may generalize poorly to previously unseen domains.

SGOU

Recap

- Evaluation measures the effectiveness and reliability of a relation extraction system.
- Benchmark datasets provide a common basis for training, testing, and comparing relation extraction systems.
- Gold-standard annotations are human-annotated relation labels used as the reference for system evaluation.
- Popular benchmark datasets such as ACE 2005, SemEval-2010 Task 8, TACRED, and DocRED are widely used for evaluating relation extraction models.
- Evaluation outcomes are commonly expressed in terms of True Positives (TP), False Positives (FP), and False Negatives (FN).
- False Positives (FP) are incorrectly predicted relations, whereas False Negatives (FN) are valid relations that the system failed to identify.
- Precision measures the proportion of predicted relations that are correct.
- Recall measures the proportion of actual relations that are successfully extracted.
- F1-score combines precision and recall into a single balanced measure.
- Micro-averaging computes evaluation metrics by pooling relation instances across all relation types, whereas macro-averaging computes the metrics separately for each relation type and then averages them.
- Standard evaluation methodologies enable researchers to compare relation extraction systems fairly using common datasets, evaluation procedures, and performance measures.
- A good benchmark dataset should have high-quality annotations, clearly defined relation types, a documented annotation scheme, appropriate training and test splits, and a representative class distribution.
- Error analysis helps identify the strengths and weaknesses of a relation extraction system by examining its incorrect predictions.

SGOU

Objective Type Questions

1. What type of dataset provides a common standard for evaluating relation extraction systems?
2. Who prepares gold-standard annotations?
3. Which benchmark dataset is commonly used for evaluating semantic relation classification?
4. Which evaluation metric measures the correctness of extracted relations?
5. Which evaluation metric measures the completeness of extracted relations?
6. Which metric combines precision and recall into a single measure?
7. What does TP stand for?
8. What does FP stand for?
9. What does FN stand for?
10. Which dataset is designed for document-level relation extraction?
11. Which relation extraction dataset was developed from the TAC Knowledge Base Population evaluations?
12. Which averaging method gives equal importance to every relation type?
13. Which averaging method evaluates all relation instances together?
14. Which portion of a dataset is used to measure the final performance of a trained model?
15. What type of annotation serves as the reference for system evaluation?

SGOU

Answers to Objective Type Questions

1. Benchmark dataset
2. Human experts or trained annotators
3. SemEval-2010 Task 8
4. Precision

5. Recall
6. F1-score
7. True Positive
8. False Positive
9. False Negative
10. DocRED
11. TACRED
12. Macro-averaging
13. Micro-averaging
14. Test set
15. Gold standard annotation

Assignments

1. Explain the importance of benchmark datasets in evaluating relation extraction systems. Discuss the characteristics of a good benchmark dataset with suitable examples.

2. Consider the following sentence: "Elon Musk founded SpaceX in 2002."

   Assume the gold-standard relation is: (Elon Musk, FOUNDER_OF, SpaceX)

   A relation extraction system predicts: (Elon\ Musk,\ FOUNDER_OF,\ SpaceX)

   Identify the evaluation outcome (TP, FP, or FN) and explain your answer.

3. Consider the following sentence: "Marie Curie discovered the radioactive elements polonium and radium."

   Assume the gold-standard relation is: (Marie Curie, DISCOVERED, Radium). A relation extraction system produces no relation for Marie Curie and Radium.

   Identify the evaluation outcome (TP, FP, or FN) and explain your answer.

4. A relation extraction system produced the following results:

   TP = 45, FP = 15, FN = 10

   Calculate Precision, Recall, and F1-score. Interpret the results in two or three sentences.

SGOU

5. Compare Micro-averaging and Macro-averaging. In what situations is each method more appropriate for evaluating relation extraction systems?

6. Choose any news article or Wikipedia article containing at least five entity pairs. Manually identify the expected relations in the text and treat them as the reference relations. Compare these with the relations extracted by a system. Classify the results as True Positives (TP), False Positives (FP), or False Negatives (FN) and briefly discuss the system's performance.

Reference

1. Jurafsky, D., & Martin, J. H. Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition.

2. Stanford Machine Learning website, http://www.holehouse.org/mlclass/index.html

3. Vajjala, S., Majumder, B., Gupta, A., & Surana, H. (2020). *Practical natural language processing: a comprehensive guide to building real-world NLP systems*. O'Reilly Media.

4. Diaz-Garcia, J. A., & Lopez, J. A. D. (2025). A survey on cutting-edge relation extraction techniques based on language models. *Artificial Intelligence Review*, *58*(9), 287.

5. Luan, Y., He, L., Ostendorf, M., & Hajishirzi, H. (2018). Multi-task identification of entities, relations, and coreference for scientific knowledge graph construction. In *Proceedings of the 2018 conference on empirical methods in natural language processing* (pp. 3219-3232).

6. Stoica, G., Platanios, E. A., & Póczos, B. (2021, May). Re-traced: Addressing shortcomings of the tracked dataset. In *Proceedings of the AAAI conference on artificial intelligence* (Vol. 35, No. 15, pp. 13843-13850).

7. https://nlp.stanford.edu/projects/tacred/

8. Hendrickx, I., Kim, S. N., Kozareva, Z., Nakov, P., Séaghdha, D. O., Padó, S., ... & Szpakowicz, S. (2010, July). Semeval-2010 task 8: Multi-way classification of semantic relations between pairs of nominals. In *Proceedings of the 5th international workshop on semantic evaluation* (pp. 33-38).

SGOU

Suggested Reading

1. Sarawagi, S. (2008). Information extraction. *Foundations and trends in databases*, *1*(3), 261-377.

2. Manning, C. D., Raghavan, P., & Schütze, H. (2008). Introduction to Information Retrieval. Cambridge University Press.

SGOU

Unit 4
Information Retrieval

Learning Outcomes

After completing this unit, learner will able to:

- explain the concept of Information Retrieval (IR) and describe the architecture of an ad hoc retrieval system
- define the fundamental terms of information retrieval: document, collection, term, and query
- compute Term Frequency (TF) and Inverse Document Frequency (IDF) and explain the intuition behind each
- calculate TF-IDF weights and rank documents for a query using cosine similarity
- explain the BM25 weighting scheme and describe how its parameters control term-frequency saturation and document-length normalization
- describe the structure of an inverted index and explain how it enables efficient document retrieval
- evaluate a ranked information retrieval system using precision, recall, interpolated precision, Mean Average Precision (MAP), and Mean Reciprocal Rank (MRR)

SGOU

Prerequisites

This unit assumes a basic understanding of text documents, keywords, queries, and digital document collections. Learners should be familiar with the general idea of using a search engine to locate relevant information from a large collection of documents.

An Information Retrieval (IR) system connects users seeking information with collections of stored documents. Users express their information needs through keywords or queries, and the system identifies and ranks documents that are likely to be relevant.

A basic understanding of Python and elementary mathematical operations will also help learners follow the numerical examples and code presented in this unit.

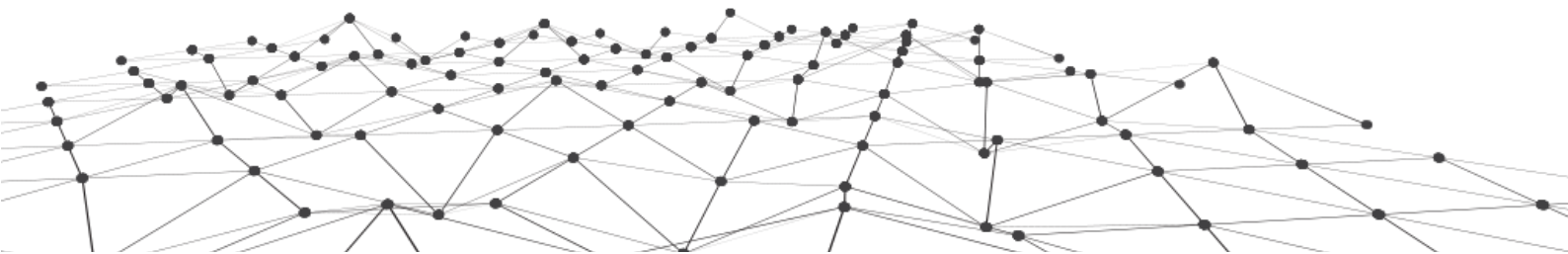




Fig 4.4.1: Basic Architecture of an Information Retrieval system

Figure 4.4.1 illustrates the basic structure of an Information Retrieval (IR) system. The central component, represented as a black box, performs the processing required to match a user's query with documents in the collection and return a ranked list of relevant documents. The components within this IR black box are explored in detail throughout this unit.

This unit also builds upon concepts introduced in earlier blocks. In Block II, you learned about the Vector Space Model, in which documents are represented as numerical vectors based on term occurrences, and about TF-IDF (Term Frequency–Inverse Document Frequency) weighting. In this unit, TF-IDF is revisited from an information retrieval perspective, where it measures term importance and helps rank documents with respect to a user's query. You will also revisit cosine similarity, which measures the similarity between query and document vectors. In addition, preprocessing techniques introduced in Block II, including tokenization, normalization, and stop-word removal, continue to play an important role in preparing text for retrieval.

In Block III, you studied the evaluation measures precision, recall, and F1-score. These metrics are revisited in this unit to evaluate the effectiveness of information retrieval systems. Unlike classification tasks, retrieval systems must not only identify relevant documents but also rank them according to their relevance to the user's query.

No prior knowledge of search engines or information retrieval systems is assumed. However, a basic understanding of Python programming will help you follow the practical examples, while familiarity with logarithms will help you understand weighting schemes such as TF-IDF.

SGOU

Keywords

Ad Hoc Retrieval, Document, Collection, Term, Query, Term Weight, Term Frequency (TF), Document Frequency (DF), Inverse Document Frequency (IDF), TF-IDF, BM25, Cosine Similarity, Vector Space Model, Stop List, Inverted Index, Dictionary, Precision, Recall, F1-Score, Interpolated Precision, Mean Average Precision (MAP), Mean Reciprocal Rank (MRR)

Discussion

4.4.1 Introduction

When a user enters a query into a search box, the system searches a very large collection of documents. It quickly returns a small set of relevant results, usually ranking the most relevant documents at the top. Although people use this process regularly, several important operations take place behind the scenes.

This unit examines how Information Retrieval (IR) systems represent, search, score, rank, and evaluate documents in response to a user's query.

Information Retrieval (IR) is the field concerned with finding relevant information from large collections of documents in response to a user's information need, as illustrated in Figure 4.4.2. Systems that perform this task are commonly known as search engines. IR is one of the oldest and most important areas of language technology. In traditional IR systems, both queries and documents are represented using terms and their associated weights, allowing the system to estimate how well each document matches the query.

The main focus of this unit is ad hoc retrieval. In ad hoc retrieval, a user expresses an information need as a query, and the system returns a ranked list of documents from a collection according to their estimated relevance.

The term ad hoc refers to the fact that the system must respond to arbitrary user queries rather than a fixed set of predefined information needs.

Since the system must determine which documents are most relevant to each query, the retrieved results may include both relevant and non-relevant documents. In the following sections, we examine methods for measuring how effectively the system identifies relevant documents and ranks them above less relevant ones. We also introduce four fundamental terms that will be used throughout this unit.

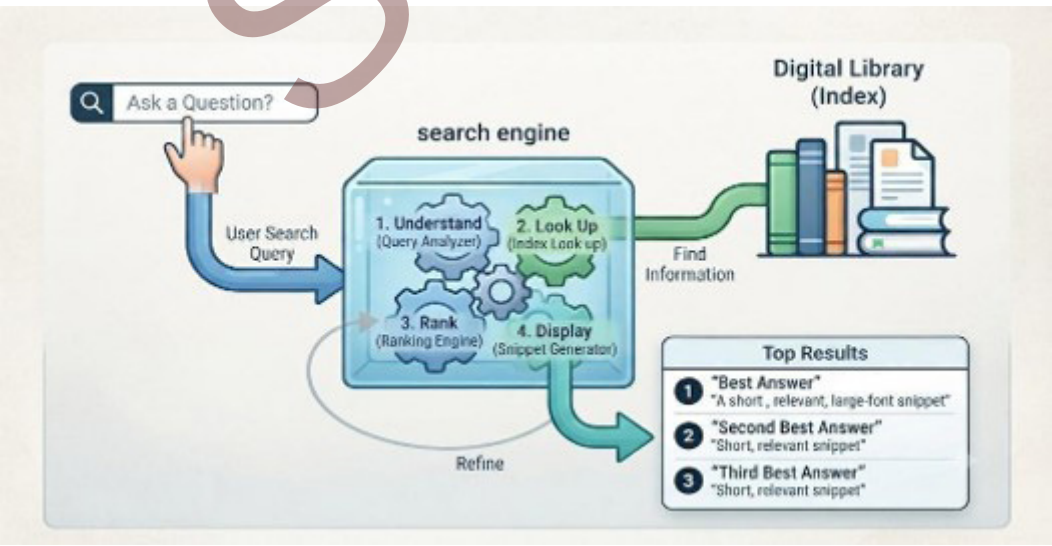


Fig 4.4.2: Basic workflow of an Information Retrieval system

Table 4.4.1: Four basic terms used in Information Retrieval

Term	Meaning	Example
Document	The unit of text that the system indexes and retrieves	A web page, news article, or single paragraph
Collection	The set of documents being searched	All articles on a news website
Term	A word or token used to represent the content of documents and queries	kerala, tourism, backwaters
Query	A user's information need expressed as one or more search terms	“kerala tourism”

4.4.1.1 The Architecture of a Retrieval System

An Information Retrieval system generally performs its work in two main phases. Understanding this separation helps explain why search can be performed quickly.

- The first phase takes place before a user submits a query. Documents in the collection are processed, broken into terms, and stored in a data structure called an inverted index. Building the index can be computationally expensive, but it is performed offline and can be updated when the document collection changes.
- The second phase begins when a query is submitted. The query is processed, and the inverted index is consulted to identify documents containing the query terms. These candidate documents are then **scored and ranked** according to their estimated relevance. Since the index has already been constructed, retrieval can be performed efficiently.

The basic IR architecture shown in Figure 4.4.4 uses the Vector Space Model introduced in Block II. In this model, queries and documents are represented as vectors based on term weights, and cosine similarity can be used to rank documents according to their similarity to the query. This is a form of the bag-of-words model, in which terms are represented largely independently of their positions in the text.

Note: **Let’s Recollect,**

Cosine Similarity: A measure that determines the similarity between two vector representations by calculating the cosine of the angle between them.

$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}},$$

SGOU

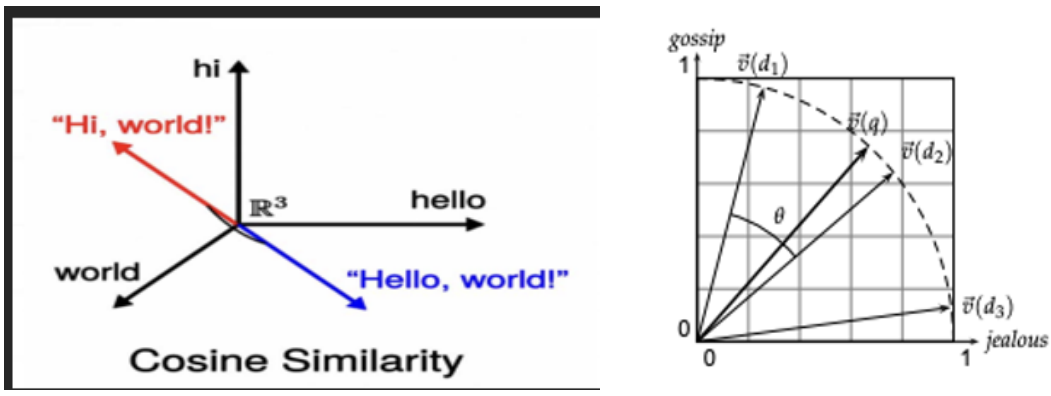


Fig 4.4.3: Illustration of cosine similarity between query and document vectors in the Vector Space Model

Figure 4.4.3 illustrates cosine similarity, where the angle θ between the query and document vectors indicates their similarity. A smaller angle corresponds to a higher cosine similarity and therefore a stronger match between the query and the document.

- The similarity between a query and a document can be measured by first converting them into vector representations.
- These vectors can then be compared using the cosine similarity formula.
- For non-negative term-weight vectors, a cosine similarity close to 1 indicates that the query and document vectors have similar directions, where-as a value of 0 indicates that they share no weighted terms.

Figure 4.4.4 illustrates the architecture of an Information Retrieval (IR) system. The user expresses an information need through a query, while the document collection is processed separately through indexing. The IR system then matches the query against the indexed documents and returns the most relevant results. A feedback mechanism may also be used to refine subsequent retrieval results.

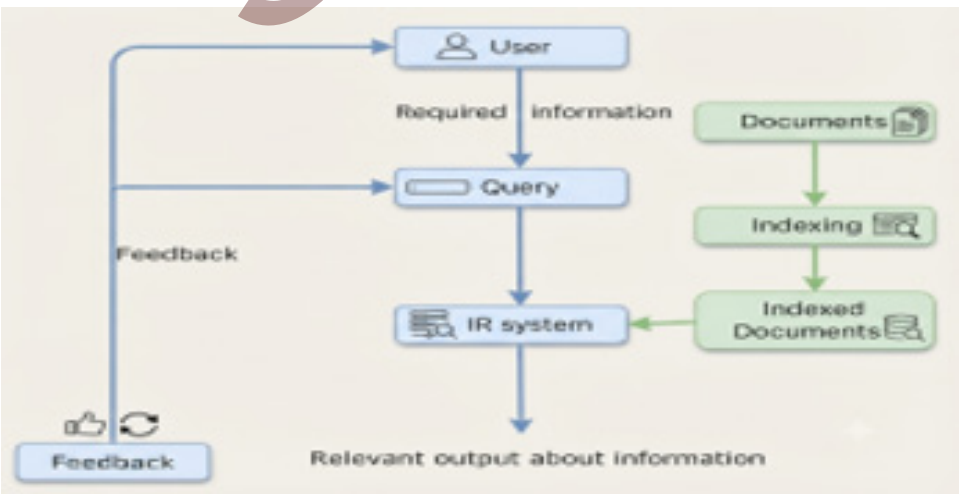


Fig 4.4.4: Architecture of an Information Retrieval System

SGOU

4.4.2 Term Frequency and Weighting

Before measuring how well a document matches a query, it is necessary to determine the importance of each term within the document. This process is known as term weighting.

In information retrieval systems, both documents and queries are represented using terms. However, not all terms contribute equally to determining relevance. A term that appears several times in a document may provide stronger evidence about the document's content than a term that appears only once. Therefore, information retrieval systems assign weights to terms rather than treating all terms as equally important.

The simplest approach is to use Term Frequency (TF), which counts how many times a term appears in a document. However, raw term frequency has two important limitations.

First, a term that appears one hundred times is not necessarily one hundred times more important than a term that appears once.

Second, a term that appears in almost every document is not very useful for distinguishing relevant documents from non-relevant ones.

The architecture of an ad hoc information retrieval system is illustrated with an example in Figure 4.4.5.

Information Retrieval therefore uses weighting schemes that consider both how frequently a term occurs within a document and how widely it occurs across the document collection. One of the most widely used classical approaches is TF-IDF (Term Frequency–Inverse Document Frequency).

The following sections describe these components in detail.

4.4.2.1 Term Frequency

Term Frequency (TF) indicates how often a term occurs in a document. A term that

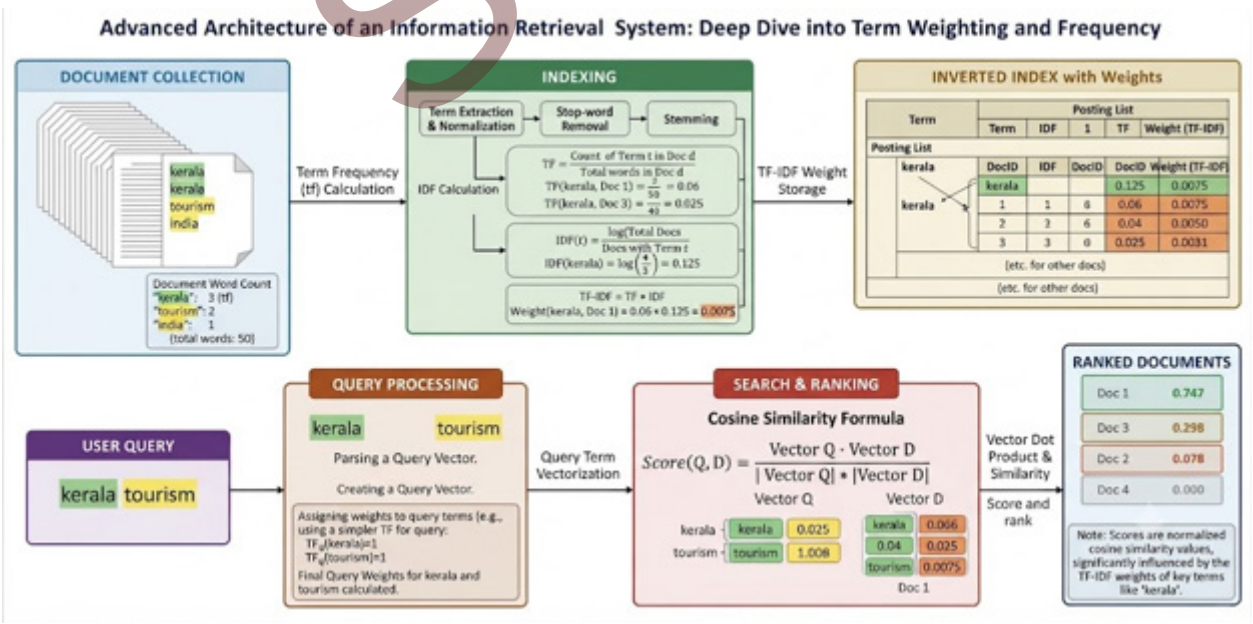


Fig 4.4.5: Architecture of an ad hoc Information Retrieval system

SGOU

occurs more frequently in a document may provide stronger evidence about the document's content. However, instead of using the raw count directly, a logarithmic transformation can be applied. This is because a term appearing 100 times should not necessarily receive 100 times the weight of a term appearing once.

Since the logarithm of zero is undefined, a zero count is handled separately.

The term frequency of term t in document d is therefore defined as:

$$tf(t,d) = \begin{cases} 1 + \log_{10}(\text{count}(t,d)), & \text{if count}(t,d) > 0 \\ 0, & \text{otherwise} \end{cases} \quad \textbf{(1)}$$

For example, the effect of logarithmic scaling can be seen clearly. A term occurring once has a TF weight of 1. A term occurring 10 times has a weight of 2, 100 occurrences produce a weight of 3, and 1,000 occurrences produce a weight of 4. Thus, each tenfold increase in the raw term count increases the TF weight by only one.

raw count = 0, TF weight = 0,

Table 4.4.2: Logarithmic damping of term frequency

Raw count	Calculation	TF Weight
1	$1 + \log_{10}(1) = 1 + 0$	1
10	$1 + \log_{10}(10) = 1 + 1$	2
100	$1 + \log_{10}(100) = 1 + 2$	3
1000	$1 + \log_{10}(1000) = 1 + 3$	4

—each tenfold increase in raw count increases the TF weight by only one

4.4.2.2 Inverse Document Frequency (IDF)

Term frequency alone is not sufficient. Consider searching a collection of Kerala tourism articles for the query *"kerala backwaters."* The term **"**kerala**"** may appear in almost every document in the collection, so its presence provides little information for distinguishing relevant documents. In contrast, the term "backwaters" may occur in only a smaller number of documents and is therefore more useful for discriminating between them.

The **document frequency** of a term, written $df(t)$, is the number of documents in the collection that contain the term. Terms that occur in only a few documents are more useful for identifying specific documents, whereas terms that occur in almost every document provide less discriminative information.

SGOU

Table 4.4.3: Example IDF values and their interpretation for terms in a corpus of 37 Shakespeare plays.

Word	DF (out of 37 plays)	IDF	Interpretation
Romeo	1	1.57	Appears in one play only - highly distinctive
salad	2	1.27	Rare - very informative
Falstaff	4	0.967	Uncommon - informative
forest	12	0.489	Moderately common
battle	21	0.246	Common - weakly informative
fool	36	0.012	Nearly universal – very low discriminative value
good	37	0	Occurs in every play – no discriminative value
sweet	37	0	Occurs in every play – no discriminative value

The Inverse Document Frequency (IDF) is defined as:

$$idf(t) = \log_{10}\left(\frac{N}{df(t)}\right) \quad \textbf{(2)}$$

where N is the total number of documents in the collection.

The fewer documents a term occurs in, the higher its IDF weight. A term that occurs in every document receives the minimum IDF value of zero because:

$$\log_{10}\left(\frac{N}{N}\right) = \log_{10}(1) = 0$$

4.4.2.3 TF-IDF

The TF-IDF weight of a term in a document is the product of its term frequency and inverse document frequency:

$$tf - idf(t,d) = tf(t,d) \times idf(t) \quad \textbf{(3)}$$

This multiplication combines the two components effectively. A term receives a high TF-IDF weight when it occurs frequently in a particular document but is relatively rare across the document collection. If either condition is not satisfied, the resulting weight will be lower.

A term that appears in every document receives a TF-IDF weight of zero, regardless of how often it occurs in a particular document, because its IDF value is zero.

SGOU

4.4.2.4 BM25

BM25 (Best Matching 25), also known as Okapi BM25, is a widely used ranking function in information retrieval. It extends basic term-weighting ideas by introducing term-frequency saturation and document-length normalization.

The BM25 score of a document d for a query q can be written as:

$$\text{Score}(d,q) = \sum_{t \in q} idf(t) \frac{tf(t,d)(k_1+1)}{tf(t,d) + k_1 \left(1 - b + b\frac{|d|}{avgdl}\right)} \quad \textbf{(4)}$$

where:

- $tf(d)$ is the frequency of term t in document d
- $|d|$ is the length of document d
- $avgdl$ is the average document length in the collection
- k_1 controls term-frequency saturation
- b controls document-length normalization

The two parameters perform different functions:

- k_1: Controls how strongly additional occurrences of a term increase the score. A small k_1 causes the term-frequency contribution to saturate quickly, while a larger k_1 gives greater importance to repeated occurrences.
- b: Controls document-length normalization. When $b = 0$, no document-length normalization is applied. When $b = 1$, full length normalization is applied.

A common simplified IDF component is:

$$idf(t) = \log\left(\frac{N}{df(t)}\right)$$

Document length normalisation matters because a very long document will naturally contain Document length normalization is important because longer documents naturally tend to contain more occurrences of a given term. Without normalization, long documents may receive higher scores simply because of their length. In the BM25 formula, $avgdl$ represents the average document length in the collection.

[1] suggests reasonable values of k_1 between 1.2 and 2.0, with $b = 0.75$.

Common implementations such as Lucene and Elasticsearch use $k_1 = 1.2$ and $b = 0.75$ as default values.

SGOU

4.4.2.5 Stop Words

In earlier information retrieval systems, it was common to remove very frequent words from both queries and documents before indexing and representation. The list of such words is called a **stop list**. The intuition is that high-frequency function words such as *"the," "a,"* and *"to"* usually have low discriminative value for retrieval and can increase the size of the inverted index.

For example, depending on the stop list used, a phrase such as *"to be or not to be"* may be reduced to only the remaining non-stop words.

In modern IR systems, the use of stop lists is less common. This is partly due to improvements in storage and processing efficiency, and partly because IDF weighting already reduces the influence of very common terms by assigning them low weights. Nevertheless, stop-word removal is still used in some retrieval and NLP applications.

4.4.3 Document Scoring

Having determined how each term is weighted, we can now measure how well a document matches a query. Both the query and the document are represented as vectors over the vocabulary, and their similarity is measured using the cosine of the angle between the two vectors.

Cosine similarity(q, d)

$$= \frac{q \cdot d}{\|q\|\|d\|} = \frac{\sum_{i=1}^{n} q_i d_i}{\sqrt{\sum_{i=1}^{n} q_i^2}\sqrt{\sum_{i=1}^{n} d_i^2}} \quad \textbf{(5)}$$

Another way to understand cosine similarity is as the dot product of two-unit vectors. The query and document vectors are first normalized to unit length by dividing each vector by its magnitude, and their dot product is then calculated.

This normalization reduces the influence of document length, preventing longer documents from receiving higher similarity scores simply because they contain more terms.

4.4.3.1 Worked Example

Consider a small collection of four documents about Kerala tourism and the query "kerala tourism." All text is converted to lowercase and punctuation is removed.

Document Collection

1. Kerala Kerala backwaters tourism
2. Kerala mountains
3. How beautiful is Kerala tourism
4. Backwaters

SGOU

Table 4.4.4: Document frequencies and IDF values for the example collection

Term	Appears in	DF	IDF = $\log_{10}(4/df)$
kerala	Doc 1, 2, 3	3	log(4/3) = 0.125
tourism	Doc 1, 3	2	log(4/2) = 0.301
backwaters	Doc 1, 4	2	log(4/2) = 0.301
mountains	Doc 2	1	log(4/1) = 0.602
how / beautiful / is	Doc 3	1	log(4/1) = 0.602

Query: "Kerala tourism"

Here, $N = 4$, since the collection contains four documents.

Steps

1. Compute the IDF of each term.
2. Build the query vector.
3. Score each document using cosine similarity.

The document frequencies and IDF values are given in table 4.4.4.

Step 1: Compute the IDF of Each Term

Notice that "kerala" has a low IDF value of 0.125 because it appears in three of the four documents. It therefore provides relatively little discriminative information. The term "tourism," which appears in only two documents, receives a higher IDF value of 0.301.

Step 2: Build the Query Vector

The query "kerala tourism" contains each term once, so $tf = 1$for both terms.

Therefore, the TF-IDF weights are:

- Kerala = 0.125
- Tourism = 0.301

The query-vector length is: $| q |= \sqrt{0.125^2 + 0.301^2} = \sqrt{0.1062} \approx 0.326$

Step 3: Score Each Document

Document 1 is:

"kerala kerala backwaters tourism"

SGOU

Here, kerala occurs twice:

$$tf = 1 + log_{10}(2) = 1.301$$

Therefore:

$$TF - IDF(kerala) = 1.301 \times 0.125 \approx 0.163$$

The terms backwaters and tourism each occur once, so their TF-IDF weights are 0.301.

The document-vector length is:

$$| d_1 | = \sqrt{0.163^2 + 0.301^2 + 0.301^2} \approx 0.456$$

The cosine similarity between Document 1 and the query is approximately:

0.747

Repeating the calculation for all four documents given in table 4.4.5

The ranking is reasonable. Document 1 receives the highest score because it contains both query terms and gives additional weight to Kerala through repeated occurrence. Document 3 also contains both query terms, but its additional weighted terms increase the document-vector length and reduce the cosine similarity after normalization. Document 2 contains only one query term, while Document 4 contains none and therefore receives a score of zero.

4.4.4 The Inverted Index

To compute document scores efficiently, it is necessary to identify the documents that contain the query terms. Documents that contain none of the query terms receive a score of zero and need not be considered. A basic search operation in information retrieval is

Table 4.4.5: Final ranking of the four documents for the query "kerala tourism."

Rank	Document	Text	Cosine Score	Why
1	Doc 1	kerala kerala back-waters tourism	0.747	Contains both query terms; Kerala occurs twice
2	Doc 3	how beautiful is kerala tourism	0.298	Contains both query terms, but also several additional weighted terms
3	Doc 2	kerala mountains	0.078	Contains only one query term
4	Doc 4	backwaters	0.000	Contains no query terms

SGOU

therefore to locate all documents in the collection that contain a specified query term.

The inverted index is the primary data structure used for this purpose. It enables efficient searching and can store information such as the document frequency of each term and the term count within each document. Given a query term, the inverted index returns the list of documents containing that term.

The index is called **"inverted"** because it maps terms to the documents in which they occur, rather than mapping each document to the terms it contains.

An inverted index consists of two main components: a dictionary and postings list.

The dictionary is an efficiently searchable list of terms, with each term linked to its corresponding postings list. It may also store the document frequency of each term, which is required for calculating Inverse Document Frequency (IDF).

A **postings list** contains the document identifiers associated with a particular term. It may also store additional information, such as the term frequency within each document or the positions at which the term occurs, which can be useful for phrase searching.

Example

The inverted index for the four-document collection is shown below. Each dictionary entry contains its document frequency in braces, followed by a postings list containing document IDs and term counts in square brackets.

backwaters {2} → Doc 1 [1] → Doc 4 [1]

beautiful {1} → Doc 3 [1]

how {1} → Doc 3 [1]

is {1} → Doc 3 [1]

kerala {3} → Doc 1 [2] → Doc 2 [1] → Doc 3 [1]

mountains {1} → Doc 2 [1]

tourism {2} → Doc 1 [1] → Doc 3 [1]

The number in braces after each term represents its document frequency, that is, the number of documents in which the term occurs. The number in square brackets after each document ID represents the term count, indicating how many times the term occurs in that document.

The entry for *kerala* illustrates the structure clearly. Its document frequency is 3, meaning that it occurs in three of the four documents. The postings list identifies these documents as Doc 1 with a count of 2, Doc 2 with a count of 1, and Doc 3 with a count of 1.

This entry provides the information required for scoring the term *kerala*. Its document frequency gives the IDF value:

SGOU

$$log_{10}(4/3) = 0.125$$

The count of 2 in Doc 1 gives the term-frequency weight:

$$1 + log_{10}(2) = 1.301$$

The absence of Doc 4 from the postings list indicates that Doc 4 does not contain the term *kerala* and therefore does not need to be considered when processing that particular query term.

Search Fast?

Given a query, the system looks up each query term in the dictionary and retrieves its corresponding postings list. The union of these postings lists gives the set of candidate documents containing at least one query term. Documents outside this set can be ignored because they will receive a score of zero.

For the query "kerala tourism," the postings list for *kerala* is {3}, while the postings list for *tourism* is {3}. Their union is:

$$\{3\} \cup \{3\} = \{3\}$$

Therefore, only Documents 1, 2, and 3 need to be scored. Document 4 does not contain either query term and can be skipped.

For very large document collections, this ability to examine only a small set of candidate documents greatly improves retrieval efficiency and enables fast search responses.

The use of union here is correct because your scoring system considers documents containing at least one query term.

4.4.5 Evaluation of Information Retrieval Systems

How can we determine whether an Information Retrieval system performs well? This question is more complex than it may initially appear because an IR system does not return a single answer; instead, it produces a ranked list of documents. The position of a relevant document in this ranking is important. A relevant document appearing at rank 1 is generally more useful to the user than the same document appearing at rank 50.

4.4.5.1 Precision and Recall

We begin with the same two evaluation metrics introduced for classification in Block III, assuming that each retrieved document is either relevant or not relevant to the user's information need.

Suppose the system returns a set T of documents for a query. Let R be the subset of retrieved documents that are relevant. Let U be the set of all relevant documents in the entire collection.

Then:

$$Recall = \frac{|R|}{|U|}$$

SGOU

Precision is the fraction of retrieved documents that are relevant.

Recall is the fraction of all relevant documents in the collection that were successfully retrieved.

However, when these metrics are computed over the entire retrieved set, they do not take the ranking order into account. Two systems that return the same documents in different orders would obtain identical precision and recall values, even if one places the relevant documents near the top and the other places them near the bottom.

Therefore, additional evaluation measures are needed to account for the rank positions of relevant documents.

4.4.5.2 Rank-Specific Precision and Recall

To evaluate a ranked retrieval system, precision and recall can be computed at each position in the ranked list.

At a given rank:

- **Precision at rank** k is the proportion of the top k retrieved documents that are relevant.
- **Recall at rank** k is the proportion of all relevant documents in the collection that have been retrieved within the top k results.

This allows us to observe how retrieval quality changes as we move down the ranked list and gives more information than computing precision and recall only over the complete set of retrieved documents.

4.4.5.3 The Precision-Recall Curve

A common way to visualize the performance of a ranked retrieval system is to plot precision against recall, producing a precision–recall curve. As we move down the ranked list, recall generally increases as more relevant documents are retrieved, while precision may rise or fall depending on whether the newly retrieved documents are relevant.

Figure 4.4.6 shows both the raw precision–recall curve and its interpolated form.

4.4.5.4 Interpolated Precision

The raw precision–recall curve can be difficult to compare across queries because it is often irregular, and different queries may produce precision values at different recall levels.

A classical approach is to compute interpolated precision at eleven fixed recall levels: 0.0, 0.1, 0.2, …, 1.0. Since an observed precision value may not occur exactly at each of these recall levels, the interpolated precision at recall level r is defined as the maximum precision obtained at any recall level r' greater than or equal to r:

$$P_{interp}(r) = \max_{r' \geq r} P(r')$$

SGOU

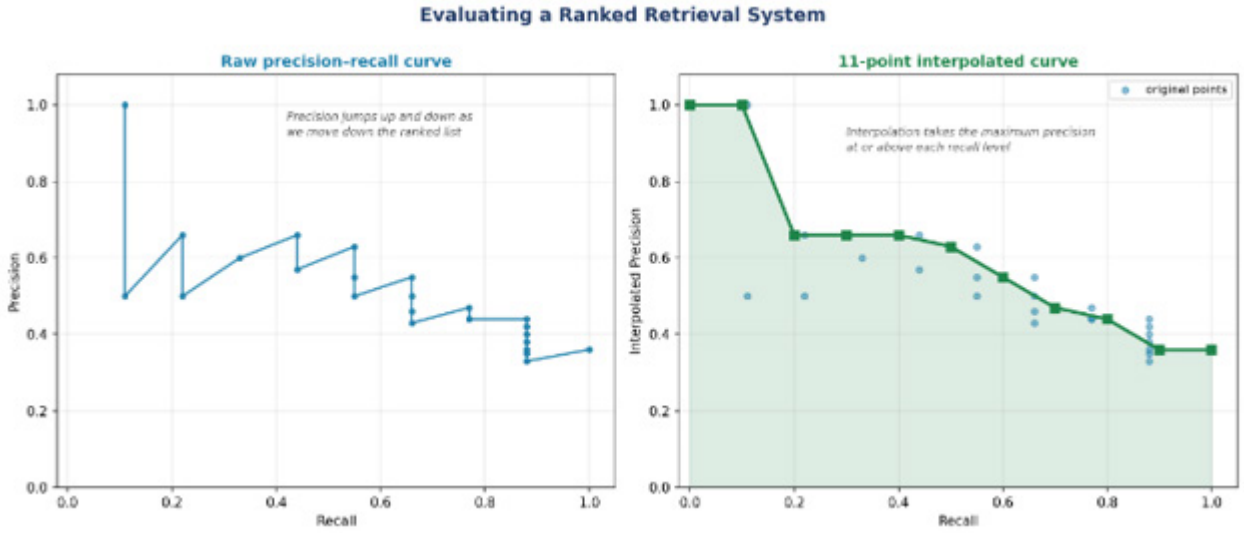


Fig 4.4.6: Raw and 11-point interpolated precision–recall curves for a ranked Information Retrieval system

This method serves two purposes. First, it allows performance across different queries to be compared and averaged at the same eleven recall levels. Second, it smooths the irregularities in the original precision–recall curve by replacing each precision value with the best precision observed at that recall level or any higher recall level.

4.4.5.5 Mean Average Precision (MAP)

A precision–recall curve provides useful information, but it does not summarize retrieval performance with a single value. Mean Average Precision (MAP) provides such a summary.

For a single query, we move down the ranked list and record the precision only at ranks where a relevant document is retrieved. For example, if relevant documents occur at ranks 1, 3, 5, and 6, precision is recorded at those four ranks, but not at ranks 2 or 4.

The Average Precision (AP) for a query is the mean of these precision values:

$$AP = \frac{1}{m}\sum_{k=1}^{m} P(R_k)$$

where:

- m is the number of relevant documents for the query.
- R_k is the rank at which the k^{th} relevant document is retrieved.
- $P(R_k)$ is the precision calculated at that rank.

For a set of queries Q, Mean Average Precision (MAP) is the average of the AP values:

SGOU

$$MAP(Q) = \frac{1}{|Q|}\sum_{j=1}^{|Q|}\frac{1}{m_j}\sum_{k=1}^{m_j}P(R_{jk})$$

4.4.5.6 Mean Reciprocal Rank (MRR)

In some retrieval tasks, particularly question answering, the user is primarily interested in finding the first correct answer as quickly as possible. Mean Reciprocal Rank (MRR) is designed to evaluate this behaviour.

For each query, the reciprocal rank is calculated as the reciprocal of the rank at which the first correct answer appears. For example, if the first correct answer appears at rank 4, the reciprocal rank is:

$$\frac{1}{4} = 0.25$$

If the first correct answer appears at rank 1, the reciprocal rank is 1.0. If no correct answer is retrieved, the reciprocal rank is 0.

For a set of queries Q, MRR is calculated as:

$$MRR = \frac{1}{|Q|}\sum_{i=1}^{|Q|}\frac{1}{rank_i}$$

where $rank_i$ is the rank of the first relevant or correct result for query i.

MRR rewards systems that place the first correct result at or near the top of the ranked list. Results appearing below the first correct answer do not affect the MRR score.

4.4.5.7 Choosing a Metric - Differentiate

Table 4.4.6 Evalation metric

Metric	What it measures	Best used when
Precision	Proportion of retrieved documents that are relevant	False positives are costly
Recall	Proportion of all relevant documents that are successfully retrieved	Missing a relevant document is costly
Interpolated precision	Smoothed precision at fixed recall levels	Comparing retrieval performance across multiple queries
MAP	Overall ranking quality across a set of queries	Comparing competing ranked retrieval systems
MRR	Rank position of the first relevant or correct result	The user primarily needs one correct answer

SGOU

Recap

- Information Retrieval (IR) finds and ranks documents that are relevant to a user's information need.
- In ad hoc retrieval, a user submits a query and the system returns a ranked list of documents from a collection.
- An IR system commonly performs document indexing before a query arrives and performs retrieval, scoring, and ranking when the query is submitted.
- A term is a word or other searchable unit used to represent documents and queries.
- Term Frequency (TF) measures how often a term occurs in a document.
- Logarithmic TF reduces the influence of very high term frequencies.
- Document Frequency (DF) is the number of documents in the collection that contain a particular term.
- Inverse Document Frequency (IDF) assigns higher weights to terms that occur in fewer documents and lower weights to terms that occur in many documents.
- TF-IDF combines term frequency and inverse document frequency to assign weights to terms.
- A term receives a high TF-IDF weight when it occurs frequently in a particular document but occurs in relatively few documents across the collection.
- BM25 is a probabilistic ranking function that considers term frequency, inverse document frequency, and document length.
- In BM25, the parameter k_1 controls term-frequency saturation, while b controls document-length normalization.
- Cosine similarity measures the similarity between query and document vectors in the Vector Space Model.
- An inverted index maps terms to the documents in which they occur and enables efficient identification of candidate documents.
- Precision measures the proportion of retrieved documents that are relevant, while recall measures the proportion of all relevant documents that are retrieved.
- Interpolated precision summarizes precision at fixed recall levels and helps compare retrieval performance across queries.

SGOU

- Mean Average Precision (MAP) summarizes ranking quality across a set of queries by averaging the Average Precision values of individual queries.
- Mean Reciprocal Rank (MRR) measures how highly the first relevant or correct result is ranked.

Objective Type Questions

1. What does Term Frequency (TF) measure?
2. What does Document Frequency (DF) measure?
3. What does Inverse Document Frequency (IDF) measure?
4. Using logarithmic TF, what is the TF weight of a term that occurs 100 times in a document?
5. Which TF weighting method reduces the influence of very frequent terms?
6. What two components are combined in TF-IDF?
7. Which terms receive higher IDF values?
8. Which terms receive lower IDF values?
9. What does BM25 rank in response to a query?
10. What does the parameter k_1 control in BM25?
11. What does the parameter b control in BM25?
12. What does cosine similarity compare in Information Retrieval?
13. How is a document represented in the Vector Space Model?
14. Which data structure maps terms to the documents in which they occur?
15. What does precision measure in Information Retrieval?
16. What does recall measure in Information Retrieval?

SGOU

Answers to Objective Type Questions

1. Frequency of a term within a document
2. Number of documents in the collection containing the term
3. How rare or discriminative a term is across the document collection
4. $tf = 1 + \log_{10}(100) = 1 + 2 = 3$.
5. Logarithmic TF weighting
6. Term Frequency (TF) and Inverse Document Frequency (IDF)
7. Terms that occur in fewer documents
8. Terms that occur in many documents
9. Documents according to their relevance to a query
10. Term-frequency saturation
11. Document-length normalization
12. The query vector and document vector
13. As a numerical vector of term weights
14. Inverted index
15. The proportion of retrieved documents that are relevant
16. The proportion of all relevant documents that are successfully retrieved

SGOU

Assignments

1. **Calculate TF-IDF:**

A collection contains 100 documents. A term occurs 5 times in document D1 and appears in 20 documents in the collection. Calculate its TF-IDF weight using:

$$tf(d) = 1 + log_{10}(count(d)) idf(t) = log_{10}\left(\frac{N}{df(t)}\right) tf - idf(d) = tf(d) \times idf(t)$$

2. Construct an inverted index for the following documents:

- D1: “information retrieval”
- D2: “information search”
- D3: “retrieval systems”

For each term, show:

- the document frequency (DF),
- The postings list, and
- the term frequency (TF) in each document.

Explain how the inverted index helps retrieve documents for the query “information retrieval.”

3. A term occurs 8 times in document D1 and appears in 20 of a collection of 200 documents. Calculate:

   1. Logarithmic TF.

   2. IDF.

   3. TF-IDF.

   Interpret the result. Briefly interpret the resulting TF-IDF value.

4. Explain the difference between Term Frequency (TF), Document Frequency (DF), and Inverse Document Frequency (IDF), with a suitable example.

5. Explain how BM25 ranks documents and describe the roles of term-frequency saturation and document-length normalization.

6. Compare Mean Average Precision (MAP) and Mean Reciprocal Rank (MRR) as evaluation metrics. Give one retrieval task for which each metric is more appropriate and justify your choice.

SGOU

Reference

1. C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. Cambridge University Press, 2008.

2. I. C. Mogotsi, “Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze: *Introduction to Information Retrieval*, Cambridge University Press, Cambridge, England, 2008, 482 pp., ISBN 978-0-521-86571-5,” 2010.

3. D. Jurafsky and J. H. Martin, *Speech and Language Processing*, 3rd ed. Pearson, 2014.

Suggested Reading

1. G. Crocetti, "Textual Spatial Cosine Similarity," *arXiv preprint arXiv:1505.03934*, 2015.

2. G. Salton and C. Buckley, "Term-Weighting Approaches in Automatic Text Retrieval," *Information Processing & Management*, vol. 24, no. 5, pp. 513–523, 1988.

SGOU



Pre-processing and Representation Models

SGOU

Unit 1

Question Answering Systems

Learning Outcomes

After completing this unit, learner will able to:

- explain the concept of Question Answering (QA) and distinguish factoid questions from other question types
- describe the two main paradigms of Question Answering and explain their respective strengths
- describe the three-stage architecture of a factoid Question-Answering system
- explain the major tasks involved in question processing, including answer type detection and the use of a hierarchical answer type taxonomy
- explain passage retrieval, including the features used to rank candidate passages, and trace a question through the complete factoid QA pipeline

Prerequisites

Think about what happens when you type a question into a search engine and receive a direct answer instead of only a list of links. First, the system determines what kind of answer is required, such as a person, place, date, or number. Next, it searches a large document collection to identify passages that may contain the answer and ranks them according to relevance. Finally, it examines the most promising passages and extracts the exact words that answer the question.

Understanding Question Answering Systems is important because modern information systems are increasingly expected to provide a direct answer to a user's question rather than simply presenting a list of documents or web pages. For example, when a user asks a digital assistant, "Who is the author of Pride and Prejudice?", the system needs to recognise that the question asks for a person, process the question to determine the required information, retrieve relevant passages from available documents, and identify the answer from those passages. Factoid Question Answering helps learners understand how systems answer questions that usually require a specific fact, such as a person, location, date, organisation, or other short answer. Question Processing explains how a natural language question can be analysed to determine what information is being requested, while Passage Retrieval helps identify the relevant portions of documents

SGOU



that may contain the answer. Understanding these concepts provides a foundation for examining how Question Answering systems transform a user's natural language question into a relevant and concise answer.

Keywords

Question Answering, Factoid Question, Information Need, IR-Based QA, Knowledge-Based QA, Question Processing, Answer Type, Answer Type Taxonomy, Query Formulation, Question Classification

Discussion

5.1.1 Introduction

People have always needed access to information, and question answering has been a computing goal since its early years. Systems developed in the early 1960s could already answer questions about domains such as baseball statistics and scientific facts. By 2011, IBM's **Watson** system demonstrated major advances in question answering by defeating human champions on the television quiz show *Jeopardy!*

The distinction between **question answering** and **search** has become increasingly blurred. When a user types a question into a search engine and receives a direct answer rather than only a list of links, question answering techniques may be involved in producing that response. Because much of the world's knowledge is represented in textual form, question answering is closely related to **Information Retrieval (IR)** and shares many of the same underlying techniques for locating and ranking relevant information.

5.1.1.1 Factoid Questions

Question Answering systems often focus on a specific type of question known as a **factoid question**. A factoid question asks for a specific piece of factual information and can usually be answered with a short response, such as a person, place, date, or number.

Examples include:

- *Where is the Napier Museum located?*
- *Who founded Infosys?*
- *When was the state of Kerala formed?*
- *How many states are there in India?*

These questions have a common characteristic: each requires a short, specific, and verifiable answer. This makes them suitable for factoid Question Answering systems.

In contrast, questions such as *"Why did the Indian economy grow rapidly after 1991?"* or *"Is nuclear power a good idea?"* require explanation, synthesis, or judgement rather

SGOU

than retrieving a single fact. Such questions fall outside the scope of factoid Question Answering.

Table 5.1.1: Examples of factoid and non-factoid questions based on expected answer type

Question	Type	Answer	Factoid?
Who founded Wipro?	PERSON	Azim Premji	Yes
Where is ISRO headquartered?	LOCATION	Bengaluru	Yes
When was Infosys founded?	DATE	1981	Yes
Why did Infosys succeed?	—	Requires explanation	No
Should India invest in AI research?	—	Requires judgement	No

5.1.1.2 Two Paradigms for Question Answering

There are two broad approaches to answering factoid questions, distinguished by the source from which the answer is obtained.

1. Information-Retrieval Based Question Answering

Information Retrieval-based QA uses large collections of text, such as Wikipedia, PubMed, or web documents, as the source of answers.

Given a user question, an information retrieval component first identifies relevant documents or passages. A reader or reading-comprehension component then examines the retrieved text and extracts the answer.

Figure 5.1.1 illustrates this two-stage process.

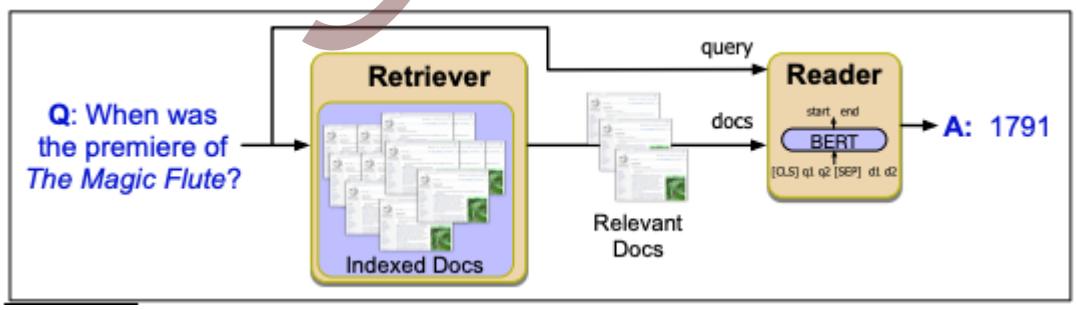


Fig 5.1.1 Retriever–reader architecture for Information Retrieval-based factoid Question Answering

IR-based factoid Question Answering commonly consists of two stages:

- **Retrieval:** Finds relevant documents or passages from the collection.

SGOU

- **Reading:** Examines the retrieved text and extracts the most likely answer span.

2. Knowledge-Based Question Answering

An alternative approach constructs a semantic representation of the question and uses it to query a structured database or knowledge base of facts. The meaning of a question may be represented using a formal logical expression or a simpler structure such as a relation triple.

Two common approaches are used for knowledge-based QA:

- Graph-based QA represents the knowledge base as a graph, with entities as nodes and relations as edges connecting those entities.
- QA by Semantic Parsing converts a natural-language question into a structured semantic representation that can be used to query the knowledge base.

For example, the question *"Who founded Infosys?"* can be transformed into a query over a knowledge base containing the triple:

(Narayana Murthy, FOUNDER_OF, Infosys)

This is the type of relation triple generated by the relation extraction systems discussed in Block IV.

These two broad QA paradigms offer complementary strengths:

- Information Retrieval-based QA can answer questions using information available in large text collections, although retrieved passages may be irrelevant or incorrectly interpreted.
- Knowledge-based QA can provide precise answers when the required fact is available in the knowledge base, but may fail when the relevant information is missing.
- Some QA systems combine retrieval-based and knowledge-based approaches.

5.1.2 The Architecture of a Factoid QA System

Figure 5.1.2 illustrates the common three-stage architecture of a factoid Question Answering (QA) system. A question passes through three main stages, each performing a distinct function:

- Question Processing: The question is analysed to determine the expected answer type and to formulate or reformulate a query suitable for the retrieval system.
- Passage Retrieval: The query is used to retrieve relevant documents and identify shorter passages that are likely to contain the answer.

SGOU

- Answer Processing: The retrieved passages are examined to identify candidate answers, which are then scored or ranked to produce the final answer.

Note: The search space is progressively reduced at each stage, beginning with a large document collection, narrowing to candidate passages, and finally selecting a short answer span. This reduces the information examined at each subsequent stage and makes the overall process more efficient.

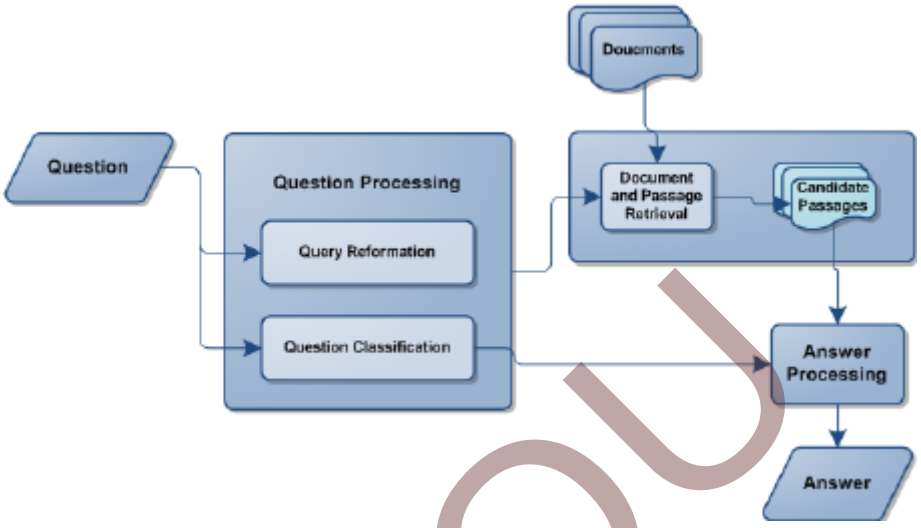


Fig. 5.1.2 Common three-stage architecture of a factoid Question Answering system

5.1.3 Question Processing

The first stage takes the user's question and extracts the information the later stages of the QA system need. Four main tasks are performed.

5.1.3.1 Answer Type Detection

The most important task is deciding what kind of thing the answer must be. If we know the answer to "Who founded Infosys?" must be a PERSON, then we can immediately discard any candidate answer that is a date, a place, or a number. This constraint is enormously useful and is applied at every subsequent stage.

An important task is determining what type of answer is expected. For example, if the question *"Who founded Infosys?"* requires a PERSON as the answer, candidate answers that are dates, locations, or numbers can be excluded. This constraint is useful throughout the subsequent stages of the system.

Answer type detection commonly uses two signals:

- the question word
- the headword of the question phrase

The Question Word

Some question words strongly indicate the expected answer type. "Who" usually signals

SGOU

a PERSON, "When" signals a DATE or TIME, and "How many" signals a NUMBER. These cues can often provide useful answer-type information directly.

The Headword

Some question words are ambiguous, so the headword of the question phrase must also be considered. For example:

"Which Indian company was founded in 1981?" → ORGANIZATION

"Which Indian city hosts the ISRO headquarters?" → LOCATION

The question word **"which"** is the same in both questions. The headword company in the first question and city in the second indicates the expected answer type. Identifying this headword is one reason why part-of-speech tagging and syntactic parsing are useful during question processing.

5.1.3.2 Query Formulation

The second task involves transforming the question into a query the retrieval engine can process. The original question may contain question words, function words, and grammatical elements that contribute little to keyword-based retrieval and can reduce search effectiveness.

For example, in the question *"Who founded Infosys in 1981?"*, the words *"who"* and *"in"* are less useful for document retrieval. Removing such terms produces the query:

"founded Infosys 1981"

This query retains the important content words that are more likely to occur in relevant passages.

Table 5.1.2 Examples of query formulation by removing question words and function words from factoid questions

Original Question	Formulated Query	What was Removed
Who founded Infosys in 1981?	founded Infosys 1981	who, in
Where is ISRO headquartered?	ISRO headquartered	where, is
When was Kerala formed as a state?	Kerala formed state	when, was, as, a

Query Expansion

A useful technique in question answering and information retrieval is query expansion. It addresses the vocabulary mismatch problem, which occurs when the words used in a question differ from those used in relevant documents.

For example, a user may ask:

SGOU

"Who started Infosys?"

while a relevant document may state:

"Infosys was founded by …"

If the system searches only for the word "started," it may fail to retrieve the relevant document. Query expansion improves retrieval by adding related words or alternative expressions such as:

started, founded, established, set up

This allows the system to retrieve relevant documents even when they use different wording from the original question.

5.1.3.3 Question Classification

Not every question asked by a user is a factoid question. Question classification determines the general type of question so that the system can select an appropriate answering strategy.

For example, if a system treats a definition question as a factoid question, it may return only a short noun phrase when the user actually requires a descriptive sentence. Classifying the question first helps the system choose the correct processing approach.

5.1.3.4 Focus Detection

The **focus** of a question is the word or phrase that identifies what the question is asking about and is typically replaced by the answer when the question is reformulated as a statement.

For example, in the question:

"Which Indian company was founded by Narayana Murthy?"

the focus is "company." The answer fills the blank in the statement:

"_____ was founded by Narayana Murthy."

Focus detection is closely related to answer type detection, because the focus often provides the headword that indicates the expected answer type.

Focus information can also support answer extraction, since passages containing words related to the focus near a candidate answer may provide stronger evidence for selecting the correct answer.

5.1.4 Passage Retrieval

The second stage uses the query produced during question processing to locate text that is likely to contain the answer. This process typically occurs in two steps. Understanding why retrieval is performed in two stages is important, because searching an entire document collection and identifying the exact answer passage are different tasks.

SGOU

5.1.4.1 Why Documents Are Not Enough

A typical Information Retrieval system, such as the one discussed in Block IV, Unit 4, returns complete documents. This is often suitable for search-engine users who want a page or article to read. A Question Answering system, however, requires a more precise result.

For example, if a user searches for *"founded Infosys 1981,"* the system may retrieve the Wikipedia article on Infosys. Although the article contains the answer, it may be several thousand words long. Returning the entire document does not directly answer the question.

The system must therefore identify the sentence, paragraph, or passage most likely to contain the required fact and, eventually, extract the exact answer span.

This is why passage retrieval is used as an intermediate step. It takes the documents returned by the retrieval engine and selects smaller text segments, such as sentences or paragraphs, that are more likely to contain the answer.

5.1.4.2 Document Retrieval

The first step uses the standard retrieval methods introduced in Block IV. The formulated query is submitted to a retrieval engine, which consults the inverted index, scores candidate documents using weighting schemes such as TF-IDF or BM25, and returns a ranked list. A fixed number of top-ranked documents, often a few dozen, are then selected for further processing.

The number of documents selected is important. If too few documents are retrieved, the correct answer may be missing, which limits the performance of later stages. On the other hand, retrieving too many documents increases the amount of text that must be examined during passage retrieval, which can slow the system and increase the chance of selecting an irrelevant passage.

The main change is replacing the fixed 20 to 50 range with "often a few dozen", so the text does not imply a universal setting

5.1.4.3 Passage Extraction and Ranking

Each retrieved document is divided into smaller passages, typically paragraphs, individual sentences, or fixed-length text segments. The choice of passage size is an important design decision. If a passage is too long, it may contain information irrelevant to the question; if it is too short, the information required to identify the answer may be distributed across multiple passages.

The resulting candidate passages are then ranked using features that may differ from those used during initial document retrieval. Document ranking measures how relevant an entire document is to the query, whereas passage ranking estimates how likely a particular passage is to contain the answer to the question.

Key Features for Passage Ranking

- **Answer type match:** Passages are evaluated for the presence of named

SGOU

entities that match the expected answer type. For example, if the required answer type is PERSON and a passage contains no person name, it is less likely to contain the answer.

- **Question term overlap:** The number and importance of the question's content words that also appear in the passage are measured.
- **Proximity:** The closeness of question terms within the passage is considered. Terms occurring near one another, particularly within the same sentence, often provide stronger evidence than terms scattered across a larger passage**.**
- **Term ordering:** The passage is examined to determine whether important question terms occur in a similar order. Similar term ordering can provide additional evidence that the passage is relevant.
- **Focus word presence:** The passage is checked for the presence of the question focus or closely related terms identified during question processing.
- **Longest common subsequence:** The length of the longest sequence of words occurring in the same order in both the question and the passage is measured.

5.1.5 Answer Processing

Passage retrieval produces a small, ranked set of short text passages that are likely to contain the answer. In the final stage, known as **answer processing**, the system examines these passages, identifies the exact text span that answers the question, and returns it to the user.

Modern Question Answering systems often use reading comprehension models for this step. These models take a question and a passage as input and predict the span of text that best answers the question. They are commonly trained on datasets such as SQuAD, where questions are paired with passages and human annotators annotate the correct answer spans.

Unit 2 of this block will discuss answer processing in greater detail, along with methods for evaluating the answers produced by the system.

SGOU

Recap

- Question Answering (QA) systems are designed to satisfy users' information needs by providing direct answers. The distinction between QA and traditional search is becoming increasingly blurred.
- A factoid question asks for a specific factual piece of information that can usually be expressed as a short answer.
- Factoid answers are typically names, places, dates, or numbers. Questions requiring explanation, synthesis, or judgement are generally considered non-factoid.
- IR-based Question Answering retrieves relevant text passages and uses them to identify an answer.
- Knowledge-based Question Answering converts a question into a semantic representation and uses it to query a structured knowledge base.
- A factoid QA system commonly consists of three stages: question processing, passage retrieval, and answer processing.
- At each stage, the search space is progressively narrowed from a large document collection to a small number of passages and finally to an answer.
- Question processing performs four main tasks: answer type detection, query formulation, question classification, and focus detection.
- Answer type detection determines the expected type of answer using information such as the question word and the headword of the question phrase.
- Question words alone may be insufficient for ambiguous forms such as *"which."* In such cases, the headword helps determine the expected answer type.
- A hierarchy of answer types can be used to represent answer categories at different levels of specificity.
- Query formulation transforms the original question into a form suitable for retrieval by retaining important content terms and removing less useful elements where appropriate.
- Query expansion adds related words or alternative expressions to reduce vocabulary mismatch between the question and relevant documents.
- Question classification distinguishes factoid questions from other question types so that an appropriate answering strategy can be selected.

SGOU

- The focus of a question identifies what the question is asking for and is typically replaced by the answer when the question is reformulated as a statement.
- Passage retrieval narrows retrieved documents to shorter text segments, such as paragraphs or sentences, that are more likely to contain the answer.
- Document retrieval uses standard IR techniques such as inverted indexes and ranking methods such as TF-IDF or BM25 to retrieve a set of highly ranked documents for further processing.
- Passage extraction and ranking divide retrieved documents into smaller passages and rank them according to their likelihood of containing the answer.
- Passage-ranking features may include answer type match, question-term overlap, proximity, term ordering, focus information, and longest common subsequence.
- Answer type match provides useful evidence: a passage containing an entity of the expected type is generally more likely to contain the correct answer.
- Passage-ranking systems combine several features because no single feature is sufficient to identify the correct passage reliably.
- After passage retrieval, answer processing examines the highest-ranked passages and identifies the most likely answer span.

Objective Type Questions

1. What type of question can be answered with a short fact?
2. What is the usual answer type for a question beginning with “Who”?
3. What is the usual answer type for a question beginning with “When”?
4. What is the usual answer type for a question beginning with “How many”?
5. Name the paradigm of QA that searches text collections for answers.
6. Name the paradigm of QA that queries a structured database.
7. How many main stages does a factoid QA system have?
8. Name the first stage of a factoid QA system.
9. Name the second stage of a factoid QA system.

SGOU

10. Name the third stage of a factoid QA system.

11. What task determines the expected type of the answer?

12. What task converts a question into a query suitable for retrieval?

13. What is the word or phrase that the answer would replace called?

14. Name one unit of text is typically used as a passage?

15. Which NLP task identifies persons and locations in a passage?

16. What technique related words and variants to a query?

17. What problem does query expansion help address?

18. What is the usual answer type for a question beginning with "Where"?

19. What data structure does the retrieval engine use to find candidate documents?

20. Name one weighting or ranking method used to score documents during retrieval.

Answers to Objective Type Questions

1. Factoid question
2. PERSON
3. DATE or TIME
4. NUMBER
5. IR-based QA
6. Knowledge-based QA
7. Three
8. Question processing
9. Passage retrieval
10. Answer processing
11. Answer type detection

SGOU

12. Query formulation

13. Focus

14. Sentence or paragraph

15. Named Entity Recognition (NER)

16. Query expansion

17. Vocabulary mismatch

18. LOCATION

19. Inverted index

20. TF-IDF or BM25

Assignments

1. What is a factoid question? Give four examples of factoid questions along with their expected answer types. Also give three examples of non-factoid questions and explain why each is not a factoid question.

2. Compare IR-based and knowledge-based question answering. Describe how each approach works, what resources each requires, and give one example of a question that each approach would handle well and one that it may handle poorly.

3. Describe the three-stage architecture of a factoid QA system. For each stage, state its input, output, and main function. Then trace the question *"Where is the Indian Institute of Science located?"* through all three stages.

4. Explain answer type detection. Describe the two main signals used, explain why the question word alone may be insufficient, and give three examples in which the headword determines the answer type. Also explain why an answer type taxonomy may be hierarchical.

5. What is query formulation? For each of the following questions, write the formulated query and state which words were removed:

   - *"Who is the Chief Minister of Kerala?"*
   - *"When did ISRO launch Chandrayaan-3?"*
   - *"How many IITs are there in India?"*

SGOU

6. Explain why passage retrieval is treated as a separate stage from document retrieval. Describe both stages and discuss the trade-offs involved in choosing the number of documents to retrieve and the appropriate passage length.

Reference

1. B. Loni, "A Survey of State-of-the-Art Methods on Question Classification," *Delft University of Technology, Technical Report*, vol. 55, p. 57, 2011.

Suggested Reading

1. I. C. Mogotsi, "Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze: *Introduction to Information Retrieval*, Cambridge University Press, Cambridge, England, 2008, 482 pp., ISBN 978-0-521-86571-5," 2010.

2. D. Jurafsky and J. H. Martin, *Speech and Language Processing*, 3rd ed., draft, ch. 14, "Question Answering, Information Retrieval, and Retrieval-Augmented Generation." Stanford University, 2024.

SGOU

Unit 2

Answer Processing

Learning Outcomes

After completing this unit, learner will able to:

- evaluate factoid answers using Mean Reciprocal Rank (MRR), Exact Match (EM), and F1-score
- explain the concept of Machine Translation (MT) and describe its major practical applications
- explain why translation cannot be performed reliably through simple word-for-word substitution
- describe word-order typology and classify languages as SVO, SOV, or VSO
- describe morphological typology and referential density as sources of difficulty in Machine Translation

Prerequisites

Answer Processing, evaluation of factoid answers, and machine translation are important for learning how Natural Language Processing systems convert retrieved information into useful answers and assess whether those answers are accurate and meaningful. In a real-life situation, when a user asks a Question Answering system, “Who is the current Prime Minister of India?”, retrieving a passage containing the correct information is not sufficient; the system must identify the appropriate answer, determine whether it correctly answers the question, and evaluate its quality. Similarly, when a person uses an online translation system to translate a sentence from Malayalam to English, the system must deal with differences in vocabulary, grammar, sentence structure, and context between the two languages. Machine translation is therefore challenging because a word or sentence may have different meanings depending on context, and a direct word-by-word translation may not convey the intended meaning. Studying these topics helps learners understand how NLP systems process answers, measure the quality of factoid answers, and handle the linguistic and contextual challenges involved in translating human language between different languages.

SGOU



Keywords

Parallel Corpus, Bitext, SQuAD, Machine Translation, Linguistic Typology, Word-Order Typology, Lexical Divergence, Lexical Gap, Morphological Typology, Isolating Language, Agglutinative Language, Referential Density

Discussion

5.2.1 Introduction

Unit 1 concluded with a Question Answering system that had not yet completed its task. The question-processing stage identified the expected answer type, while passage retrieval reduced a large document collection to a small set of relevant passages. The remaining step is to examine these passages and extract the exact words that answer the question.

This unit completes the discussion of Question Answering before introducing a second NLP application, Machine Translation (MT). Machine Translation, like Question Answering, processes input text to produce output text and must deal with the fact that meaning is not conveyed by individual words alone, as illustrated in Figure 5.2.1. The latter part of this unit examines why translation is difficult, focusing mainly on linguistic differences between languages.

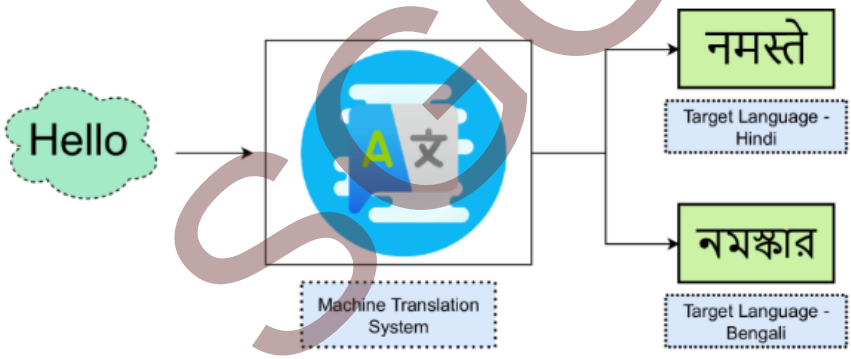


Fig 5.2.1 Overview of a Machine Translation system translating source text into different target languages

5.2.2 Answer Processing

The third and final stage of a factoid QA system takes the ranked passages produced by passage retrieval and extracts the answer. Although the task appears simple, accurately identifying the correct answer span can be challenging.

5.2.2.1 The Reading Comprehension Task

Answer processing is commonly formulated as a reading comprehension task. The system is given a question and a passage that is expected to contain the answer and must identify the answer within that passage.

SGOU

In extractive Question Answering, the system does not generate a new sentence. Instead, it selects a contiguous sequence of tokens from the passage, known as an answer span.

The model predicts two positions:

- the start position of the answer
- the end position of the answer

All tokens between these positions, inclusive, form the extracted answer.

Worked Example

Suppose the question is:

"Who co-founded Infosys?"

and passage retrieval returns:

"Infosys was co-founded by N. R. Narayana Murthy in Pune in 1981."

1	2	3	4	5	6	7	8	9	10	11	12
Infosys	was	founded	by	N.	R.	Narayana	Murthy	in	pune	in	1981

The model predicts:

START = 5

END = 8

Output: N. R. Narayana Murthy

No new words are generated. The answer already appears in the passage, so the model identifies its boundaries.

The expected answer type can also help distinguish suitable candidates. Infosys is an organisation, Pune is a location, and 1981 is a date, whereas N. R. Narayana Murthy is a person.

Why Span Labeling Rather Than Generation

There are two important advantages of extractive QA for fact-based questions:

- **Verifiability:** The answer is copied directly from a source passage, making it easier to trace and verify.
- **Factual precision:** Names, dates, and numbers are taken directly from the source rather than newly generated.

5.2.2.2 Training Data for Answer Processing

Span-labeling models are trained using datasets in which the correct answer span is

SGOU

identified for each question. A well-known example is SQuAD (Stanford Question Answering Dataset), which contains questions associated with passages and annotated answer spans.

For each training example, the model learns to predict two positions: the token at which the answer begins and the token at which it ends.

5.2.3 Evaluation of Factoid Answers

Different evaluation techniques are used for factoid Question Answering systems depending on whether the system returns a ranked list of candidate answers or a single answer span.

5.2.3.1 Mean Reciprocal Rank

When a system returns a ranked list of candidate answers, the position of the first correct answer is important. An answer appearing at rank 1 is more useful than the same answer appearing at rank 5.

Mean Reciprocal Rank (MRR) measures this behaviour. For each question, the reciprocal of the rank of the first correct answer is calculated. A correct answer at rank 1 receives a score of 1.0, at rank 2 a score of 0.5, and at rank 5 a score of 0.2. If no correct answer is retrieved, the score for that question is 0.

$$MRR = \frac{1}{N}\sum_{i=1}^{N} \frac{1}{rank_i}$$

where:

- N is the number of questions.
- $rank_i$ is the position of the first correct answer for question i.
- If no correct answer is retrieved, the reciprocal-rank contribution is taken as 0.

Worked Example

Question	First correct answer at rank	Reciprocal rank
Q1	1	1/1 = 1.000
Q2	3	1/3 = 0.333
Q3	2	1/2 = 0.500
Q4	not found	0
Q5	5	1/5 = 0.200

SGOU

The sum of reciprocal ranks is:

$$1.000 + 0.333 + 0.500 + 0 + 0.200 = 2.033$$

Therefore,

$$MRR = \frac{2.033}{5} = 0.407$$

MRR gives greater reward when the first correct answer is moved closer to the top of the ranking. For example, moving an answer from rank 2 to rank 1 increases its contribution from 0.5 to 1.0, whereas moving it from rank 5 to rank 4 increases the contribution only from 0.20 to 0.25.

Note: A question for which no correct answer is retrieved contributes **0**. An answer appearing at rank 100 would contribute $1/100 = 0.01$, so it is penalized heavily but not treated the same as a missing answer.

5.2.3.2 Exact Match

When a system returns a single answer span, the predicted answer can be compared directly with the gold-standard answer.

Exact Match (EM) is a strict evaluation metric. It assigns:

- **1** if the predicted answer exactly matches the gold answer after the specified evaluation normalization.
- **0** otherwise.

No partial credit is awarded.

For example:

Gold answer: N. R. Narayana Murthy

Predicted answer: Narayana Murthy

The prediction identifies the correct person, but because it does not exactly match the complete gold span, its Exact Match score is **0**.

5.2.3.3 Token-Level F1

Token-level F1 addresses the strictness of Exact Match by awarding partial credit based on overlapping tokens between the predicted and gold answers.

Consider:

- **Gold answer:** *N. R. Narayana Murthy*

  Tokens: N., R., Narayana, Murthy
- **Predicted answer:** *Narayana Murthy*

  Tokens: Narayana, Murthy

SGOU

Both predicted tokens occur in the gold answer:

$$Precision = \frac{2}{2} = 1.000$$

Two of the four gold tokens are recovered:

$$Recall = \frac{2}{4} = 0.500$$

Therefore,

$$F1 = 2 \times \frac{1.000 \times 0.500}{1.000 + 0.500} = 0.667$$

Exact Match and token-level F1 therefore provide complementary information. An EM score of 0 combined with an F1 score of 0.667 indicates that the prediction overlaps substantially with the gold answer even though it is not an exact match.

5.2.3.4 Human Evaluation

Automatic evaluation compares system-generated answers with one or more predefined gold-standard answers. However, some questions may have several valid answers or alternative surface forms that are difficult to capture completely in advance.

For example, the location of ISRO headquarters may be expressed as *Bengaluru*, *Bangalore*, or *Bengaluru, Karnataka*. These expressions may refer to the same location but differ in wording.

For this reason, human evaluation can complement automatic metrics. Human annotators judge whether an answer is correct regardless of minor differences in wording. Automatic metrics are useful for rapid and repeatable system comparison, while human evaluation can provide additional insight when answer correctness cannot be captured adequately by string-based measures.

5.2.4 Machine Translation

Machine Translation (MT) refers to using computers to automatically translate text from one language into another. Machine translation has been an important goal of computing for many years and is now widely used in practical applications.

Translating literature or poetry is particularly challenging because it involves creativity, style, cultural knowledge, and interpretation. Therefore, modern machine translation systems are primarily designed for practical translation tasks rather than attempting to reproduce every aspect of human translation.

5.2.4.1 Applications of Machine Translation

Modern Machine Translation (MT) is widely used in a range of practical applications.

Information Access

One of the most common uses of MT is helping people access information available in

SGOU

other languages. It can translate:

- Web pages and online articles
- Recipes and instructions
- Government information
- Wikipedia articles
- News and other online resources

MT can also help reduce the digital divide by making information available across languages, particularly when much more digital content exists in high-resource languages such as English than in many lower-resource languages.

Assisting Human Translators

MT can generate an initial translation that a human translator reviews and corrects. This process is known as **post-editing**.

MT is also used in Computer-Aided Translation (CAT) tools, particularly for localization, where content or products are adapted for a particular language or cultural community.

Real-Time Communication

MT can support communication between people who speak different languages. Translation systems can process speech as it is produced, supporting applications such as real-time multilingual communication and interpretation.

Image-Centric Translation

Text contained in an image can first be recognized using Optical Character Recognition (OCR) and then translated using an MT system.

For example, a mobile phone camera can be used to translate menus, road signs, or public notices.

5.2.4.2 The Translation Task

Many Machine Translation systems treat each sentence as an individual translation unit. Given a sentence in a source language, the task is to generate a corresponding sentence in a target language, as illustrated in Figure 5.2.2.

For example, an English sentence can be provided as input to a translation system, which then produces its corresponding Malayalam translation.

Machine translation uses supervised machine learning. During training, the system is given a large set of parallel sentences, with each sentence in the source language matched with its translation in the target language, and learns to map source sentences to target sentences. A text that appears in two or more languages in this way is called a parallel corpus or a bitext.

SGOU

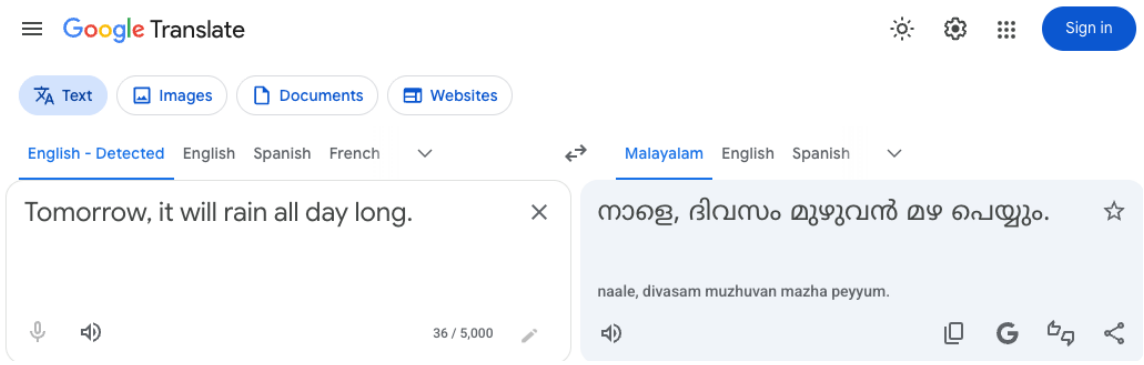


Fig 5.2.2: Example of English-to-Malayalam translation using Google Translate

Large parallel corpora are available for well-resourced language pairs. The Europarl corpus, extracted from the proceedings of the European Parliament, contains between 400,000 and 2 million sentences from each of 21 European languages. The United Nations Parallel Corpus contains about 10 million sentences in the six official languages of the United Nations. Other parallel corpora have been assembled from film and television subtitles and from general web text.

Parallel corpora play a key role for Indian languages, especially since many have fewer digital resources than English and other major languages.

For example, the Samanantar corpus offers large-scale English–Indian-language parallel data for languages such as *Hindi, Bengali, Gujarati, Malayalam, Marathi, Tamil, Telugu, Kannada, Punjabi, Odia,* and *Assamese.* These resources help develop machine translation and other NLP applications for Indian languages.

5.2.5 Why Machine Translation Is Difficult?

Many people think translation is simply about replacing words from one language with words from another using a dictionary. However, even a simple example shows that this approach does not work.

Consider the English sentence:

"He wrote a letter to a friend."

Now consider its translation into Japanese:

"Tomodachi ni tegami-o kaita."

(Friend to letter wrote.)

Notice that the elements of the two sentences occur in very different positions.

- In English, the verb occurs in the middle of the sentence.
- In Japanese, the verb kaita occurs at the end.
- The Japanese sentence does not require the pronoun he, whereas English does.

SGOU

- Word-for-word substitution would produce an incorrect or unnatural translation in either direction.

Understanding why these translation differences occur can help us develop better translation systems. It is useful to distinguish individual differences, such as how the word for *"dog"* varies from one language to another, from broader patterns that can be described more generally. The study of these patterns and differences is called linguistic typology.

5.2.5.1 Word Order Typology

Languages vary in how they arrange verbs, subjects, and objects in simple sentences. Word order is one of the most obvious ways to compare different language types.

Word order typology studies how languages arrange the major components of a sentence, particularly the Subject (S), Object (O), and Verb (V).

English is predominantly Subject–Verb–Object (SVO):

Ravi eats rice.

Ravi → eats → rice

S → V → O

Hindi and Malayalam are generally Subject–Object–Verb (SOV) languages:

Hindi:

रवि चावल खाता है।

Ravi chawal khata hai.

Ravi → rice → eats

S → O → V

Malayalam:

രവി ചോറ് കഴിക്കുന്നു

Ravi choru kazhikkunnu.

Ravi → rice → eats

S → O → V

These differences often require the translation system to perform substantial structural reordering as it generates its output. In particular, translation between English and Hindi may require considerable rearrangement because of their different basic word orders.

5.2.5.2 Lexical Divergence

Beyond word order, individual words seldom correspond precisely across languages. A term in one language may have multiple possible translations in another, with the

SGOU

appropriate choice depending on context, semantic nuance, grammatical features, or cultural knowledge.

This phenomenon is called lexical divergence.

One Word, Several Translations

Consider the English word uncle.

Hindi distinguishes several relationships using different lexical items:

- मामा (māmā) – mother's brother
- चाचा (chāchā) – father's younger brother
- ताऊ (tāū) – father's elder brother

Malayalam similarly uses more specific kinship terms, with usage varying by family and region. For example:

ചിറ്റപ്പൻ (chittappan) – commonly father's younger brother

വല്യച്ഛൻ (valyachchan) – commonly father's elder brother

അമ്മാവൻ (ammāvan) – commonly mother's brother or certain older male relatives, depending on usage

Consider:

My uncle came yesterday.

The English word uncle does not specify the exact familial relationship.

If the intended person is the mother's brother, a Hindi translation could be:

मेरे मामा कल आए।

If the intended person is the father's younger brother:

मेरे चाचा कल आए।

Therefore, the source sentence does not provide sufficient information to determine the most specific target-language term.

A Machine Translation (MT) system may require contextual information to select the appropriate target-language equivalent.

- **Lexical Ambiguity**

A single word may have multiple meanings within the same language.

Consider the English word light.

It can refer to illumination:

SGOU

Turn on the light.

Or to something having little weight:

This bag is light.

The Hindi translations for these senses differ:

लाइट जलाओ। → Turn on the light.

यह बैग हल्का है। → This bag is light in weight.

Malayalam likewise uses different expressions for each sense:

ലൈറ്റ് ഓണാക്കൂ. → Turn on the light.

ഈ ബാഗ് ഭാരം കുറഞ്ഞതാണ്. → This bag is light in weight.

Therefore, an MT system must determine the intended sense of the word *light* before selecting the appropriate target-language expression. This requirement closely links Machine Translation with the task of Word Sense Disambiguation (WSD).

♦ **Grammatical and Contextual Constraints on Word Choice**

Sometimes the source language leaves information unspecified, while the target language offers ways to express it more explicitly.

Consider:

I met my cousin yesterday.

English uses the general word cousin.

Hindi can specify the relationship more precisely when the information is known:

मामा का बेटा – mother's brother's son

चाचा की बेटी – father's younger brother's daughter

Malayalam can similarly use phrases such as:

അമ്മാവന്റെ മകൻ – mother's brother's son

അച്ഛന്റെ സഹോദരന്റെ മകൾ – father's brother's daughter

The important point is that Hindi or Malayalam does not always use these longer expressions. Rather, the target language provides lexical or phrasal distinctions that a single English word may not express.

Consequently, an MT system must determine whether the additional information is available from the context before choosing a target-language expression.

Many-to-Many Lexical Mappings

SGOU

Lexical relationships can be even more complicated than one word having several translations.

Consider the English kinship terms:

brother, sister, uncle, aunt, cousin

These are relatively broad categories.

Hindi and Malayalam can express more specific family relationships through different words or combinations of words.

For example:

English: brother

Hindi:

भाई

bhāī

Additional information can be expressed as:

बड़ा भाई – elder brother

छोटा भाई – younger brother

Malayalam similarly has:

ചേട്ടൻ – elder brother

അനിയൻ – younger brother

Therefore:

English brother → Hindi भाई / बड़ा भाई / छोटा भाइ

and

English brother → Malayalam ചേട്ടൻ / അനിയൻ, etc.

depending on context.

This is an example of a one-to-many lexical mapping.

At the same time, several source-language expressions may correspond to a single target-language expression in a particular context. Consequently, lexical mappings between languages can be many-to-many rather than one-to-one.

Lexical Gaps

A lexical gap occurs when one language lacks a single lexical item that exactly matches the meaning and usage of an expression in another language.

SGOU

This does not mean that the concept cannot be translated. It means that the translation may require:

- a phrase,
- a descriptive expression,
- a culturally adapted expression, or
- additional contextual information.

Kinship terminology provides a useful Indian-language example.

English has the broad term:

cousin

A Malayalam speaker may specify the relationship through a phrase such as:

അമ്മാവന്റെ മകൻ

mother's brother's son

Hindi can similarly use:

मामा का बेटा

mother's brother's son

Thus, the English single-word expression may correspond to a multi-word expression in Hindi or Malayalam.

Lexical gaps are especially important in translation involving culture-specific concepts, social relationships, food, customs and institutions.

- **Agglutinative Structure and Translation**

In an agglutinative language, grammatical information can be added through sequences of affixes or suffixes, often with relatively identifiable morphological functions.

Malayalam is commonly described as an agglutinative language. Its morphology can involve multiple suffixes and morphological changes, and identifying morpheme boundaries can be challenging for computational systems.

noun/root + case/direction + other grammatical information

5.2.5.3 Morphological Typology

Languages differ in how they build words from smaller meaningful units called morphemes.

For example, in English:

books = book + -s

Malayalam

SGOU

Malayalam commonly expresses grammatical relationships through case suffixes and related morphological forms.

വീട്

veedu – house

വീട്ടിൽ

veettil – in/at the house

വീട്ടിലേക്ക്

veetilekku – to/towards the house

Malayalam is widely described as an agglutinative and highly inflectional language, and morphological analysis is an important computational problem for Malayalam NLP.

Hindi

Hindi also expresses grammatical information using inflection and postpositions.

Consider:

घर में

ghar mein

in the house

घर से

ghar se

from the house

घर + से

house + from

Hindi therefore often expresses grammatical relationships through postpositions, which occur after the noun phrase.

5.2.5.4 Referential Density

The fourth dimension concerns not what languages explicitly express, but what they can leave implicit.

Languages differ in how often they explicitly mention a participant or referent that can already be understood from context.

The term referential density describes how explicitly a language or text tends to express references to entities and discourse participants.

English normally requires an overt subject in ordinary finite clauses:

SGOU

Rahul went home. He slept.

The second sentence explicitly contains He to identify the subject.

This phenomenon is commonly discussed using terms such as pro-drop, null subject, or zero anaphora, although the exact conditions differ by language.

Why Referential Differences Matter in MT

Consider the Malayalam sentence:

പാൽ വാങ്ങി

Without context, it may be unclear who bought the milk.

Possible interpretations could include:

I bought milk.

He bought milk.

She bought milk.

They bought milk.

The preceding discourse may resolve the ambiguity.

For example:

അനു കടയിൽ പോയി. പാൽ വാങ്ങി.

A natural English translation is:

Anu went to the shop. She bought milk.

The MT system must therefore perform a form of cross-sentence reference resolution.

This shows that translation sometimes requires information beyond the current sentence.

For example:

Rahul entered the room. [He] sat down. [He] opened his laptop.

5.2.5.5 Bringing the Four Dimensions Together

The four dimensions can be summarized as follows:

Word Order Typology

Question: How are sentence constituents arranged?

English: SVO

Hindi: SOV

Malayalam: SOV

SGOU

Lexical Divergence

Question: Which target-language expression represents the intended meaning?

English: uncle

Hindi: मामा / चाचा / ताऊ … …

Malayalam: different kinship terms depending on the relationship

Morphological Typology

Question: How is grammatical information encoded?

English: to the house

Malayalam: വീട്ടിലേക്ക്

A phrase in one language may correspond to a single morphologically complex word in another.

Referential Density

Question: Which participants are explicitly mentioned?

English: Rahul went home. He slept.

Malayalam: രാഹുൽ വീട്ടിൽ പോയി ഉറങ്ങി.

The second Malayalam clause can omit the subject when it is recoverable from context.

A translation system may need to handle all four dimensions simultaneously. This is why Machine Translation, despite decades of development and substantial practical progress, remains challenging.

SGOU

Recap

- Answer processing is the final stage of a Question Answering system, in which candidate passages are examined to identify the exact answer to a question.
- Factoid Question Answering focuses on questions that generally expect a short, specific answer such as a person, place, organisation, date, or number.
- Mean Reciprocal Rank (MRR) evaluates a ranked list of candidate answers by considering the rank position of the first correct answer.
- The reciprocal rank for a question is calculated as 1/rank, and a question for which no correct answer is found contributes 0.
- MRR gives greater importance to correct answers appearing near the top of the ranked list, reflecting the usefulness of highly ranked answers to users.
- Machine Translation (MT) is the automatic translation of text or speech from one natural language into another.
- Common applications of Machine Translation include information access, computer-aided translation, localisation, real-time communication, and image-based translation.
- Large parallel corpora containing aligned texts in two or more languages provide important training resources for Machine Translation systems.
- Word order typology describes how languages arrange major sentence constituents such as Subject (S), Verb (V), and Object (O).
- English predominantly follows SVO word order, whereas Hindi and Malayalam generally follow SOV word order, often requiring structural reordering during translation.
- Lexical divergence occurs when words or concepts do not have direct one-to-one equivalents between languages.
- Lexical divergence may arise from lexical ambiguity, context-dependent word choice, many-to-many mappings, and lexical gaps.
- Morphological typology examines how languages construct words from morphemes and how grammatical information is represented within words or through separate grammatical elements.
- Referential differences occur when one language explicitly expresses a participant using a pronoun, while another language can omit the pronoun when the referent is recoverable from context.

SGOU

- Machine Translation is difficult because languages differ in word order, lexical choice, morphology, and reference, requiring a system to transfer meaning rather than simply translate individual words.

Objective Type Questions

1. What is the final stage of a Question Answering system that identifies the exact answer?
2. What type of Question Answering generally expects a short, specific answer?
3. Which evaluation measure considers the rank of the first correct answer?
4. What is the reciprocal rank when the first correct answer occurs at rank 1?
5. What is the reciprocal rank when the first correct answer occurs at rank 5?
6. What score does a question receive in MRR when no correct answer is found?
7. What does MRR stand for?
8. What technology automatically translates text from one natural language to another?
9. What type of corpus contains aligned texts in two or more languages?
10. What is the predominant word order of English?
11. What is the predominant word order of Hindi?
12. What is the predominant word order of Malayalam?
13. What is the term for a lack of one-to-one correspondence between words across languages?
14. Which NLP problem determines the intended meaning of an ambiguous word?
15. What is the smallest meaningful unit used in morphological analysis?
16. Which type of language commonly builds words using sequences of morphemes or affixes?
17. What type of information may be omitted in a pro-drop language?

SGOU

18. What must an MT system recover when an omitted pronoun refers to a previously mentioned entity?

19. Name one major application of Machine Translation.

20. Which linguistic dimension deals with the arrangement of Subject, Object and Verb?

Answers to Objective Type Questions

1. Answer processing
2. Factoid Question Answering
3. Mean Reciprocal Rank (MRR)
4. 1.0
5. 0.2
6. Zero
7. Mean Reciprocal Rank
8. Machine Translation (MT)
9. Parallel corpus or bitext
10. SVO
11. SOV
12. SOV
13. Lexical divergence
14. Word Sense Disambiguation (WSD)
15. Morpheme
16. Agglutinative language
17. Subject or pronoun
18. The referent

SGOU

19. Information access / localization / real-time communication / image-based translation / assisting human translators

20. Word order typology

Assignments

1. Explain the role of answer processing in a Question Answering system with a suitable example.

2. Explain Mean Reciprocal Rank (MRR) with a suitable example.

3. Write a short note on Machine Translation and list any four applications of Machine Translation.

4. Compare the word order of English, Hindi, and Malayalam with suitable examples.

5. Explain any two linguistic differences that make Machine Translation difficult, with suitable examples.

Reference

1. S. Vajjala, B. Majumder, A. Gupta, and H. Surana, *Practical Natural Language Processing: A Comprehensive Guide to Building Real-World NLP Systems*. O'Reilly Media, 2020.

2. D. Jurafsky and J. H. Martin, *Speech and Language Processing*. Pearson, 2014.

SGOU

Suggested Reading

1. Stanford Question Answering Dataset (SQuAD), "SQuAD Explorer." [Online]. Available: https://rajpurkar.github.io/SQuAD-explorer/ [Accessed: 6 September 2026].

2. S. Bird, E. Klein, and E. Loper, *Natural Language Processing with Python*. O'Reilly Media, 2009.

3. G. Ramesh, S. Doddapaneni, A. Bheemaraj, M. Jobanputra, R. Ak, A. Sharma, et al., "Samanantar: The Largest Publicly Available Parallel Corpora Collection for 11 Indic Languages," *Transactions of the Association for Computational Linguistics*, vol. 10, pp. 145–162, 2022.

SGOU

Unit 3
Translation Techniques

Learning Outcomes

After completing this unit, learner will able to:

- trace the historical development of Machine Translation (MT) from rule-based classical systems to statistical approaches
- explain the Direct, Transfer, and Interlingua strategies of classical Machine Translation and place them on the Vauquois triangle
- describe the noisy-channel formulation that underlying Statistical Machine Translation (SMT)
- define word alignment and identify different alignment patterns between a source sentence and a target sentence
- describe how alignment models (the IBM Models) are trained using the Expectation-Maximisation (EM) algorithm
- compare the strengths and limitations of rule-based and statistical approaches to Machine Translation

SGOU

Prerequisites

Studying translation techniques is important for understanding how computers convert text from one natural language into another while preserving its meaning and context. In real life, when a person translates a Malayalam sentence into English using an online translation system, the system must determine how words, phrases, and sentence structures in one language correspond to those in another language. Classical Machine Translation introduces the basic approaches used in early translation systems, while Direct Translation translates source language expressions more directly into the target language. Transfer based translation uses linguistic rules to transform the structure of the source language into an appropriate target language structure, whereas Statistical Machine Translation learns translation patterns from previously translated language data. Phrase based translation improves translation by considering groups of words rather than translating each word independently, which is useful when expressions have different meanings as a group. Alignment in Machine Translation identifies relationships between words and phrases in parallel texts, such as determining which Malayalam



words correspond to particular English words. Training alignment models enables a system to learn these word or phrase correspondences automatically from large collections of translated sentences. For example, while translating a sentence containing the Malayalam expression "സുഖമാണോ?" into English, a translation system needs to learn that the appropriate English expression is "How are you?" rather than translating each word separately. Learning these techniques provides the foundation for understanding how statistical and rule based translation systems are designed, trained, and evaluated. Therefore, these topics prepare learners to study more advanced approaches to machine translation and understand how translation systems handle differences in vocabulary, grammar, word order, and context.

Keywords

Direct Translation, Transfer, Interlingua, Vauquois Triangle, Bilingual Dictionary, Transfer Rule, Statistical Machine Translation (SMT), Noisy-Channel Model, Language Model, Translation Model, Fluency, Fidelity, Decoder

Discussion

5.3.1 Introduction to Translation Techniques

Unit 2 showed why Machine Translation is challenging. Languages arrange verbs differently, divide meanings in different ways, use word endings for grammar to different extents, and follow different rules about leaving out pronouns. Understanding these issues shows what a translation system must deal with, but it does not explain how such a system can be built.

This unit focuses on how translation systems are built. It examines the main methods developed since the 1950s, beginning with rule-based systems and then moving towards statistical methods, which gained prominence during the 1990s. Each successive approach attempted to address important limitations of earlier approaches, so the sequence presents the development of translation techniques rather than simply listing separate methods.

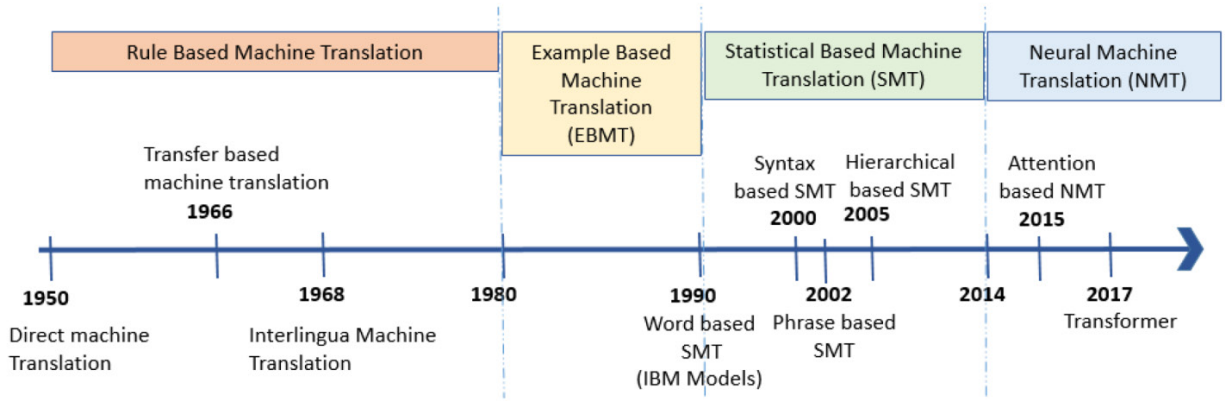


Fig 5.3.1: Evolution of Machine Translation (MT) approaches over the years

Figure 5.3.1 provides a visual overview of this evolution. The figure is not intended to represent a perfectly linear replacement of one technology by another; rather, it highlights the major methodological shifts in the development of MT systems.

Most real-world systems now use neural methods instead of the approaches covered in this unit. However, these earlier methods are still important to study for two reasons. First, concepts such as alignment, phrase tables, and the distinction between fluency and fidelity are important in understanding translation. Second, the linguistic problems addressed by these methods still exist; neural systems handle them in different ways.

5.3.2 Classical Machine Translation

Machine Translation began to be seriously explored in the late 1940s, soon after the development of the computer. In 1954, the first public demonstration of an MT system prototype generated considerable interest, and the following decade saw the development of many ideas that influenced later approaches.

The earliest systems were based on what is now called the classical approach, in which translation is driven by hand-crafted linguistic knowledge. A direct system relies mainly on a large bilingual dictionary of the source and target languages. A transfer system additionally uses rules written by linguists to capture syntactic, semantic, and morphological regularities and the correspondences between languages.

Classical MT architectures can be grouped into three general models, which differ in how deeply they analyse the source text before producing the target text. A common way to visualise their relationship is the Vauquois triangle, shown in Figure 5.3.2.

The triangle illustrates two aspects. Moving upwards represents an increasing depth of linguistic analysis on the source side and generation on the target side. At the same time, moving upwards represents a decreasing need for language-pair-specific transfer rules. At the base, much of the system's knowledge concerns direct correspondences between the source and target languages. At the apex, translation passes through an interlingual representation, so no language-pair-specific transfer rules are required.

The three architectures are examined in turn below. Sections 5.3.3 and 5.3.4 then discuss the statistical approaches that followed them.

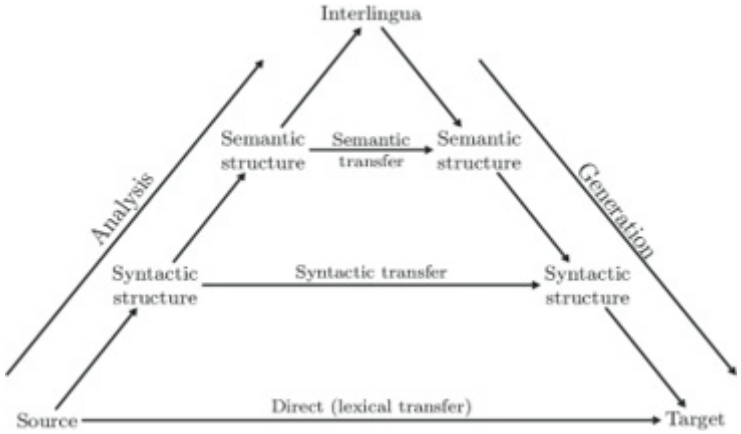


Fig 5.3.2: Schematic representation of Machine Translation architectures using the Vauquois triangle

SGOU

5.3.2.1 Direct Translation

Direct translation is one of the earliest and simplest approaches to Machine Translation. It forms the foundation of the translation hierarchy discussed in this unit. The basic idea is straightforward: the system processes the source-language sentence and translates its words using a bilingual dictionary, together with a limited set of morphological and local reordering rules.

A direct translation system does not normally construct a complete representation of the sentence. Instead, it works mainly with individual words and small local sequences. A dictionary entry may therefore contain more than a simple word-to-word substitution. It may specify the appropriate target-language form, identify a grammatical feature, or trigger a small local reordering rule.

The basic process can be represented as:

Source sentence → Word lookup → Local rules → Target sentence

Illustrative Example 1: English to Malayalam

English sentence:

The boy reads a book.

First, look up the individual words in the bilingual dictionary:

the → may not have a direct equivalent in this context

boy → ബാലൻ / കുട്ടി

reads → വായിക്കുന്നു

a → ഒരു

book → പുസ്തകം

If the system simply concatenates the translated words:
ബാലൻ വായിക്കുന്നു ഒരു പുസ്തകം.

The individual words are understandable, but the word order is not the most natural Malayalam expression. A more appropriate translation would be:

ബാലൻ ഒരു പുസ്തകം വായിക്കുന്നു.

The example shows an important limitation of direct translation. The system cannot simply replace English words with Malayalam words and expect the resulting sequence to be a natural Malayalam sentence. It must also account for the structural differences between the two languages.

Example 2: She drinks tea.

A simple word-level mapping might be:

she → അവൾ

SGOU

drinks → കുടിക്കുന്നു

tea → ചായ

Result:

അവൾ കുടിക്കുന്നു ചായ.

A local reordering rule could change this to:

അവൾ ചായ കുടിക്കുന്നു.

This demonstrates that a direct system can be improved by adding local reordering rules.

Why Does Direct Translation Need Rules?

Languages do not always express the same information in the same order.

English - Subject–Verb–Object (SVO) order:

She → drinks → tea.

Malayalam - Subject–Object–Verb (SOV) order:

അവൾ → ചായ → കുടിക്കുന്നു.

Therefore, even a simple English-to-Malayalam system needs some knowledge of word order.

5.3.2.2 The Transfer Approach

The transfer approach is a middle-ground method that addresses the reordering problem directly. Instead of translating word by word, a transfer system uses three main steps.

First, the system analyses the source text to determine how the words are related grammatically. Next, it applies transfer rules to transform this structure into one that matches the target language. Finally, it generates the target sentence from the transformed structure.

What a Transfer Rule Looks Like

A transfer rule operates on sentence structure rather than on individual words. For example, to handle English–Spanish adjective order, a transfer rule could state:

Noun Phrase [Adjective + Noun] → Noun Phrase [Noun + Adjective]

This rule can reorder adjective–noun combinations without requiring a separate rule for each word. The main advantage of using this structure is that one rule can apply to many similar cases.

Similarly, a rule for English-to-Hindi verb order might specify that a Subject–Verb–Object (SVO) structure should be transformed into a Subject–Object–Verb (SOV) structure. This rule can be applied to sentences that follow this pattern.

SGOU

Limitations

Transfer systems are more powerful than direct systems, but they still require transfer rules for each language pair.

For example, a system that supports five languages may require rules for 20 directed language pairs. Creating these rules requires expert knowledge of both languages and can therefore be costly and time-consuming.

5.3.2.3 The Interlingua Approach

The interlingua approach builds on the logic of transfer by going a step further. Instead of mapping a source structure directly onto a target structure, the system analyses the source text into an abstract meaning representation called an interlingua. This interlingua is not tied to any specific language. The system then generates the target-language sentence from this representation.

One advantage of this approach is that it reduces the language-pair problem.

For example, to support five languages, the system requires five analysers and five generators, for a total of ten components, instead of twenty sets of directed transfer rules. If a sixth language is added, only two additional components are required rather than ten new transfer-rule sets.

However, creating an interlingua that works well for real translation is very challenging. The representation must capture the distinctions required by different languages, even when some languages do not express those distinctions explicitly. Because of this, practical interlingua systems have mainly been developed for restricted domains with a limited range of meanings.

5.3.2.4 Why Classical Approaches Gave Way

As high-quality Machine Translation proved difficult to achieve, greater attention was given to evaluation and fundamental research. This development culminated in the ALPAC report of 1966, which criticized the progress and cost-effectiveness of Machine Translation at the time and recommended greater emphasis on basic research and tools for human translators. In the United States, the report contributed to a major reduction in MT funding, and the field entered a period sometimes described as the MT winter.

Some developers continued their work, and successful narrow-domain systems were built. Météo, which translates weather forecasts from English into French, is a well-known example. However, the general translation problem remained open until a different approach became possible.

5.3.3 Statistical Machine Translation

Statistical Machine Translation (SMT) is a Machine Translation approach that translates text from one language to another by learning statistical patterns from large collections of bilingual text. It selects the target-language sentence that has the highest probability of being the correct translation of the source sentence. This is achieved using probabilistic models learned from bilingual training data.

SGOU

The objective of SMT is to find the target sentence (f^*) that maximises the conditional probability:

$$f^* = \arg\max_f P(f \mid e)$$

where:

- e = Source language sentence
- f = Target language sentence
- $P(f|e)$ = Probability that the target sentence
- f is the correct translation of the source sentence e

Using Bayes' Theorem, the formula can be rewritten as:

$$f^* = arg\ arg\ P(e|f)\ P(f)$$

where:

- $P(e|f)$ = Translation Model, which measures how well the target sentence (f) accounts for the source sentence (e)
- $P(f)$ = Language Model, which estimates how probable and fluent the target-language sentence is

Example

Statistical Machine Translation selects a target-language sentence using probabilities learned from a bilingual corpus. Different SMT models use different translation units. Early models operated primarily at the word level, while later phrase-based models learned translations of sequences of words.

The translated phrases are then rearranged to match the grammatical structure of the target language. The figure below illustrates this process using an English-to-French translation example.

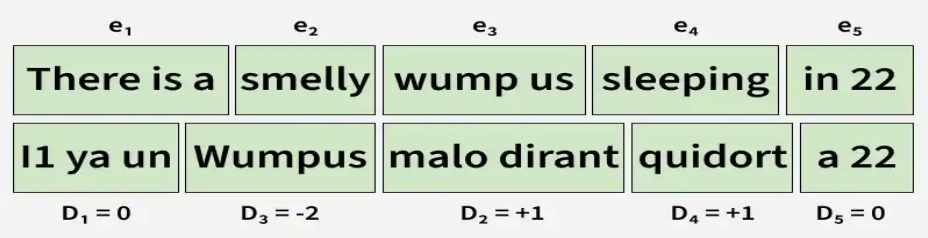


Fig. 5.3.3: Example of a statistical approach to Machine Translation

Step 1: Divide the Source Sentence into Phrases:

The English input sentence is divided into smaller phrases represented as e_1, e_2, e_3, e_4, and e_5. Each phrase contains one or more words that can be translated as a single unit.

SGOU

Step 2: Translate Each Phrase

Each English phrase is translated into its corresponding French phrase (f_1, f_2, f_3, f_4, and f_5) using the translation model learned from bilingual corpora. Phrase-level translation helps preserve local context better than translating individual words separately.

Step 3: Reorder the Translated Phrases

Since English and French follow different word orders, the translated phrases cannot always remain in the same sequence. The distortion model rearranges the phrases to follow the target language's grammar.

Step 4: Apply Distortion Values

In the simplified illustration, each translated phrase has a distortion value (D) indicating its movement during reordering.

- D = 0 → The phrase remains in the same position.
- D > 0 → The phrase moves in one direction.
- D < 0 → The phrase moves in the opposite direction.

In the figure, the distortion values illustrate how some phrases retain their positions while others are reordered during translation.

Step 5: Generate the Final Translation

After the phrases have been translated and reordered, the decoder combines them to produce the final French sentence. The aim is to produce a translation that is both grammatically fluent and faithful to the meaning of the source sentence.

5.3.4 The Phrase-Based Translation Model

The earliest statistical systems, such as the IBM Models of the early 1990s, operated mainly at the word level. This was a serious limitation because translations do not always correspond word by word.

Example 1:

English: She drinks tea.

Malayalam: അവൾ ചായ കുടിക്കുന്നു.

drinks tea ↔ ചായ കുടിക്കുന്നു

Example 2:

English : "slap"

Spanish: "daba una bofetada"

Here, several Spanish words together correspond to a single English expression. This shows why translating larger sequences of words can be more effective than translating individual words separately.

SGOU

Step 1: Segmentation.

The source sentence is divided into a sequence of phrases. The segmentation is a choice because the same sentence can be divided in different ways, and the system may consider several possible segmentations.

Step 2: Translation.

Each phrase is translated using a phrase table, a collection of phrase pairs learned from a parallel corpus, along with their associated probabilities.

Step 3: Reordering.

The translated phrases are rearranged to suit the word order of the target language. The model that governs this rearrangement is called the distortion model. It assigns a cost to moving a phrase away from its original position, discouraging excessive reordering while allowing the movements required by the target language.

5.3.5 Word Alignment in Machine Translation

Both the translation model P(f|e) and the phrase table depends on knowing which words in a source sentence correspond to which words in its translation. This correspondence is called word alignment, and it is one of the central concepts in Statistical Machine Translation.

Consider the following English–Malayalam sentence pair:

English: The boy reads a book.

Malayalam: കുട്ടി ഒരു പുസ്തകം വായിക്കുന്നു.

The parallel corpus tells us that these two sentences are translations of each other. It does not tell the system that boy corresponds to കുട്ടി, that book corresponds to പുസ്തകം, or that reads corresponds to വായിക്കുന്നു. These correspondences must be inferred from many sentence pairs across the corpus, and this inference is the task of word alignment.

The problem becomes more difficult when the two languages differ in word order:

English: She drinks tea.

Malayalam: അവൾ ചായ കുടിക്കുന്നു.

English follows SVO order, so the verb drinks comes second. Malayalam follows SOV order, so the verb കുടിക്കുന്നു comes last. The alignment must therefore connect the second English word to the third Malayalam word. An alignment model must represent correspondences that cross positions in this way rather than simply pairing words that occur in the same position.

5.3.5.1 Types of Alignment

Before inferring alignments, it is important to understand the different forms they can take. Four useful cases are one-to-one, one-to-many, many-to-one, and many-to-many alignment. One-to-One Alignment

SGOU

The simplest case occurs when one source word corresponds approximately to one target word.

A simplified alignment is:

book → പുസ്തകം,

book → किताब

One-to-one alignment is easy to understand, but it is not sufficient to describe translation in general. Languages frequently express grammatical information differently.

♦ **One-to-Many Alignment**

A single word in the source language may correspond to several words in the target language.

Example:

English Malayalam

reading → വായിച്ചു കൊണ്ടിരിക്കുന്നു

♦ **Many-to-one**

Several source-language words may correspond to a single target-language word or form.

Example:

English: did not go

Malayalam: പോയില്ല

♦ **Many-to-many alignment**

In this case, groups of words or phrases correspond across the two languages, but a simple one-to-one word mapping may not be possible.

Example:

English: is not able to go

Malayalam: പോകാൻ കഴിയുന്നില്ല

5.3.5.2 Representing an Alignment

To work with alignments computationally we need a compact representation. The IBM models use an alignment vector.

Suppose the source sentence contains (l) words and the target sentence contains (m) words. The alignment is represented as a vector of length (m), with one entry for each target word. Each entry records the source-word position to which that target word is aligned. The value **0** is reserved for a special NULL position.

SGOU

Example

English: Mary reads a book

Malayalam: മേരി ഒരു പുസ്തകം വായിക്കുന്നു

Numbering both sentences:

Position	1	2	3	4
English	Mary	reads	a	book
Malayalam	മേരി	ഒരു	പുസ്തകം	വായിക്കുന്നു

Reading the alignment, മേരി is aligned with English position 1, ഒരു with position 3, പുസ്തകം with position 4, and വായിക്കുന്നു with position 2. The alignment vector is therefore:

a = (1, 3, 4, 2)

Notice that the fourth entry is 2. The Malayalam verb occurs at the end of the sentence, while the English verb occurs in the second position. The alignment vector records this reordering.

The NULL position

Sometimes a target word has no counterpart in the source sentence. For example, Malayalam may require a grammatical element that is not explicitly expressed in English. A target word of this kind is aligned to position **0**, meaning NULL, a notional empty position indicating that the target word has no corresponding source word.

What the vector can and cannot express

Several target positions may point to the same source position, so the alignment vector can naturally represent one-to-many alignments. If no target position points to a particular source word, that word has zero fertility; it is not translated.

In the standard alignment-vector representation, each target position points to a single source position. Therefore, a target word cannot directly point to several source words at the same time.

This limitation is one reason phrase-based models became important. By using phrases rather than individual words as the unit of translation, they can represent correspondences between sequences of words in the source and target languages.

5.3.6 Training Alignment Models

We now reach a problem that has been present from the beginning. To estimate translation probabilities, we need aligned text. However, aligning text requires translation probabilities. Initially, we have neither.

The Expectation-Maximisation (EM) algorithm solves this circular problem. It starts

SGOU

with an initial estimate and lets the alignment and translation probabilities improve as it repeatedly processes the training data.

5.3.6.1 The Intuition

Before looking at the mathematics, consider what a person might infer from a small bitext without any additional information. Consider the following English–French examples:

the house ↔ la maison

the flower ↔ la fleur

The word la appears in both French sentences, while the appears in both English sentences. This repeated co-occurrence suggests that the two words may correspond.

Now consider maison. It appears only in the first sentence pair, together with the and house. However, the also appears in the second sentence pair, where maison does not occur. This provides stronger evidence for an association between maison and house than between maison and the.

This is the basic intuition behind EM. The algorithm performs this kind of reasoning using probabilities and repeatedly re-estimates them until the values become stable.

5.3.6.2 The Algorithm

EM alternates between two steps. When training an alignment model with EM, these two steps help each other until the probabilities settle down.

Initialization

All translation probabilities t(f|e) are initially set to equal values. The model begins by assuming every French word is equally likely to translate any English word. In other words, it starts with no prior knowledge.

The E-Step

Using the current probabilities, the algorithm estimates how likely each possible alignment is for every sentence pair and uses these estimates to collect fractional counts. If a French word could have originated from three different English words, the count is distributed among them according to their current probabilities rather than being assigned entirely to one word.

The M-step

Using these fractional counts, the translation probabilities are updated by normalization. The count for a particular word pair is divided by the total count associated with that English word across all French words.

Iteration

The E-step and M-step are repeated. Each iteration produces updated translation probabilities, which improve alignment estimates and, in turn, further probability updates. This cycle continues until the probabilities change very little.

SGOU

This general procedure is applied with increasing sophistication across successive IBM Models, with each model adding structural information the previous model did not include.

Table 5.3.1: Key ideas introduced in IBM Models 1–4 for statistical word alignment

Model	Key Idea Added
IBM Model 1	Uses purely lexical translation probabilities t(f|e) and assumes all possible alignments are equally likely. It has no notion of word position.
IBM Model 2	Adds an alignment probability that depends on the positions of the source and target words and the sentence lengths.
IBM Model 3	Introduces fertility, which represents the number of target words that a source word may generate. A source word may generate several target words or no target word.
IBM Model 4	Refines the reordering model by considering word classes and relative word positions, providing a more realistic treatment of word reordering.

5.3.6.3 A Worked Example

If we run EM on the two-sentence bitext above, we obtain the following results, using three English words and three French words in the vocabulary.

At the beginning, every translation probability (t(f|e)) is set to (1/3), or 0.333. The model initially assumes that la, maison, and fleur are equally likely translations of the, house, and flower.

After the first iteration, the probabilities begin to differ:

$t(la|the) = 0.500$

$t(maison|house) = 0.500$

$t(fleur|flower) = 0.500 t(maison|the) = 0.250$

$t(fleur|the) = 0.250$

After the third iteration, the differences between the probabilities have grown:

$t(la|the) = 0.692$

$t(maison|house) = 0.640$

$t(fleur|flower) = 0.640$

SGOU

$t(maison|the) = 0.154$

$t(fleur|the) = 0.154$

This pattern agrees with the earlier intuition. The probability of maison given house increases, while the probability of maison given the decreases. The word the occurs with several French words and therefore distributes its probability among them, whereas house has a stronger association with maison.

With repeated iterations and sufficient data, stronger alignment preferences emerge even though the alignments are not explicitly marked in the training data.

5.3.6.4 Why Model 1 Is Trained First

One practical point explains the order in which the IBM Models are usually trained. **IBM Model 1** has a useful mathematical property: its likelihood function does not suffer from the same local-optimum problems as the more advanced models. As a result, EM training for Model 1 provides a reliable starting point.

More advanced IBM Models are more complex and can converge to less favourable local solutions. Therefore, the usual approach is to train Model 1 first, use its estimated probabilities to initialise Model 2, and then continue through the later models. Each model thus begins from a better starting point provided by the previous one.

A widely used toolkit for training these alignment models was GIZA++, while phrase-based translation systems built on top of such alignments were commonly developed using the Moses toolkit.

Recap

- Classical Machine Translation (MT) relies mainly on linguistic rules and language-specific resources such as dictionaries and grammatical rules.
- Direct Translation is one of the simplest rule-based approaches, in which source-language words are translated using a bilingual dictionary together with limited local reordering.
- Direct translation has difficulty handling large-scale syntactic differences between languages such as English, Hindi, and Malayalam.
- Transfer-based translation analyses the source sentence and transforms its syntactic representation into a structure suitable for the target language.
- Statistical Machine Translation (SMT) learns translation patterns automatically from large collections of parallel text.
- A parallel corpus contains sentences in two or more languages that are translations of one another.

SGOU

- SMT estimates the likelihood of possible translations using statistical models rather than relying entirely on manually written linguistic rules.
- Phrase-based SMT translates sequences of words or phrases instead of treating every word as an independent translation unit.
- Word alignment identifies correspondences between words in a source sentence and their translations in a target sentence.
- A one-to-one alignment occurs when one source word corresponds to one target word.
- A one-to-many alignment occurs when one source word corresponds to several target words.
- Many-to-one and many-to-many alignments demonstrate why simple word-level translation is sometimes insufficient.
- IBM Model 1 primarily learns lexical translation probabilities, while later models progressively introduce position, fertility, and reordering information.
- Fertility refers to the number of target-language words generated by a source-language word in the alignment model.
- Alignment training is necessary because parallel corpora normally identify corresponding sentences but do not explicitly provide word-level correspondences.

Objective Type Questions

1. Which MT approach primarily uses a bilingual dictionary and local translation rules?
2. What type of corpus contains corresponding texts in two languages?
3. What is the term used for correspondence between source and target words in SMT?
4. Which alignment occurs when one source word corresponds to one target word?
5. What is meant by zero fertility?
6. What is the main translation unit in phrase-based SMT?
7. What special position is represented by 0 in the IBM alignment vector?
8. Which language generally follows SVO word order: English or Hindi?

SGOU

9. Which word-order pattern is commonly associated with Hindi and Malayalam?

10. Why are word alignments learned from a parallel corpus rather than read from it?

11. What is the main purpose of a bilingual corpus in Statistical Machine Translation?

12. What is the basic idea of phrase-based translation?

13. What does a parallel corpus contain?

14. What is the process of identifying correspondences between source and target words called?

15. Why is word alignment important in Statistical Machine Translation?

Answers to Objective Type Questions

1. Direct translation
2. Parallel corpus
3. Word alignment
4. One-to-one alignment
5. A source word produces no target-language word.
6. Phrase
7. NULL position
8. English
9. SOV
10. Because parallel corpora usually provide sentence-level correspondences, not word-level alignments.
11. To provide training data for learning translation probabilities and patterns.
12. To translate sequences of words as phrases rather than translating each word independently.

SGOU

13. Corresponding sentences or texts in two or more languages.

14. Word alignment

15. It helps identify which source-language words correspond to target-language words and supports the learning of translation models.

Assignments

1. Write a short note on Direct Machine Translation with a suitable example.

2. Explain the difference between Direct Translation and Transfer-based Translation.

3. Take any five English sentences and write their Hindi or Malayalam translations. Identify the words or phrases that change position during translation.

4. What is word alignment? Explain the four basic types of alignment with suitable examples.

5. Write a short note on Statistical Machine Translation and explain the role of a parallel corpus.

6. Consider the following English–Hindi and English–Malayalam sentence pairs:

   English: She drinks tea.

   Hindi: वह चाय पीती है।

   Malayalam:അവൾ ചായ കുടിക്കുന്നു

   Identify the likely word correspondences and explain how the word order differs between English and the two target languages.

Reference

1. S. Sharma, M. Diwakar, P. Singh, V. Singh, S. Kadry, and J. Kim, "Machine Translation Systems Based on Classical-Statistical-Deep-Learning Approaches," *Electronics*, vol. 12, no. 7, Art. no. 1716, 2023.

2. A. Lopez, "Statistical Machine Translation," *ACM Computing Surveys*, vol. 40, no. 3, pp. 1–49, 2008.

SGOU

3. P. Koehn, *Statistical Machine Translation*. Cambridge University Press, 2010.

4. S. Vajjala, B. Majumder, A. Gupta, and H. Surana, *Practical Natural Language Processing: A Comprehensive Guide to Building Real-World NLP Systems*. O'Reilly Media, 2020.

5. D. Jurafsky and J. H. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*.

6. C. D. Manning, P. Raghavan, and H. Schütze, "XML Retrieval," in *Introduction to Information Retrieval*. Cambridge University Press, 2008.

7. M. Elbayad, *Rethinking the Design of Sequence-to-Sequence Models for Efficient Machine Translation*, Ph.D. dissertation, Université Grenoble Alpes, 2020.

8. GeeksforGeeks, "Statistical Machine Translation of Languages in Artificial Intelligence." [Online]. Available: https://www.geeksforgeeks.org/artificial-intelligence/statistical-machine-translation-of-languages-in-artificial-intelligence/ [Accessed: 6 September 2026].

Suggested Reading

1. S. Bird, E. Klein, and E. Loper, *Natural Language Processing with Python*. O'Reilly Media, 2009.

2. M. Song, W. C. Kim, D. Lee, G. E. Heo, and K. Y. Kang, "PKDE4J: Entity and Relation Extraction for Public Knowledge Discovery," *Journal of Biomedical Informatics*, vol. 57, pp. 320–332, 2015.

SGOU

Unit 4

Symmetrizing Alignments for Phrase-Based MT

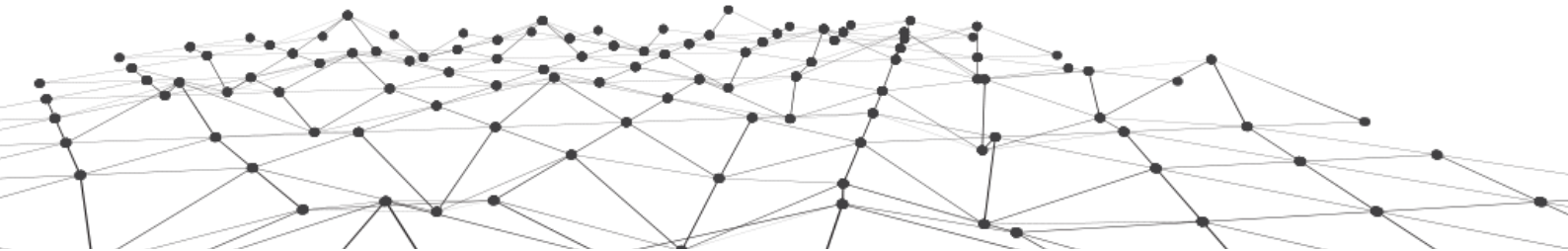
Learning Outcomes

After completing this unit, learner will able to:

- explain the need for alignment symmetrization and the limitations of one-direction alignment, including intersection, union, and growdiag- final heuristics
- define alignment consistency and explain how it is used to extract phrase pairs from a symmetrized alignment
- explain decoding in Phrase Based Statistical Machine Translation and describe translation hypotheses and the role of the coverage vector
- describe stack decoding and explain how hypotheses are organized according to their coverage count
- explain histogram pruning and threshold pruning and their importance in reducing the computational complexity of decoding

Prerequisites

Knowledge of symmetrizing alignments and decoding for phrase-based statistical machine translation is important for understanding how translation systems identify meaningful relationships between words and phrases in different languages. In real life, when a user translates a Malayalam sentence into English using an online translation system, words in the two languages may not follow the same order or have a direct one-to-one correspondence. Symmetrizing alignments combines alignment information from both translation directions to produce more reliable word and phrase relationships. Phrase-based translation uses these relationships to translate groups of words rather than translating each word independently, which can help preserve the intended meaning of a sentence. Decoding is then required to select the most suitable sequence of translated phrases from the available alternatives. For example, a Malayalam expression may need to be translated into an English phrase with a different word order than the original sentence. Knowledge of these concepts provides the foundation for understanding how statistical machine translation systems generate meaningful translations by considering phrase relationships, word order, and alternative translation possibilities.



SGOU

Keywords

Symmetrization, One-Directional Alignment, Intersection, Union, Parallel Corpus, Grow-Diag-Final, Alignment Consistency, Phrase Extraction, Phrase Pair, Decoder, Translation Hypothesis, Coverage Vector, Stack Decoding, Histogram Pruning, Threshold Pruning

Discussion

5.4.1 Introduction

Unit 3 left two gaps in the discussion of phrase-based translation, and this unit addresses them.

The first gap concerns the phrase table. A phrase-based system translates by looking up phrase pairs, such as the English–Hindi phrase segments illustrated in Figure 5.4.1, along with thousands of similar pairs. However, the IBM Models trained using EM do not directly produce phrase pairs. They produce word alignments, in which each target word is aligned to a source word. Moving from these word alignments to a phrase table requires further processing, and the first step is symmetrization.

The second gap concerns decoding. The noisy-channel formulation tells us that we want the target sentence that maximises:

$$P(e|f) \times P(f)$$

It does not tell us how to find that sentence. Searching every possible translation is impractical. With several possible translations for each phrase and different possible reorderings of those phrases, the number of candidate translations grows rapidly even for a short sentence. The decoder must therefore find a good translation without examining every possible candidate.

These two topics complete the phrase-based translation pipeline. With them in place, the process from parallel corpus to translated sentence becomes a complete system.

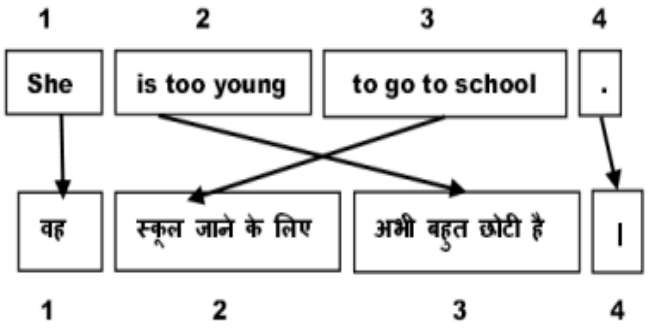


Fig 5.4.1: Example of phrase segmentation and reordering in English–Hindi translation

SGOU

5.4.2 Symmetrizing Alignments for Phrase-Based MT

Recall the limitation of the alignment vector discussed in Unit 3. Each target position contains exactly one source position so that a target word can point to only one source word. The representation can express one-to-many alignments, but not many-to-one alignments.

This is not a minor technical limitation. It makes the alignment model asymmetric in a way that natural-language translation is not. When translating from English into Spanish, the word "slap" may correspond to several Spanish words. When translating from Spanish into English, those same Spanish words may correspond to a single English word, which the alignment representation cannot express directly.

Thus, whichever translation direction is chosen, some alignment information may be lost.

5.4.2.1 Running the Alignment Twice

The solution is straightforward. Instead of choosing only one alignment direction, the alignment process is run twice: once from source to target and once from target to source. The two resulting alignments are then combined into a single symmetric alignment. This combination process is called symmetrization.

Each direction produces its own alignment for the same sentence pair, and the two alignments may not agree. Figure 5.4.2 illustrates the two directional alignments together with their combined alignment.

Figure 5.4.2 shows the word alignments produced in both translation directions. The English-to-Spanish and Spanish-to-English alignments do not always identify the same word correspondences. Symmetrization combines the information from both directions to produce a more complete alignment, which is then used for phrase extraction.

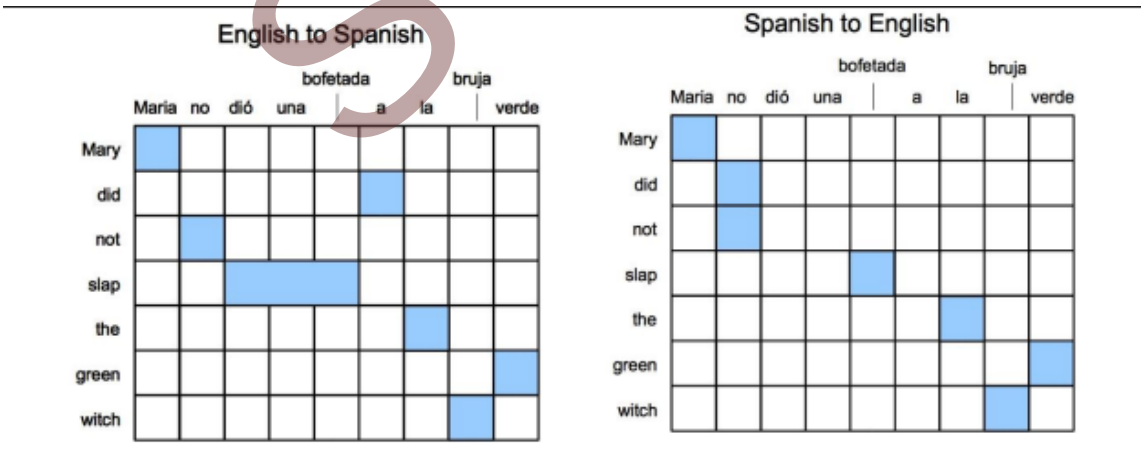


Fig 5.4.2: Source-to-target and target-to-source word alignments for the same English–Spanish sentence pair

SGOU

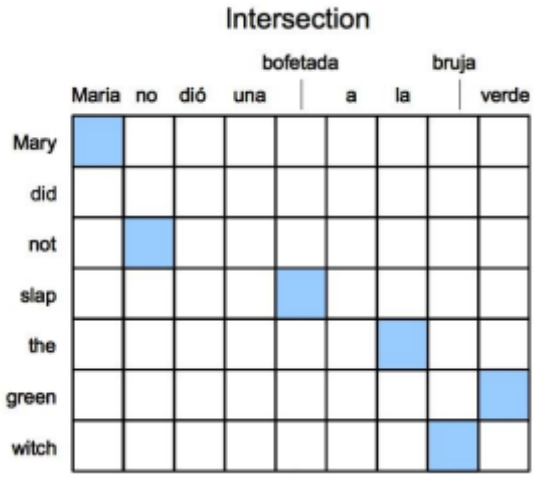


Fig 5.4.3: Intersection of source-to-target and target-to-source word alignments

Figure 5.4.3 shows the intersection of the source-to-target and target-to-source alignments. Only the word pairs that appear in both directional alignments are retained. Because both alignment directions support these links, the intersection usually gives high-precision alignments, although some valid correspondences may be missed.

5.4.2.2 Intersection and Union

The two most direct ways to combine the directional alignments are intersection and union.

Intersection keeps only those alignment links that appear in both directions. If both alignment runs identify that Mary corresponds to Maria, that link is retained. If only one direction identifies a link, it is discarded.

Union keeps every alignment link that appears in either direction. A link found by only one of the two alignment runs is also retained.

Intersection usually produces a more conservative alignment with high precision and lower recall. Since a link is kept only when both directions agree, the retained links are more likely to be correct. However, some valid links may be lost, which can reduce the number of useful phrase pairs that can be extracted.

Union usually gives higher recall and lower precision. It keeps more valid links, including those found by only one direction, but it can also retain incorrect links. These incorrect links may lead to noisy phrase pairs in the phrase table.

Therefore, neither intersection nor union alone is ideal. Intersection may discard too many useful links, while union may retain too many incorrect ones.

5.4.2.3 Grow-Diag-Final

The standard solution takes a middle path. The grow-diag-final heuristic begins with the intersection and then selectively adds alignment points from the union.

SGOU

The name describes the main phases of the procedure:

- Grow: Begin with the intersection and consider adding union points that are adjacent to an existing alignment link. Adjacency includes diagonal neighbours, which is what diag refers to.
- Final: After the growing stage, make a final pass and add remaining union points when at least one of the two words involved is still unaligned.

The crucial restriction is that a point is added only when at least one of the two words it links is currently unaligned. This prevents the heuristic from adding unnecessary links between words that are already well aligned.

The result is an alignment that retains much of the precision of intersection while recovering much of the recall of union. Grow-diag-final and related heuristics became widely used in phrase-based MT systems, including systems based on the Moses toolkit.

Heuristic What it Keeps

Table 5.4.1: Comparison of intersection, union, and grow-diag-final alignment symmetrization heuristics

Heuristic	What it keeps	Precision	Recall
Intersection	Only links found in both directions	High	Low
Union	Links found in either direction	Low	High
Grow-diag-final	Intersection plus selected union points	Good	Good

SGOU

5.4.2.4 Extracting Phrase Pairs

With a symmetric alignment in hand, we can build the phrase table. The rule that determines which phrase pairs may be extracted is called alignment consistency.

The alignment can be represented as a grid, with source words along one axis and target words along the other. A mark is placed in each cell where an alignment link exists. A candidate phrase pair corresponds to a rectangular region of this grid, containing a contiguous sequence of source words and a contiguous sequence of target words.

A phrase pair is consistent if every alignment link involving a word inside the selected source or target phrase remains within the corresponding phrase pair. If a word inside the selected phrase is aligned to a word outside it, the phrase pair is inconsistent and must be rejected.

Figure 5.4.4 illustrates extracting phrase pairs consistent with the word alignment.

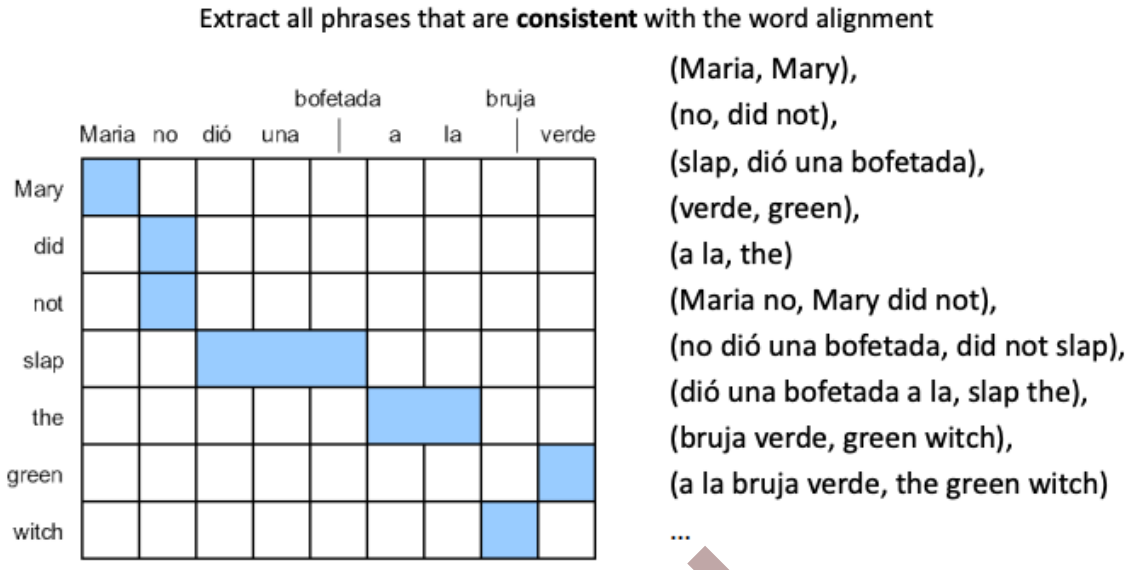


Fig 5.4.4: Extraction of phrase pairs consistent with the word alignment

5.4.2.5 Scoring the Phrase Table

Phrase extraction gives us the phrase pairs, but not their probabilities. These probabilities are estimated by counting how often each phrase pair occurs across the corpus.

The probability of a target phrase given a source phrase is calculated as:

$$\frac{count(source\ phrase,\ target\ phrase)}{count(source\ phrase)}$$

If "did not slap" is extracted 1,000 times from the corpus, and in 850 of those cases its corresponding target phrase is "no daba una bofetada", then the probability of that phrase pair is:

$$P(no\ daba\ una\ bofetada \mid did\ not\ slap) = \frac{850}{1000} = 0.85$$

The remaining **150** occurrences are distributed among the other translations observed for the same source phrase.

Production systems usually store several scores for each phrase pair rather than a single probability. These commonly include translation probabilities in both directions and lexical weights computed from the individual word probabilities within the phrase. The decoder combines these scores during translation.

5.4.3 Decoding for Phrase-Based Statistical MT

We now have everything the model needs: a phrase table containing translation probabilities, a language model, and a distortion model that scores possible reorderings. The remaining task is to use these components to produce the final translation.

SGOU

Recall the noisy-channel formulation from Unit 3. The objective is to find the target sentence that maximises the combined score of the translation model and the language model. The component that performs this search is called the decoder, and the process is known as decoding.

Figure 5.4.5 illustrates phrase-based translation from Spanish to English, showing how source phrases are translated and reordered to form the target sentence.

Slide adapted from Philipp Koehn

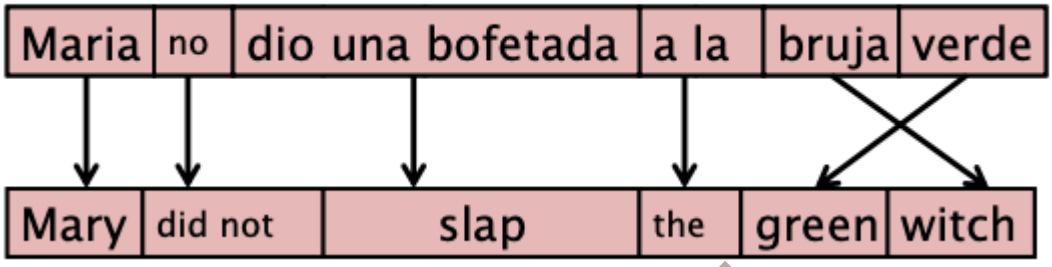


Fig 5.4.5: Example of phrase-based translation showing phrase segmentation and reordering from Spanish to English

5.4.3.1 Why Decoding is Hard

Consider what the decoder is required to do. For the seven-word sentence "Mary did not slap the green witch", it must consider:

- Every possible way of segmenting the source sentence into phrases
- For each phrase, every translation offered by the phrase table
- Every possible ordering in which the translated phrases may be arranged

These choices combine and make the search space grow rapidly. Even if the phrase table offers only five translations for each phrase, a sentence that can be segmented in several ways and whose translated phrases can be reordered in different ways produces far too many possible translations to examine exhaustively.

Exhaustive search is therefore impractical. The decoder must find a very good translation while examining only a small fraction of all possible candidates.

5.4.3.2 Translation Hypotheses

The decoder builds a translation step by step. Each partial translation is called a hypothesis, and it records three things:

- **The translation so far** – the target-language words produced up to that point
- **The coverage vector** – which source words have already been translated
- **The score** – the combined translation-model, language-model, and distortion score accumulated so far

The coverage vector is a key data structure. It is a sequence of bits, one for each source

SGOU

word, indicating whether each word has already been covered or is still untranslated.

The target sentence is built strictly from left to right. At each step, new target-language words are appended to the end of the translation generated so far; words are not inserted into the middle. This is important because the language model scores each word based on the words that precede it.

The source sentence, however, may be covered in any order. The decoder may translate a phrase from the middle or end of the source sentence before translating one from the beginning. This flexibility allows reordering. For example, to move a verb from the middle of an English sentence to the end of a Hindi sentence, the decoder can translate the other source phrases first and leave the verb until the final stage.

A hypothesis is complete when every bit in its coverage vector is set, indicating that all source words have been translated. The decoder then compares the complete hypotheses and returns the one with the highest score.

5.4.3.3 Expanding a Hypothesis

At each step, a hypothesis is expanded by choosing a source phrase that has not yet been covered and selecting one of its translations from the phrase table. The new hypothesis records:

- The translation so far, with the new target phrase appended
- The coverage vector, with the bits corresponding to the newly covered source words set
- The score, updated using three costs

The three costs added during each expansion are:

- **Translation cost** – obtained from the phrase-table probability
- **Language model cost** – based on the newly appended words and their preceding context
- **Distortion cost** – penalising large jumps from the previously covered source position

Because probabilities are often very small and may underflow when multiplied repeatedly, the scores are represented as logarithms and combined by addition. This is the same technique used for Naïve Bayes in Block III.

The distortion limit

If reordering is left completely unconstrained, the decoder may consider many unrealistic phrase orders, such as moving the final phrase of a sentence to the beginning.

To reduce such unnecessary possibilities, a distortion limit is imposed. This restricts how far the decoder may jump from the position of the previously translated source phrase.

SGOU

A distortion limit of six words is commonly used. This greatly reduces the search space while usually having little effect on translation quality.

5.4.3.4 Stack Decoding

We now come to the central algorithmic idea. The decoder generates many hypotheses, and most of them are poor. To keep the search manageable, poor hypotheses must be discarded early. However, hypotheses must be compared carefully.

Consider two hypotheses. The first has covered two source words and has a score of −0.9. The second has covered six source words and has a score of −2.4. Which one is better?

The question cannot be answered directly. The second hypothesis has a lower score partly because it has already covered more source words, and each expansion adds additional cost. Comparing the two hypotheses directly could therefore lead to an incorrect decision.

The solution is to compare only hypotheses that have covered the same number of source words. This is achieved by organising hypotheses into stacks, with one stack for each possible coverage count.

5.4.3.5 Pruning

Even with stacks, the number of hypotheses can grow too quickly. Each stack must therefore be limited in size. Two pruning strategies are commonly used, often together.

- **Histogram pruning** keeps a fixed number of the best hypotheses in each stack, commonly around 100, and discards the rest. The stack size controls the speed–quality trade-off: a larger stack explores more possibilities and may produce better translations, but requires more computation.
- **Threshold pruning** discards any hypothesis whose score is worse than the best score in the stack by more than a fixed threshold. When one hypothesis is clearly better than the others, pruning is more aggressive; when several hypotheses have similar scores, more are retained.

Both strategies are heuristics. Either may discard a hypothesis that could eventually have produced the best translation. Therefore, beam-search decoding is not guaranteed to find the globally optimal translation. This is accepted as a practical trade-off for keeping the search manageable.

Future cost estimation

One refinement addresses a limitation of pruning based only on the score accumulated so far. A hypothesis that has already translated easy source words may have a better current score than one that has translated more difficult words, even though the second hypothesis may ultimately lead to a better complete translation.

Decoders therefore estimate a future cost for each hypothesis. This approximates the cost of translating the remaining uncovered source words.

SGOU

Hypotheses are then ranked using their accumulated score and the estimated future cost, allowing the decoder to compare them based on overall prospects rather than only progress so far.

5.4.3.6 Limitations

Phrase-based systems worked well and were widely deployed, but several limitations eventually motivated the move towards neural approaches.

- **Long-distance reordering remained difficult.** The distortion limit that makes decoding tractable can also restrict the large rearrangements required by some language pairs. Translation between English and Hindi, where verbs often move from the middle of the sentence to the end, illustrates this challenge.
- **Context beyond the phrase was limited.** Each phrase was translated mainly using its phrase probability together with language-model information from the surrounding target words. This made it difficult to use broader sentence-level context when choosing the best translation.
- **The pipeline was long and errors could accumulate.** An error in word alignment could produce an incorrect phrase pair, which could then lead to a poor translation. Because the different stages were trained separately, errors introduced at one stage were difficult to correct later.

Neural Machine Translation replaced much of this multi-stage pipeline with end-to-end models, allowing translation decisions to be learned jointly within a single system.

Recap

- Symmetrization combines alignment information obtained in both translation directions into a single alignment.
- One-directional word alignment is limited because the alignment representation is asymmetric and cannot adequately capture all word correspondences.
- Intersection keeps only the alignment links that are present in both directional alignments.
- Intersection generally provides high precision but lower recall because it removes links that occur in only one direction.
- Union keeps all links found in either directional alignment.
- Union generally provides higher recall but lower precision because it may also retain incorrect alignment links.

SGOU

- Grow-diag-final provides a compromise between intersection and union by starting with the intersection and selectively adding links from the union.
- In the grow stage, neighbouring alignment points are added to the existing alignment.
- In the final stage, additional links may be added when at least one of the words involved is still unaligned.
- Phrase extraction uses a symmetrized alignment to identify valid source-language and target-language phrase pairs.
- A phrase pair is consistent when alignment links involving words inside the candidate phrase do not extend outside the corresponding phrase boundaries.
- Phrase-pair probabilities can be estimated from their frequency in the parallel corpus.
- Decoding is the process of searching for the best target-language translation using the phrase table, language model, and distortion information.
- A translation hypothesis represents a partial translation and records the translation produced so far, the coverage vector, and its accumulated score.
- Stack decoding and pruning make decoding computationally practical by keeping promising hypotheses and discarding less promising ones. Histogram pruning limits the number of hypotheses, while threshold pruning removes hypotheses whose scores are sufficiently worse than the best hypothesis.

Objective Type Questions

1. What is the process of combining alignments from two translation directions called?
2. Which heuristic keeps only links found in both alignment directions?
3. Which heuristic keeps links found in either alignment direction?
4. Which alignment heuristic provides high precision but low recall?
5. Which alignment heuristic provides high recall but lower precision?
6. Which heuristic starts with the intersection and adds selected links from the union?
7. What does the “diag” in grow-diag-final refer to?

SGOU

8. What is the main purpose of alignment symmetrization in phrase-based MT?

9. What property must a phrase pair satisfy before it can be extracted from an alignment?

10. What is the process of finding the best translation using the phrase table and language model called?

11. What records which source words have already been translated into a hypothesis?

12. What is a partial translation produced during decoding called?

13. Which pruning method keeps a fixed number of hypotheses in each stack?

14. Which pruning method removes hypotheses whose scores are worse than the best score by a specified margin?

Answers to Objective Type Questions

1. Symmetrization
2. Intersection
3. Union
4. Intersection
5. Union
6. Grow-diag-final
7. Diagonal neighbouring alignment points
8. To combine alignment information from both translation directions into a more complete alignment
9. Alignment consistency
10. Decoding
11. Coverage vector
12. Translation hypothesis
13. Histogram pruning
14. Threshold pruning

SGOU

Assignments

1. Explain the difference between intersection and union in alignment symmetrization.
2. Explain the grow-diag-final alignment heuristic with a suitable example.
3. Explain how phrase pairs are extracted from a symmetrized word alignment.
4. What is a translation hypothesis? Explain the role of the coverage vector and score.
5. Explain histogram pruning and threshold pruning used in Statistical Machine Translation decoding.

Reference

1. P. F. Brown, S. A. Della Pietra, V. J. Della Pietra, and R. L. Mercer, "The Mathematics of Statistical Machine Translation: Parameter Estimation," *Computational Linguistics*, vol. 19, no. 2, pp. 263–311, 1993.
2. C. D. Manning, "Phrase-Based Machine Translation," Stanford University, CS224N course material. [Online]. Available: https://web.stanford.edu/class/archive/cs/cs224n/
3. P. Koehn, *Statistical Machine Translation*. Cambridge University Press, 2010.
4. F. J. Och and H. Ney, "A Systematic Comparison of Various Statistical Alignment Models," *Computational Linguistics*, vol. 29, no. 1, pp. 19–51, 2003.
5. P. Koehn, F. J. Och, and D. Marcu, "Statistical Phrase-Based Translation," in *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology (NAACL-HLT)*, pp. 48–54, 2003.

SGOU

Suggested Reading

1. S. Bird, E. Klein, and E. Loper, *Natural Language Processing with Python*. O'Reilly Media, 2009.

SGOU

SREENARAYANAGURU OPEN UNIVERSITY

QP CODE: …… Reg. No : .....

Name : …….....

Model Question Paper- set-I

End Semester Examination

Bachelor of Computer Applications - Fifth Semester

B24DS04DE: Natural Language Processing

(CBCS - UG)

2025 -26 - Admission Onwards

Time: 3 Hours Max Marks: 70

Section A

Answer any 10 questions. Each carries one mark **(10 x 1 = 10)**

1. What is Natural Language Processing (NLP)?
2. Define unstructured data.
3. What is Named Entity Recognition (NER)?
4. What is the purpose of text cleaning?
5. What is tokenization?
6. What is the full form of VSM
7. What type of machine learning uses labelled training data?
8. What does P(y|X) represented in Bayes' theorem?
9. Which evaluation metric measures the proportion of actual entities that are correctly identified?
10. Which NLP task identifies named entities in a sentence?
11. Name one knowledge base used in distant supervision.
12. What type of annotation serves as the reference for system evaluation?

SGOU

13. What is word order typology?

14. What is meant by referential density?

15. What is a phrase-based translation model?

Section B

Answer any 5 questions. Each carries two marks **(5x2=10)**

16. Explain the purpose of text classification in NLP with an example.

17. Differentiate between heuristic-based NLP and machine learning-based NLP.

18. What is stemming?

19. Mention any two common text preprocessing techniques.

20. What is Naive Bayes Classifier?

21. Which algorithm is commonly used to optimize the weights in Logistic Regression?

22. Differentiate between Relation Detection and Relation Classification in NLP.

23. List the major steps involved in the ReVerb algorithm for relation extraction.

24. What is Statistical Machine Translation? Briefly explain its basic working.

25. Why is decoding hard?

Section C

Answer any 5 questions. Each carries four marks **(5x4=20)**

26. Describe any five core NLP tasks with suitable examples.

27. Explain the major steps involved in an NLP system pipeline, including data acquisition.

28. Differentiate between word tokenization and sentence tokenization with examples

29. Explain One-Hot Encoding and illustrate it with a suitable example

30. This question is grammatically correct and suitable for an examination.

31. Explain different methods of Text Classification.

SGOU

32. Explain the bootstrapping technique in semi-supervised relation extraction with a suitable example

33. Explain advantages and limitations of supervised approaches in relation extraction.

34. Explain the two broad approaches to answering factoid questions.

35. Write the applications of machine translation.

Section D

Answer any 2 questions. Each carries fifteen mark **(15x2=30)**

36. Explain the major tasks and applications of NLP, including text classification, machine translation, text summarization, question answering, and Named Entity Recognition.

37. Discuss the major text-cleaning techniques used in NLP. Explain the removal of unwanted characters, punctuation, noise, and other irrelevant information with examples.

38. Describe the six stages of the text classification pipeline. Explain what happens at each stage and why the order matters.

39. Discuss the evaluation metrics used to measure the performance of Information Retrieval systems, including Precision, Recall, Interpolated Precision, Mean Average Precision (MAP), and Mean Reciprocal Rank (MRR), with suitable examples.

SGOU

SREENARAYANAGURU OPEN UNIVERSITY

QP CODE: …… Reg. No : .....

Name : …….....

Model Question Paper- set-II

End Semester Examination

Bachelor of Computer Applications - Fifth Semester

B24DS04DE: Natural Language Processing

(CBCS - UG)

2025 -26 - Admission Onwards

Time: 3 Hours Max Marks: 70

Section A

Answer any 10 questions. Each carries one mark **(10x1=10)**

1. What is machine translation?
2. What is text summarization?
3. What is heuristic-based NLP?
4. Name any two tools used for web scraping
5. What is text preprocessing?
6. What is the full form of BoW?
7. In binary classification, how many categories can a text be assigned to?
8. What is the first stage of the text classification pipeline?
9. Write the formula for the sigmoid function.
10. Which Open IE system discussed in this unit extracts verb-based relation phrases?
11. Which TF weighting method reduces the influence of very frequent terms?
12. Which evaluation metric measures the correctness of extracted relations?

SGOU

13. What is a factoid question?

14. What is the purpose of machine translation in NLP?

15. List different types of Alignment.

Section B

Answer any 5 questions. Each carries two marks **(5x2=10)**

16. Explain any two applications of Natural Language Processing.

17. Differentiate between natural language and computer language.

18. Differentiate between structured and unstructured data.

19. What is lemmatization?

20. What is a text classifier?

21. Explain Support Vector Machine (SVM) for Text Classification.

22. What is TF-IDF, and how is it used to determine the importance of a term in a document?

23. What is Relation Extraction, and how does it differ from Named Entity Recognition (NER)?

24. Explain Mean Reciprocal Rank.

25. What is symmetrization in word alignment?

Section C

Answer any 5 questions. Each carries four marks **(5x4=20)**

26. Explain the major building blocks of NLP and the challenges involved in processing human language

27. Describe the text preprocessing techniques used in NLP, including cleaning, tokenization, stemming, lemmatization, and normalization.

28. Describe the major steps involved in text cleaning with suitable examples.

29. Explain the major stages of the NLP system pipeline

30. Explain the Applications of Information Extraction.

31. Explain Named Entity Recognition (NER) and the ambiguity in NER with examples.

SGOU

32. Differentiate supervised and unsupervised relation extraction with suitable examples.

33. Explain Precision, Recall, and F1-score as evaluation metrics for relation extraction in Natural Language Processing (NLP).

34. Explain the common three-stage architecture of a factoid Question Answering system.

35. Explain the decoding process in phrase based statistical machine translation.

Section D

Answer any 2 questions. Each carries fifteen mark **(15x2=30)**

36. Discuss the different approaches to NLP, explaining heuristic-based NLP and machine learning-based NLP, their working principles, advantages, limitations, and applications.

37. Explain text representation in NLP. Discuss the Vector Space Model, including the representation of documents as numerical vectors, with suitable examples.

38. Describe the optimization techniques used to improve Naïve Bayes performance for sentiment analysis. Explain techniques such as binary word counts, negation handling, and sentiment lexicons.

39. Explain the question processing stage in a Question Answering (QA) system and describe its four main tasks.

SGOU

സർവ്വകലാശാലാഗീതം

വിദ്യയാൽ സ്വതന്ത്രരാകണം
വിശ്വപൗരരായി മാറണം
ഗ്രഹപ്രസാദമായ് വിളങ്ങണം
ഗുരുപ്രകാശമേ നയിക്കണേ

കൂരിരുട്ടിൽ നിന്നു ഞങ്ങളെ
സൂര്യവീഥിയിൽ തെളിക്കണം
സ്നേഹദീപ്തിയായ്വിളങ്ങണം
നീതിവൈജയന്തി പാറണം

ശാസ്ത്രവ്യാപ്തിയെന്നുമേകണം
ജാതിഭേദമാകെ മാറണം
ബോധരശ്മിയിൽ തിളങ്ങുവാൻ
ജ്ഞാനകേന്ദ്രമേ ജ്വലിക്കണേ

കുരീപ്പുഴ ശ്രീകുമാർ

SGOU

SREENARAYANAGURU OPEN UNIVERSITY

Regional Centres

Kollam
Sreenarayanaguru Open University
Headquarters Building
BSNL Bhavan, Vellayittambalam,
Kollam, Kerala, Pin : 691012
Ph: 9188912554
email:rckollam@sgou.ac.in

Tripunithura
Govt. College
Tirpunithura, Ernakulam
Kerala, Pin : 682301
Ph: 04842927436
email:rcedirector@sgou.ac.in

Thalassery
Govt. Brennen College
Dharmadam, Thalassery
Kannur, Pin : 670106
Ph: 04952990494
email:rctdirector@sgou.ac.in

Pattambi
Sree Neelakanta Govt. Sanskrit College
Pattambi, Palakkad
Kerala, Pin : 679303
Ph: 04662912009
email:rcpdirector@sgou.ac.in

Kozhikode
Govt. Arts and Science College
Meenchantha, Kozhikode,
Kerala, Pin : 673002
Ph: 04952920228
email:rckdirector@sgou.ac.in

SAY NO TO RAGGING

BEFORE YOU EVEN THINK OF RAGGING



Humiliation

Blacklisting

Ruined Career

Download

ANTI RAGGING

App

Don't just stand and watch. Stop Ragging! Show Character

Visit UGC Website i.e. www.ugc.ac.in & www.antiraging.in to UGC Anti Ragging regulations.

Ministry of Education
Government of India

Are You Being Ragged ?

विश्वविद्यालय अनुदान आयोग
University Grants Commission

SSGOU

DON'T LET IT BE TOO LATE

SAY NO TO DRUGS

LOVE YOURSELF AND ALWAYS BE HEALTHY



SREENARAYANAGURU OPEN UNIVERSITY

The State University for Education, Training and Research in Blended Format, Kerala

Natural Language Processing

COURSE CODE: B24DS04DE

SGOU

Sreenarayanaguru Open University
Kollam, Kerala Pin- 691601, email: info@sgou.ac.in, www.sgou.ac.in Ph: +91 474 2966841

ISBN 978-81-180900-6-9