الجمهوريـــة الجزائريـــة الديمقراطيـــة الشعبيـــة
People's Democratic Republic of Algeria
وزارة التعليـــم العالـــي والبحـــث العلمـــي
Ministry of Higher Education and Scientific Research
جامعـة محمـد البشـير الابراهيمـي – برج بوعريريج
University of Mohamed El Bachir El Ibrahimi - Bordj Bou Arréridj

Lecture notes on
Text and Web Mining

Compiled by
SABRI Lyazid

Department of Mathematics and Information Technology
Mohamed El Bachir El Ibrahimi university

Course Objectives

This course provides students with a means to understand and assimilate the various technologies associated with several fields, including linguistics and Natural Language Processing (NLP). In addition, it covers the fundamentals of text mining for information retrieval, knowledge extraction from text corpora, web exploration, and more.

The course also demonstrates how tools from Data Mining and Machine Learning can be used to design more intelligent platforms. Therefore, for pedagogical reasons, we have chosen to write this document in a way that enables students to both understand the theory and practice it progressively.

In the appendix, students will find R language scripts for the implementation of all the exercise solutions.

Level of study

This course targets Master 1 students specializing in Information and Communication Technologies (ICT).

Assessment procedures

1. Students will be assessed based on practical work, workshop activities, and continuous assessment.
2. A final exam will be held at the end of the module.

Prerequisites for the Course

It is recommended that students have basic knowledge of statistics, linear algebra, and machine learning to better understand the principles of text mining. Basic notions in linguistics, particularly regarding the structure and meaning of texts, are also desirable. Proficiency in a programming language such as Python or R, along with familiarity with handling textual data, is strongly recommended for the practical implementation of algorithms. Finally, curiosity about language analysis and information retrieval will greatly facilitate the understanding of the concepts presented in this module.

Content

Introduction

Natural Language Processing (NLP) faces a major challenge: human language is unstructured, implicit, and often ambiguous. Unlike numerical data, a text does not explicitly indicate its internal relationships; its understanding depends on context, shared knowledge, and the inferences made by the reader.

For example: "A 76-year-old woman suffering from hypertension and angina. Possible heart attack two years ago. Admitted to hospital with central chest pain without radiation. She mentioned that her daughter had similar symptoms. She was treated and is now doing well."

Even for a human reader, it is uncertain whether the final *she* refers to the elderly woman or to her daughter. The ambiguity remains due to the lack of clear clues in the text. For a machine, this uncertainty becomes an even greater challenge: it must detect the entities (the woman, her daughter), identify the anaphors (*she*, *her*), and estimate their probable co-reference based on syntactic structure, contextual proximity, and sometimes even domain knowledge (for example, age and medical history).

This example illustrates that understanding a text is not only about recognizing words, but also about reasoning over their implicit relationships — a task that even human intelligence finds difficult, and one in which artificial intelligence must learn to manage linguistic uncertainty

Text Mining (TM) is a field of Artificial Intelligence that combines several disciplines such as statistics, linguistics, and Natural Language Processing (NLP). TM is widely used today for information retrieval, knowledge extraction from text corpora, web exploration, and more. It is also applied in data analysis (medical data, trends, sentiment and opinion analysis, marketing, etc.) and document annotation—the latter being essential for accurate text analysis.

Information extraction, on the other hand, remains an open research field, as it involves discovering semantic and causal relationships within naturally written language (i.e., unstructured documents). To implement classification and categorization processes, TM distinguishes between supervised and unsupervised learning.

- **Supervised learning** aims to predict the class of a source (document, observation, etc.) from a set of predefined categories.
- **Unsupervised learning** seeks to determine to which class a document or theme belongs when **no predefined categories** exist.

There are many methods for text processing, both for classification and categorization — some based on statistical, probabilistic, or linguistic approaches, while others rely on machine learning (ML) techniques.

In this course, you will use the Python and R programming languages to understand the basic principles of text mining. Following this material, other useful tools and algorithms dedicated to text mining will be introduced — particularly those based on ontologies and deep learning approaches.

Simple examples will be used and analyzed throughout this document. However, for better understanding, readers are encouraged to study the content sequentially and to reproduce all tests and exercises in the given order.

1. Chapter 1 : From Unstructured Texts to Features: Preprocessing and Linguistic Analysis

Learning—whether biological or artificial—can be understood as a process through which a system acquires, structures, and uses information from its environment. It is a fundamental mechanism shared by humans, animals, and intelligent machines.

This process begins with perception: the system captures signals or data from the external world—sounds, images, texts, smells, measurements, and so on. This raw input is not yet knowledge but forms the raw material from which knowledge is built.

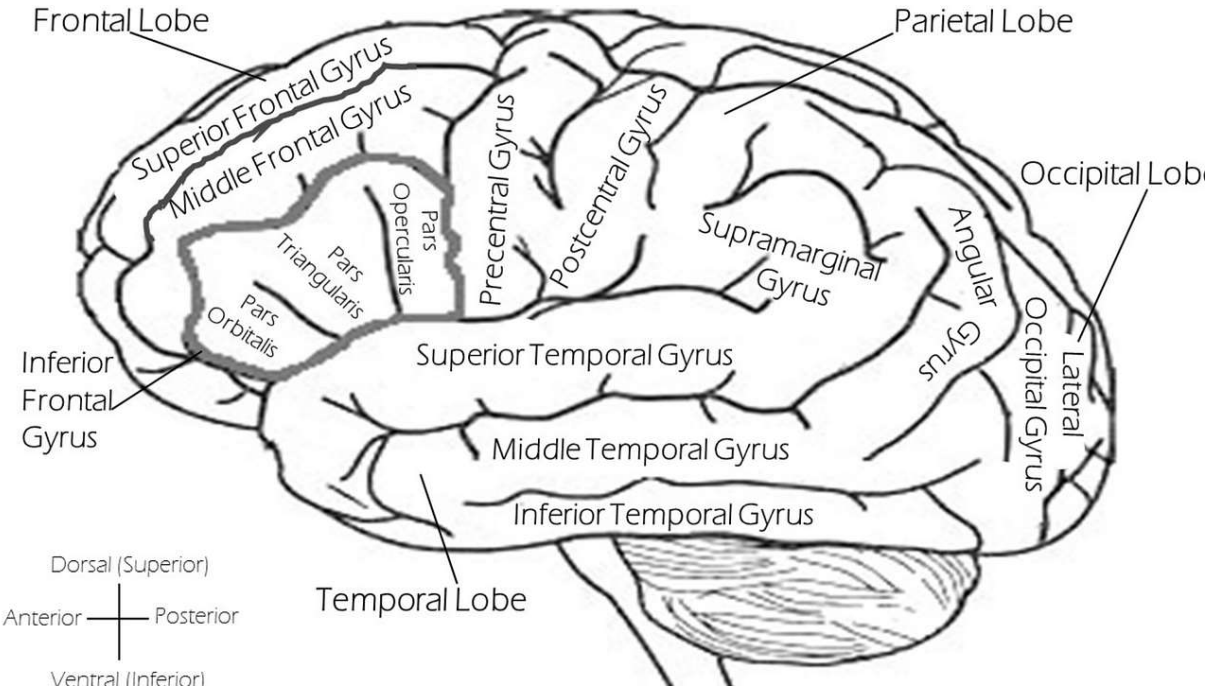
The next stage is the construction of internal representations: the perceived information is organized, filtered, and translated into symbols, models, or mental schemas that enable the system to represent the world in a usable way. In humans, this corresponds to the formation of concepts, categories, and abstract relationships; in artificial systems, it takes the form of knowledge graphs, neural networks, or probabilistic models.

These representations are then transformed into knowledge through processing, reasoning, and learning operations. Knowledge allows generalization from experience, the discovery of causal relationships, and the extraction of regularities. In other words, knowledge is information interpreted and integrated within a conceptual framework that supports appropriate action.

Finally, the system implements this knowledge in its activities, behaviors, or operations. In humans, this manifests in decision-making, communication, or problem-solving. In machines, it appears as image classification, natural language dialogue, event prediction, or adaptive robotic behavior.

1.1.From Sensing to Understanding: The Stages of Learning

Wether natural or artificial, all cognitive systems follow the same fundamental cycle: perceive → represent → learn → reason → act. This cycle illustrates the gradual transition from raw data to operational knowledge—from the world as observed to the world as understood and transformed.



The figure illustrates the functioning of the brain as a cognitive system: it perceives sensory information, processes it to build internal representations, extracts knowledge, and uses it to guide actions and behaviors. This diagram highlights the fundamental stages of the learning process: perception, representation, reasoning, and action.

1.2. Text Mining : Applications

Text mining encompasses a set of techniques that automatically extract relevant information, hidden patterns, and knowledge from large collections of unstructured text. Its applications now span almost every field of activity.

In the medical domain, it can be used to analyze clinical records, scientific publications, or hospital reports to detect correlations between symptoms, treatments, and diagnoses, or even to detect early warning signs of epidemics.

In business, it supports sentiment analysis, competitive intelligence, and the detection of emerging events in media and social networks.

In academic research, text mining helps map scientific domains, identify potential collaborations, and uncover thematic trends. In security and intelligence, it helps detect weak signals, hate speech, or suspicious content in massive text streams. Finally, in the legal and administrative sectors, it facilitates precedent research, case law analysis, and the partial automation of document drafting.

All these applications rely on techniques from natural language processing (NLP), computational linguistics, and data science, such as classification, clustering, semantic analysis, named entity recognition, and relationship detection.

In short, text mining transforms an ocean of unstructured text into usable knowledge, supporting decision-making, research, and innovation in increasingly complex and data-rich contexts.

A particularly promising application concerns the Internet of Things (IoT). IoT systems generate a growing volume of text messages: event logs, alert notifications, fault descriptions, and human annotations on sensor data. Text mining makes it possible to analyze this content to detect abnormal behavior, correlate physical and textual events, and improve predictive maintenance. For example, automatic analysis of intervention reports and sensor messages can help identify recurring causes of malfunctions in industrial or urban networks.

1.3. Intelligent Internet of Robotic Things (IoRT) and the Role of the Robot

The Intelligent Internet of Robotic Things (IoRT) represents the natural evolution of the IoT towards systems capable not only of collecting and transmitting data, but also of perceiving, reasoning, and acting autonomously. In this paradigm, robots become intelligent connected agents within an ecosystem of communicating objects, digital infrastructures, and cloud services.

The role of the robot in the IoRT goes far beyond simple mechanical execution. Through interconnection with IoT sensors (cameras, temperature sensors, biomedical sensors, GPS, etc.), the robot accesses a stream of contextual information that it can analyze using machine learning and text mining techniques. For example, an industrial maintenance robot can combine sensor data (temperature, vibration, power consumption) with text reports from operators or alerts from the IoT network to diagnose a fault before it occurs.

Text mining plays a key role here: it allows the robot to interpret symbolic information (instructions, messages, status descriptions) written in natural language and link it to physical data from the connected environment. Thus, the robot becomes not only a physical actor, but also a semantic interpreter capable of understanding the meaning of interactions and adapting its behavior.

IoRT is therefore based on vertical integration between:

• the sensory level (IoT objects and sensors),

• the cognitive level (processing, text mining, semantic reasoning),

• and the action level (robots and autonomous systems).

This convergence paves the way for advanced applications: intelligent care robots, connected logistics, smart cities, or autonomous agricultural systems. In these environments, robots no longer act in isolation, but in cooperation with the network of intelligent objects, sharing their knowledge and continuously learning from the global flow of information.

1.4.Text Mining: Motivation and Text Exploration

Text mining is primarily driven by the need to explore, understand, and exploit the growing.

With the widespread adoption of digital technologies, textual information now comes from multiple sources, such as:

- social media,
- product or service reviews,
- emails,
- blogs and forums,
- medical records and clinical opinions.

These texts contain valuable information on opinions, behaviors, trends, and knowledge, but they often remain unusable without automated analysis.

The rise of connected devices—such as smartphones, IoT devices, and virtual platforms—has further expanded the scope of text mining. Users now communicate anytime, anywhere, generating a continuous stream of text data. These technologies enable the development of intelligent applications to:

1- Interact With Government Institutions (Online Services, E-Administration),

2- Offer Personalized Assistance To Customers, Patients, Or Citizens,

3- Facilitate The Digitization And Automatic Analysis Of Documents,

4- And Support New Forms Of Human-Machine Interaction In Connected Environments.

In summary, text mining is an essential response to the information overload of the digital age. It transforms a heterogeneous set of texts—from social, medical, institutional, or industrial sources—into actionable knowledge for decision-making, research, and the improvement of smart services.

1.5.Text Mining: Problems and Limitations

Limitations Related to Language Understanding

Despite its many advantages, text mining presents several fundamental limitations due to the complex nature of human language.

Machines can generally only analyze the syntactic aspect of a text, that is, the way words are arranged according to grammatical rules. These approaches can identify the structure of a sentence (subject, verb, object), but they remain incapable of grasping the true meaning conveyed by these words in a given context.

Processing systems often rely on the use of grammar or statistical models describing the relationships between words, but lack the conceptual or cultural understanding that a human naturally mobilizes.

However, semantics—which studies the meaning of a word or group of words in a particular context—is essential for correctly interpreting a text. Without a detailed understanding of the language, it is impossible to fully grasp the overall meaning of a document.

Thus, even if machines can segment, annotate, or label sentences, they encounter difficulties when it comes to understanding implicit relationships, anaphora (such as he, she, her), metaphors, or innuendos. The transition from syntax to semantics therefore remains a major challenge in natural language processing (NLP).

Analysis Methods: Statistics and Linguistics

Text documents (or corpora) can be analyzed using two broad, complementary approaches:

Mathematical and Statistical Approach

This approach represents texts in digital form, allowing the application of machine learning or statistical methods. Techniques such as TF-IDF, Word2Vec, doc2vec, or neural language models (BERT, GPT, etc.) transform words into vectors to identify similarities, classify documents, or detect dominant themes.

However, these models primarily address the form of the text (frequency, co-occurrence, vector structure), without necessarily understanding its deeper meaning.

Linguistic and Symbolic Approach

This method is based on natural language processing techniques and seeks to preserve the linguistic richness of the text. It uses grammar, morphology, syntax, and semantic models to represent the relationships between entities and concepts. The goal is to produce a structured semantic representation, where each sentence is interpreted according to its meaning and context, thus enabling a more detailed analysis of intentions, relationships, and events.

The current challenge of text mining is to combine these two approaches: harnessing the power of statistical methods to manage large volumes of data, while integrating linguistic models to preserve the meaning and semantic coherence of texts. This hybridization is at the heart of recent research in linguistic artificial intelligence.

1.6.Conceptual Links Between Text Mining and Data Mining

Data Mining Definition : (It is an automatic or semi-automatic process for discovering implicit, previously unknown, and potentially useful knowledge within electronically stored data collections. This knowledge takes the form of structural patterns in the data, which can also be used to make predictions or provide answers.) *Ian Witten and Eibe Frank Data Mining: Practical Machine Learning Tools and Techniques. Morgan Kaufmann, San Francisco, 2011.*

Text mining and data mining share the same objectives: extracting useful knowledge from large amounts of data. The main difference lies in the nature of the data:

1- Data mining processes structured data (databases, tables, numerical measurements).

2- Text mining applies to unstructured data (free texts, documents, messages, articles, etc.).
3- The objective of text mining is therefore to transform the text into usable data, in order to then apply data mining techniques to discover patterns, regularities, and knowledge.

1.7.Main Tasks of Text Mining

a. Document Categorization (Classification)

1- Automatically assign a document to one or more predefined categories.
2- Identifying whether an email is spam or not.
3- Classifying a text according to its scientific field or its sentiment (positive, negative, neutral).

b. Clustering

1- Automatically group documents based on their content similarity.
2- No prior categories are required.
3- Allows you to discover themes or group similar discussions within a large corpus.

c. Automatic Summarization

1- Identify the most important parts of one or more documents.
2- Produce a concise summary preserving essential information.

d. Structured Information Extraction

1- Extract named entities (people, places, organizations, dates, etc.), events, or relationships from unstructured text.

2- Example: Identify all interactions between patients and doctors in medical records.

e. Association and Trend Extraction

1- Find associations between concepts or terms (co-occurrences, correlations).
2- Detect temporal trends: how a concept or theme evolves over time in documents.

1.8.Text Mining: Methods, Tools, and Algorithms

The data used in text mining takes the form of character strings written in a natural language. These texts are composed of meaningful words, organized according to syntactic rules and often marked by the subjectivity, ambiguity, or redundancy specific to human language.

A text unit can correspond to different levels: an isolated sentence, a set of sentences, a paragraph, or even a complete document (for example, web pages, emails, medical reports, publications, opinions, or messages).

In other cases, the texts may be very short and not form a grammatically correct sentence, this is typically the case with social media messages, text messages, or abbreviated medical terms used in clinical reports.

This linguistic and structural variability makes text analysis complex: raw text is unstructured and therefore cannot be directly used by machine learning algorithms.

1.9.Requirements for Applying Data Mining Methods

To apply techniques from data mining or statistical modeling, it is essential to convert texts into a structured representation.

This step, called preprocessing or vectorization, consists of transforming texts into digital forms that can be used by algorithms—for example, through term matrices (TF-IDF), semantic representations (Word2Vec, BERT), or co-occurrence graphs.

Thus, text mining relies on a series of methods and tools that allow the transition from natural language to a formal representation, paving the way for quantitative analyses and knowledge discovery from textual documents.

1.10. *Text Mining & Machine Learning: Principles*

Text mining relies heavily on machine learning techniques to discover new knowledge from large volumes of unstructured text.

The goal is to replicate certain human cognitive abilities, such as comprehension, categorization, or the detection of semantic relationships.

The main principles can be summarized as follows:

Imitation of Human Reasoning

Algorithms seek to artificially replicate the human ability to recognize patterns, connect concepts, and interpret implicit meanings in texts.

Learning Underlying Structures

Machine learning aims to identify the structure of models and optimally adjust their parameters to represent the hidden relationships between words, sentences, or documents.

Iterative Learning Process

Learning is based on training samples (training data), analyzed iteratively until the model achieves satisfactory performance.

Validation and Generalization

A testing phase is then conducted using a separate data set (test set), not used during training, to verify the model's ability to generalize to new texts.

Knowledge Extraction

Through this training-testing-evaluation cycle, the system learns to automatically identify linguistic regularities, categories, sentiments, or themes, helping to transform textual data into usable knowledge.

1.11. *Text Mining & Machine Learning : Définition*

Text mining, combined with machine learning methods, can be defined as a knowledge-intensive process in which a user interacts with a collection of textual documents using analytical and algorithmic tools. The goal is to identify, extract, and explore hidden patterns, linguistic regularities, or semantic relationships within texts.

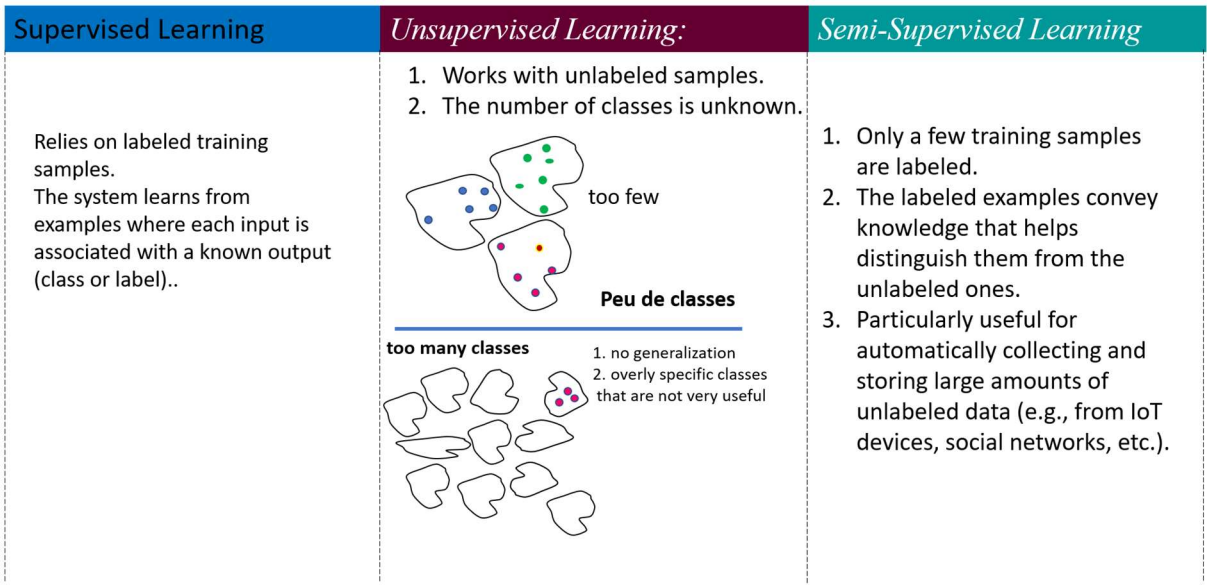
This process combines:

• techniques from natural language processing (NLP) to represent textual content in a usable manner;

• and machine learning algorithms (classification, clustering, dimension reduction, language models, etc.) to automatically discover knowledge from these representations.

Text mining is not just a simple statistical analysis of words, but a learning and interpretation process that transforms unstructured textual data into structured, understandable, and decision-useful information.

1.12. Machine Learning Sphere



Within the context of text mining, machine learning methods are divided into three main spheres depending on the nature of the available data and the analysis objectives.

1.13. Supervised Learning

The model learns from a labeled data set, meaning that each text is associated with a known category (e.g., "spam/non-spam," "positive/negative," "medical/legal").

The goal is to predict the class of a new, unknown text based on the regularities observed in the training examples.

Examples in text mining:

1. Email classification (spam filtering)

2. Sentiment analysis (positive or negative opinion)

3. Theme detection (automatic article categorization)

1.14. Unsupervised learning

The data has no prior labels: the algorithm must discover the hidden structure or natural groupings in the texts on its own.

The goal is to reveal patterns, clusters, or implicit themes in the corpora.

1- Examples in text mining:

2- Grouping similar documents (clustering)

3- Theme discovery through topic modeling (e.g., LDA)

4- Analysis of word co-occurrences or semantic networks

1.15. Semi-supervised learning

Combines the two previous approaches: a small portion of the data is labeled, while the rest is unlabeled.

1- The algorithm uses known information to guide learning on unlabeled data.

2- Very useful when the cost of manually annotating texts is high (a common case in linguistics or medicine).

3- Text mining examples:

4- Automatic corpora annotation based on a few human examples

5- Relationship or entity detection with partially guided learning

1.16. Text Mining Approaches and Data Transformation

In the field of text mining, numerous applications — such as email filtering, topic extraction from web pages, or analysis of medical reports — rely on data mining methods, often referred to as statistical or probabilistic techniques, while others are based on linguistic approaches. Unlike linguistic methods, statistical ones do not take semantics into account. To apply data mining techniques efficiently, it is therefore necessary to transform unstructured textual data into structured representations that describe documents in a computationally usable way.

In this representation, documents are expressed as vectors within a Vector Space Model (VSM), usually containing the frequency of word occurrences within the corpus. These vectors are called feature vectors or term weights.

The concept of word vectors emerged in the mid-1970s. It involves submitting a corpus to a preprocessing phase aimed at reducing the dimensionality of the resulting matrix.

However, this approach presents a major limitation: word order is not preserved. As a result, sentences such as *"the child eats an apple"* and *"an apple eats the child"* are represented identically. Furthermore, punctuation marks and grammatical information (such as word synonyms) are ignored. The number of columns can grow dramatically, and the contextual meaning of words within sentences is completely lost.

Information loss can also result from typical preprocessing operations such as lemmatization, stemming, and stop word removal. Stop words — such as *what*, *who*, *this*, *at*, *in*, *the*, etc. are considered non-informative terms that do not help distinguish between documents and are thus eliminated.

While this reduction significantly simplifies the data and improves computational performance, it may also remove words that carry important contextual meaning, potentially altering the interpretation of some texts.

1.17. Representation Model and Document-Term Matrix

Despite the absence of semantic understanding, the results obtained from algorithms based on these representations remain acceptable in many practical applications. Grouping documents with similar structures or lexical contents makes it easier to analyze, retrieve, or categorize large collections of texts efficiently.

The underlying representation is the Document-Term Matrix (DTM), also known as the Bag-of-Words (BoW) model. In this matrix:

- Rows represent the documents in the corpus,
- Columns represent the terms (or words),

- and each cell (t_i,d_j) indicates the frequency or weight of the term t_i in document d_j.

documents	animal	To be	friend	human	machine	To do	intelligent	inspire
Text1	1	2	1	1	1	0	0	0
Text2	0	1	0	1	2	1	1	0
Text3	1	1	0	1	0	0	1	1
Text4	1	2	0	1	1	0	0	0
Text5	1	2	0	2	1	0	0	1

The cell (human, text5) represents an entry indicating the frequency of the word *human* in document *text5*, which corresponds to a matrix attribute–value pair

Each document is thus represented as a vector of features — a numerical description of its content. Although this representation ignores grammar and word order, it remains one of the most widely used models in text mining because it allows the application of machine learning algorithms such as clustering, classification, or topic modeling.

1.18. Knowledge Extraction Process from Texts

Knowledge extraction from **narrative texts** is a linguistically and analytically intensive process. Its goal is to transform unstructured textual data into structured, machine-readable information suitable for data mining or machine learning algorithms. The success of this process largely depends on data preparation and the quality of the extracted features.

1.19. Text Preparation and Linguistic Analysis

Before any extraction, text must be cleaned, normalized, and analyzed at several complementary linguistic levels:

a) **Morphology**

Morphology studies the form of words, including their grammatical categories (verbs, nouns, adjectives, pronouns, etc.) and inflectional variations (conjugations, declensions, gender, number).
This step allows reducing words to their base forms (*lemmatization* or *stemming*), simplifying grouping and lexical comparison.

b) **Structural Syntax**

Syntax focuses on the organization and structure of sentences. It aims to determine hierarchical relationships between words (e.g., subject–verb–object dependencies) and identify syntactic units (noun, verb, or prepositional phrases). This structuring is essential for understanding the role and function of each word in a sentence.

c) **Semantics**

Semantics studies the meaning of words and expressions in context. It links linguistic forms to their meanings: synonymy, antonymy, polysemy, or conceptual

relations between terms.
This level analyzes semantic relationships such as *"the doctor treats the patient."*

d) **Pragmatics**

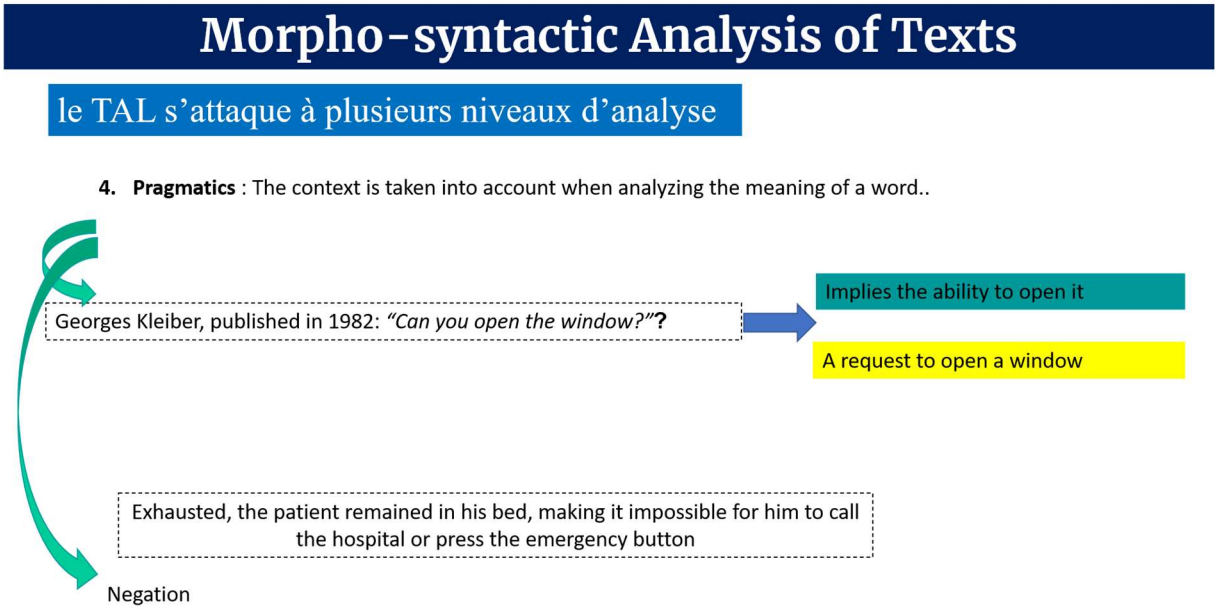
Pragmatics considers the context of utterance to interpret meaning.
It studies how a word or sentence's meaning depends on the situation, the speaker's intentions, or implicit references (e.g., resolving anaphora: *she* → *the elderly woman*).
This dimension is crucial for understanding narrative texts, where temporal sequences, actors, and intentions vary with context.

1.20. Combined Morpho-Syntactic Features

By combining morphological and syntactic information, texts can be represented with richer features:

1. Part-of-speech tags linked to word forms
2. Dependency relations between words (who does what to whom)
3. Phrase structures capturing functional units in sentences

These morpho-syntactic features serve as a foundation for semantic analysis, feature vector construction, and higher-level NLP applications such as text classification, summarization, or knowledge extraction.



1.21. Text Preprocessing: The Problem of Words

In text preprocessing, one of the fundamental challenges is the **identification and treatment of words**, which are not always straightforward in natural language.

1. **Word Boundaries and Separators**
    - **Word**: A sequence of letters delimited by separators (spaces, punctuation, etc.).
    - **Apostrophe (')**: Can indicate either a single word or two words contracted (e.g., *don't → do + not*).
    - **Period (.)**: May indicate the end of a sentence, but not always (e.g., abbreviations like *Dr.*, *Mr.*).
    - **Whitespace**: Can separate single words or multi-word expressions.
2. **Influence of Letters on Meaning**
    - A single letter can change the meaning of a sentence.
    - Some letters can serve as negation markers, modifying the sense of a sentence (e.g., *un-* in *unfortunate*).

3. **3. Morphological Units: Morphemes**

A morpheme is the smallest linguistic unit that carries both form and meaning.

Examples:

- *robots* → 2 morphemes:
    - Root / lemma: *robot*
    - Suffix: *-s* (plural)
- *unfortunately* → 3 morphemes:
    - Prefix: *un-*
    - Root: *fortunate*
    - Suffix: *-ly*

Morphemes are essential in text preprocessing because they allow the **normalization of words** (lemmatization, stemming) and reduce dimensionality in feature extraction.

1.22. Text Preprocessing: Inflection and Derivation

In text preprocessing, understanding **morphological variations** of words is crucial for normalization and feature extraction. Two key concepts are **inflection** and **derivation**.

1. Inflection

Inflection does not change the word category: a noun remains a noun, a verb remains a verb.

- Plural: *cat* → *cats*
- Possessive: *John* → *John's*
- Comparative / superlative: *fast* → *faster* → *fastest*
- Past participle: *go* → *gone*

2. Derivation

Derivation creates a new word by adding prefixes or suffixes, often changing its part of speech.

- Prefix: *healthy* → *unhealthy*
- Suffix: *happy* → *happiness*
- Adjective to noun: *fortunate* → *fortunately*
- French example: *heureux* → *malheureux* (adjective, positive → negative)

1.23. Text Preprocessing: Lemmatization and Morpho-Syntactic Analysis

Lemmatization is a key step in text preprocessing, aimed at converting words into their canonical dictionary form, called a lemma. This process is crucial for improving the consistency of textual features in text mining and natural language processing (NLP)

1.24. Text Preprocessing: Stemming

Stemming (also called racinisation) is a text preprocessing technique that reduces different forms of a word to a common base form, often called the stem or root

- A stem is the part of a word that carries its core meaning.
- When a root consists of a single morpheme, it is identical to the stem.
- Words derived from the same stem are generally closely related in meaning, although this may not always hold for machines.
- The generated stem may not exist as a valid word in the language dictionary.

Example of Stemming

Original text:

"Robotic sensors are used to estimate a robot's condition and environment. These signals are passed to a controller to enable appropriate behavior."

After stemming:

```
robot sensor use estim robot condit environ signal pass control enabl appropri
behavior
```

Notice that some stems may not be valid dictionary words (e.g., *estim*, *behav*), but they still capture the core meaning for text mining purposes.

1.25. Lemmatization

Definition:

Lemmatization associates each token with its canonical form, such as the infinitive for verbs, the singular form for nouns, or the base adjective. Unlike stemming, lemmatization ensures that the converted word is a valid dictionary word.

Examples:

- *maisons* → *maison* (French, plural → singular)
- *robots* → *robot*
- English verbs: *am, was, is* → *be*
- Example sentence:

  "Robots are close to oven" → "robot be close to oven"
- A lemmatizer generally requires a complete lexical and morphological analysis.
- It can convert words to correct dictionary forms, which a stemmer may not achieve.

- Lemmatization typically precedes stemming when preprocessing text.

1.26. Words and Morphological Units

- A word is a unit carrying meaning, often composed of subunits like roots, prefixes, suffixes.
- A morpheme is the minimal meaningful linguistic unit, either free or bound.
- Some expressions go beyond single words, e.g., *porte-plume*, *s'il vous plaît*.
- Language-specific differences: some languages treat words as independent units, others blend them into syntagms or lexies, requiring analysis at higher levels.

Historical references:

- Ferdinand de Saussure (1916): Words are units of meaning combining a signifier (sound/image) and a signified (concept).
- Joseph Vendryes: Morphological variations affect how words are defined in different languages.

1.27. Challenges in Natural Language Processing: Context, Punctuation, and Named Entities

Consider the following example: "The monitoring system is designed to assist and supervise the home of a person living alone, named John. At time t0, it detects the presence of a person in the living room. This is John, who is in front of his television. At the same time, the system detects another person in the kitchen. It then notifies John's daughter."

From this paragraph, a human can quickly infer that another person is present in John's apartment, even though John is supposed to be alone. Humans can draw this conclusion based on contextual cues in the text. The question arises: Can a machine reach the same inference?

Punctuation and Text Preprocessing Challenges

Punctuation introduces several difficulties in NLP:

- Some punctuation marks may or may not belong to a word.
- Characters such as hyphens or periods appear in numbers, IP addresses, or abbreviations, making automatic parsing challenging.
- Aggressive lemmatization may change the real meaning of sentences.

Example (medical report):

"The patient was taking treatment X."

The past tense (*was taking*) implies that the patient may no longer be taking the treatment, highlighting that tense and conjugation are crucial for correct interpretation.

Another example:

"Exhausted, the patient remained in bed; it was thus impossible for him to call the hospital or press the emergency button."

- Here, negation is expressed indirectly.
- The reason for the patient's inability (*exhausted*) is carried by a lexical morpheme, i.e., a word with semantic content.
- Even single letters (e.g., *s* in plurals) are morphemes if they convey grammatical meaning.

1.28. Stemming Tools and Algorithms

The most well-known stemming algorithms were developed by Lovins, Porter, and Paice [6][7][8]. For the French language, the algorithm CARY is used, which is based on the Porter algorithm. CARY was proposed in 2002 by M. Paternostre et al. [9].

These algorithms are widely applied in information retrieval, named entity recognition, and even machine translation. By reducing words to their root form, stemming allows different forms of a word to share the same root, although they may not always have exactly the same semantic meaning.

In summary, these algorithms generally proceed in multiple steps, and some of them rely on dictionaries.

- Dictionary-based approaches are slower but have the advantage of not making errors on words already listed.
- Non-dictionary-based approaches are faster but have a higher error rate.

The Porter algorithm is based, among other things, on five contextual rules to decide whether a suffix should be removed. For example, a suffix like -ing will only be removed if the stem contains at least one vowel. In this way:

- *troubling* becomes *troubl*
- *sing* remains *sing*

The CARY algorithm also uses rules and conditions for suffix removal or transformation. When the analyzer recognizes a suffix from its list, it either removes it or modifies it.

Therefore, stemming should be applied as the final step in the preprocessing pipeline, after other text normalization processes such as tokenization and lemmatization.

1.29. Term Weighting in Text Mining

In text mining, before applying similarity measures, it is crucial to determine the importance of words in documents. One of the most intuitive approaches is to calculate the frequency of each word, which forms the basis for term weighting. Term weighting allows us to represent each document as a vector, where each component corresponds to a specific word.

Binary Term Weighting

The simplest form of weighting is binary weighting:

- Assign a weight of **1** if the term is present in the document.
- Assign a weight of **0** if the term is absent.

While easy to implement, this approach has a major limitation: all terms are considered equally important. For instance, a word appearing once in a document is treated the same as a word appearing 100 times.

documents	animal	To be	friend	human	machine	To do	intelligent	inspired
Text1	1	1	1	1	1	0	0	0
Text2	0	1	0	1	1	1	1	0
Text3	1	1	0	1	0	0	1	1
Text4	1	1	0	1	1	0	0	0
Text5	1	1	0	1	1	0	0	1

Term Frequency (TF)

To overcome the limitations of binary weighting, more sophisticated methods account for how frequently a term appears:

- In the document itself
- Across a category
- Within the entire corpus

This is called Term Frequency (TF), which gives a measure of a term's weight relative to the corpus. TF counts the actual occurrences of a term, but it has some drawbacks:

- Using TF with Euclidean distance can be problematic if documents differ significantly in length.
- TF does not consider the prevalence of terms across the corpus.

Other formulas to compute TF (Term Frequency)

Formula	Expression	Explanation
TF brut	TF(t,d) = f{t,d}	Nombre d'apparitions du mot dans le document
TF normalisé	TF(t,d) = $\frac{f_{t,d}}{\sum_{t'} f_{t',d}}$	Fréquence relative par rapport au nombre total de mots
TF logarithmique	TF(t,d) = $\begin{cases} 1 \; si \; t \; apparaît \\ 0 \; sion \end{cases}$	Reduces influence of very frequent terms
TF binaire	TF(t,d) = 1 if t appears else 0	Présence ou absence du mot
TF double normalisé	TF(t,d) = 0,5+0,5 * $\frac{f_{t,d}}{mat_{t'} f_{t',d}}$	Adjusted by most frequent term

1.30. Inverse Document Frequency (IDF) and TF-IDF

To solve these issues, rare terms in the corpus that appear in a document are considered more important, because common terms provide little differentiation between documents. This leads to Inverse Document Frequency (IDF).

The TF-IDF weighting scheme combines both concepts:

- Terms frequent locally (in a document or paragraph) but rare globally are given higher importance.
- **IDF:** Gives more weight to rare terms in a corpus that appear in a document, as they are more discriminative.
- TF-IDF is widely used in document classification, clustering, and information retrieval.

IDF Formula	Description
$\mathrm{IDF}(t) = \log \frac{N}{df_t}$	Classic formula: natural logarithm of total number of documents N divided by the number of documents containing term t (df_t).
$\mathrm{IDF}(t) = \log_{10} \frac{N}{df_t}$	Uses base-10 logarithm instead of natural log. Often chosen for simplicity and interpretability, because log base 10 scales values in a more intuitive range for human understanding.
$\mathrm{IDF}(t) = \log \frac{N}{1+df_t}$	Adds 1 to the denominator to avoid division by zero if the term does not appear in any document.
$\mathrm{IDF}(t) = \log \frac{N}{df_t} + 1$	Adjusted formula ensuring IDF value is always greater than 1.

TF*IDF: Indicates that terms that occur frequently in a document (TF), relative to how often they appear in the entire collection (IDF), are more important than terms that are common

across many documents. It emphasizes terms that are less frequent throughout the whole collection.

Other formulas to compute IDF (Inverse Document Frequency)

Formula	Expression	Explanation
IDF classic	IDF(t) = log(N /nt)	The rarer the word, the higher its IDF
IDF smoothed	IDF(t) = $\log(\frac{N}{1+nt})$	Évite la division par zéro
IDF probabilistic	IDF(t) = $\log(\frac{N-nt}{1+nt})$	Based on the inverse likelihood of occurrence
IDF probabilistic	IDF(t) = $\log(1+\frac{N}{nt})$	Variant to dampen extremes

Example

Term Weighting (Features)

Terme Frequency TF vs IDF (Inverse document frequency)

$$IDF_{t,d} = \log_{10} \frac{DCs}{D_t}$$

DCS The entire set of documents.

D_t: Number of documents in which term t appears.

It should be noted that a high occurrence of a term helps to reveal the domain or topic of a document.

Weights are assigned to less frequent terms since they are the most discriminative..

$$TFIDF_{t,d} = f_{t,d} * \text{idf}_t$$

	T1	T2	T3
D1	1	7	1
D2	8	1	0
D3	3	5	0
IDF	**0**	**0**	**0.47**

	T1	T2	T3
D1	0	0	**0.47**
D2	0	0	0
D3	0	0	0

To avoid a zero value for IDF, 1 is added in the logarithm calculation..

Experimental studies show that even using the top 10% most frequent words often maintains classifier performance, suggesting that highly frequent words can carry sufficient discriminative information. A high TF-IDF value indicates that a term is important in a document and rare in the corpus, making it highly informative for text analysis.

1.31. Normalization

Normalization is a fundamental process in Text Mining and in the vector representation of documents. It is used for several practical and theoretical reasons:

Impact of Document Length on Term Relevance

Longer documents naturally contain more distinct terms, which can heavily influence similarity calculations between documents. Without adjustment, queries are more likely to match longer documents simply because they have higher term counts.

Example:
Consider two documents containing the term **BBA**:

- Document A: 100 words, with **BBA** appearing 8 times.
- Document B: 2,000,000 words, with **BBA** appearing 20 times.

Although **BBA** occurs more frequently in absolute terms in Document B, it is proportionally much more significant in Document A. Therefore, Document A is actually more relevant to **BBA** than Document B.

This illustrates why normalization of term frequencies is essential when comparing documents of different lengths.

Comparability Between Documents

- Documents can have very different lengths (e.g., a 5-word SMS vs. a 1000-word report).
- Without normalization, longer documents automatically have higher term weights, which distorts similarity measures.
- Normalization scales all vectors to a common range, allowing for fair comparison.

Reducing the Impact of Frequent Terms

- Some words appear very frequently in a document or across the corpus (e.g., “the,” “of,” “a”).
- Logarithmic normalization reduces the effect of these frequent terms, emphasizing more informative and discriminative terms.

Stability of Machine Learning Algorithms

- Algorithms such as k-NN, clustering, SVM, or neural networks are sensitive to the scale of the data.
- Normalization prevents extreme values from dominating the learning process, improving performance and convergence.

Logarithmic Normalization

Logarithmic normalization is used to reduce the effect of very frequent terms. The standard formula is:

$$w_{i,j} = \begin{cases} 0 & \text{if } tf_{i,j} = 0 \\ 1 + \log(tf_{i,j}) & \text{if } tf_{i,j} > 0 \end{cases}$$

Where:

- $tf_{i,j}$ = frequency of term i in document j
- $w_{i,j}$ = normalized weight of the term

Simple Normalization

Simple normalization consists of dividing the frequency of each term by the sum of frequencies in the document:

$$w_{i,j} = \frac{tf_{i,j}}{\sum_k tf_{k,j}}$$

Each weight becomes a proportion of the total terms in the document.

This method ensures that the sum of weights in a document equals 1.

1.32. Similarity: Case Study

Document1: I have my coffee at Zila
Document2: Zila is located at the corner of the street where I live
Document3: Many of us go to Zila early in the morning

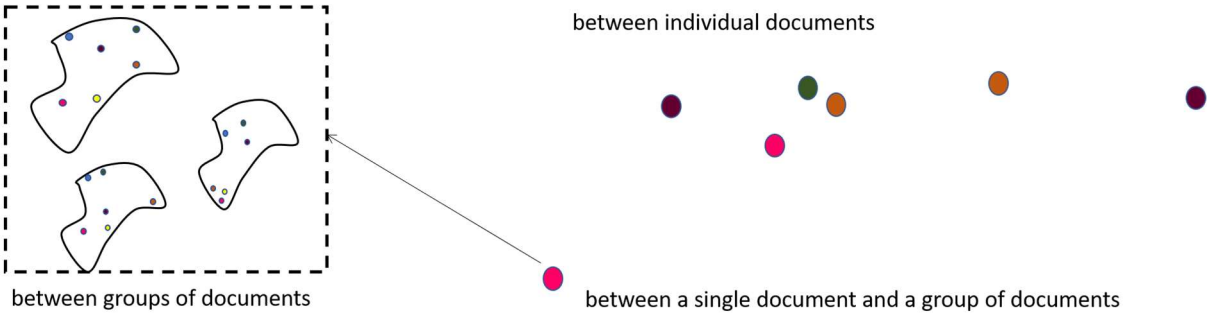
Question: To what/whom does “Zila” refer?

Although "Zila" is not a widely recognized name, its meaning can be inferred from the context provided in the documents. Document1 mentions having coffee at Zila, suggesting it is a place where beverages are served. Document2 provides additional spatial context, indicating that Zila is a physical location at the corner of a street. Document3 reinforces this interpretation by describing people going there early in the morning, a typical behavior associated with cafés or breakfast spots. Taken together, these contextual clues allow a human reader—or, in advanced natural language processing, a system analyzing the text—to reasonably conclude that Zila refers to a café, even without prior knowledge of the place.

Text1: "Early in the morning, I drink milk with a chocolate croissant"
Text2: "I have my breakfast consisting of milk and a small bread"
Text3: "Every morning I neither drink milk nor eat a chocolate croissant"
Text4: "Milk and some bread are better than chocolate"

Question: What is the degree of similarity between these texts?

1.33. Similarity Measurement



Similarity can be assessed in different ways:

- **Between groups of documents** – evaluating how closely related two sets of documents are.
- **Between individual documents** – comparing the content of two single documents.
- **Between a document and a group of documents** – determining how closely a document aligns with a collection or cluster of documents.

Grouping similar documents helps accelerate the search process.

Documents that are more similar (i.e., coherent) are more likely to be relevant, especially if one of the documents in the same category is considered a response to a given query.

Consequently, it is essential to define an appropriate similarity measure.

1.34. Term Weighting (Feature Weighting)

Assigning weights to terms (features) helps quantify their importance within a document or across a collection of documents. Common weighting schemes include TF (Term Frequency), IDF (Inverse Document Frequency), and their combination TF-IDF. These weights are essential for representing documents as vectors in a vector space model.

Several similarity measurement methods exist, and the choice of method depends on the characteristics of the data. In a vector space, similarity calculation is based on computing the distance between vectors. The default method is the Euclidean distance, but there are many other methods such as cosine, maximum, Manhattan, Minkowski, Pearson, etc. However,

caution is required: selecting the distance measure is a crucial step since it directly influences the final results.

Euclidean Distance and Manhattan Distance

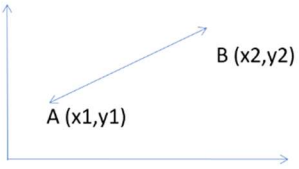
Euclidean distance is a geometric measure used to quantify the similarity (or dissimilarity) between two document vectors. It calculates the straight-line distance between points (vectors) in a multi-dimensional space, where each dimension corresponds to a term's weight. The smaller the distance, the more similar the documents are.

Term Weighting
(Feature Weighting)

Euclidean distance

$$d(u,v) = \sqrt{\sum_{j=1}^{t}(u_i - v_j)^2}$$

$$\sqrt{(x2-x1)^2 + (y2-y1)^2}$$

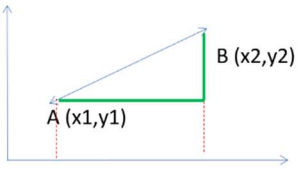


Distance Euclide donne plus d'importance au terme fréquent.

Manhattan distance

$$|x1-x2| + |y1-y2|$$

	T1	T2	T3
D1	1	7	1
D2	8	1	0
D3	3	5	0



Manhattan distance measures the distance between two vectors by summing the absolute differences of their corresponding components. Unlike Euclidean distance, which considers the straight-line distance, Manhattan distance computes the "grid-like" path (like moving along streets in a city, hence the name).

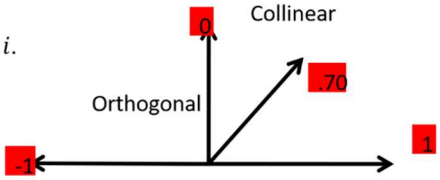
Cosine Similarity

Cosine similarity measures the cosine of the angle between two vectors in a multidimensional space. In text mining, these vectors represent documents, and each component corresponds to a term (feature).

$$cos(i,j) = \frac{\sum_{k=1}^{n} i_k \cdot j_k}{\sqrt{\sum_{k=1}^{n} i_k^2}\ \sqrt{\sum_{k=1}^{n} j_k^2}}$$

ik represents the number of occurrences of term k in document i.

The cosine value lies between –1 and +1



	T1	T2	T3
D1	1	3	2
D2	0	4	1
D3	2	1	0

$$cos(d1,d2) = \frac{1*0+3*4+2*1}{\sqrt{1+9+4}\sqrt{0+16+1}} =$$

2. Chapiter 2 Machine Learning for Text Mining

Machine learning is a key pillar of artificial intelligence. Inspired by the human capacity to learn and adapt, it aims to develop algorithms capable of simulating cognitive processes that enable the acquisition, generalization, and application of knowledge to new problems. Unlike traditional computer programs, which are based on fixed instructions and deterministic functions $f(x) = y$ machine learning relies on adaptive models that improve based on data. It does not require complete knowledge of the underlying function: it approximates it by learning from examples. This approach is particularly suited to dynamic contexts, such as text analysis, where data is continually evolving (new expressions, language changes, different contexts). When a conventional program fails in the face of these variations, a machine learning model can be retrained to adapt to the new conditions.

Machine learning draws on a multidisciplinary body of knowledge: probability theory, statistics, mathematical analysis, combinatorics, etc. It now encompasses dozens of algorithms—supervised, unsupervised, or hybrid—continuously improved by research to meet practical real-world needs. Its fundamental objective is to build models capable of extracting regularities in training data and then generalizing this knowledge to previously unobserved cases. The success of this learning is evaluated through independent tests, which measure the model's ability to handle new examples.

Heuristics at the heart of adaptive reasoning

The term heuristic (from the Greek heuriskein, "to find") refers to both a method of discovery and a pragmatic approach to problem-solving. In philosophy and science, it refers to any process or reasoning that aids in the search for new knowledge, often without a guarantee of absolute certainty. A heuristic method proceeds by successive approximations, progressively eliminating alternatives to converge toward a satisfactory or optimal solution. Unlike algorithmic methods, which are strictly deterministic and formal, heuristics rely on intuition, empirical experience, and practical rules adapted to the context.

In computer science and artificial intelligence, heuristics refers to strategies designed to reduce the complexity of the solution search by relying on similarities with previous cases or on filtering criteria. It allows for the finding of plausible solutions in situations where exhaustive computation would be too costly or impossible (combinatorial problems, dynamic environments, reinforcement learning, etc.). As Jean-Louis Le Moigne (famous French specialist in epistemology and systems) points out, a heuristic is a formalized, plausible, but not certain, reasoning approach that tends toward a satisfactory solution. Thus, in intelligent systems, heuristic methods play a fundamental role: they mimic the adaptive strategies of human reasoning, promoting discovery, adaptation, and decision-making in uncertain contexts—all essential qualities of machine learning and artificial cognition.

2.1.Document Categorization and Classification

Document categorization and classification are fundamental tasks in text mining, information retrieval, and natural language processing (NLP). They aim to automatically organize large collections of texts into meaningful categories or classes to facilitate retrieval, analysis, and decision-making.

Beyond simple organization, these methods play a crucial role in applications such as spam filtering, sentiment analysis, topic identification, medical report classification, and news categorization.

By assigning documents or data streams to predefined or emerging categories, classification systems enable intelligent information management and support real-time decision-making in complex environments.

In the emerging context of the Artificial Intelligence of Things (AIoT), document and context classification extend beyond traditional textual data. Here, heterogeneous information coming from IoT devices, sensors, user interactions, and ambient systems can be semantically analyzed and categorized to infer the current context of activity, user intention, or environmental state. This process, known as **context mining**, allows AIoT systems to dynamically adapt behaviors, automate reasoning, and provide personalized or context-aware services.

For example, in ambient intelligence environments, sensor-generated events (e.g., temperature, movement, sound, physiological signals) can be treated as "documents" describing the state of the environment. By applying Bayesian, fuzzy, or deep learning classifiers, the system can categorize these data streams into high-level situations such as *"user studying,"* or *"emergency detected".* This form of classification bridges reasoning and data-driven inference, forming the core of ubiquitous computing and intelligent decision support.

Ultimately, integrating document classification with IoT-driven context mining enables a seamless interaction between humans, devices, and intelligent environments, where knowledge extraction, prediction, and adaptation operate continuously and autonomously.

2.2.Naïve bayes (supervised algorithm)

Bayesian classification is both a supervised learning method and a statistical classification approach. It relies on an underlying probabilistic model that represents the uncertainty of predictions in a rigorous way, by determining the probabilities associated with different possible outcomes. Its main goal is to minimize the probability of misclassification (or risk) by assigning each observation to the most probable class according to **Bayes' theorem**, proposed by *Thomas Bayes (1702–1761)*.

This approach provides practical learning algorithms capable of combining prior knowledge with observed data. It is also robust to noise in input data and incremental, since each new example can adjust the probabilities of a given hypothesis.

The Bayesian classifier relies on the conditional densities $P(x|\omega_i)$

and the prior probabilities $P(\omega_i)$

In the general case with risks, we define the discriminant function as:

g $_i(x) = -R\ (\alpha_i|x)$

Thus, maximizing g $_i(x)$is equivalent to minimizing the conditional risk.
In the case of minimal error, this simplifies to:

g $_i(x) = P\ (\omega_i|x)$ which means that the chosen class is the one with the maximum posterior probability.

Mathematically, **Bayes' theorem** expresses the probability of a hypothesis $Y = y_i$

given the data $X = x_k$ as:

$$P(Y = y_i \mid X = x_k) = \frac{P(X = x_k \mid Y = y_i)\, P(Y = y_i)}{\sum_j P(X = x_k \mid Y = y_j)\, P(Y = y_j)}$$

where x_k denotes the vector of observed attributes, and the denominator's sum extends over all possible values of Y.

Learning consists in estimating $P\ (Y\ |X = x_k)$ for new examples.

The Bayesian classifier is sometimes referred to as:

- Naïve Bayes (or Idiot Bayes),
- Simple Bayes,

when it assumes that the attributes X_i are conditionally independent of one another.

In summary, Bayesian classification provides:

- Probabilistic learning with explicit estimation of hypothesis probabilities,
- Probabilistic prediction that considers multiple weighted hypotheses,
- A theoretically optimal decision rule, even when other methods are inaccurate or heuristic.

The Naive Bayes classifier is called *"naive"* because it relies on a very simplifying (and unrealistic) assumption:

"It assumes that all features (words or attributes) are conditionally independent of one another given the class."

Example

Consider the following medical report:

"The patient shows high fever and cough."

The naive model assumes that, given the class (e.g., *infection*), the presence of the word *"fever"* does not affect the probability of seeing *"cough"*, *"patient"*, or *"high."* However, in real medical language, such symptoms are often **correlated** — for instance, *"fever"* increases the likelihood of *"infection"* or *"inflammation."*

Why this assumption?

- It makes the computation extremely simple and efficient.
- It avoids estimating complex dependencies between medical terms or symptoms.
- Despite being unrealistic, this assumption works remarkably well in practice, especially with large clinical datasets.

Principle

Naive Bayes applies **Bayes' theorem** under the conditional independence assumption between features.

Advantages

- Fast and efficient for high-dimensional text data (e.g., sparse representations using TF or TF-IDF).
- Particularly effective for tasks such as medical text classification, symptom analysis, or diagnostic prediction.

Variants

- **Multinomial Naive Bayes:** suitable for word-count or TF/TF-IDF features.

  The multinomial Naive Bayes classifier is suitable for classification with discrete features (e.g., word counts for text classification). The multinomial distribution normally requires integer feature counts. However, in practice, fractional counts such as tf-idf may also work.

  https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html

- **Bernoulli Naive Bayes:** used when features are binary (e.g., presence or absence of a term). This classifier is suitable for discrete data. The difference is that while MultinomialNB works with occurrence counts, BernoulliNB is designed for binary/boolean features.

https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.BernoulliNB.html#sklearn.naive_bayes.BernoulliNB

Applications

Naive Bayes is one of the most widely used probabilistic models in medical text mining, particularly for:

- Automatic classification of medical reports (e.g., infection vs. non-infection),
- Disease or symptom detection,
- Clinical decision support systems,
- Categorization of electronic health records (EHR).

Naïve Bayes Formulas

Conditional Probability with Laplace Smoothing

The first equation defines the conditional probability of a word www given a class c:

$$P(w \mid c) = \frac{\text{count}(w, c) + 1}{\text{count}(c) + |V|}$$

This formula estimates how likely a word is to appear in a document belonging to class c. The term **"+1"** is the **Laplace smoothing**, used to avoid assigning a zero probability to words that do not occur in the training data for that class.

- count(w,c): number of times the word www appears in documents of class ccc.
- |V|: total number of distinct words in the vocabulary.

Bayesian Decision Rule

The second equation expresses the **Bayesian classification rule**:

$$P(d \mid c) = \arg\max_{c \in C} P(d \mid c)\, P(c)$$

This means that, for a given document d, the classifier chooses the class ccc that maximizes the posterior probability P(c|d).

In practice, this is equivalent to finding the class that has both:

- a high prior probability P(c), and
- a high likelihood P(d|c) (i.e., the document fits well within that class).

Logarithmic Form for Stability

To avoid numerical underflow caused by multiplying many small probabilities, the Naïve Bayes formula is often expressed in logarithmic form:

Formula 1 :

$$\hat{y} = \arg\max_{y} \left(\log P(y) + \sum_{i=1}^{n} \log P(x_i \mid y) \right)$$

Here, the predicted class $(\hat{c})$ is the one that **maximizes the sum** of:

- the logarithm of the prior probability log P(y) and
- the sum of the logarithms of the conditional probabilities log $P\ (x_i|y)$ for each feature (or word).

Formula 2 :

$$\hat{c} = \arg\max_{c \in C} P(c) \prod_{i=1}^{n} P(x_i \mid c)$$

It assumes that a document d is composed of several words x1,x2,…,xn.
The Naive Bayes model estimates the probability of the document as the product of the conditional probabilities of each word, assuming independence between them:

$$P(d \mid c) = \prod_{i=1}^{n} P(x_i \mid c)$$

Then, the predicted class is the one that maximizes the posterior probability:

$$\hat{c} = \arg\max_{c \in C} P(c)\, P(d \mid c)$$

Practical example

We trained the Naive Bayes algorithm — see the Python code — and it gave us the following result.

	T1	T2	T3	T4	class
Text1	1	2	3	0	C1
Text2	0	1	1	0	C2
Text3	1	2	0	1	C1
Text4	0	0	0	1	C2
Text5	0	1	0	1	C2

Classify this document

Text	0	2	1	0	?

Step1 : Calculation of class probabilities

	T1	T2	T3	T4	class
Text1	1	2	3	0	C1
Text2	0	1	1	0	C2
Text3	1	2	0	1	C1
Text4	0	0	0	1	C2
Text5	0	1	0	1	C2

	T1	T2	T3	T4	class
Text1	1	2	3	0	C1
Text3	1	2	0	1	C1

P(c1) = 2/5 = 0.4

	T1	T2	T3	T4	class
Text2	0	1	1	0	C2
Text4	0	0	0	1	C2
Text5	0	1	0	1	C2

P(c2) = 3/5 = 0.6

P(c1)+P(c2) = 1

Step 2 : Compute the probabilities of the terms

$$P(w|c) = \frac{count(w,c)+1}{count(w)+|V|}$$

(w, c): Number of occurrences of the word *w* in class *c*
|V|: Number of words in the vocabulary = 4

	T1	T2	T3	T4	class
Texte1	1	2	3	0	C1
Texte3	1	2	0	1	C1

4 terms (tokens):
What is the probability of the occurrence of each term in class C1?
P(T1|C1)= (2+1)/10+4 = **0.21** P(t2|C1)=(4+1)/14=**0.35**. etc.

	T1	T2	T3	T4	class
Text2	0	1	1	0	C2
Text4	0	0	0	1	C2
Text5	0	1	0	1	C2

Do the same for C2

Step3 : Classify the document

$$P(d|c) = arg\,max_{c \in C}\, P(d|c)P(c)$$

	T1	T2	T3	T4	class
Texte	0	2	1	0	?

Probabilités[c1,c2] : [[0.49595885 0.50404115]]

The text belongs to C2

2.3. K-Nearst Neighbors (KNN) (supervised algorithm)

The **K-Nearest Neighbors (KNN)** algorithm can be mathematically described as follows:

1. Input:

- A training dataset $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$, where each $x_i \in \mathbb{R}^d$ is a feature vector of dimension d (for example, $x_i = (x_{i1}, x_{i2}, \ldots, x_{id})$) and $y_i \in \{C_1, C_2, \ldots, C_k\}$ is the class label of the point x_i.
- A test point $x_{\text{test}} \in \mathbb{R}^d$ that we want to classify.

Selecting the K Nearest Neighbors:

Let $N_K(x_{\text{test}})$ be the set of indices of the K nearest neighbors, where:

$$N_K(x_{\text{test}}) = \{i_1, i_2, \ldots, i_K\} \quad \text{such that} \quad d(x_{\text{test}}, x_{i_j}) \leq d(x_{\text{test}}, x_{i_l}) \quad \forall j < l$$

Classifying the Test Point x_{test}

$$C_{\text{maj}} = \arg\max_C \left(\sum_{i=1}^{K} \mathbf{1}_{y_i = C} \right)$$

where $\mathbf{1}_{y_i=C}$ is the indicator function, which is 1 if $y_i = C$ and 0 otherwise, and $\arg\max$ returns the class with the highest number of occurrences among the K neighbors.

Example

Create a small medical *dataset*

"The patient shows symptoms of high blood pressure and mild chest pain.",

"MRI results indicate possible brain inflammation and neural lesions.",

"Blood tests reveal elevated glucose levels suggesting diabetes.",

"X-ray shows fracture in the right arm after an accident.",

"The patient was given antibiotics and intravenous fluids for infection.",

"Therapy includes physical rehabilitation after knee surgery.",

"Treatment involves chemotherapy sessions twice a week.",

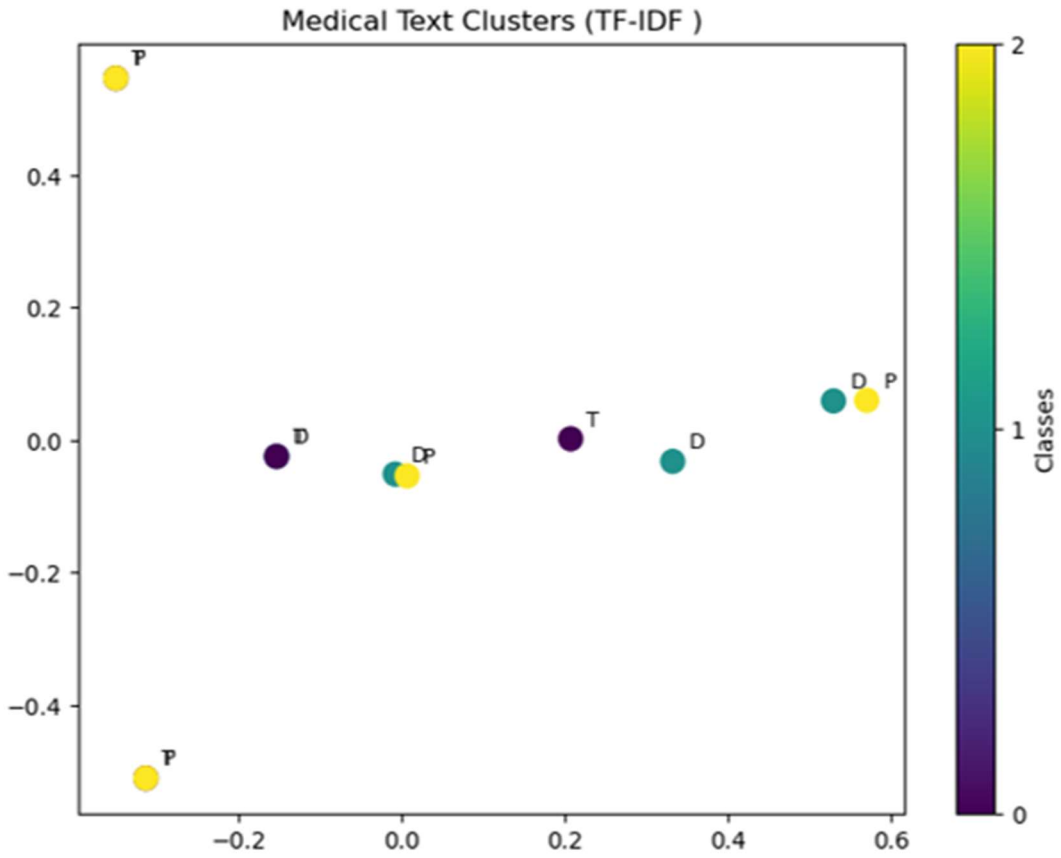
"Prescribed physiotherapy to recover muscle strength.",

"Prescribed 5mg of amlodipine once daily to control hypertension.",

"Patient should take insulin before meals and monitor glucose levels.",

"Administered paracetamol for fever and pain management.",

"Medication includes vitamin D supplements and calcium tablets."



We have three labels : "Diagnosis", "Treatment", and "Prescription"

We want predict the following text : "The patient received antibiotics for a bacterial infection."

En principe on obtient Treatment, mais cela dépendera fortement

Des mesures de similarités utilisées (Cosine, TF, TFIDF, etc.)

Cross-validation

The performance of the k-nearest neighbors algorithm can also be assessed using cross-validation, a method for measuring its generalization capacity. The principle involves extracting a small subset of individuals from the dataset whose classes are known, then applying the

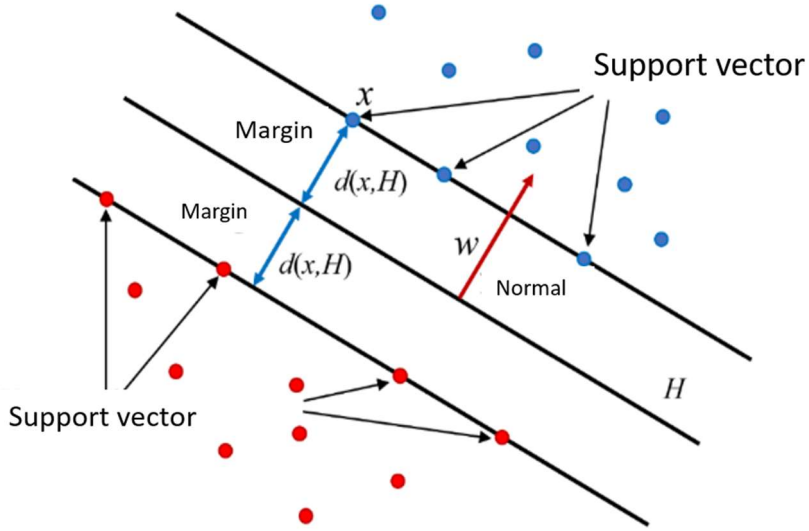
algorithm to this group, treating it as a test set. The predictions obtained are then compared with the actual classes to determine the number of times the algorithm is incorrect.

From these results, the average misclassification rate, denoted t, is calculated, defined as the ratio of the number of misclassified individuals to the total number of individuals. The closer the value of t to 0, the better the predictive quality of the model. Conversely, when $t > 0.5$, the classification is considered to be of poor quality, indicating that the model is unable to correctly distinguish between classes. Thus, cross-validation is an essential tool to evaluate the reliability and robustness of the k-NN classifier.

2.4. Support Vector Machine "SVM" (Supervised algorithm)

Case of binary classification (linearly separable)

For two given classes of examples, the goal of SVM is to find a classifier that separates the data and maximizes the distance between these two classes. With SVM, this classifier is a linear classifier called a hyperplane. In the figure below, a hyperplane is determined that separates the two sets of points. The closest points, which alone are used to determine the hyperplane, are called support vectors.



$$f(x) = \sum_{i=1}^{n} w_i x_i + b = \langle w, x \rangle + b$$

- w is the vector orthogonal (normal) to the hyperplane.
- b is the offset (bias) from the origin.
- ⟨·,·⟩ denotes the usual dot (inner) product in \[\mathbb{R}^n\].

$$x = \mathbb{R}^d : \langle x, y \rangle = \Sigma_{i=1}^{d} x_i y_i$$

Once the separator f(x) is found, the classification of a new data point is done using a simple zero-threshold decision:

- f(x)=0 : the element lies on the separating boundary, no decision.
- f(x)>0: class 1.
- f(x)<0: class 0.

Find a decision function

$$(\boldsymbol{x}) = \boldsymbol{w^T} + \boldsymbol{b}$$

Hard Margin SVM

The optimal separator is the solution to the optimization problem.

$$\min_{(\boldsymbol{w},\boldsymbol{b})\in\mathbb{R}^{d+1}} \frac{1}{2}\|\boldsymbol{w}\|^2$$

Under the constraint $\forall\, \boldsymbol{i} = \mathbf{1}, \dots, \boldsymbol{n}: \boldsymbol{y_i}\left(\boldsymbol{w^T x_i} + \boldsymbol{b}\right) \geq \mathbf{1}$

1. $\mathrm{w} \in \mathbb{R}^{\mathrm{d}}$ weight vector of the classifier
2. $\mathrm{b} \in \mathbb{R}^{\mathrm{d}}$: bias
3. $\mathrm{x_i} \in \mathbb{R}^{\mathrm{d}}$: input vector (training data)
4. $\mathrm{y_i}\{-1, +1\}$: class label
5. $\|\mathrm{w}\|_2^2 = \mathrm{w^T w}$: quantity to minimize.
6. The distance between the two marginal hyperplanes Margin = $\frac{2}{\|\mathbf{w}\|}$ because the points closest to the hyperplane satisfy the equality constraint in the optimization problem $\mathbf{y_i}\left(\mathbf{w^T x_i} + \mathbf{b}\right) = \mathbf{1}$ so they are at a distance equal to the margin from the separating hyperplane $\frac{1}{\|\mathrm{w}\|}$ and since we have one on each side, therefore $\frac{2}{\|\mathrm{w}\|}$

And thus minimize W => maximize$\frac{2}{\|\mathrm{w}\|}$ for computational reasons, we do min $\frac{\mathbf{1}}{\mathbf{2}}\|\mathbf{w}\|^{\mathbf{2}}$

3. Chapter 3 unsupervised algorithm

An unsupervised algorithm is a type of machine learning method that learns patterns or structures from data without using labeled outputs. Unlike supervised learning, where each training example has an associated label, unsupervised algorithms only have input data and must infer relationships, groupings, or distributions on their own. Common unsupervised tasks include clustering, dimensionality reduction, and anomaly detection.

In text mining, unsupervised algorithms are used to analyze and extract patterns from text data without relying on predefined labels. They help in discovering hidden structures, such as grouping similar documents (clustering), identifying underlying topics (topic modeling), or reducing dimensionality of large term-document matrices (e.g., via Latent Semantic Analysis). These methods are particularly useful when labeled datasets are not available or when the goal is exploratory analysis.

3.1.Cluster creation (class) with K-means.

Is an unsupervised machine learning method. It allows grouping elements such as topics, documents, images, or web pages into **K distinct clusters (classes)** that do not overlap. The number of clusters K must be specified at the start of the algorithm—to optimize the choice of the number of clusters. Unlike HCA (see the next section), within each cluster no hierarchical ordering between documents is applied. However, clustering is based on minimizing the distance between an element and the cluster centroid (i.e., the intra-cluster value) to which it belongs. With K-means, data within each group are as similar as possible (the intra-class value is minimized), and the inter-class value between groups is maximized. The principle of this algorithm can be summarized as follows:

1. Random Initialization

- The algorithm randomly selects kkk points as initial centroids.
- These points serve as references for forming the clusters.

2. Assignment

- Each point in the dataset is assigned to the nearest centroid (often based on Euclidean distance).
- Result: all points are grouped into kkk clusters.

3. Recalculation of Centroids

- For each cluster, the mean (centroid) of the points assigned to it is calculated.
- This point becomes the new centroid.

4. Iteration

- Steps 2 and 3 are repeated until:
    - the centroids move very little (convergence), or

- a maximum number of iterations is reached.

Note that if you do not specify the default algorithm parameter, the Hartigan and Wong algorithm is used by default. It should be noted that some authors use k-means to refer to a specific algorithm rather than the general method: most often the algorithm proposed by MacQueen , and sometimes the one proposed by Lloyd and Forgy. The Hartigan-Wong algorithm generally performs better than either of the others.

Data manipulation in R

```
> matrice = as.matrix(data.frame(c(2,0,5,1,2,4), c(1,3,2,1,3,0)))
> k<-kmeans(matrice, centers =rbind(c(1,2),c(3,1)), iter.max = 1,algorithm = c("Lloyd"),
trace=TRUE)
Warning message:
pas de convergence en 1 itération
> k$cluster
[1] 2 1 2 1 1 2
> k$centers
  c.2..0..5..1..2..4. c.1..3..2..1..3..0.
1            1.000000            2.333333
2            3.666667            1.000000
```

Categorization with K-means

The variable your_matrix_name refers to the matrix created after cleaning the texts, which can come from CSV files, data frames, or other sources. Refer to the practical exercises to learn how to create this type of matrix. In the following, we simply want to demonstrate the possibilities offered by R.

We will use the term-document matrix to represent the texts. To measure similarity between documents, we use the Euclidean distance. The first command calculates the distances between documents and stores the result in a variable:

distance_matrix <- dist(your_matrix_name, method = "euclidean")

Next, we apply the K-means algorithm, specifying the desired number of clusters via the centers parameter. The cluster centers can be explicitly set or left to default. Another useful parameter is nstart, which tries multiple initial configurations and selects the best one. The recommended value is 25 (see R help for details).

A minimal set of commands in R would then be:

ca <- kmeans(your_matrix_name, centers = 3, nstart = 25)

Shows the structure of the result

```
str(ca)

# Returns the cluster assignments for each text

ca$cluster

# Indicates the number of items in each cluster

ca$size

# Shows cluster centers rounded to two decimal places

round(ca$centers, digits = 2)
```

1. The number 3 represents the number of clusters (centers).
2. You can also use the cluster library to visualize clusters:

```
clusplot(as.matrix(your_matrix_name), ca$cluster, color = TRUE, shade = TRUE, labels = 2, lines = 1)
```

1. str(ca) displays the names of components you can access from ca and summarizes the operations performed by K-means.
2. ca$cluster shows how each text is assigned to a cluster. You can explore other components as well.
3. Note that cluster centers are chosen randomly; you can compare ca$cluster with your expected assignments.

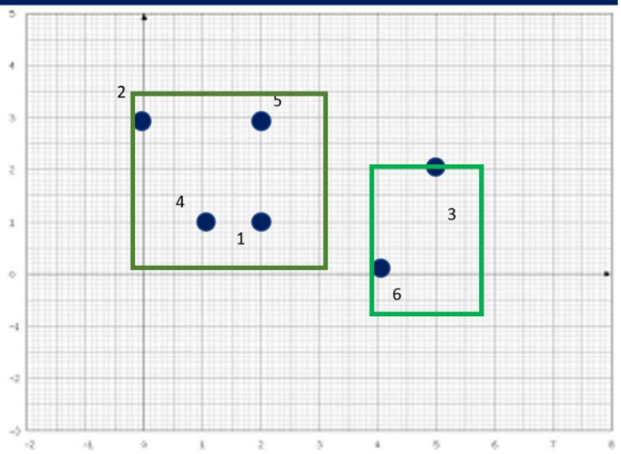
Finally, to visualize the distribution of your texts, you can execute the visualization commands above.

This minimal R example focuses on showing the features and possibilities of K-means in R. In the lab sessions, Python will be used to explore K-means and other algorithms in different programming languages, allowing students to see diverse implementations and applications.

Example

K-mean

	t1	t2
Text 1	2	1
Text 2	0	3
Text 3	5	2
Text 4	1	1
Text 4	2	3
Text 5	4	0




```
> dist(rbind(mm),method="euclidean")
          1         2         3         4         5
2 2.828427
3 3.162278 5.099020
4 1.000000 2.236068 4.123106
5 2.000000 2.000000 3.162278 2.236068
6 2.236068 5.000000 2.236068 3.162278 3.605551
```

43

- Compute the centroid of each C_i with respect to the centers.
  The best trick is to position them as far apart from each other as possible

	t1	t2
Text 1	2	1
Text 2	0	3
Text 3	5	2
Text 4	1	1
Text 4	2	3
Text 5	4	0

c1(1,2) et **c2**(3,1)

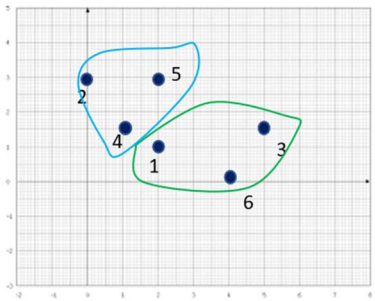
- Assign each entity to C_i such that the centroid is the closest to the entity (i.e., $d(e, centroid)$ is minimized)

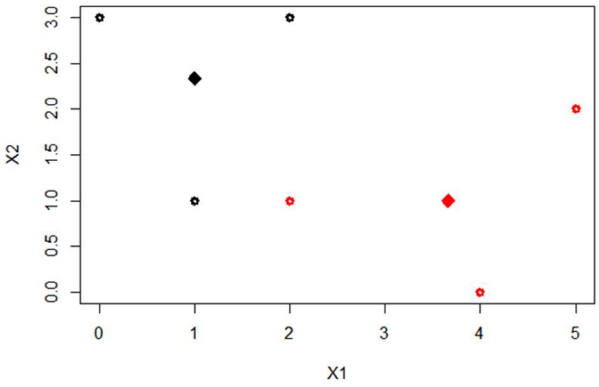
	C1	C2	cluster
Text 1	1.41	1	2
Text 2	1.41	3.6	1
Text 3	4	2.23	2
Text 4	1	2	1
Text 5	1.41	2.23	1
Text 6	3.6	1.41	2

Step 3: Recalculate the New Center
In each new cluster, compute the updated centroid.

$$\overrightarrow{X_i} = \left(\frac{\sum x_i}{z_i}, \frac{\sum y_j}{z_j} \right) \qquad \overrightarrow{X_i} = \left(\frac{0+1+2}{3}, \frac{3+1+3}{3} \right) \qquad c1\_1 = (1,3)$$

$$\overrightarrow{X_i} = \left(\frac{\sum x_i}{z_i}, \frac{\sum y_j}{z_j} \right) \qquad \overrightarrow{X_i} = \left(\frac{2+5+4}{3}, \frac{1+2+0}{3} \right) \qquad c1\_1 = (3.4,1)$$





Repeat Steps 2–3 until:
The centroids stabilize,
A maximum number of iterations is reached, or
The inter-class inertia does not improve significantly.

3.2.How to Determine the Minimal Number of Clusters

For a very large corpus (thousands of texts), choosing the optimal number of clusters k can be challenging.

- If the number of clusters is too small, the resulting groups may be too general, grouping texts that are not homogeneous.
- Conversely, too many clusters can fragment information unnecessarily, making it difficult to detect interesting patterns in the texts.

The key question is: how to choose kkk to obtain an optimal partition?

One approach is to run the K-means algorithm several times with different values of k and examine the results to identify the best choice. The idea is to calculate the variance within clusters: a better partition maximizes inter-class distances while minimizing intra-class distances.

Among the existing methods, the most well-known is the **Elbow method**.

Using R with the factoextra Library

The **factoextra** library is very useful for:

- Extracting and visualizing results from multivariate analyses (PCA, CA, MCA, FAMD, MFA, HMFA, etc.).
- Simplifying certain steps of clustering analysis.
- Providing elegant visualizations based on **ggplot2**.

Elbow Method (Within-Cluster Sum of Squares – WCSS)

Steps to follow:

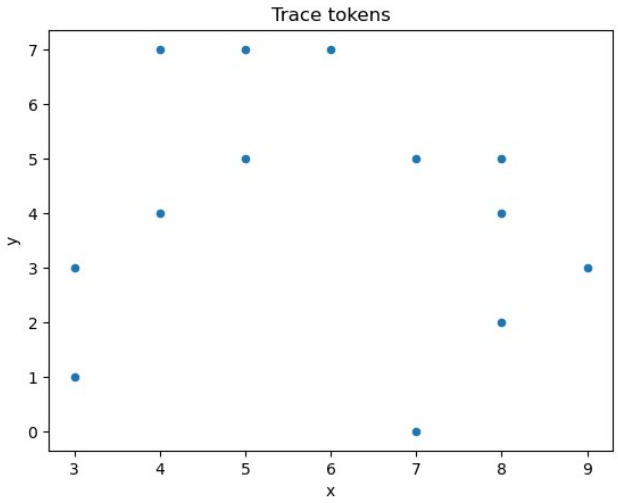
1. Apply K-means for different values of k (e.g., from 1 to 10).
2. Calculate the within-cluster inertia for each kkk:
    - This is the sum of squared distances between points and their cluster centroid.
3. Plot a graph:
    - X-axis: number of clusters k
    - Y-axis: within-cluster inertia
4. Identify the **"elbow"**:
    - This is the point where the curve starts to flatten.
    - It represents the best compromise between complexity (number of clusters) and performance (minimal inertia).

Example

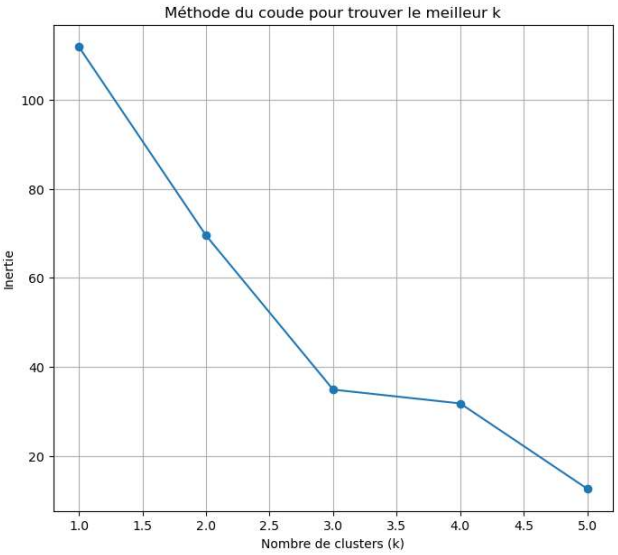
Let us consider a **very simple dataset** as follows:

```
data = {
  'x': [3, 5, 8, 3, 4, 4, 6, 7, 5, 8, 8, 7, 9],
  'y': [1, 7, 4, 3, 4, 7, 7, 0, 5, 2, 5, 5, 3]
}
```

We can first **plot the scatter plot** of these points, as shown in the figure below.



By applying the **Elbow method**, we can determine the optimal number of clusters to choose. In this simple case, we observe that **k = 3** is the best choice, because the convergence (within-cluster inertia) does not improve significantly beyond this point.



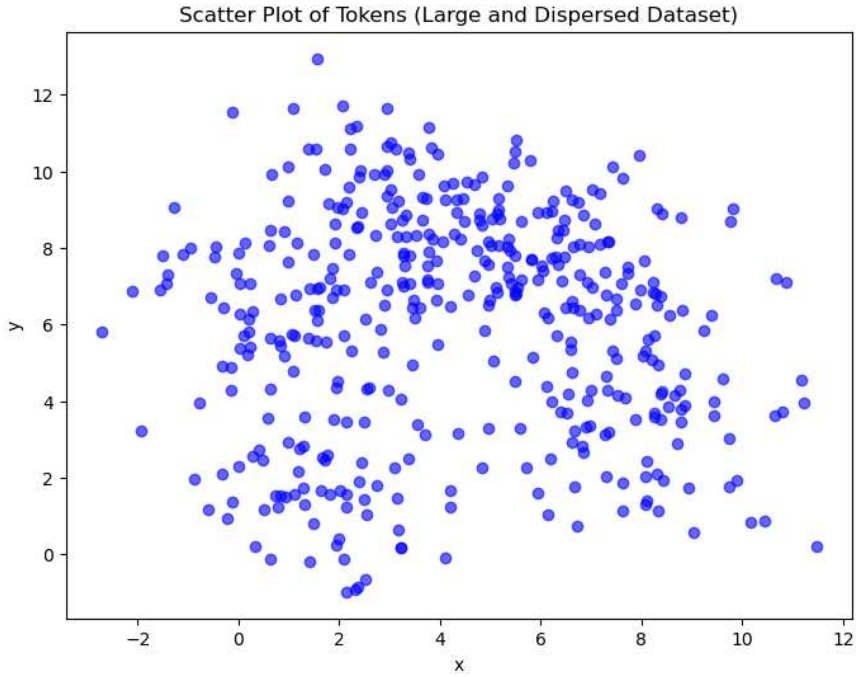
Of course, this dataset is very small, so the result is easy to interpret. However, for larger datasets, as illustrated by the next scatter plot, it becomes difficult to visually distinguish the best number of clusters.

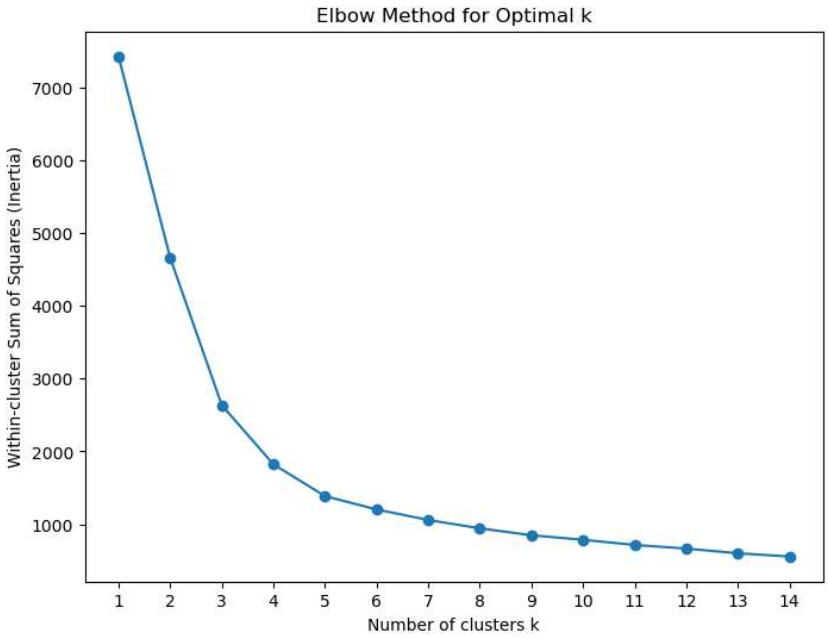
In such cases, the **Elbow method** provides a clear visual indication, allowing us to accurately determine the **optimal number of clusters (k)**. The resulting Elbow plot is shown in the following figure.

Determining the Optimal Number of Clusters with the Elbow Method

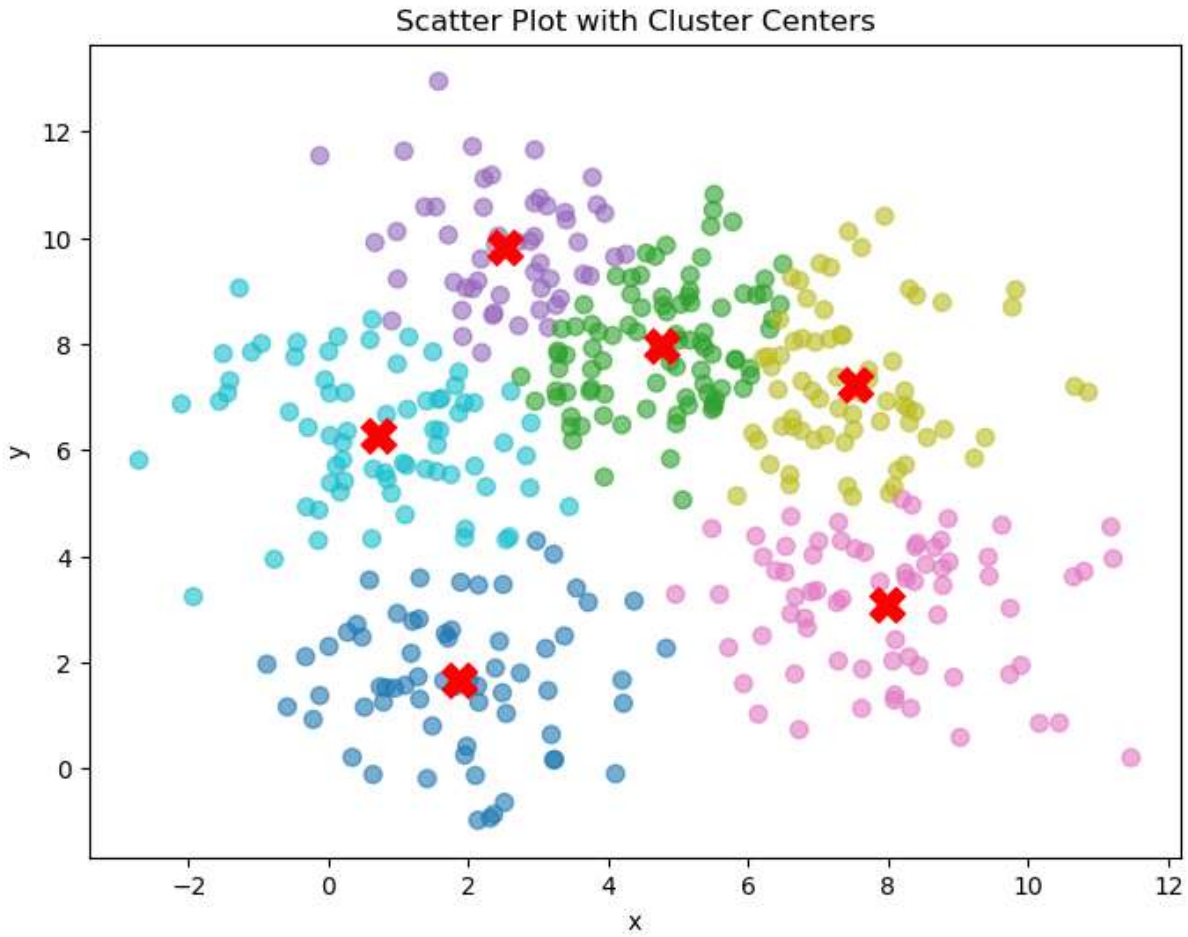
Let us consider a **larger dataset**, as illustrated in the following figures. With such a dataset, it becomes difficult to visually distinguish the optimal number of clusters. In this context, the

Elbow method is particularly useful, as it provides a clear visual indication of the **best number of clusters (k)**, allowing us to determine the optimal partition of the data accurately.

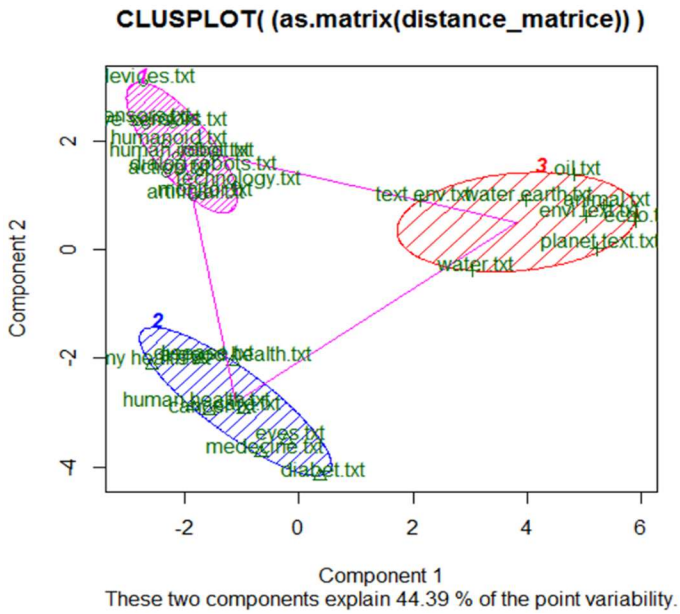


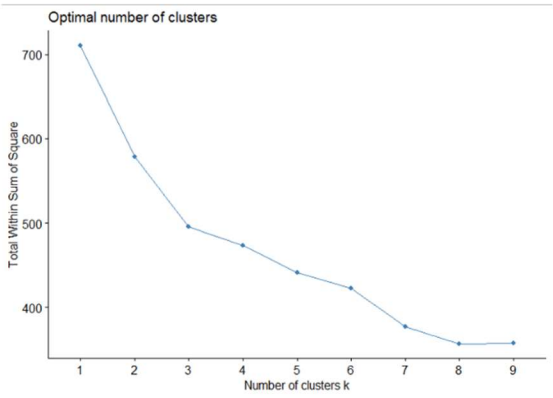


The graph shows that after k = 6, the improvement in convergence (within-cluster inertia) becomes negligible, indicating that 6 clusters is a reasonable choice for this dataset



Another example





3.3.Evaluating Cluster Quality with the Silhouette Score

The Silhouette Score is a metric used to evaluate the quality of clustering. It measures how similar an object is to its own cluster compared to other clusters. The score ranges from -1 to 1:

- A value close to 1 indicates that the object is well matched to its own cluster and poorly matched to neighboring clusters.
- A value close to 0 indicates that the object lies on or near the boundary between two clusters.
- A value close to -1 indicates that the object may have been assigned to the wrong cluster.

The average silhouette score over all points provides an overall measure of how well the clusters are separated. It is often used alongside methods like Elbow to choose the optimal number of clusters (k).

By defining these sets, we can compute clustering metrics such as intra-cluster distances, centroids, and the Silhouette Coefficient.

$$I_k = \{ i \in [[1, N]] \mid C(i) = k \}$$

This notation represents the set of indices of elements belonging to cluster k:

- N is the total number of points.
- C(i) is the function that returns the cluster assigned to point i.
- I_k contains all indices iii such that point iii belongs to cluster k.

The Silhouette Coefficient measures how similar a sample is to its own cluster compared to other clusters. It is calculated using the mean intra-cluster distance a and the mean nearest-cluster distance b for each sample. The coefficient for a sample is:

$$s = \frac{b - a}{\max(a, b)}$$

where b is the distance between the sample and the nearest cluster that the sample is not part of. The mean Silhouette Coefficient over all samples gives an overall measure of clustering quality.

Silhouette and Global Silhouette

The silhouette is a measure used to evaluate the quality of a clustering by combining cluster cohesion and cluster separation. For each point xix_ixi, we compute:

1. **Intra-cluster distance** (cohesion):

$$a(x_i) = \frac{1}{|I_k| - 1} \sum_{\substack{x_j \in I_k \\ j \neq i}} d(x_i, x_j)$$

where I_k is the set of points in the cluster C_k to which xi belongs. This distance corresponds to the average distance between x_i and all other points in its cluster.

2. **Distance to the nearest cluster** (separation):

$$b(x_i) = \min_{k' \neq k} \frac{1}{|I_{k'}|} \sum_{x_j \in I_{k'}} d(x_i, x_j)$$

The distance is computed for all clusters k' different from the cluster of x_i , and the minimum is taken. It represents the proximity of the point to the closest cluster to which it does not belong.

3. **Silhouette of the point**:

$$s(x_i) = \frac{b(x_i) - a(x_i)}{\max(a(x_i), b(x_i))}$$

This value ranges from −1 to 1:

- $x_i \approx 1$: the point is well-clustered and well-separated from other clusters.
- $x_i \approx 0$: the point lies on the boundary between two clusters.
- $x_i < 1$: the point is likely misassigned to its cluster.

4. **Global Average Silhouette (Ssil)**

To obtain a global measure of clustering quality, we compute the average silhouette per cluster, and then the average over all clusters:

$$S_{\text{sil}} = \frac{1}{K} \sum_{k=1}^{K} \frac{1}{|I_k|} \sum_{i \in I_k} s(x_i)$$

This formula provides a measure of overall clustering quality, taking into account both internal cohesion and separation between clusters:

1. First, compute $s(x_i)$ for each point x_i .
2. Next, take the average of the silhouettes within each cluster C_k.
3. Finally, average over all K clusters to obtain the global silhouette.

3.4.Fuzzy C-means Clustering Algorithm

Unlike K-means, which is hard clustering, **Fuzzy C-means** allows each document to belong to all clusters but with different degrees of membership. This is particularly useful for ambiguous data, which makes it well-suited for text data.

Sometimes, in an implementation, a small ε is added. Why? The "division by zero" error can occur if a point is located exactly at a cluster center, making its distance zero and thus the denominator in the calculation of μ_{ij} becomes zero.

Step 1 to 3

1. **Propose a number of clusters** and the parameter mmm (fuzziness degree), see the formula in point 3.
2. **Randomly assign membership values** to the clusters (μ_{ij}), where μ_{ij} represents the degree of membership of element i to cluster j. This matrix is called U0 (initial matrix).
3. **Compute the distances** to the new centers C1 and C2 using the following formula :

$$c_j = \frac{\sum_{i=1}^{n} (\mu_{ij})^m x_i}{\sum_{i=1}^{n} (\mu_{ij})^m}, \quad j = 1, 2, \ldots, c$$

The best approach is to calculate (μ_{ij}) ,$(\mu_{ij})^m$, $(\mu_{ij})^m \ x_i$, $(\mu_{ij})^m \ y_i$, $\sum_{i=1}^{n} \ (\mu_{ij})^m \ x_i$, $\sum_{i=1}^{n} \ (\mu_{ij})^m \ y_i$ and divide for each j (number of clusters), for example obtaining this for m=2 and c=2

Documents	Term1	Term2	u1	$(\mu_{ij})^2$	$(\mu_{ij})^2 \ x$	$(\mu_{ij})^2 \ y$	U2	$(\mu_{ij})^2$	$(\mu_{ij})^2 \ x$	$(\mu_{ij})^2 \ y$
Doc1	3	7	0.6	0.36	1.08	2.52	0.4	0.16	0.48	1.12
.....	.....	.....	.....	.....	.....	.....	....	.....	.....	.....

$$\mathbf{C1}\left(\frac{(\mu_{ij})^2 \ x}{(\mu_{ij})^2} \quad , \quad \frac{(\mu_{ij})^2 \ y}{(\mu_{ij})^2}\right) \ , \mathbf{C2}\left(\frac{(\mu_{ij})^2 \ x}{(\mu_{ij})^2} \quad , \quad \frac{(\mu_{ij})^2 \ y}{(\mu_{ij})^2}\right)$$

Modified distance by mmm: By raising the distance to the power of mmm, smaller distances (which indicate that the point is closer to the cluster center) are given a higher weight, increasing the point's membership to that cluster. Conversely, larger distances are given a lower weight, reducing the point's membership to that cluster.

Step 4 to 5

Update the fuzzy memberships: To do this, use the following formula

$$\boldsymbol{\mu_{ij}} = \frac{1}{\sum_{k=1}^{c} \ \left(\frac{\boldsymbol{dij}}{\boldsymbol{dik}}\right)^{\frac{2}{m-1}}} \quad \boldsymbol{c: cluster}$$

And the Euclidean distance

$$d_{ij} = \sqrt{\left(x_i - c_{j1}\right)^2 + \left(y_i - c_{j2}\right)^2}$$

Begin by computing the Euclidean distance from each data point to the updated cluster centroids (denoted as new distances d).

c1 (3.31, 4.95)

c2 (5.89, 3.48)

c=2 et m=2 (our case), therefore,

Documents	Term1	Term2	d1=dist(c1)	d2=dist(c2)
Doc1	3	7	2.08	4.83
.....	.....	.....	.....	.....

$$u_{ij} = \frac{1}{\sum_{k=1}^{c} \left(\frac{d_{ij}}{d_{ik}}\right)^{\frac{2}{m-1}}} = \frac{1}{\left(\frac{d_{ij}}{d_{i1}}\right)^2 + \left(\frac{d_{ij}}{d_{i2}}\right)^2}$$

compute $\mu_2 = 1 - \mu_1 = 0.156$

We obtain the new matrix U

Documents	d1	d2	u1	u2
Doc1	2.08	4.83	0.844	0.156

Repeat steps 3 and 4 until the algorithm converges

Step 6 to 7

6. Update the centers C1C_1C1 and C2C_2C2 until convergence does not change significantly

a) Convergence occurs when the cluster centers no longer change significantly between two successive iterations. You can check this in two ways:

i. Compute the variation of the membership matrix U between two successive iterations. If this variation is smaller than a threshold ε (for example, ε =0.001), the algorithm has converged.

$$max_{ij} \left\| U^{(k+1)} - U^{(k)} \right\| < \varepsilon / \quad \varepsilon = 0.01 \; et \; 0.001$$

The initial membership matrix U and the final one

documents	Term1	Term2	$u_1^{(1)}$	$u_2^{(1)}$	$u_1^{(2)}$	$u_2^{(2)}$
Doc1	3	7	0.6	0.4	0.835	0.165

Variation

Doc0 : |0.835−0.6|=0.235 |0.165−0.4|=0.235	Doc2 : |0.011−0.3|=0.289 |0.989−0.7|=0.289

The complete variation is as follows:

documents	$u_1^{(1)}$	$u_2^{(1)}$	Δ_{μ_1}	$u_1^{(2)}$	$u_2^{(2)}$	Δ_{μ_2}
Doc1	0.600	0.835	0.235	0.400	0.165	0.235
.....	.....	.....	.....	.....	.....	.....

Suppose we have found in a document the max(Δ_μ)=0.323(at the level of Doc7) So the variation is still large; we need to perform another iteration, and possibly more, to try to obtain a stable result max < (0.0001= ε)

b) Verify whether the cluster centers have converged by comparing the current centroids with the previous ones. If the difference is below a specified threshold ε, the algorithm is considered to have converged. Compute the differences using the Euclidean distance:

$$\left\| c_{j,nouveau} - c_{j,ancien} \right\| < \varepsilon$$

c) Limit the number of iterations. If the iteration count reaches a specified maximum TTT, the algorithm can be stopped even if full convergence has not been achieved.

7. Repeat steps 4–5 for the next iteration. That is, recalculate the cluster centers and membership degrees using the updated cluster centers until the algorithm converges or the maximum number of iterations is reached

3.5.Hierarchical Cluster Analysis (HCA)

HCA offers two main approaches: the first is known as *agglomerative*. This method, also called AGNES (*Agglomerative Nesting*), assumes that each text initially forms a separate cluster, called a *singleton*. Texts are then successively merged two by two to form larger clusters, and this process continues until a stopping criterion is met.

The second approach is called DIANA (*Divisive Analysis*). In contrast to AGNES, DIANA starts with all texts grouped into a single cluster and successively divides this cluster until a stopping criterion is satisfied.

Unlike the K-means algorithm, it is not necessary to specify the number of clusters to be created in advance. Furthermore, HCA provides a tree-like structure of the data known as a dendrogram. The distance at which clusters are merged or split (called the *height*) is shown on the y-axis of the dendrogram.

The principle of HCA used in our case can be summarized as follows:

1. Initialization: each individual initially represents a class (*singleton*).
2. Compute the pairwise similarity/dissimilarity between individuals using an aggregation criterion based on distances (e.g., *single linkage*, *complete linkage*, or *average linkage*).
3. At each step, form a new partition by merging the two most similar individuals (or clusters).
4. Repeat steps 2 and 3 iteratively.
5. Finally, merge all individuals into a single cluster.

The following measures are used in the HCA algorithm for computing distances:

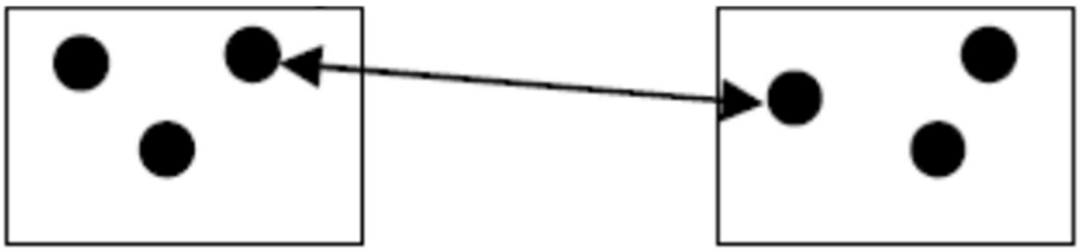
(the list of distance measures would follow here).

Single linkage (minimum distance or nearest neighbor method):

In the **single linkage** approach — also called the **minimum distance** or **nearest neighbor** method — the distance between two clusters is defined as the **smallest distance** between any pair of elements, one from each cluster.

Formally, if C_i and C_j are two clusters, the distance between them is given by:

$$d(C_i, C_j) = \min_{x \in C_i,\, y \in C_j} d(x, y)$$



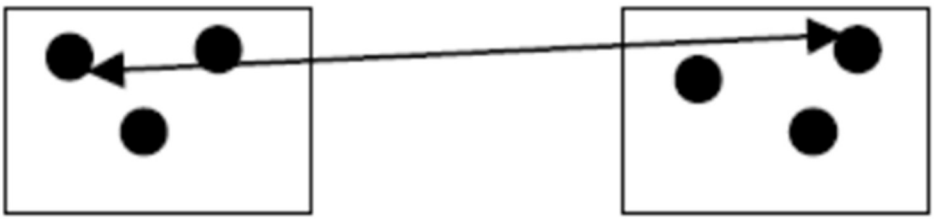
This method tends to **form elongated or chain-like clusters**, since it merges clusters based on the closest pair of points, even if other members are far apart.

Complete linkage (maximum distance or farthest neighbor method):

In the **complete linkage** approach — also called the **maximum distance** or **farthest neighbor** method — the distance between two clusters is defined as the **largest distance** between any pair of elements, one from each cluster.

Formally, if C_i and C_j are two clusters, the distance between them is given by:

$$d(C_i, C_j) = \max_{x \in C_i,\, y \in C_j} d(x, y)$$

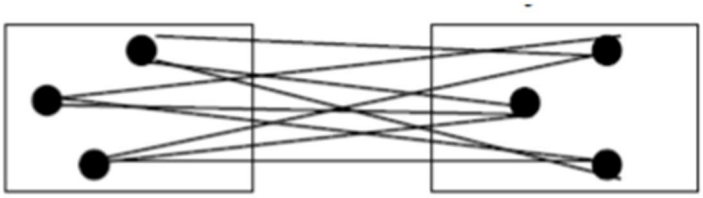


Average linkage (mean distance or group average method):

In the **average linkage** approach — also known as the **mean distance** or **group average** method — the distance between two clusters is defined as the **average of all pairwise distances** between elements belonging to the two clusters.

Formally, if C_i and C_j are two clusters containing nin_ini and njn_jnj elements respectively, the distance between them is:

$$d(c_i, c_j) = \frac{1}{|c_i|\,|c_j|} \sum_{x \in c_i} \sum_{y \in c_j} \|x - y\|^2$$



This method provides a **balance** between single linkage (which can produce long, chain-like clusters) and complete linkage (which tends to create very compact clusters). As a result, **average linkage** often yields **moderately compact and well-separated clusters**, making it a popular compromise in hierarchical clustering.

Example Categorization using HCA

Let us assume that we have a matrix called matrice_documents, which represents the data to be classified. Each row of this matrix corresponds to a document, and each column corresponds to a feature (for example, a term or a word frequency).

Run the following commands. Each command will be followed by a comment to introduce an explanation. Other commands can be used for graphical representation, but they are not covered in this section.

```
matde <- dist(matrice_documents, method = "euclidean")

hcTree = hclust(matde, "average")
```

- The hclust function allows you to specify the aggregation criterion.
- We can replace "average" with "complete", "ward", or "single".
- Each option produces a different dendrogram depending on the chosen linkage method.

```
hcTree$height
```

This command returns the **height values**, which correspond to the inertia jumps (the distances at which clusters are merged).

plot(hcTree)

This function displays the **dendrogram**. Additional parameters can be added to customize the visualization.

You can also use the **agnes** method instead of **hclust** or **diana** (see previous sections), as follows:

```
>  hcagnes <- agnes(matde, method = "average")
> hcagnes$height
 [1] 6.146395 4.472136 4.990959 6.177169 6.645826 4.898979 6.077754 6.868793
 [9] 7.066370 5.477226 5.523204 4.582576 4.943284 5.204455 4.582576 5.844617
[17] 5.930492 6.052728 6.159373 6.352959 7.802122 5.099020 5.611236 3.872983
[25] 6.115677 6.612639 6.662134 6.774242
>  pltree(hcagnes, cex = 0.7, hang = -1, main = "Dendrogram of agnes")
> #pour utiliser la méthode plot, il faut transforemr le résultat de agne
s  # à un hclust comme suit:
>  treeAgnes<-as.hclust(hcagnes)
>  plot(treeAgnes, cex=0.7,hang=-1)
>
>  # méthode diana
> hcdiana<-diana(matde)
> hcdiana$height
 [1] 6.782330 4.472136 5.291503 6.928203 7.211103 7.874008 4.898979 6.324555
 [9] 9.273618 5.196152 6.082763 4.582576 5.196152 5.196152 5.656854 6.244998
[17] 6.324555 6.557439 6.708204 6.782330 8.544004 5.099020 6.324555 3.872983
[25] 6.557439 6.855655 6.928203 7.280110
> pltree(hcdiana, cex = 0.6, hang = -1, main = "Dendrogram of diana")
```

Let us note that for each cluster formed, the paragon is defined as the individual whose coordinates are closest to the cluster's center of gravity. It serves to characterize the cluster to which it belongs.

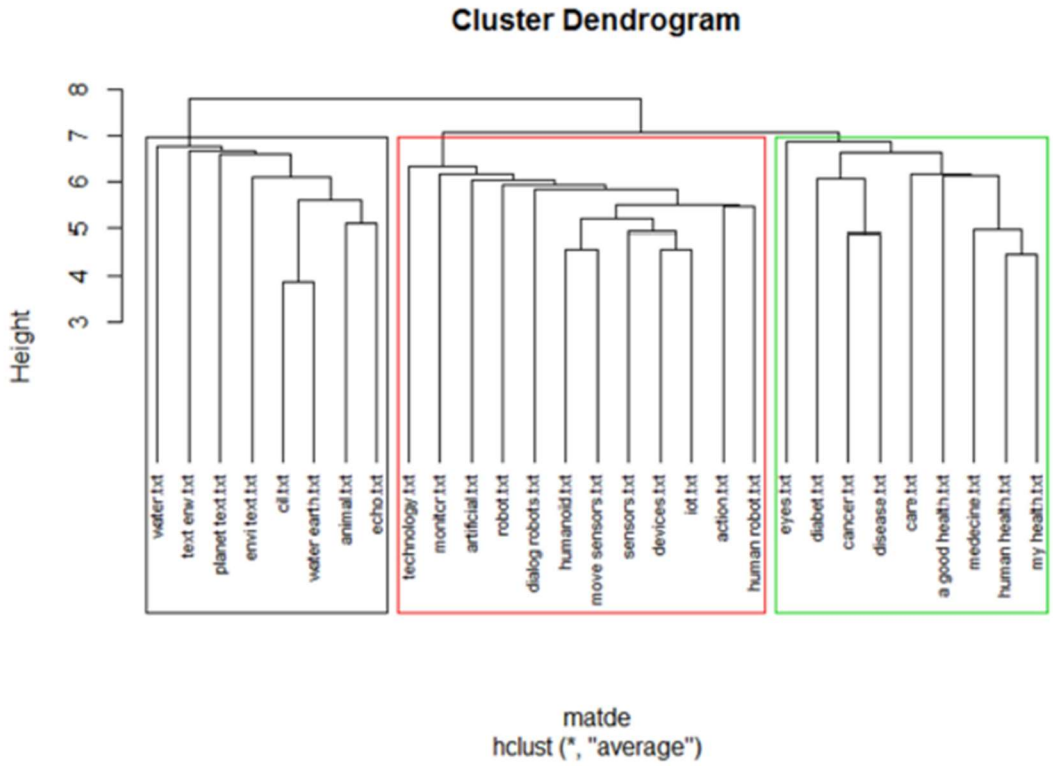
Let us now try to obtain three classes, and then four classes using HCA. To do this, we simply need to cut the dendrogram by using the following functions. An explanation is provided just after:

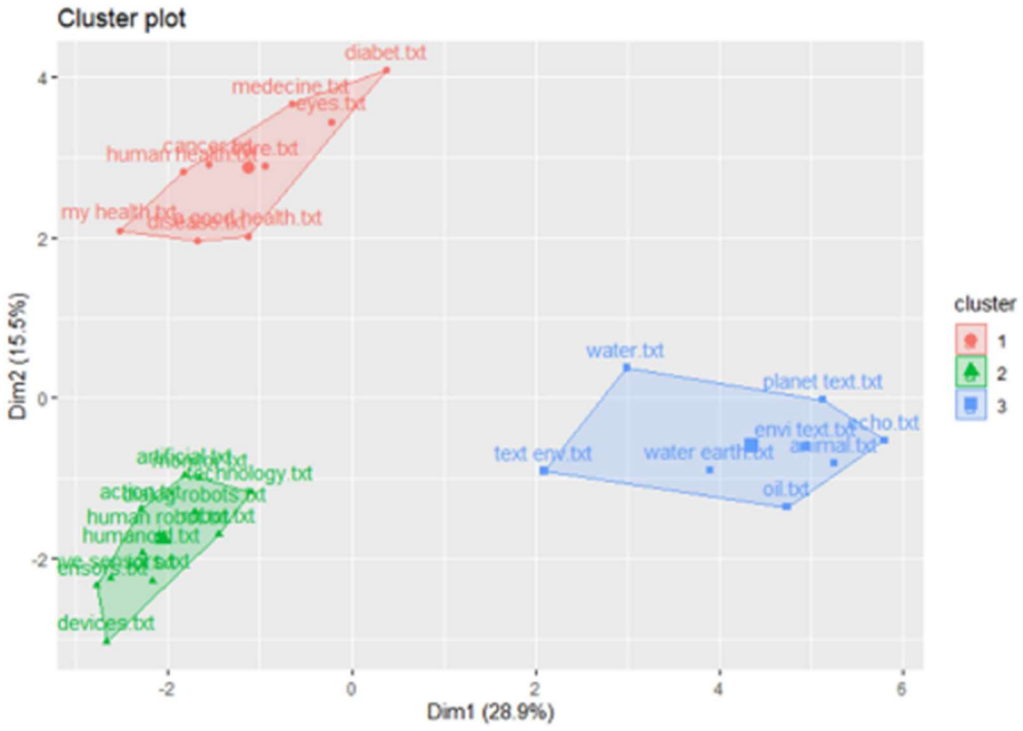
```
> plot(hcTree, cex=0.7,hang=-1)
> rect.hclust(hcTree, k = 3, border = 1:5)
> myclasses <- cutree(hcTree , k = 3)
> table(myclasses)
myclasses
 1  2  3
 9 12  8
> fviz_cluster(list(data = matde, cluster = myclasses))
> fviz_nbclust(matrice, FUN = hcut, method = "wss")

>
```

The first command draws the tree (dendrogram) with the parameter hang = -1, which aligns all the individual names at the same level, and cex is used to reduce the label size.

The cutree function returns a vector indicating cluster membership when k or h are scalar values.
Otherwise, it returns a matrix of group memberships, where each column corresponds to the elements of k or h, respectively (these are also used as column names).

Finally, the last two commands were already presented earlier for selecting the optimal number of clusters (best k). The following figures illustrate the results produced by these commands.





3.6.Practical example

Here is a **complete example** of applying **Hierarchical Cluster Analysis (HCA)** using a small **Term Frequency (TF)** table:

	t1	t2
doc1	3	6
doc2	2	0
doc3	1	2
doc4	2	7
doc5	0	4

	doc1	doc2	doc3	doc4	doc5
doc1	0.00	6.08	4.47	1.41	3.61
doc2	6.08	0.00	2.24	7.00	4.47
doc3	4.47	2.24	0.00	5.10	2.24
doc4	1.41	7.00	5.10	0.00	3.61
doc5	3.61	4.47	2.24	3.61	0.00

distance matrix

Each document (doc1 to doc5) is represented by two features (t1 and t2) corresponding to term frequencies. The goal is to group documents based on their similarity computed from these coordinates.

Computing the distances

We start by computing the **Euclidean distance** between each pair of documents:

$$d(doc_i, doc_j) = \sqrt{(t1_i - t1_j)^2 + (t2_i - t2_j)^2}$$

From the distance matrix, the smallest cell corresponds to (doc1, doc4), therefore doc1 and doc4 are the first documents to be grouped together.

Step: Applying the complete Linkage Criterion

From the distance matrix, the smallest distance corresponds to the pair **(doc_1, doc_4)**, so these two documents are merged first.

Next, we apply the **minimum average linkage** criterion to determine the distance between the new cluster {doc_1, doc_4} and another document, such as **doc_3**. The computation is as follows:

min(dist(doc1,doc3),dist(doc4,doc3))=min(4.472136,5.099020)=4.472136

min((doc_1, doc_3), (doc_4, doc_3)) = min(4.472136, 5.099020) = 4.472136

min(dist(doc1,doc3),dist(doc4,doc3))=min(4.472136,5.099020)=4.472136

If instead we use the **maximum linkage** method, we take:

max(dist(doc1,doc3),dist(doc4,doc3))=max(4.472136,5.099020)=5.099020

max((doc_1, doc_3), (doc_4, doc_3)) = max(4.472136, 5.099020) = 5.099020

max(dist(doc1,doc3),dist(doc4,doc3))=max(4.472136,5.099020)=5.099020

At each step:

- We compute the distance matrix using complete linkage (the maximum distance between elements of the two clusters).
- We display this matrix.
- We look for the smallest distance, which identifies the two closest clusters.
- We merge these clusters.
- We recalculate the new distance matrix and repeat the process.

Applying the maximum linkage we obtain :

	doc2	doc3	doc5	doc1+doc4
doc2	0.000	2.236	4.472	7.000
doc3	2.236	0.000	2.236	5.099
doc5	4.472	2.236	0.000	3.606
doc1+doc4	7.000	5.099	3.606	0.000

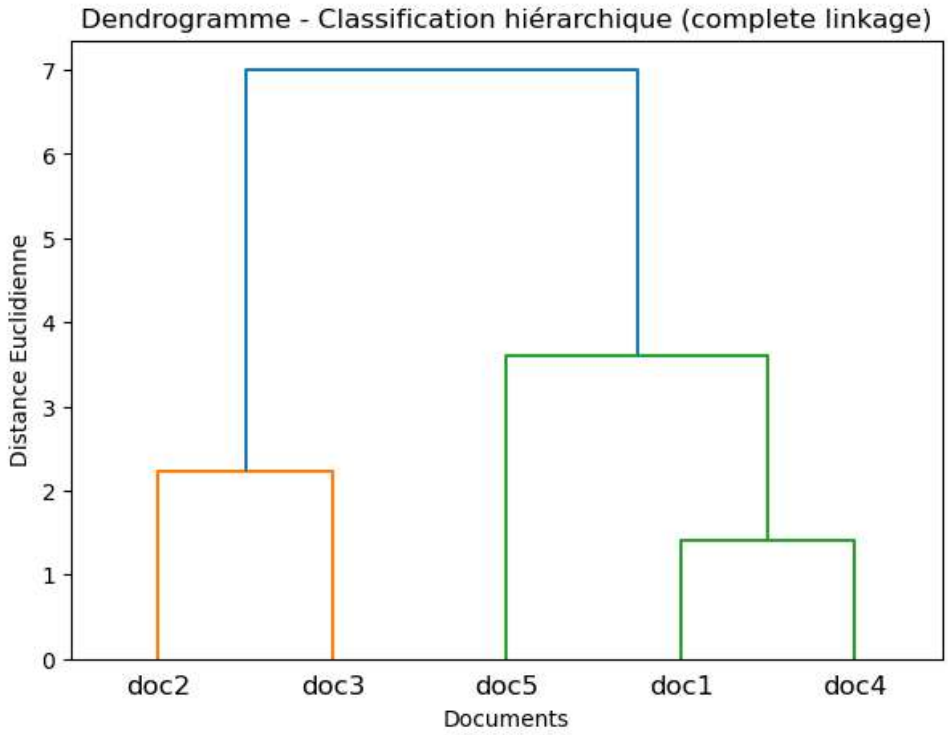
In this case, the smallest distance observed in the matrix after the first merge corresponds to the pair (doc_3, doc_2) (or doc_3, doc_5 depending on the values), leading to the next merge between doc_3 and doc_2.

	doc5	doc1+doc4	doc2+doc3
doc5	0.000	3.606	4.472
doc1+doc4	3.606	0.000	7.000
doc2+doc3	4.472	7.000	0.000

The final merge occurs between the clusters

[doc1, doc4] and [doc2, doc3, doc5]

at a distance of 7.00.



3.7.HCA (Ward's method)

Let us now reconsider the same example to illustrate how Ward's method operates using this criterion.

Minimizing the within-cluster inertia is equivalent to maximizing the between-cluster inertia.

- \[mu_{A}\] = singleton, \[mu_{B}\] = singleton (each singleton forms its own centroid).

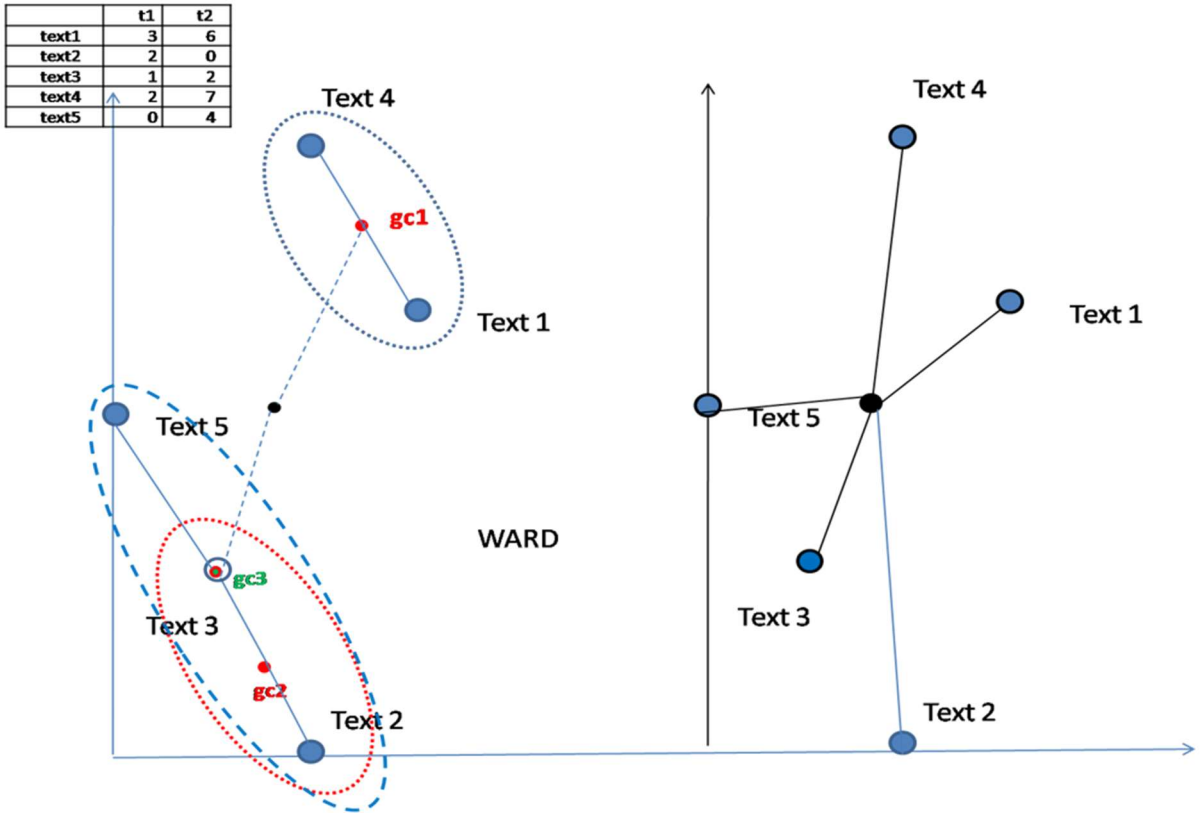
Steps:

1. From the distance matrix (minimizing the between-cluster inertia), scan the matrix to find the smallest value (i.e., identify the pair to merge).
2. Cross the corresponding columns (inspect those columns/entries).
3. Compute the centroid of the new pair.
4. Compute the within-cluster inertia (take the minimal matrix value ÷ number of singletons).
5. Recalculate the distance of every point to this new centroid.
6. Repeat the procedure starting from step 2.

$$d^2(A,B) = \frac{\mu_A \mu_B}{\mu_A + \mu_B} \|g_A - g_B\|^2$$

μA and μ_B are the sizes (number of points) of clusters A and B.
g_A and g_B are their centroids (centers of gravity).
$\|\cdot\|$ denotes the Euclidean norm.

This quantity is proportional to the increase in intra-cluster inertia caused by the merging, this is precisely the criterion used by Ward's method, which selects the merge that minimizes this value.



We are going to calculate all the Ward distances between the 5 texts (text1 to text5) step by step.

STEP 1

Reminder: for two singletons A and B

$$d^2(A,B) = \frac{\mu_A \mu_B}{\mu_A + \mu_B} \|g_A - g_B\|^2$$

Texts	t1	t2
text1	3	6
text2	2	0
text3	1	2
text4	2	7
text5	0	4

	text1	text2	text3	text4	text5
text1	0	18.5	10	1	6.5
text2	18.5	0	2.5	24.5	10
text3	10	2.5	0	13	2.5
text4	1	24.5	13	0	6.5
text5	6.5	10	2.5	6.5	0

$$d(\text{text1},\text{text2}) = \frac{1\cdot 1}{1+1}\left[(3-2)^2+(6-0)^2\right] = \frac{1+36}{2} = 18.5$$
$$d(\text{text1},\text{text3}) = \frac{1\cdot 1}{1+1}\left[(3-1)^2+(6-2)^2\right] = \frac{4+16}{2} = 10$$
$$d(\text{text1},\text{text4}) = \frac{1\cdot 1}{1+1}\left[(3-2)^2+(6-7)^2\right] = \frac{1+1}{2} = 1$$
$$d(\text{text1},\text{text5}) = \frac{1\cdot 1}{1+1}\left[(3-0)^2+(6-4)^2\right] = \frac{9+4}{2} = 6.5$$
$$d(\text{text2},\text{text3}) = \frac{1\cdot 1}{1+1}\left[(2-1)^2+(0-2)^2\right] = \frac{1+4}{2} = 2.5$$
$$d(\text{text2},\text{text4}) = \frac{1\cdot 1}{1+1}\left[(2-2)^2+(0-7)^2\right] = \frac{0+49}{2} = 24.5$$
$$d(\text{text2},\text{text5}) = \frac{1\cdot 1}{1+1}\left[(2-0)^2+(0-4)^2\right] = \frac{4+16}{2} = 10$$
$$d(\text{text3},\text{text4}) = \frac{1\cdot 1}{1+1}\left[(1-2)^2+(2-7)^2\right] = \frac{1+25}{2} = 13$$
$$d(\text{text3},\text{text5}) = \frac{1\cdot 1}{1+1}\left[(1-0)^2+(2-4)^2\right] = \frac{1+4}{2} = 2.5$$
$$d(\text{text4},\text{text5}) = \frac{1\cdot 1}{1+1}\left[(2-0)^2+(7-4)^2\right] = \frac{4+9}{2} = 6.5$$

STEP 2

The centroid of cluster

C1={text1,text4} then gC1=(2.5,6.5)

Cluster size: μC1=2

Intra(G1) =Intra(doc2, doc3, doc4, c1)=1/5 * 0.5 =1/ 10

Total group size: 5 elements (doc2, doc3, doc5, and C1 counts as 2 docs? → if we count the actual documents, then doc1 and doc4 are part of C1, so total = 5 docs).

Average intra-cluster distance: 0.5

Overall within-cluster (intra-class) value:

Redo step 1 for the centroid g_{c_1}

C1 = {doc1, doc4} → $\mu_{c_1} = \mathbf{2}$ $\mu_{c_1} = (2.5, 6.5)$

Singletons:

- **Text2 = (2,0), μ = 1 Text3 = (1,2), μ = 1 Text5 = (0,4), μ = 1**

Calculations

1. d²(C1, doc2)

$$\|g_{C1} - g_{doc2}\|^2 = (2.5-2)^2 + (6.5-0)^2 = 0.25 + 42.25 = 42.5$$

$$d^2 = \frac{2 \cdot 1}{2+1} \cdot 42.5 = \frac{2}{3} \cdot 42.5 \approx 28.33$$

2. d²(C1, doc3)

$$\|g_{C1} - g_{doc3}\|^2 = (2.5-1)^2 + (6.5-2)^2 = 2.25 + 20.25 = 22.5$$

$$d^2 = \frac{2 \cdot 1}{2+1} \cdot 22.5 = \frac{2}{3} \cdot 22.5 = 15$$

3. d²(C1, doc5)

$$\|g_{C1} - g_{doc5}\|^2 = (2.5-0)^2 + (6.5-4)^2 = 6.25 + 6.25 = 12.5$$

$$d^2 = \frac{2 \cdot 1}{2+1} \cdot 12.5 = \frac{2}{3} \cdot 12.5 \approx 8.33$$

Cluster – Singleton	Ward distance (d^2)
C1 – doc2	28.33
C1 – doc3	15
C1 – doc5	8.33

The next fusion will be C1 and doc5, because it has the smallest Ward distance (8.33).

Step 3: Merge C1 and doc5

Cluster before fusion:

- C1={text1,text4}, gC1=(2.5,6.5), µC1=2
- Singleton to merge: text5 = (0,4), µ = 1

Then the new cluster: {text1,text4,text5}, \[\mu_{C_1, text5} \]=3

Compute the new center of gravity

$$g_{C_1,\text{text5}} = \frac{1}{\mu_{C_1,\text{text5}}} \left(g_{\text{text1}} + g_{\text{text4}} + g_{\text{text5}}\right)$$

$$g_{\text{new}} = \frac{1}{3}\Big((3,6) + (2,7) + (0,4)\Big)$$
$$= \frac{1}{3}(5,17)$$
$$\approx (1.67, 5.67)$$

The **center of gravity** of the new cluster $\mathbf{g}_{new} = \mathbf{g}_{c_1,\,\text{text}_5}$ et $\boldsymbol{\mu}_{new} = \boldsymbol{\mu}_{\boldsymbol{c_1},\,\text{text}_5}$ is **(1.667, 5.667)**

Step 3b: Distances from the new cluster to remaining singletons

Remaining singletons: text2 = (2,0), text3 = (1,2)
Formula:

$$d^2(A,B) = \frac{\mu_A \cdot \mu_B}{\mu_A + \mu_B} \| g_A - g_B \|^2$$

1.d²(Cnew, text2)

$$\| g_{new} - g_{text2} \|^2 = \big(1.667 - 2\big)^2 + \big(5.667 - 0\big)^2 = 0.111 + 32.111 \approx 32.222$$

$$d^2 = \frac{3 \cdot 1}{3+1} \cdot 32.222 = \frac{3}{4} \cdot 32.222 \approx 24.167$$

2.d²(Cnew, text3)

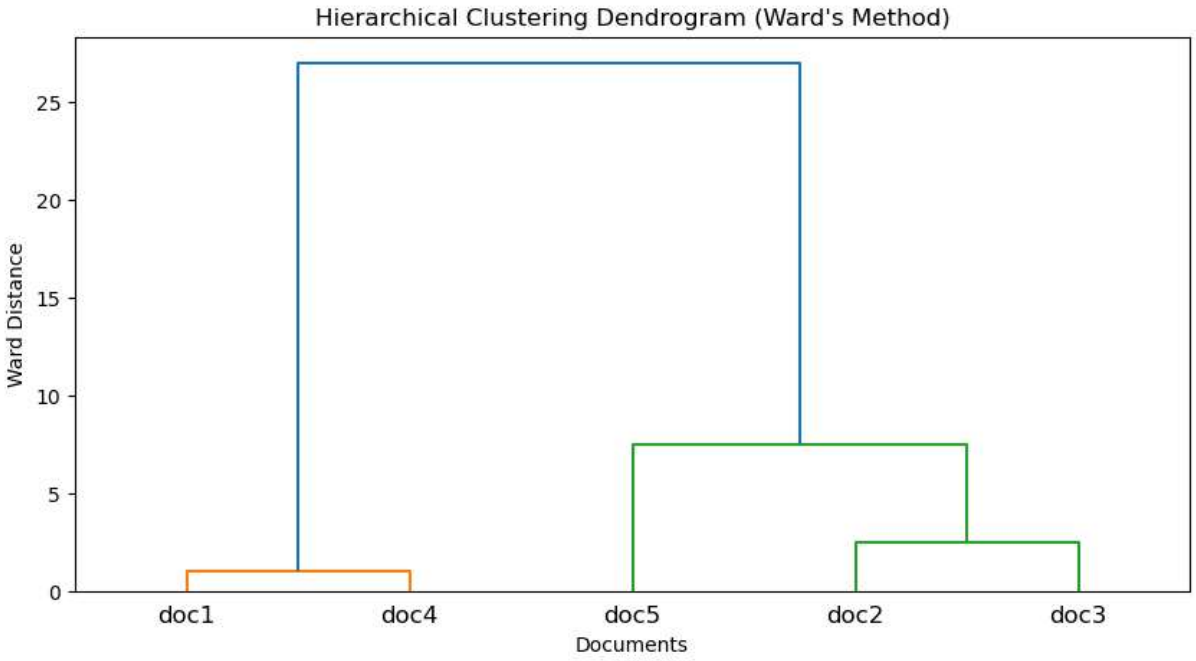
$$\| g_{new} - g_{text3} \|^2 = \big(1.667 - 1\big)^2 + \big(5.667 - 2\big)^2 = 0.444 + 13.444 \approx 13.888$$

$$d^2 = \frac{3 \cdot 1}{3+1} \cdot 13.888 = \frac{3}{4} \cdot 13.888 \approx 10.416$$

Cluster – Singleton	Ward distance (d^2)
Cnew – doc2	24.167
Cnew – doc3	10.416

The **next fusion** will be $\boldsymbol{\mu}_{\boldsymbol{c_1},\,\text{text}_5}$ **and doc3**, because it has the smaller distance (10.416).

Continue the calculation after merging



Why your merges are the same

- In your small 5-point example:
- Text1=(3,6), text2=(2,0), text3=(1,2), text4=(2,7), text5=(0,4)
- Because the distances are well separated, the cluster pairs chosen by Ward are the same as if you just used HCA with Euclidean distances.
- In small, simple datasets, other linkage methods (like single or complete linkage) may also give the same merges at first steps, but differences appear in larger or less regular datasets.

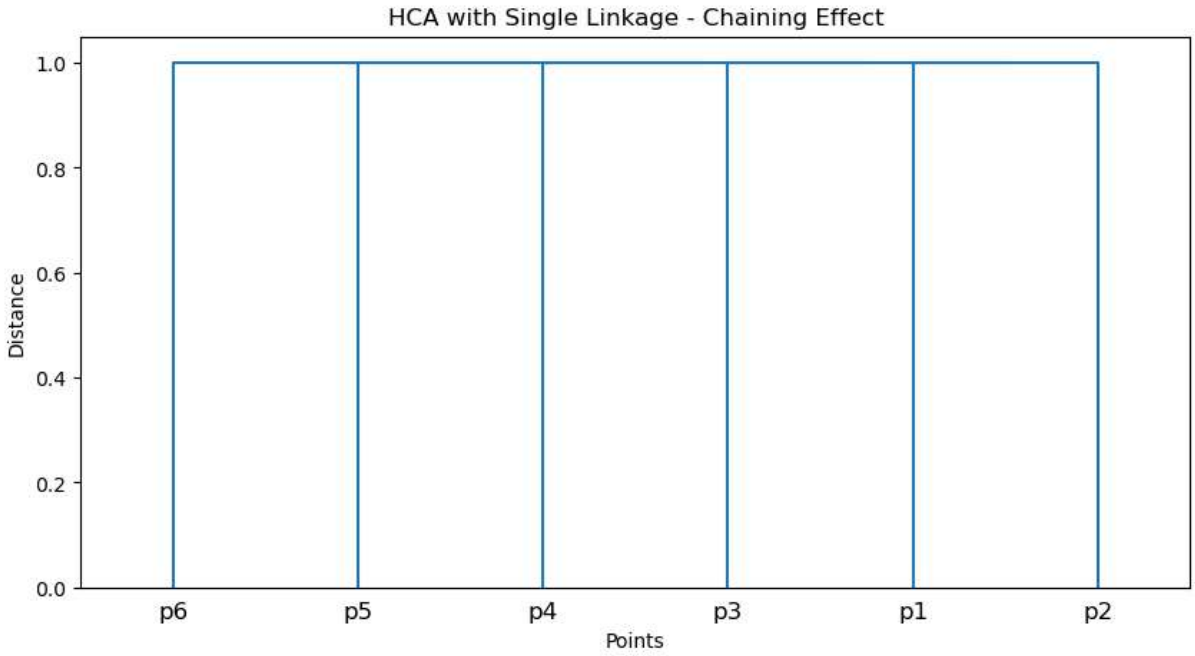
If we take the following example:

points=[(0,0),(1,0),(2,0),(3,0),(4,0),(5,0)]

- The points are **arranged in a line**, forming a natural “chain.”

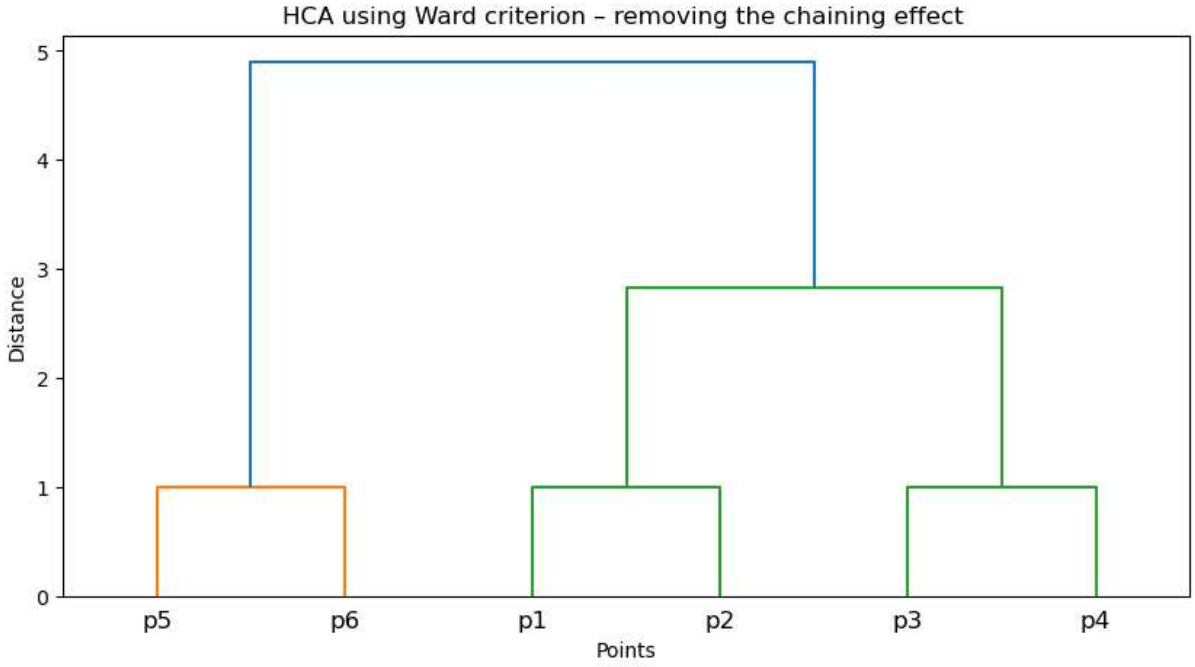
Using single linkage:

- Clusters are merged based on the **minimum distance between points**.
- Since each point is closest to its immediate neighbor, merges happen **one by one along the line**, creating a **chaining effect**.
- The dendrogram shows a long sequence of small merges instead of compact clusters.



Using Ward's method:

- Ward merges clusters to **minimize the increase of within-cluster variance**.
- This encourages forming **compact clusters**, avoiding merges that create long chains.
- The dendrogram shows **more balanced cluster merges**, eliminating the chaining effect.



The chaining effect

The chaining effect occurs when clusters form elongated chains due to merging based solely on the smallest pairwise distances, often seen in single linkage HCA. Using Ward's criterion mitigates this effect by favoring compact, balanced clusters that minimize within-cluster variance.

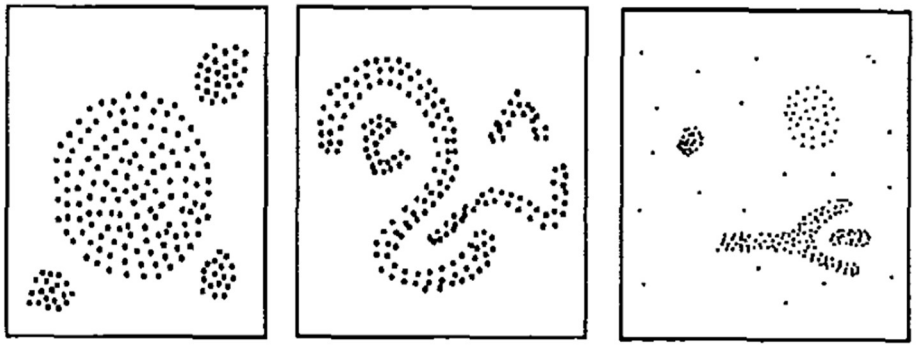
3.8.DBSCAN algorithm

Density-based Spatial Clustering of Applications with Noise (DBSCAN), Ester et al. (1996), derived from human natural clustering

Objective:
The goal is to identify dense regions, which can be measured by the number of objects close to a given point.

Principle:
Given a set of points and an integer k, the algorithm aims to divide the points into k groups, called clusters, that are homogeneous and compact. Consider the example below:

When talking about density, points that are close to a given point have very small distances between them



Required parameters:

- Epsilon ("eps"): The eps parameter defines the neighborhood radius around a point x. This is called ε (Euclidean distance is used).
- Minimum points ("MinPts" / min sample): The minimum number of neighbors within the "eps" radius. The center of a group is called a Core Point.

Secondary parameters:

- **Border Point**: Not a Core Point, it lies on the edge of a cluster and cannot form a full neighborhood circle.
- **Noise Point (outlier)**: Neither a Core Point nor a Border Point; it has no chance of being part of any cluster.

3.9. DBSCAN Algorithm

For each point in the dataset:

1. Calculate the distance between the current point (pip_ipi) and all other points. Find all points within a distance **eps** from the starting point (pip_ipi). Any point with a number of neighbors greater than or equal to **MinPts** is marked as a **Core Point** or as visited — this defines the ε -**neighborhood** of the observation.
2. For each Core Point, if it is not already assigned to a cluster, create a new cluster. Recursively find all density-connected points and assign them to the same cluster as the Core Point.
3. Continue with all unvisited points. Any point that does not belong to a cluster is considered a **Noise Point (outlier)**.

Advantages:

1. Unlike K-means, DBSCAN does not require the user to specify the number of clusters in advance.
2. DBSCAN can detect clusters of any shape; clusters do not need to be circular.
3. DBSCAN can identify outliers.

Note: The choice of **eps** is critical — setting it too high or too low can result in too few or too many noise points.

Method for determining the optimal eps value

The method proposed here consists of calculating the distances to the k nearest neighbors in a point matrix.

The idea is to compute the average distance of each point to its k nearest neighbors. The value of **k** is specified by the user and corresponds to **MinPts**.

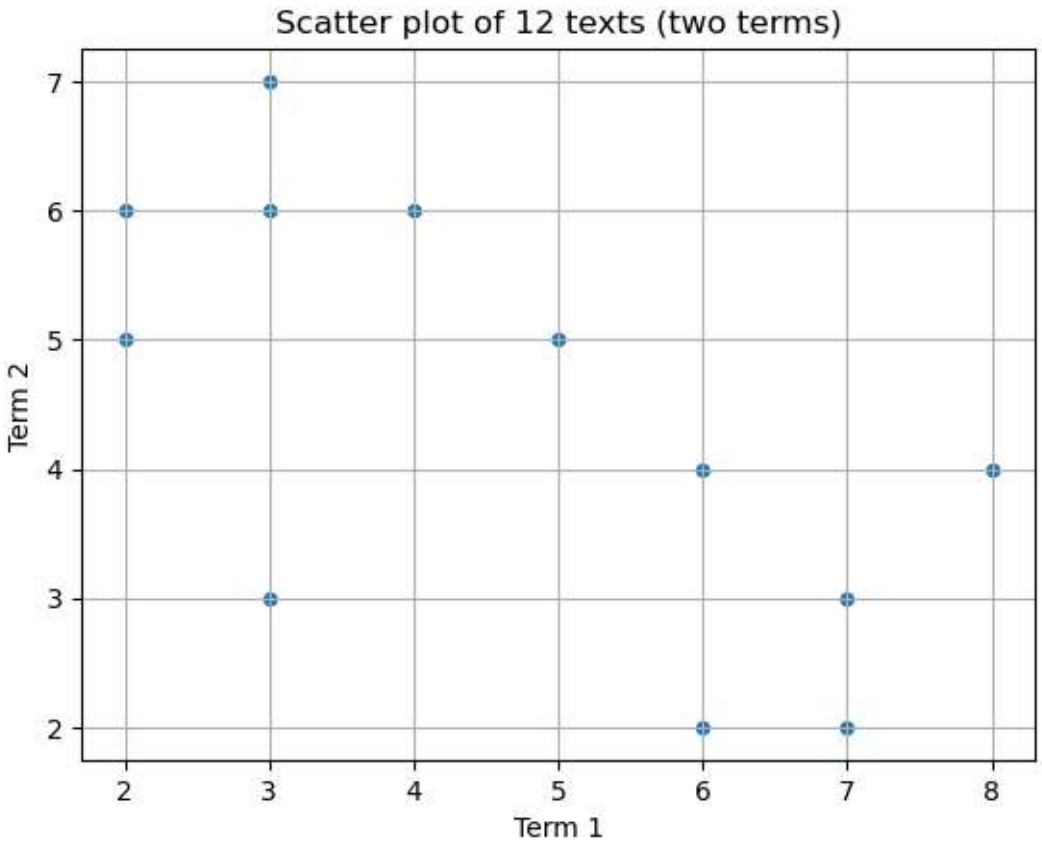
Next, these k-distances are plotted in ascending order. The goal is to identify the **"knee"**, which corresponds to the optimal **eps** parameter.

A knee represents a threshold where a sudden change occurs along the k-distance curve.

Example

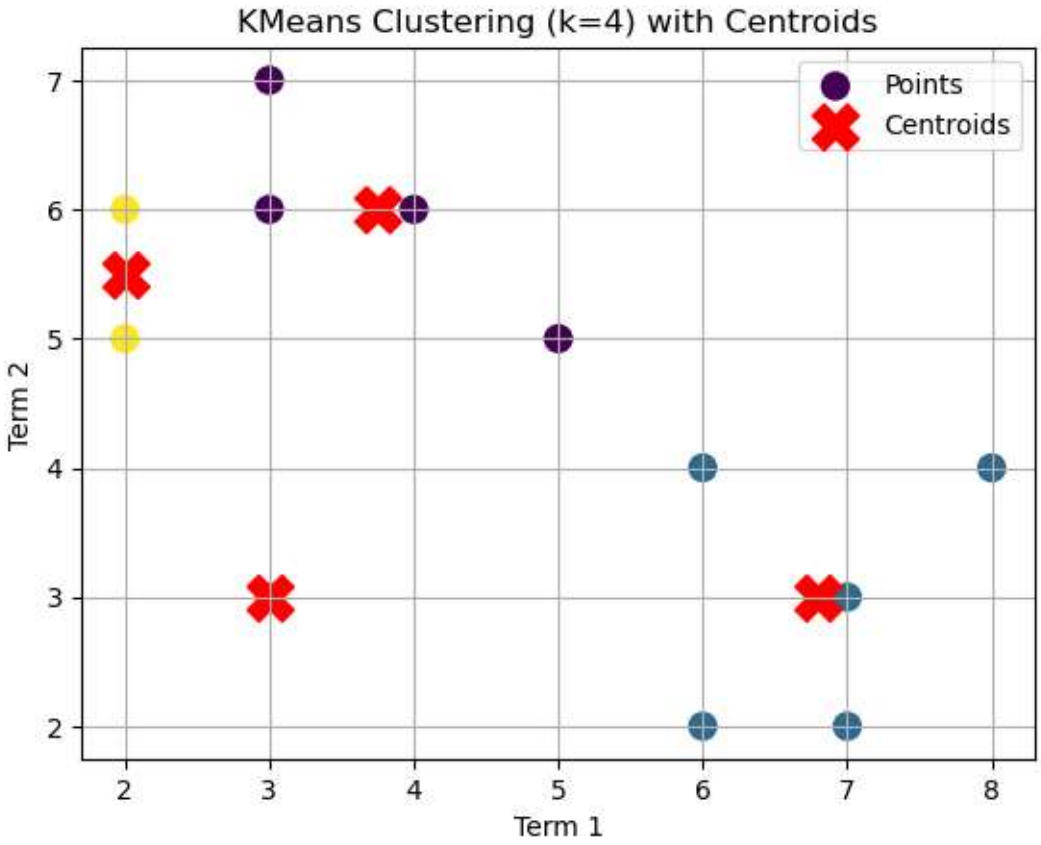
Let us assume a dataset:

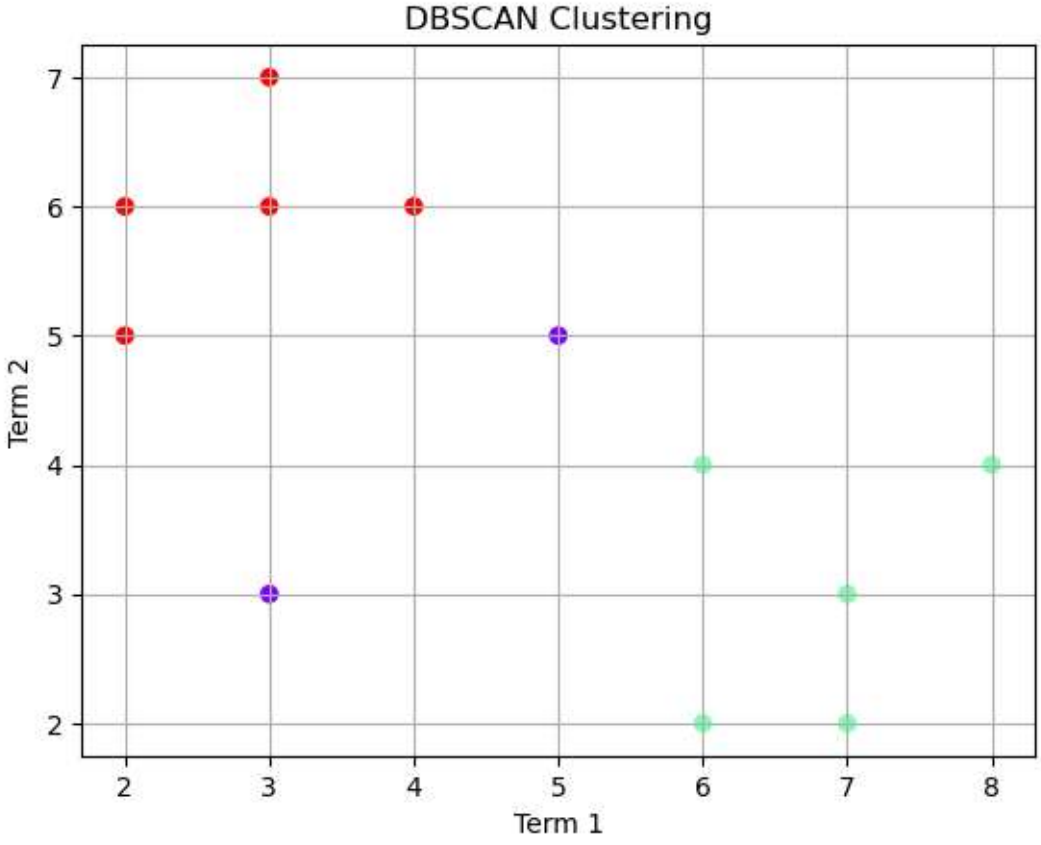
```
data2 = {
   'text1': [3,5,7,7,3,3,4,6,6,8,2,2],
   'text2': [7,5,3,2,3,6,6,4,2,4,6,5]
}
```



This dataset contains two variables, text1 and text2, each represented by a list of numerical values. We will use this data as an example to illustrate the calculations.

The two figures show the differences between clustering with K-means (k = 4) and DBSCAN


KMeans Clustering (k=4) with Centroids
Points
Centroids
Term 1
Term 2
2
3
4
5
6
7
8


DBSCAN Clustering
Term 1
Term 2
2
3
4
5
6
7
8

3.10. Introduction to LSA

Latent Semantic Analysis (LSA) is a technique in natural language processing that enables the detection of hidden semantic relationships between words and documents.
It is based on the idea that words appearing in similar contexts have a close meaning, even if they do not co-occur directly in the same document.

Main Objective

LSA is a statistical technique that allows to:

- Reduce the dimensionality of a text corpus,
- Identify latent semantic relationships between words that do not necessarily appear together in the documents,
- Represent texts in a semantic vector space.

Applications:

- Information Retrieval (IR)
- Semantic Indexing
- Synonym Detection
- Automatic Summarization

Advantages

- Handles synonymy well (different words, same meaning).
- Noise reduction through projection into the latent space.
- Good performance on medium-sized corpora.

Limitations

- Does not take word order into account (no grammar or syntactic context).
- Sensitive to the choice of the number of latent dimensions.
- SVD is computationally expensive for very large corpora.

Steps of LSA:

1. Construction of a term-document matrix
    - Generally, each row corresponds to a word in the vocabulary,
    - Each column represents a document,
    - Values are often weighted by TF-IDF.

2. Dimensionality reduction via SVD (Singular Value Decomposition)
    - The singular value decomposition allows projecting the matrix into a lower-dimensional latent space,
    - This extracts latent concepts by combining the co-occurrence information of terms.

Mathematical Foundations

Diagonalization of a Matrix (Eigenvalues, Eigenvectors)

The matrix **I** (square matrix)

$$I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$A\vec{v} = \vec{v}\lambda \qquad \vec{v} \neq 0, \text{An eigenvector;otherwise } we\ have\ 0$$

eigenvector

- **Keeps its direction** (collinear)
- **Only its length changes** by a factor of **λ**
- Therefore, **v** is stretched or shrunk
- It is **intrinsic to this transformation** (like an axis of the matrix)
- **λ** is the **eigenvalue**

$$A\vec{v} = \vec{v}\lambda \Rightarrow A\vec{v} - \vec{v}\lambda = 0 \Rightarrow (A - \lambda)\vec{v} = 0 \qquad \lambda I\,\vec{v} = \overrightarrow{\lambda v}$$

$$(\boldsymbol{A - \lambda I})\vec{v} = 0 \quad , \vec{v} \neq 0$$

The **eigenvalues** $\lambda$of a matrix $A$are the scalars for which there exists a non-zero vector $\vec{v}$such that $A\vec{v} = \lambda\vec{v}$ .This means that applying the matrix to a vector **does not change its direction** (it remains collinear), but only its length. Thus, this vector is an **eigenvector** and $\lambda$is its corresponding **eigenvalue**

$(\boldsymbol{A - \lambda I}) \equiv$ Assume it is non-invertible $\equiv \boldsymbol{det(A - \lambda I) = 0}$

otherwise $\vec{v}$ =0 characteristic equation

Key Points to Remember

Subtracting λI allows us to find when the transformation A acts like a simple stretching:

$$A\vec{v} = \lambda\vec{v}$$

for certain vectors.

1. Since λ is real, it is **collinear** with $\vec{v}$)so $\vec{v}$ is an **eigenvector**).
   λ is the **eigenvalue**.
2. Even if the resulting vector points in the opposite direction, as long as it remains on the same line, it is still an eigenvector.
3. For textual contexts, we might translate this idea as **different "vectors" representing different texts**, but the underlying principle remains the same.

Example

$$A = \begin{bmatrix} 2 & 3 \\ 2 & 1 \end{bmatrix} \qquad I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$A - \lambda I = \begin{bmatrix} 2 & 3 \\ 2 & 1 \end{bmatrix} - \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix} = \begin{bmatrix} 2-\lambda & 3-0 \\ 2-0 & 1-\lambda \end{bmatrix}$$

Compute $\det\left(\begin{bmatrix} 2-\lambda & 3 \\ 2 & 1-\lambda \end{bmatrix}\right)$

$$(2-\lambda)(1-\lambda) - 3.2 = (2-\lambda)(1-\lambda) - 6 = \lambda^2 - 3\lambda - 6$$

Solve the Equation $\lambda^2 - 3\lambda - 6$ Use the **discriminant** to obtain two solutions (eigenvalues).

$$\lambda_1 = -1 \quad et \quad \lambda_2 = 4 \text{ (eigenvectors)}$$

Calculate the eigenvectors

$\lambda_1 = -1$ Solve $A\,v = \lambda\,v$

$$\begin{bmatrix} 2 & 3 \\ 2 & 1 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} = -1\begin{bmatrix} x \\ y \end{bmatrix} = -\begin{bmatrix} x \\ y \end{bmatrix}$$

$$\begin{bmatrix} 2x+3y \\ 2x+y \end{bmatrix} = \begin{bmatrix} -x \\ -y \end{bmatrix}$$

Solve the corresponding **linear system**

$$\begin{cases} 2x+3y = -x \\ 2x+y = -y \end{cases}$$

$\begin{cases} 2x+3y=-x \\ 2x+y=-y \end{cases}$ $\begin{cases} 3x+3y=0 \\ 2x+2y=0 \end{cases}$ We notice that we are working on the same

$\begin{cases} x+y=0 \ \ divide\ \ by\ 3 \\ x+y=0 \ \ divide\ by\ 2 \end{cases}$ $x+y=0$

So if we are given x=-1 then **y=1**, x=3 then y =-3 since x=-y

Let's verify this

$$\begin{bmatrix} 2 & 3 \\ 2 & 1 \end{bmatrix}\begin{bmatrix} -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2*-1+3*1 \\ 2*-1+1*1) \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \end{bmatrix} \vec{v} = \begin{pmatrix} \mathbf{1} \\ \mathbf{-1} \end{pmatrix}$$

Draw the two vectors in the coordinate system (they are collinear but not necessarily pointing in the same direction)

For $\lambda_1 = 4$ $\begin{bmatrix} 2 & 3 \\ 2 & 1 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} = 4\begin{bmatrix} x \\ y \end{bmatrix}$

$$\begin{bmatrix} 2 & 3 \\ 2 & 1 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 4x \\ 4y \end{bmatrix} = \begin{bmatrix} 2x+3y \\ 2x+y \end{bmatrix} = \begin{bmatrix} 4x \\ 4y \end{bmatrix}$$

$\begin{cases} 2x+3y=4x \\ 2x+y=4y \end{cases}$ $\begin{cases} -2x+3y=0 \\ 2x-2y=0 \end{cases}$ $-2x+3y=0 \Rightarrow y=\frac{2}{3}x$

To find the eigenvector, we will assign a value to $y$so that we obtain a natural number. Here, if we set $y=2$, then $x=3$, giving the vector

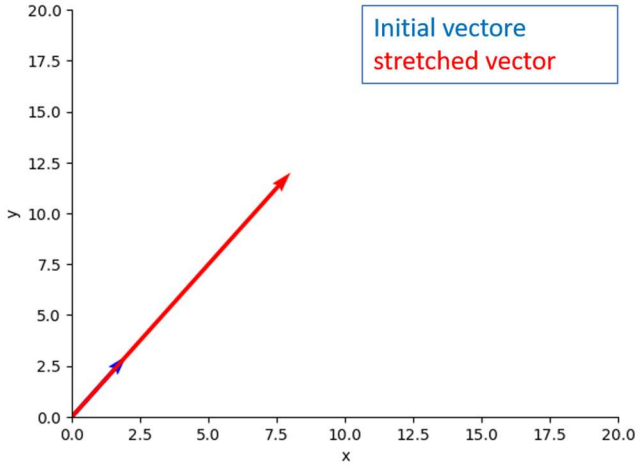
$\vec{v} = \begin{pmatrix} \mathbf{3} \\ \mathbf{2} \end{pmatrix}$ eigenvector

To verify this $\begin{bmatrix} 2 & 3 \\ 2 & 1 \end{bmatrix}\begin{bmatrix} 3 \\ 2 \end{bmatrix} = \begin{bmatrix} 4*3 \\ 4*2 \end{bmatrix} = \begin{bmatrix} 12 \\ 8 \end{bmatrix} =$

Draw on a plane to see that the second vector is 4 times the initial vector

$$\vec{v} = \begin{pmatrix} a \\ b \end{pmatrix} \Rightarrow \alpha\vec{v} = \begin{pmatrix} \alpha a \\ \alpha b \end{pmatrix}$$

If I have an eigenvector, then all values of αthat satisfy this relationship also give eigenvectors, because $\vec{v}$ and $\alpha\vec{v}$ are collinear



SVD computation

Initial matrix $A = \begin{bmatrix} 2 & 3 \\ 2 & 1 \end{bmatrix}$ and thus we are looking for $A = U\Sigma V^T$

1. Compute $A^T A$
2. Compute the eigenvalues to find Σ
3. Find the eigenvectors to determine V
4. Find U from V and Σ

SVD is used everywhere:
In **image compression** (JPEG)
In **text retrieval** (LSA)
In **machine learning** (PCA)

$$A = U\Sigma V^T$$

The columns of V are the **eigenvectors** of the matrix $A^T A$,and the **eigenvalues** of $A^T A$ are the **squares of the singular values** in Σ.
Thus, $A^T A$ allows us to determine V and Σ.

Thus, $A^T A$ allows us to compute the eigenvalues $\lambda_1, \lambda_2, \ldots$
From $A^T A$,we can extract the singular $\sigma_i = \sqrt{\lambda_i}$.
We compute the eigenvalues, then the vectors that become the columns of V

$$\boldsymbol{A^T A}=\begin{bmatrix}2 & 2\\ 3 & 1\end{bmatrix}\begin{bmatrix}2 & 3\\ 2 & 1\end{bmatrix}=\begin{bmatrix}2*2+2*2 & 2*3+2*1\\ 3*2+1*2 & 3*3+1*1\end{bmatrix}=\begin{bmatrix}8 & 8\\ 8 & 10\end{bmatrix}$$

Find the eigenvalues of $A^T A = \begin{bmatrix}8 & 8\\ 8 & 10\end{bmatrix}$

$$\det(A^T A - \lambda I) = (8-\lambda)(10-y) - 8\cdot 8 = \boldsymbol{\lambda^2 - 18\lambda + 16}$$

$$\lambda = \frac{18 \pm \sqrt{324-64}}{2} = 9 \pm \sqrt{65}\ \ then$$

$$\lambda_1 = 9+\sqrt{65}, \quad \lambda_2 = 9-\sqrt{65}$$

remember

$$A - \lambda I = \begin{bmatrix}2 & 3\\ 2 & 1\end{bmatrix} - \begin{bmatrix}\lambda & 0\\ 0 & \lambda\end{bmatrix} = \begin{bmatrix}2-\lambda & 3-0\\ 2-0 & 1-\lambda\end{bmatrix}$$

The singular values

are simply the square roots of the eigenvalues of $\lambda_1 et\ \lambda_2$

$$\sigma_1 = \sqrt{9+\sqrt{65}} \quad et \quad \sigma_2 = \sqrt{9-\sqrt{65}}$$

$$\Sigma = \begin{bmatrix}\sigma_1 & 0\\ 0 & \sigma_2\end{bmatrix}$$

Compute the eigenvectors of V

$\lambda_1 = 9 + \sqrt{65}$ $\quad \begin{bmatrix} 8 & 8 \\ 8 & 10 \end{bmatrix} \equiv V$

Other method

$$(A^T A - \lambda_1 I)\vec{v} = 0$$

$$\begin{bmatrix} 8-\lambda_1 & 8 \\ 8 & 10-\lambda_1 \end{bmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

$$\begin{cases} (8-\lambda_1)x + 8y = 0 \\ 8x + (10-\lambda_1)y = 0 \end{cases} \qquad (8-\lambda_1)x + 8y = 0 \Rightarrow x = \frac{-8}{8-\lambda_1} y$$

$\vec{v}\begin{pmatrix} 0.883 \\ 1 \end{pmatrix} \Rightarrow$ it must be normalized

$$\|\vec{v}_1\| = \sqrt{0.883^2 + 1^2} \approx \sqrt{0.78 + 1} = \sqrt{1.78} \approx 1.334$$

$$\vec{v}_1^{\text{norm}} = \frac{1}{1.334} \begin{bmatrix} 0.883 \\ 1 \end{bmatrix} \approx \begin{bmatrix} 0.662 \\ 0.75 \end{bmatrix}$$

$$\overrightarrow{v_1}\begin{pmatrix} 0.662 \\ 0.75 \end{pmatrix} \Rightarrow$$

- The columns of V (and also those of U) must be unit vectors (norm = 1)
- and orthogonal to each other (perpendicular).

$\lambda_2 = 9 - \sqrt{65}$ $\qquad \overrightarrow{v_2}\begin{pmatrix} -0.75 \\ 0.662 \end{pmatrix} \Rightarrow$

$$V = \begin{bmatrix} 0.662 & -0.75 \\ 0.75 & 0.662 \end{bmatrix}$$

Compute U(the columns of the Umatrix) from $A\,\overrightarrow{v} = \sigma_i \overrightarrow{u_i}$

$\overrightarrow{v_1}\begin{pmatrix}0.662\\0.75\end{pmatrix} \Rightarrow$

$\overrightarrow{v_2}\begin{pmatrix}-0.75\\0.662\end{pmatrix} \Rightarrow$

$A = \begin{bmatrix}2 & 3\\2 & 1\end{bmatrix}$

$$U_i = \frac{1}{\sigma_i} A\vec{v}_i$$

$A\vec{v}_1 = \begin{bmatrix}2 & 3\\2 & 1\end{bmatrix}\begin{pmatrix}-0.75\\0.662\end{pmatrix} = \begin{pmatrix}2.824\\4.324\end{pmatrix}$

$U_i = \frac{1}{\sqrt{17,06}}\begin{pmatrix}\mathbf{2.824}\\\mathbf{4.324}\end{pmatrix} = \begin{pmatrix}\mathbf{0.684}\\\mathbf{1.047}\end{pmatrix}$ using normalization $\begin{pmatrix}\mathbf{0.684}\\\mathbf{1.047}\end{pmatrix} = \|\overrightarrow{u_1}\| \approx$

$\sqrt{(0,684)^2 + (1.047)^2} = 1.26\ \bar{a} > \frac{1}{1,26}\begin{pmatrix}\mathbf{0.684}\\\mathbf{1.047}\end{pmatrix} = \begin{pmatrix}\mathbf{0.543}\\\mathbf{0.831}\end{pmatrix} = U_1$

$U_2 = \begin{pmatrix}-0.151\\0.99\end{pmatrix}$

Et donc $U \approx \begin{bmatrix}0.543 & -0.151\\0.831 & 0.99\end{bmatrix}$

$$A = \begin{bmatrix}0.543 & -0.151\\0.831 & 0.99\end{bmatrix}\begin{bmatrix}4.13 & 0\\0 & 0.97\end{bmatrix}\begin{bmatrix}0.662 & -0.75\\0.75 & 0.662\end{bmatrix}$$

Note: If we have rounded the values, we do not necessarily obtain the exact same A, but it remains powerful.

3.11. Example: Apply LSA to text mining

"the cat drinks milk",

"the cat, friend of the human",

"human gives milk to the cat"

Step 1 :

Preprocess the texts: convert all to lowercase, remove punctuation, No lemmatization/stemming

ID	Cleaned text
c1	cat drink milk
c2	cat friend human
c3	human give milk cat

[friend, drink, cat, give, human, milk]

The DTM (Document-Term Matrix)

Term	c1	c2	c3
friend	0	1	0
drink	1	0	0
cat	1	1	1
human	0	1	1
milk	1	0	1
give	0	0	1

[friend, drink, cat, give, human, milk]

Apply TF-IDF:
frequency of a word in a document
rarity of a word across all documents
standard method

df(cat)=3 IDF (cat)= $\log(\frac{5}{3}) = 0.2218$

df(drink)=1 IDF (drink)= $\log(\frac{5}{1}) = 0.6989$

IDF(drink) > IDF(cat)

Weight the importance of words in a document corpus

$$IDF(t) = \log\left(\frac{N}{df(t)}\right)$$

N: number of documents, $df(t)$: number of documents containing the term t

TF-IDF of our texts

Terme	C1	C2	C3
friend	0.000	0.477	0.000
drink	0.477	0.000	0.000
cat	0.000	0.000	0.000
give	0.000	0.000	0.477
human	0.000	0.176	0.176
milk	0.176	0.000	0.176

Term	c1	c2	c3
friend	0	1	0
drink	1	0	0
cat	1	1	1
human	0	1	1
milk	1	0	1
give	0	0	1

If we apply scikit-learn, we will get a different TF-IDF calculation.

```
TF-IDF matrix with fixed vocabulary
   cat  drink      milk    friend     human  give
0  1.0    0.0  1.405465  0.000000  0.000000   0.0
1  1.0    0.0  0.000000  2.098612  1.405465   0.0
2  1.0    0.0  1.405465  0.000000  1.405465   0.0
```

Calculation of $A^T A$, a square matrix (3×3)

Term	friend	Drink	Cat	Give	Human	milk
C1	0.000	0.477	0.000	0.000	0.000	0.176
C2	0.477	0.000	0.000	0.000	0.176	0
C3	0.000	0.000	0.000	0.477	0.176	0.176

A

documents × terms

Term	C1	C2	C3
Friend	0.000	0.477	0.000
Drink	0.477	0.000	0.000
Cat	0.000	0.000	0.000
Give	0.000	0.000	0.477
Human	0.000	0.176	0.176
milk	0.176	0.000	0.176

A^T

(terms × documents)

(1,1) = $c_1 \cdot c_1$

$0^2 + 0.477^2 + 0 + 0 + 0 + 0.176^2 = 0.2275 + 0.0310 \approx 0.2585$

(1,2) = $c_1 \cdot c_2$

$0 \times 0.477 + 0.477 \times 0 + 0 + 0 + 0 \times 0.176 + 0.176 \times 0 = 0$

(1,3) = $c_1 \cdot c_3$

$0 \times 0 + 0.477 \times 0 + 0 + 0 \times 0.477 + 0 \times 0.176 + 0.176 \times 0.176 \approx 0.0310$

(2,2) = $c_2 \cdot c_2$

$0.477^2 + 0 + 0 + 0 + 0.176^2 + \downarrow = 0.2275 + 0.0310 \approx 0.2585$

$A^T A$ Used to find:
Eigenvalues and eigenvectors ➜ to obtain V and Σ

$$A^T A = \begin{bmatrix} 0.2585 & 0 & 0.0310 \\ 0 & 0.2585 & 0.0310 \\ 0.0310 & 0.0310 & 0.2895 \end{bmatrix}$$

(terms × terms)

MATRIX A: documents × terms

	Friend	Drink	Cat	Give	Human	milk
C1	0.000000	2.098612	1.0	0.000000	0.000000	1.405465
C2	2.098612	0.000000	1.0	0.000000	1.405465	0.000000
C3	0.000000	0.000000	1.0	2.098612	1.405465	1.405465

Matrixe A^T

(terms × documents

	C1	C2	C3
ami	0.000000	2.098612	0.000000
boit	2.098612	0.000000	0.000000
chat	1.000000	1.000000	1.000000
donne	0.000000	0.000000	2.098612
humain	0.000000	1.405465	1.405465
lait	1.405465	0.000000	1.405465

$A^T * A$ = matrice symétrique

	Friend	Drink	Cat	Give	Human	millk
ami	4.404174	0.000000	2.098612	0.000000	2.949526	0.000000
boit	0.000000	4.404174	2.098612	0.000000	0.000000	2.949526
chat	2.098612	2.098612	3.000000	2.098612	2.810930	2.810930
donne	0.000000	0.000000	2.098612	4.404174	2.949526	2.949526
humain	2.949526	0.000000	2.810930	2.949526	3.950664	1.975332
lait	0.000000	2.949526	2.810930	2.949526	1.975332	3.950664

Each cell shows you the correlation between two words with respect to the documents.

(friend, human) = 0*0+2.0986*1.4055+0*1.4055= 2.9495 (without rounding), but in Python, there are rounding effects, hence a slight difference.

(friend, friend) = 0*0+2.0986*2.0986+0*0 = 4.40412

The columns of V are the **eigenvectors** of the matrix $A^T A$,and the **eigenvalues** of $A^T A$ are the **squares of the singular values** in Σ.
Thus, $A^T A$ allows us to determine V and Σ.

Thus, $A^T A$ allows us to compute the eigenvalues $\lambda_1, \lambda_2, \ldots$
From $A^T A$,we can extract the singular $\sigma_i = \sqrt{\lambda_i}$.
We compute the eigenvalues, then the vectors that become the columns of V

	Friend	Drink	Cat	Give	Human	milk
Friend	4.404174	0.000000	2.098612	0.000000	2.949526	0.000000
Drink	0.000000	4.404174	2.098612	0.000000	0.000000	2.949526
Cat	2.098612	2.098612	3.000000	2.098612	2.810930	2.810930
Give	0.000000	0.000000	2.098612	4.404174	2.949526	2.949526
Human	2.949526	0.000000	2.810930	2.949526	3.950664	1.975332
milk	0.000000	2.949526	2.810930	2.949526	1.975332	3.950664

$$(\boldsymbol{A - \lambda I})\vec{v} = 0 \quad \equiv \boldsymbol{det(A - \lambda I) = 0}$$

$$\vec{v} = \begin{pmatrix} a \\ b \end{pmatrix} \Rightarrow \alpha\vec{v} = \begin{pmatrix} \alpha a \\ \alpha b \end{pmatrix} \qquad U_i = \frac{1}{\sigma_i} A\vec{v}_i$$

$$A = U\Sigma V^T$$

	Friend	Drink	Cat	Give	Human	milk
C1	0.000000	2.098612	1.0	0.000000	0.000000	1.405465
C2	2.098612	0.000000	1.0	0.000000	1.405465	0.000000
C3	0.000000	0.000000	1.0	2.098612	1.405465	1.405465

Matrixe	content
U	Document expressed by concepts
Σ	Strength (weight) of each concept
V^T	Words expressed by concepts

```
Reconstructed A :
           friend         drink  cat          give         human          milk
C1 -2.715306e-15  2.098612e+00  1.0  3.780284e-15  9.691286e-16  1.405465e+00
C2  2.098612e+00  2.435169e-15  1.0 -3.598235e-15  1.405465e+00 -3.603957e-16
C3 -3.719364e-17 -5.671516e-16  1.0  2.098612e+00  1.405465e+00  1.405465e+00

Reconstruction error :
 7.833174222381017e-15
```

- U is a matrix (documents × concepts).
- Each row of U represents a document (C1, C2, C3).

Graphical interpretation

	Friend	Drink	Cat	Give	Human	milk
C1	0.000000	2.098612	1.0	0.000000	0.000000	1.405465
C2	2.098612	0.000000	1.0	0.000000	1.405465	0.000000
C3	0.000000	0.000000	1.0	2.098612	1.405465	1.405465

- Link between words and documents,
- each row is a document (e.g., C1, C2, C3),
- each column is a word (e.g., 'cat', 'milk', 'human', etc.),
- each cell is a TF-IDF weight.

- Each row of U represents a document (C1, C2, C3),
- Each column represents a 'latent concept' automatically detected by SVD.
- If $U_{2,1}$ is large, it means that document C2 is strongly related to the first concept found by SVD.

- Each diagonal value (called σi) is a 'weight' or 'importance' of concept number i.

Object	Meaning
Document	Actual text you are analyzing (e.g., C1, C2, C3)
word	Vocabulary keyword (e.g., cat, human, milk)
Concept	Idea / abstract theme constructed from co-occurrences between words and documents

Documents words expressed by concepts $\Sigma \times V^T$

	Friend	Drink	Cat	Give	Human	milk
Concept 1	-0.9875	-0.9875	-1.6866	-1.5671	-1.7095	-1.7095
Concept 2	-1.4835	1.4835	0.0	0.0	-0.9938	0.9938
Concept 3	-1.1089	-1.1089	-0.3908	1.3943	0.1928	0.1928

•"The cat drinks milk"
•"The cat, friend of the human"
•"Human gives milk to the cat"

This means that concept 1 is largely about cat, human, milk, friend, drinks, gives, etc.

```
Sigma =
 [[3.6198 0.     0.    ]
 [0.     2.5258 0.    ]
 [0.     0.     2.152 ]]
```

σ1 = 3.6198 is the largest value: it is therefore the main concept.

Vector of a single word for k=2

- A word is represented by a vector in the concept space.
- It comes from the V^T matrix after projection byΣ
- Mathematically, the vector of a word = Σ_k V_i where:
  - V_i is the column corresponding to the word in V^T,
  - Σ_k is the column corresponding to the word in

```
Sigma =
 [[3.6198 0.     0.    ]
 [0.     2.5258 0.    ]
 [0.     0.     2.152 ]]
```

```
V^T =
 [[-0.2727 -0.2727 -0.4662 -0.4329 -0.4725 -0.4725]
 [-0.5875  0.5875 -0.      0.     -0.3935  0.3935]
 [-0.5149 -0.5149 -0.1816  0.6486  0.0896  0.0896]]
```

Exemple concret : "chat"

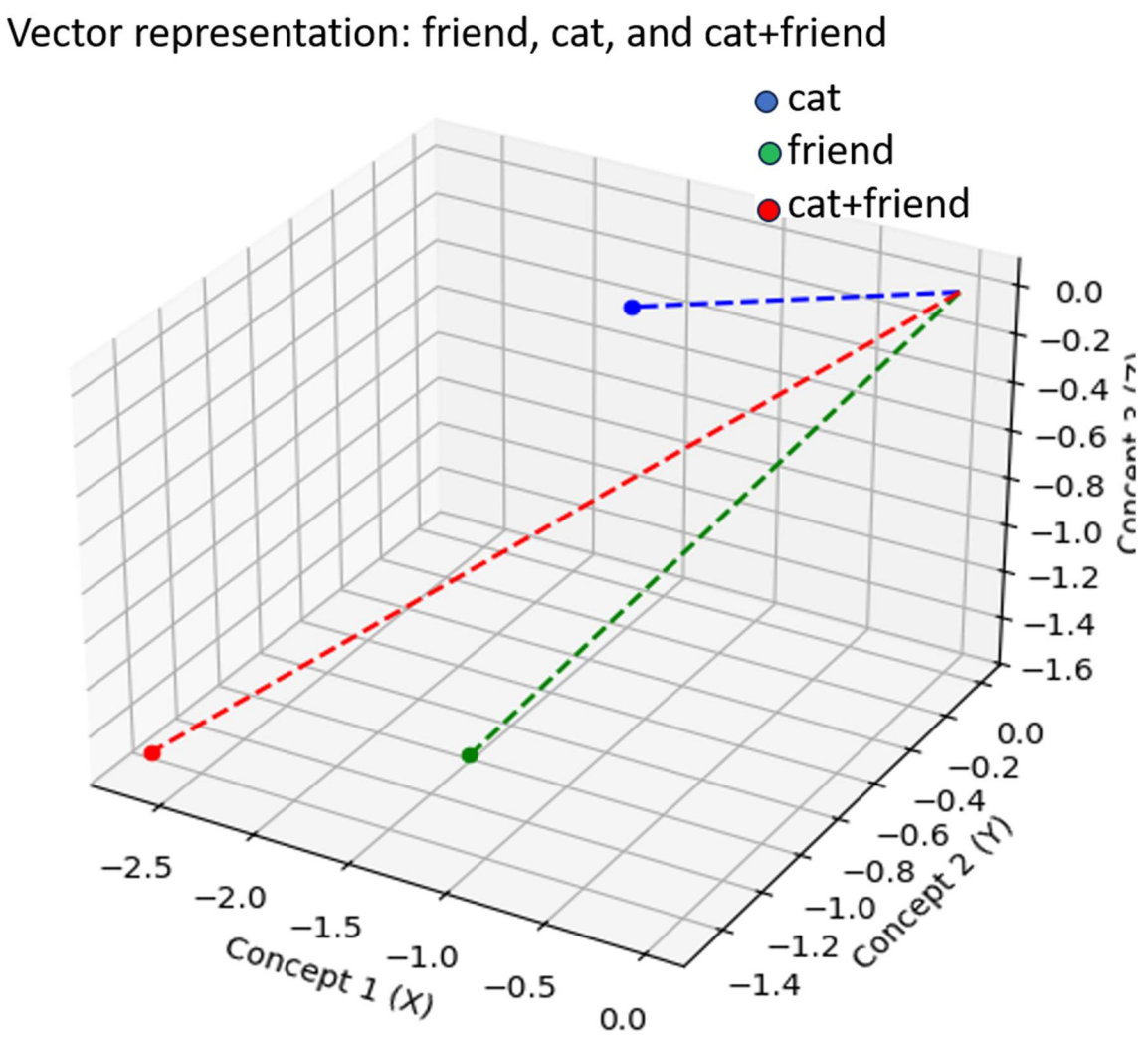
he "cat" column in V^T is: cat= $\begin{bmatrix} -0,4662 \\ 0.0 \\ -0.1816 \end{bmatrix}$ For k=2, we keep only the first two components: cat= $\begin{bmatrix} -0,4662 \\ 0.0 \end{bmatrix}$

Σ_k V_i

$$\text{cat (projected)} = \Sigma_k \times cat_k = \begin{bmatrix} 3.6198 & 0 \\ 0 & 2,5258 \end{bmatrix} * \begin{bmatrix} -0,4662 \\ 0.0 \end{bmatrix}$$

$$= \begin{bmatrix} 3.6198 * (-0.4662) + 0 * 0 \\ 0 * 0 + 2.5258 * 0 \end{bmatrix} = \begin{bmatrix} -1.687 \\ 0.0 \end{bmatrix}$$

The word 'cat' is represented by the vector(-1.687, 0)



3.12. Discover the latent meaning

- ❑ In a text corpus, words and documents share implicit meanings.
- ❑ However, a term-document matrix (or TF-IDF) is **sparse** (contains many zeros), heavy, and very sensitive to synonyms or variations.
- ❑ For example, two documents may talk about "human" and "science," or "animal," yet have no words in common.

However, they are talking about related things, aren't they?

SVD in LSA

- Projection of documents and words into a lower-dimensional latent vector space (e.g., 2 or 3),
- where semantic proximities are captured,
- even if the words do not co-occur directly.

Singular Value Decomposition (SVD)

$$A = U\Sigma V^T$$

A: TF-IDF matrix (words × documents)
U: Words represented in the latent space
Σ: Singular values (importance of the concepts)
V^T: Documents in the latent space

- Hidden semantic links,
- Clusters of similar words or documents,
- A better measure of similarity than raw co-occurrence

Réduire l'espace

By keeping only the k largest values of Σ (e.g., 2 or 3), we obtain an approximation

$$A = U_k \Sigma_k V_k^T$$

3.13. syntactic similarity or semantic similarity

But the question that always remains is: should we use syntactic similarity or semantic similarity?
Text similarity measures have been studied since the early 1970s. However, it is undoubtedly clear that lexical similarity is not at all the same as semantic similarity. For example, based on the bag-of-words principle, the sentence *"Le patient est souffrant"* (The patient is suffering) has no link with the sentence *"Le malade est hospitalisé"* (The patient is hospitalized). Although these two sentences are semantically similar, it is almost certain that existing lexical measures cannot detect this link.

Thus, the need to automatically extract information and perform semantic analysis became a very active research area during the MUC (Message Understanding Conference) in 1987. The goal here is to extract logical or semantic relationships between words in a text. Moreover, Named Entity Recognition (NER) can, among other things, determine the semantic context and extract textual information. For example, the term *Dr* should designate a person (a human) whose profession is associated with medicine.

Statistical, probabilistic, and ontological approaches

The use of rules, statistical, probabilistic, and ontological approaches can help discover these entities.
To ensure a semantic representation and faithfully reproduce the meaning of texts, linguistic methods use natural language processing (NLP) techniques. These methods also rely on external resources such as dictionaries, ontologies, etc., with the aim of obtaining domain knowledge.

This knowledge may include the context in which a text is written, the intended purpose, the actors, and the geographical location of the authors and the studied topic. Furthermore, it is also about absorbing knowledge to achieve reasoning similar to human reasoning.

Thus, it is useful to detect relationships (associations) between concepts/terms, detect internal contextual relations within texts to build the meaning of statements, count the number of documents containing common terms, and so on.

POS tagging

The use of POS tagging, which is a grammatical annotation of words based on the context of their occurrence, allows words to be categorized according to their role in a sentence. These methods do not reduce the dimensionality of the entire function space but simply remove the "unimportant" components of individual document vectors. Since the complexity of computing similarity between documents is proportional to the number of non-zero components in document vectors, such truncation is efficient. In practice, the document vectors themselves are already quite sparse, and only the centroids, which can be very dense, need to be truncated.

The alternative approach is global dimensionality reduction. Its disadvantage is that it does not adapt to the unique characteristics of each document. Its advantage is that this method better preserves the ability to compare different documents, as each document undergoes the same transformation. One increasingly popular dimensionality reduction technique is Latent Semantic Analysis (LSA). Its applications are varied, such as information retrieval, automated text summarization, text categorization and classification, and discovering relationships between texts. Indeed, studying word occurrence measures is one of the tasks of LSA.

n-grams vs LSA

Unlike n-grams, LSA does not take into account the order of term appearance. It focuses solely on extracting contextual relationships from the usage of words in a document. This allows for establishing a kind of semantic relationship between terms and associating keywords (concepts) with documents.

We will cover these topics in a second document that follows this one. However, I will address n-grams below.

The objective of n-grams is to find contexts (meaning) by associating terms. In both French and English, the composition of words can have a different meaning if considered individually. Non-exhaustive examples include:

- *watching machine* (washing machine)
- *take off* (if *off* is treated as a stop word, the sense of "takeoff" is lost in French)
- *human being* (loses meaning if *being* is lemmatized to *is*)

The n-gram principle consists of finding consequences of consecutive terms. Thus, a bi-gram associates two terms, a tri-gram associates three terms, and so on.

Bibliography

1. Marcus, M. P., Marcinkiewicz, M. A., & Santorini, B. (1993). Annotated Corpus of English: The Penn Treebank. *Computational Linguistics, 19*(2), 313–330.
2. [Auteur inconnu]. (2013). Études sur la linguistique et les corpus. *Actes*, 271–284.
3. Tournier, J. (1996). Recherches en linguistique étrangère. Vol. XIX. Mélanges. ISBN 2-251-60643-2.
4. [Auteur inconnu]. (1998). La pragmatique et l'analyse linguistique. *Revue de Linguistique*, 2, pp. 109–111.
5. Civoeurs, J. (1978). Linguistique, langage et informatique: approches et applications. *Escande Pratiques*, 20, 109–111.
6. Lanodv, C. (1996). Algorithmic approaches in mechanical translation. *Journal of Computational Linguistics*, 11(2), 22–31.
7. [Auteur inconnu]. (1996). Evaluation of information processing algorithms. *Journal of Information Science*, 4(7), 349–362.
8. Porter, M. (1980). An algorithm for suffix stripping. *Program, 14*(3), 130–137.
9. [Auteur inconnu]. (2002). Approches de traitement automatique du langage. *Journal of French Linguistics*, July 2002.
10. Troen, B., & Landmining, H. (2006–2007). Text mining and structural analysis. Massachusetts: Academic Press. ISBN 9780511546914.
11. Salton, G., Wong, A., & Yang, C. S. (1975). A vector space model for automatic indexing. *Communications of the ACM, 18*(11), 613–620.
12. Salton, G., Buckley, C., & Fox, E. A. (1983). Term-weighting approaches in automatic text retrieval. *Information Processing & Management, 19*(5), 513–523.
13. Maron, M. E. (1961). Automatic Indexing: An Experimental Inquiry. *Journal of the Association for Computing Machinery, 8*(3), 404–417.
14. Schütze, H., Hull, D. A., & Pedersen, J. O. (1995). A comparison of classifiers for text categorization. In E. A. Fox, P. Ingwersen, & R. Fidel (Eds.), *Proceedings of the ACM SIGIR Conference* (pp. 229–237). Seattle: ACM Press.
15. Wiener, E. D., Pedersen, J. O., & Weigend, A. S. (1995). A neural network approach to topic spotting. In *Proceedings of SDAIR-95, 4th Annual Symposium on Document Analysis and Information Retrieval* (pp. 317–332). Las Vegas: University of Nevada.
16. Forgy, E. W. (1965). Cluster analysis of multivariate data: Efficiency vs. interpretability of classifications. *Biometrics, 21*, 768–769.
17. Hartigan, J. A., & Wong, M. A. (1979). Algorithm AS 136: A K-means clustering algorithm. *Applied Statistics, 28*, 100–108. https://doi.org/10.2307/2346830
18. Lloyd, S. P. (1957, 1982). Least squares quantization in PCM. *Bell Laboratories Technical Note*. Published in IEEE Transactions on Information Theory, 28, 128–137.

19. MacQueen, J. (1967). Some methods for classification and analysis of multivariate observations. In L. M. Le Cam & J. Neyman (Eds.), *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Vol. 1* (pp. 281–297). Berkeley, CA: University of California Press.
20. Goldberg, Y. (2016). A primer on neural network models for natural language processing. *Journal of Artificial Intelligence Research, 57*(1), 345–420.
21. T. *M. Mitchell. Machine Learning. WCB/McGraw-Hill, 1997.*
22. *P. Larranaga, D. Atienza, J. Diaz-Rozo, A. Ogbechie, C. E. Puerto-Santana, and C. Bielza. Industrial Applications of Machine Learning. Chapman & Hall/CRC Data Mining and Knowledge Discovery Series. CRC Press, Boca Raton, FL, 2018.*

External Links

https://www.ibm.com/think/topics/text-mining

https://www.lexalytics.com/blog/context-analysis-nlp/

https://doi.org/10.1016/j.csl.2024.101638

https://opensemanticsearch.org/doc/analytics/textmining/

Improving Named Entity Recognition by External Context Retrieving and Cooperative Learning