Bachelor's Degree Final Project

Machine Learning Techniques for Natural Language Processing

Faculty of Mathematics

Statistics and Operational Research Department

Presented by:

Amparo Jareño García

Supervised by:

Dr. Rafael Pino Mejías

Dual Degree Mathematics and Statistics
Academic year 2022-23
Seville, July 2023

Abstract

Language is such a powerful tool that enables communication between humans. However, there have been some barriers, historically, which arose the need for automatisation of linguistic tasks. After overcoming setbacks, a balance between linguistics and computer science was achieved, which was based mainly on artificial intelligence. The objective of this work is to understand the text preprocessing so that machine learning algorithms can be applied to draw insights. For this purpose, we will dive into supervised learning theory together with discriminative and generative algorithms that will be perform text classification tasks: a fake news classifier and a film reviews sentiment analysis. Furthermore, we will explore the variety of purposes and the industrial applications that Natural Language Processing has.

Resumen

El lenguaje es una herramienta tan poderosa que permite la comunicación entre los seres humanos. Sin embargo, históricamente han existido algunas barreras que fomentaron la necesidad de automatizar las tareas lingüísticas. Tras superar los contratiempos, se logró un equilibrio entre la lingüística y la informática, el cual se fundamentó principalmente en la inteligencia artificial. El objetivo de este trabajo es comprender el preprocesamiento de textos para poder aplicar algoritmos de aprendizaje automático que permitan extraer conclusiones. Para ello, nos sumergiremos en la teoría del aprendizaje supervisado junto con algoritmos discriminativos y generativos que realizarán tareas de clasificación de textos: un clasificador de noticias falsas y un análisis de sentimientos de críticas de películas. Además, exploraremos la variedad de propósitos y de aplicaciones industriales que tiene el Procesamiento del Lenguaje Natural.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Text Basics

This chapter explains the basic concepts of NLP together with a deep understanding of its history. This would allow us to better understand which are the setbacks that NLP has overcome during the history and how NLP has been evolving as a result. By understanding the historical context and evolution of NLP, we gain valuable insights into the challenges, breakthroughs and paradigm shifts that have shaped this dynamic field. We witness how NLP has evolved from rule-based systems to data-driven approaches, unlocking new opportunities for understanding, analysing and generating human language.

1.1 Introduction

This section introduces the fields of computational linguistics and natural language processing, which are closely related, but each still retains their differences in motivation and purpose. Both combine linguistics and artificial intelligence.

Computational Linguistics (CL) is an interdisciplinary field that combines the study of linguistics with the techniques of computer science and artificial intelligence to study the structure, meaning, and use of human language. Its main objective is to understand language modelling by humans, including written and spoken natural language. In other words, it focuses on the system or concept that machines can compute to understand, learn or output languages. This field has evolved from being considered within computer science until it has become part of applied science such as psychology, neuroscience or philosophy. Since the rise of social media, Computational Linguistics is playing a vital role in modelling human language.

Natural Language Processing (NLP) refers to the branch of artificial intelligence focused on giving computers the ability to understand text and speech in the same way humans can. NLP integrate computational linguistics (rule-based modelling of human language) with statistical, machine learning and deep learning models to analyse and extract information.

1.2 History

The history of Natural Language Processing is a story full of twists and turns. It started with non-productive research, progressed through years of fruitful work, and finally ended at a time when we are still trying to determine the limits of this field.

The area of Natural Language Processing began in the 1940s, since during World War II, people became aware of the need for machine translation. Therefore, researchers began to train computers to emulate natural language understanding by applying linguistic rules. This is the main reason why this era is known as **Symbolic** or **rule-based NLP** as it was focused on generating syntax with the help of linguists and computer scientists. One of these researchers, Noam Chomsky, published his influential work *Syntactic Structures* [2] in 1957. He argued for the independence of syntax (the study of sentence structure) from semantics (the study of meaning), since sentences that were nonsense but grammatically correct were classified as improbable to the same extent as sentences that were nonsense and not grammatically correct. As a result, Chomsky created the Phase-Structure Grammar which is a type of generative grammar where a set of ordered rules known as rewrite rules are applied stepwise. At the same time, Alan Turing published the seminal paper *Computing Machinery and Intelligence* [3], which introduced the Turing Test or also known as the Imitation Game whose main objective was testing machine's ability to exhibit intelligent behaviour equivalent to a human. In addition to this, John McCarthy organised the Dartmouth Conference in 1956, where he created the term *artificial intelligence* and started brainstorming about this immense field.

Due to the development of linguistic syntactic theory and parsing algorithms, the 1950s were filled with outrageous enthusiasm. It was widely believed that high-quality fully automated translation systems would be able to produce results that are indistinguishable from those of translators. However, given the language skills and computer systems available at the time, this thought was completely unrealistic. These are the reasons why there was a huge investment in NLP, which started to decrease sharply as a result of unfruitful research.

In the late 1980s, there was a revolution in NLP with the introduction of machine learning algorithms, which led to the **statistical era of NLP**. This was due to both the increase in computing power and the gradual decline in Chomskyan language theories. Although some of the earliest used machine learning algorithms produced systems with performance similar to the old-fashioned handwritten rules, statistical models overcame the complexity barrier of hand-coded rules by creating them through automatic learning. The main reason is that it required the use of statistical inference to automatically learn such rules through the analysis of large numbers of typical real-world examples. For this, the latter can be made more accurate simply by providing more input data. On the other hand, the others can only be made more accurate by increasing the complexity of the rules, which is a much more laborious task.

Regarding the success of statistical models in NLP, it mainly occurred in the field of machine translation, primarily through work at IBM Research [4]. Moreover, with the growth of the World Wide Web, increasing amounts of raw language data became available, which made research focused on unsupervised learning algorithms and semi-supervised learning algorithms. Such algorithms can learn from data that has not been hand-annotated with the desired answers or by using a combination of annotated and non-annotated data. In general, this task is much more difficult than supervised learning, and usually produces less accurate results for certain input data.

During the 2000s, **neural networks** acquired importance in machine learning models and then, in 2010, neural network-like representation and machine learning methods became widespread in NLP, in part due to some results showing that such techniques can achieve state-of-the-art results on many tests such as language modelling and parsing. Furthermore, three types of neural networks appeared: recurrent neural networks, convolutional neural networks and recursive neural networks. These completely revolutionised the NLP world. At the same time, the most popular word embedding model, *Word2Vec*, was introduced, which refers to the representation of a word as a real-valued vector to be processed by neural networks.

In addition to this, reinforcement learning has also an impact in NLP, which is a subfield of deep learning that focuses on training agents to make sequential decisions in an environment to maximize a reward signal. It is particularly well-suited for tasks with temporal dependencies and non-differentiable optimisation zones, where traditional gradient-based methods may not be effective.

Undeniably, NLP together with AI is under constant evolution and it is used in a wide range of applications, which will be discussed in section (4). With the proliferation of social media and online communication, NLP has become increasingly important for understanding, translating and processing large volumes of text data. Recent advances have enabled NLP to perform complex tasks and natural language generation with high accuracy. Scientists want to develop algorithms that can understand the meaning and intent of sentences with as few words as possible. They intend to capture the meaning and intent of sentences and develop a set of algorithms that can extract information from them. This is the reason why there is still no limit on what we want to achieve with NLP as long as it supports human activities in everyday life. They say that the development of NLP has greatly helped people on a daily basics. Nevertheless, there are some dangers behind the development of NLP, but there are also many opportunities. Surely, we can expect to see in the future even more sophisticated applications and human-like dialogue systems, which will completely revolutionise the way we live.

Chapter 2

Language Normalisation

It is crucial to understand the basic concept of *natural language*, which refers to any language that has evolved naturally in humans through use and repetition without conscious planning or forethought. Natural languages can take different forms, such as speech, written or signing. Estimates of the number of human languages in the world vary between 5,000 and 7,000, depending on distinction between languages and dialects, respectively. Despite the variety of vocabulary in our mother language, we only use at most 10% of words in a day-to-day setting. This ambiguity makes it inherently hard for computers to process and understand natural language. This difficulty increases when sentiments influence human language with sarcasm, irony, metaphors and humour. Moreover, social networks are changing human language with the introduction of emojis and new terminology resulting from current trends.

To analyse a natural language, it has to be distinguished the different linguistic characteristics for text and speech analysis. These categories found in table 2.1 are useful to provide a rich set of representations for machine learning algorithms.

Text analysis	**Morphology**	Word structure
	Lexical	Segmenting words
	Syntax	Sentence structure
	Semantics	Linguistic meaning
	Discourse	Meaning among sentences
	Pragmatics	Meaning through speaker intent
Speech analysis	**Acoustics**	Representations of sound
	Phonetics	Speech sounds
	Phonemics	Sound patterns and changes
	Prosodics	Intonation, stress, and rhythm

Table 2.1: Summary of language analysis categories. Source: [1]

Natural language processing is the mapping of language to representations which capture the characteristics of morphology, dictionaries, syntax, semantics or discourse. The choice of the representation can have a major impact on subsequent tasks and will depend on the selected machine learning algorithms.

In the following sections of this chapter, we will dive into these representations to better understand their role in linguistics and purpose in natural language processing. In this work, we will focus on the most common text-based representations and the preprocess methodology to follow.

2.1 Morphological Analysis

Within the discipline of linguistics, morphology is the study of the structure and formation of words. This refers to the rules and conventions used to shape words depending on their contexts, such as plural, gender, contraction and conjugation.

Words are composed of subcomponents called **morphemes**, which are the smallest unit of meaning in a language. Morphemes can be classified into free or bound morphemes:

- **Free morpheme**: a morpheme which is a word by themself, such as “run”, “dog” or “book”. They can also appear within lexemes such as “bookstore”, which is the compound of two free morphemes: “book” and “store”.
- **Bound morpheme**: a morpheme that cannot stand alone as a word but must be attached to a free morpheme to create a word. Here, it is important to make a distinction between two classes: inflectional and derivational.
    - **Inflectional bound morphemes**: they add information about grammatical number, gender or verbal tense. Some examples are “dogs”, “prince” to “princess” or “booked”.
    - **Derivational bound morphemes**: create new words by adding **affixes**, which are prefixes when added to the front of a word and suffixes when added to the back. Another less common types are infixes, when added in the middle of a word, and circumfixes, which consists of two parts, one attached to the beginning and the other to the end of the word. They directly change the meaning of words as well as the part of speech. For instance, adding the suffix *-er* to the root *run* creates the word “runner”, which has been tranformmed from a verb to a noun.

Despite the diversity of possible morpheme combinations, English has simple morphologic rules in comparison to other languages like Arabic [6]. For this reason, computers have to consider morphological analysis, whose main normalisation approaches are **stemming** and **lemmatisation**. Both truncating techniques aim to simplify text processing by reducing words to their root or base form and are widely used in different applications such as information retrieval, machine translation, text classification and tagging systems. These and even more applications will be discussed in section (4).

2.1.1 Stemming

Researcher J.B. Lovins defines stemming in his publication *Development of a Stemming Algorithm* [7] as:

> "A **stemming algorithm** is a computational procedure which reduces all words with the same root (or, if prefixes are left untouched, the same **stem**) to a common form, usually by stripping each word of its derivational and inflectional suffixes."

Stemming can be useful for standardising vocabulary since the stem and their inflected words have related meaning, which allows to build relationships between them. However, due to the cut of the end beginning of a word, this process can result in stems that are not real words. In addition to this, stemming can introduce ambiguity, as it is shown in the following example:

$$\begin{aligned}\text{photographs} &\rightarrow \text{photograph}\\ \text{photographed} &\rightarrow \text{photograph}\\ \text{photographers} &\rightarrow \text{photograph}\end{aligned}$$

Although humans can easily identify differences in meaning, it is still difficult for computers to do so through stemming. It is relevant to notice that words with different meanings have the same stem: "photographers" and "photographs" (people versus items). On the contrary, this shows that stemming is robust to spelling errors, as the correct root may still be inferred correctly.

From 1960s, several stemming algorithms, which are known as **stemmers**, have been developed. The most popular algorithm is the **Porter stemmer**, which was published in 1980 by Martin F. Porter, in the paper *An algorithm for suffix stripping* [8]. It consists of five steps to remove suffixes, within each step, rules are tested until a condition is satisfied. For this particular rule, the suffix is removed, and the algorithm goes to the next step. The rules have the following structure:

$$\text{(condition) suffix} \quad \longrightarrow \quad \text{new suffix}$$

For instance, the rule (2.1.1) means that the words whose stem contains a vowel should be removed the *ing* suffix. So, "motoring" becomes "motor", while "sing" remains unchanged. The algorithm consists of 60 rules approximately, which can be found in [9].

$$\text{(*v*) ING} \quad \longrightarrow \tag{2.1.1}$$

A few years later, Martin Porter implemented some corrections in a new algorithm known as **Snowball Stemmer**, as a tribute to *SNOBOL*[1]. While Porter

[1]SNOBOL ("StriNg Oriented and symBOlic Language") is a programming language developed in the 1960s for text and string manipulation. It has built-in support for pattern-matching and search capabilities, and is designed to handle complex pattern matching and text processing tasks that are difficult or impossible to express in other programming languages.

Stemmer can only be used in English, Snowball Stemmer supports 13 languages, incluiding French, German and Spanish. Snowball is characterised to be more aggressive but also more accurate than Porter stemmer. Another relevant algorithm is **Lancaster stemmer**, which was developed by C.D. Paice at Lancaster University in 1990. This is an iterative algorithm containing 120 rules, which are only applicable to English. Compared with the previous stemmers, Lancaster is the most aggressive.

Python:
With the aim of implementing these stemmer algorithms in Python, NLTK (Natural Language Toolkit) is used. This is a Python library used for natural language processing into we will dive in the subsection (5.1). Therefore, the first step is to download and import this library. To use stemmers, import the corresponding algorithm from the module `nltk.stem`. For instance, Martin Porter algorithm corresponds with the `PorterStemmer` class, from which we have to create an instance. Then, use the `stem()` method for each word by passing it through a `for` loop to evaluate different examples. An analogous implementation can be made for the rest of stemmers.

Word	**Porter**	**Snowball**	**Lancaster**
program	program	program	program
programming	program	program	program
programer	program	program	program
programs	program	program	program
programmed	program	program	program
people	peopl	peopl	peopl
procedure	procedur	procedur	proc
successful	success	success	success
photographers	photograph	photograph	photograph
photographs	photograph	photograph	photograph
protographed	protograph	protograph	protograph
unacceptable	unaccept	unaccept	unacceiv
fairly	fairli	fair	fair
sportingly	sportingli	sport	sport
universal	univers	univers	univers
university	univers	univers	univers
data	data	data	dat
datum	datum	datum	dat

Table 2.2: Porter, Snowball and Lancaster algorithms results for a list of words. Source: Own elaboration (see section (7)).

As it is displayed in table 2.2, the stemmers were able to correctly reduce each word to its base form, but some are not perfect and can sometimes result in stem words that are not actual words or might be too aggressive in their stemming, as it is the case of Lancaster algorithm.

Regarding incorrect stemmisation, there are two types of errors:

- **Over-stemming**: is when two words with different stems are stemmed to the same root. Statistically, this definition matches what is known as a **false positive**. In the table 2.2, “universal” and “university” are a clear over-stemming example. This would imply that both words have a similar meaning, which is not correct.

- **Under-stemming**: is when two words that should be stemmed to the same root are not. This corresponds to a **false negative**. If we consider the stemming of “data” and “datum” shown in table 2.2, we can see that only the Lancaster algorithm does in the correct way since both words have the same stem and meaning. However, this occurs mainly because Lancaster is the most straightforward stemmer.

On general, stemmers which reduce over-stemming errors, increase under-stemming, and viceversa. A good stemmer algorithm would be one that finds the balance between these two extremes.

2.1.2 Lemmatisation

In the chapter *Lemmatisation as a Tagging Task* [13], lemmatisation is defined as the following:

> “**Lemmatisation** is the task of grouping together word forms that belong to the same inflectional morphological paradigm and assigning to each paradigm its corresponding canonical form called **lemma**.”

Lemma refers to a word form that exists and has meaning by itself, being found in the dictionary. For this purpose, lemmatisation relies more on morphological analysis of words, using a sophisticated approach that takes into account the part of speech and the word context in the sentence. This requires the existence of dictionaries for each language to provide this type of analysis. However, such dictionaries are created manually and cannot be developed quickly in many languages.

To understand the differences with stemming, we can analyse the previous example:

$$\text{photographs} \rightarrow \text{photograph}$$
$$\text{photographed} \rightarrow \text{photograph}$$
$$\text{photographers} \rightarrow \text{photographer}$$

Now, we can see that the meaning of each word is considered and remains. The results are similar to those of stemming, but they are actually words. In contrast to stemming, meaning and context are considered. On the other hand, lemmatisation is sensitive to spelling errors, which requires a preprocessing task for spell correction and it is slower because of looking up in a dictionary.

Python:
We consider the implementation in Python with NLTK package, from which we download **WordNet** (see section (2.1.2.1)) lemmatiser. Firstly, we create an instance of the lemmatiser and use its `lemmatize()` method to lemmatise a word. For this, we have to indicate in the argument `pos` the grammatical category, or also known as part of speech (POS) tags of each word. These grammatical categories are `v` for verbs, `a` for adjectives, `r` for adverbs and `n` for nouns, which is the default option. With this information, the words are lemmatised accordingly and the result are actual words. The results are shown in table 2.3, which can be compared with the stemming methods in in table 2.2.

Word	**POS**	**Lemmatisation**
program	n	program
programming	v	program
programer	n	programer
programs	v	program
programmed	v	program
people	n	people
procedure	n	procedure
successful	a	successful
photographers	n	photographer
photographs	n	photograph
photographed	v	photograph
unacceptable	a	unacceptable
fairly	r	fairly
sportingly	r	sportingly
universal	a	universal
university	n	university
data	n	data
datum	n	datum

Table 2.3: Lemmatisation results indicating the corresponding POS tag. Source: Own elaboration (see section (7)).

Words are reduced to their canonical form, lemma, which has meaning by itself. The main problem is that words that cannot be found in WordNet will remain unchanged and that the POS tag has to be included in this method to obtain better results.

To summarise, here you can find a table 2.4, which compares the benefits and drawbacks between stemming and lemmatisation.

	Advantages	**Disadvantages**
Stemming	Model performance	Overstemming
	Grouping similar words	Understemming
	Easier to analyse and understand	Language difficulties
	Robust to spelling errors	Not actual words
Lemmatisation	Accuracy	Time-consuming
	Actual words	Sensitive to spelling errors

Table 2.4: Advantages and disadvantages of stemming and lemmatisation. Source: Own elaboration.

2.1.2.1 WordNet

Since there are words that can have multiple different lemmas, lemmatisation algorithms or **lemmatisers** need the part of speech tag in that specific context. For this purpose, **WordNet** [15] is used as an English dictionary for lemmatisation. WordNet was released in 1986 at Princeton University by Christiane Fellbaum. It consists of a large semantic network or *graph* in which words are interconnected by means of labelled arcs that represent meaning relations. This database is organised in **synsets**, which each refers to a set of words which are semantically equivalent (or, also known as synonyms). This can be seen as a many-to-one mapping of word forms and concepts. Synsets can be seen as the nodes of WordNet and edges as meaning-based relations. Nowadays, there are 117000 synsets, each one corresponding to one category: nouns, verbs, adjectives and adverbs. Regarding another semantic representation present in table 2.5, polysemy can be captured when the same word appears in different synsets, each with their own synonyms, since this would imply that it has distinct meanings. In contrast, antonyms are linked by an "antonym" pointer between words.

Synonymy	Words having the same meaning
Antonymy	Words having the opposite meaning
Polysemy	Word with two or more distinct meanings
Hypernymy	Generic word whose meaning includes other words (hyponyms)
Hyponymy	Specific word whose meaning is included in a broad category (hypernym)

Table 2.5: Semantic relations between words. Source: [1]

Considering that WordNet is a graph, there is an important semantic relation which can be represented through directed edges: hypernymy and hyponymy. In this way, we link specific concepts to more general ones, but it could also be seen the other way around, as a bi-directional relation, from a global category to a particular case. However, with the aim of representing it as a directed graph, we will consider that each directed edge $v \rightarrow w$ means that w is a hypernym of v, or equivalently, v is a hyponym of w. For example, the synset gym shoe, sneaker, tennis shoe is a hyponym of shoe, which in turn is a hyponym of footwear, footgear, etc. And gym shoe, sneaker, tennis shoe is a hypernym of plimsoll, which denotes a specific type of sneaker ([15] pp. 233). This is how this hierarchical relation is constructed,

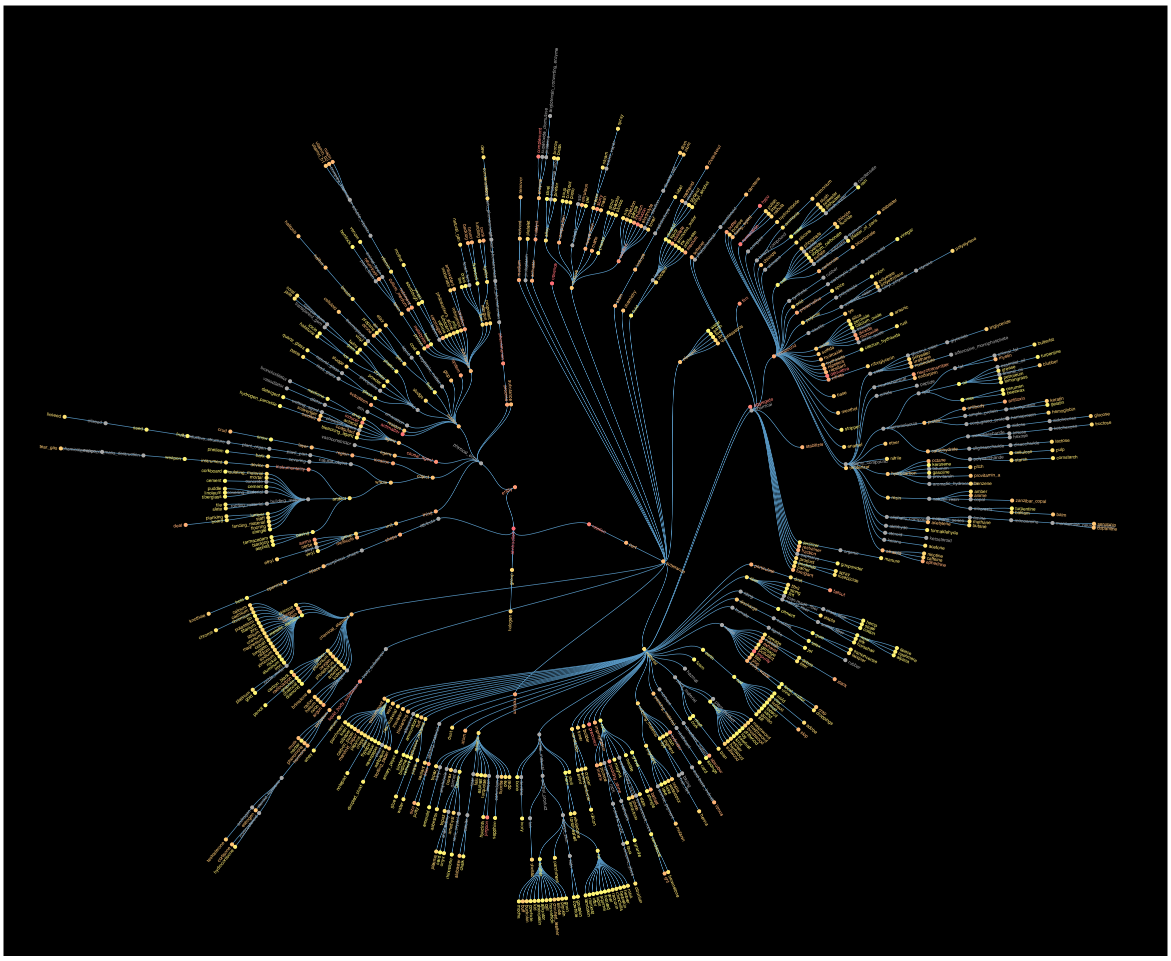


Figure 2.1: The WordNet example representation as a rooted DAG. Source: [16]

where scaling through all noun synsets, we can reach to a common node. This root corresponds to the concept *entity*, which encompassess all things that exist in the world, incluiding *physical entity*, *abstract entity* and *thing.* For these reasons, WordNet can be seen as a directed acyclic graph (DAG), though not a tree since each synset can have several hypernyms. In figure 2.2, there is a graphical example of the *is-a* relationship, which joins a hyponym to a hypernym [17]. As before, for this particular case, a birdcage is a hyponym to cage, and cage is a hyponym to enclosure.

These mathematical concepts are explained below, and are useful because it provide researchers with DAG properties and tools, which can be very useful to implement in NLP.

Definition 2.1.1. *A* ***directed graph*** *G is a pair G=(V, E) comprising:*

- *V: a set of vertices (**nodes**).*
- *E: a set of directed edges, which are ordered pairs of distinct vertices.*
  $E \subseteq \{(x, y) \mid (x, y) \in V^2, x \neq y\}$

Definition 2.1.2. *A* ***cycle*** *in a graph G = (V, E) is a sequence of distinct vertices* $v_1, v_2, \ldots, v_k \in V$ *such that* $v_i v_{i+1} \in E(G) \quad \forall i = 1, \ldots, k-1$ *and* $v_1 v_k \in E(G)$.

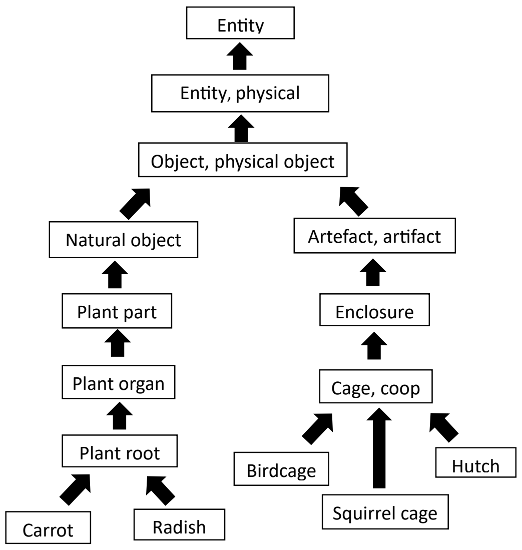


Figure 2.2: The “is a” relationship. Source: [17]

Definition 2.1.3. *A* ***directed acyclic graph (DAG)*** *is a directed graph, with no directed cycles.*

Definition 2.1.4. *A vertex v is an* ***ancestor*** *of w in graph G if $v = w$ or there is a directed path in G from w to v.*

Definition 2.1.5. *A* ***root*** *in a graph is a vertex which is an ancestor of every other vertex.*

Regarding the NLP problem of establishing similarity relationships between words, WordNet DAG plays a vital role since it allows to find the shortest path which connects two words. For this, new concepts are introduced:

Definition 2.1.6. *An* ***ancestral path*** *between two vertices v and w in a digraph (directed graph) is a directed path from v to a common ancestor x, together with a directed path from w to the same ancestor x.*

Definition 2.1.7. *A* ***shortest ancestral path*** *between two vertices v and w is an ancestral path of minimum total length. In other words, it is the shortest path between these two vertices that leads to their closest common ancestor.*

Note that the length of the shortest path between v and w is the combination of the lengths from both v and w to their nearest common ancestor. In relation to NLP, if u is an ancestor of v, this means that u is a hypernym of v, or the other way around, v is an hyponym of u.

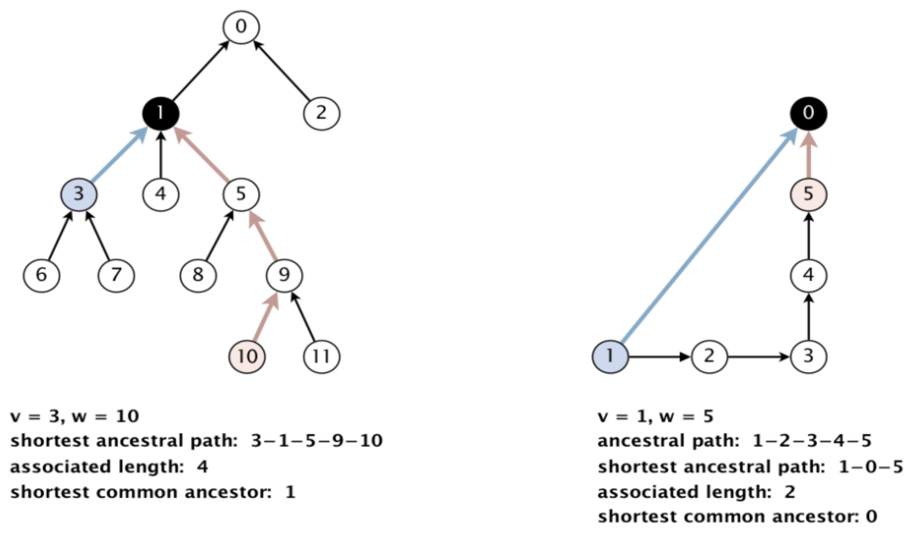

v = 3, w = 10
shortest ancestral path: 3–1–5–9–10
associated length: 4
shortest common ancestor: 1
v = 1, w = 5
ancestral path: 1–2–3–4–5
shortest ancestral path: 1–0–5
associated length: 2
shortest common ancestor: 0

Figure 2.3: Example of shortest ancestral path. Source: [18]

Python:

The NLTK package provides a wide range of functions and algorithms [19] developed to dive in WordNet:

- `synsets()`: it is a method that returns a list of synsets for a given word, which has an optional `pos` argument which lets you constrain the part of speech of the word. If `pos` is not provided, all synsets for all parts of speech will be returned. Another optional parameter is `lang`, which determines the language of the synsets and by default is set to English. Once the method is called, the output will be a list of Synset objects, where each represents a collection of synonymous words that are semantically related. These Synset objects provide different methods to access and retrieve information:
  - `definition()`: provides a brief explanation of the meaning of the concept represented by the synset.
  - `examples()`: returns a list of example sentences that illustrate the use of the words in the synset.
  - `lemmas()`: returns a list of Lemma objects that represent the different words in the synset. A Lemma object contains the spelling of the word, the part of speech and the synset ID.
  - `hypernyms()`: returns the synsets that represent more general concepts that include the concept represented by the synset.
  - `hyponyms()`: returns the synsets that represent more specific concepts that are included in the concept represented by the synset.
- `path_similarity()`: determines the similarity between two synsets by examining the length of the shortest path that connects them in the WordNet DAG.

- `common_hypernyms()`: determines the synsets that are shared by two synsets, which represent concepts that are more general and higher up in the WordNet hierarchy.

2.2 Lexical Representations

Lexical representations in NLP refer to the way words and phrases are represented in computational systems, which can be useful for further processing. For this, lexical analysis consists of identifying the structure of text to segment it into its lexical expressions. The segmentation can be done by words, sentences or even documents. This is the first step to preprocess or normalise a text.

2.2.1 Tokenisation

Tokenisation is the process of breaking down a text into smaller units of meaning, which are called **tokens**. Tokens are usually words, but they can also be sentences, documents or even individual characters such as numbers or punctuation marks. The main reason why tokenisation is vital is that it splits the text into a list from which we can use statistical tools to draw insights.

The tokenisation techniques rely on the specific language to be analysed since each language has unique challenges. Therefore, depending on the language, a particular technique will be used.

The most common approach is achieved by splitting text through whitespace. Although whitespace-based tokenisation may work correctly for most cases, it is not ideal if we do not want to split open compound words. Below, there is an example for better understanding:

A token is a sequence of characters that are grouped together.
`Tokens:` |A|, |token|, |is|, |a|, |sequence|, |of|, |characters|, |that|, |are|, |grouped|, |together|, |.|

Problems

Los Angeles and New York are located in the United States.
`Tokens:` |Los|, |Angeles|, |and|, |New|, |York|, |are|, |located|, |in|, |the|, |United|, |States|, |.|

Notice that Los Angeles, New York and the United States refer to specific locations and each one should have been considered as a single token. Therefore, whitespace-based tokenisation does not perform well in cases where tokens must consist of multiple words which are separated by whitespace.

On the other hand, there are some languages that do not use whitespace such as Chinese. Alternatively, character tokenisation can be used to cope with this challenge, altough there are other advanced techniques that we are not going to look into. In this work, we are focused on English tokenisation.

In reference to sentence tokenisation, punctuation plays a crucial role since it delineates the end of a sentence and beginning of another one. Unfourtunately, this still retains ambiguity as punctuation marks are also used for other purposes such as abbreviations and decimal numbers.

Problems

Mr. Henry drinks 2.5 litres of water per day.
`Sentence tokens:` |Mr.|, |Henry drinks 2.|, |5 litres of water per day.|

Sentence tokenisation algorithms should not be simple punctuation-based.

Python:

Tokenisation can be implemented using regular expressions, although more complex algorithms are required to overcome the problems mentioned above. This is the reason why `nltk.tokenize` module is frequently used since it provides a wide range of functions and classes for tokenising text. It contains the following methods:

- `word_tokenize()`: tokenisation method by words which returns a list of tokens (words).
- `sent_tokenize()`: tokenisation method by sentences which returns a list of tokens, in this case, sentences.

Regarding regular expressions and the basis of tokenistation, the `findall()` module from the `re` library allow us to search for all occurrences of a pattern in a given string and return a list of all matched substrings. For word tokenisation purpose, we implement the following regular expression:

- `[]`: a set of characters.
- `\w`: returns a match where the string contains any word characters (characters from A to Z, digits from 0 to 9, and the underscorer _ character).
- `+`: quantifier of one or more occurrences.

A simple-based sentence tokenisation method can also be easily implemented although it is not going to perform well due to other punctuation marks uses. For this, we can use the `compile()` module to compile the regular expression pattern into a regular expression object, which is then used to split the input text using the `split()` method. This method splits the input text wherever the regular expression matches, and returns a list of the resulting substrings. The regular expression we compile is the one which matches any punctuation marks followed by a space character: `‘[.!?] ’`.

2.2.2 Stop Words

Assuming that word tokenisation is performed, recall that tokens would be basically words. These words are not distributed uniformly in the text, where there are important differences in their frequency use. **Stop words** refer to the words which are commonly used and they do not provide any useful meaning for text processing. Typical examples of stop words include determiners, coordinating conjunctions and prepositions. All of them are the basis for articulating English language, which leads to the overuse compared to the actual words that add the relevant differentiating meaning. This is the reason why stop words removal is part of text normalisation. The list of words excluded is known as a **stop word list**.

This phenomenon is known as **Zipf's law**. Zipf's law is a statistical distribution in which the rank-frequency distribution is inversely related. It has several applications in biology species, cities, and of course, linguistics, where this relationship was originally noticed.

To illustrate, we are going to use the *Brown Corpus*[2], which was the first million-word electronic corpus of English, created in 1961 at Brown University. This corpus contains 500 text samples about 15 genres. NLTK has this categorised corpus available as a list of words or a list of sentences, where each sentence is shown as a list of words.

The main objective is to obtain the frequency of each word so that we could test the Zipf's law hyphotesis for this given corpus. To meet the goal, it is vital to pre-process the corpus, which includes lowercasing, removing punctuation and word-tokenisation. This can be easily done in Python using a list comprehension, which is a syntactic construction to create a list from an existing list based on some conditions. It shares some similarities with the mathematical set-buider notation. Mathematically, let w be a word, W be the set of Brown corpus words and P be the set of punctuation marks:

$$\{\text{lowercase}(w) \mid w \in W, w \notin P\}$$

In Python, we have to mimic the structure, but we have to adapt it to the programming language.

```
punctuations = list(string.punctuation)
tokens = [word.lower() for word in brown.words()
if not all(char in punctuations for char in word)]
```

The next step is to calculate the frequency of each word. `FreqDist` tells us the frequency of each item in the text. In general, it could count any kind of observable event. It is a distribution because it tells us how the total number of tokens in the text are distributed across the vocabulary items. The output is a dictionary whose values related to the frequency are sorted in decrease order. `Most_common()` is a method whose argument is the number of the most common tokens we want to extract. Applying this methodology some Python functions are defined in section (7) to show the results.

[2]A **corpus**, or **corpora** in plural, is a collection of structured texts, which are often used to do statistical analysis and hypothesis validations.

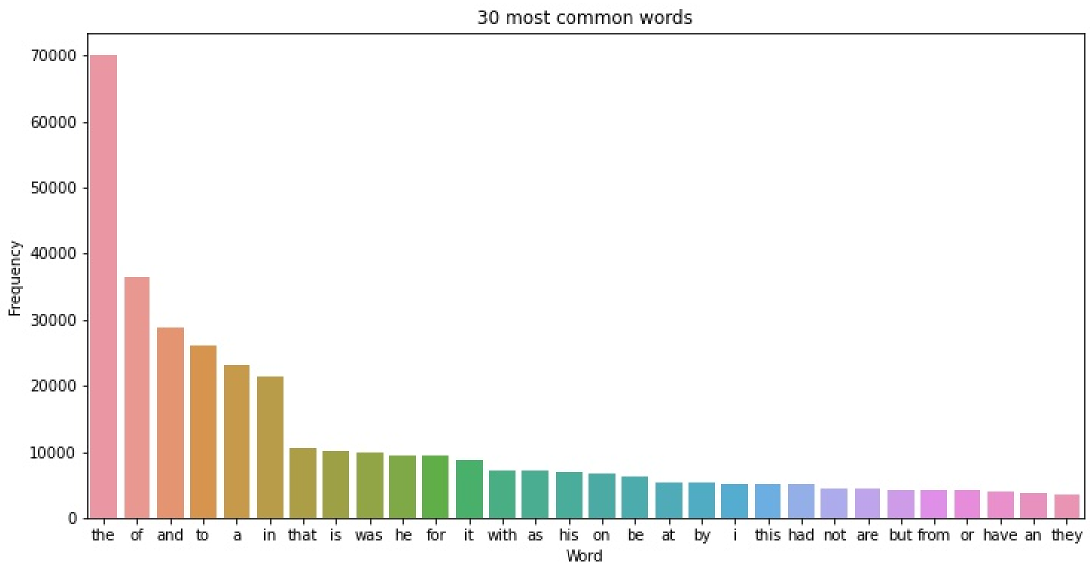


Figure 2.4: Barplot of the 30 most frequent words in the Brown Corpus. Source: Own elaboration (section (7)).

In this type of distribution, frequency declines sharply as the rank number increases. Therefore, a small number of items appear very often, and a large number rarely occur. If we focused on which are these words, we notice that the majority of them are stop words. As previously mentioned, normalisation includes removal of stop words. As a result of this process, dimensionality is strictly reduced. In fact, calculating the cumulative frequency distribution of each word and their corresponding percentage, we can see that only 133 tokens, mainly stop words, are needed to account for half the Brown corpus number of words.

	Frequency	**% of total words**	**Rank**
the	69971	6.905131	1
of	36412	3.593340	2
and	28853	2.847376	3

Table 2.6: Three most frequent words in Brown Corpus. Source: Own elaboration (section (7)).

Mathematically, the frequency of the nth-word, f_n, follows a distribution which could be approximated by

$$f_n \sim \frac{1}{n^a} \tag{2.2.1}$$

where a is a real positive number, generally slightly greater than 1.

According to Zipf's law, the most common word occurs twice as often as the second most frequent word, three times as often as the third most frequent word and so on until the least frequent one. This can be easily verified in the Brown Corpus. Out of slighly over one million total words, the word "the" represents almost

7% of them with 69971 occurences. The half of this value approximately corresponds with the frequency of the second ranked word "of", which accounts for 3.5% of words, followed by "and", with nearly 3% of representation in the corpus.

To sum up, the word in the position n appears $\frac{1}{n}$ times as often as the most frequent one. This relationship can be visually tested using a log scale for both axis: if Zipf's law is satisfied, the result is a straight line.

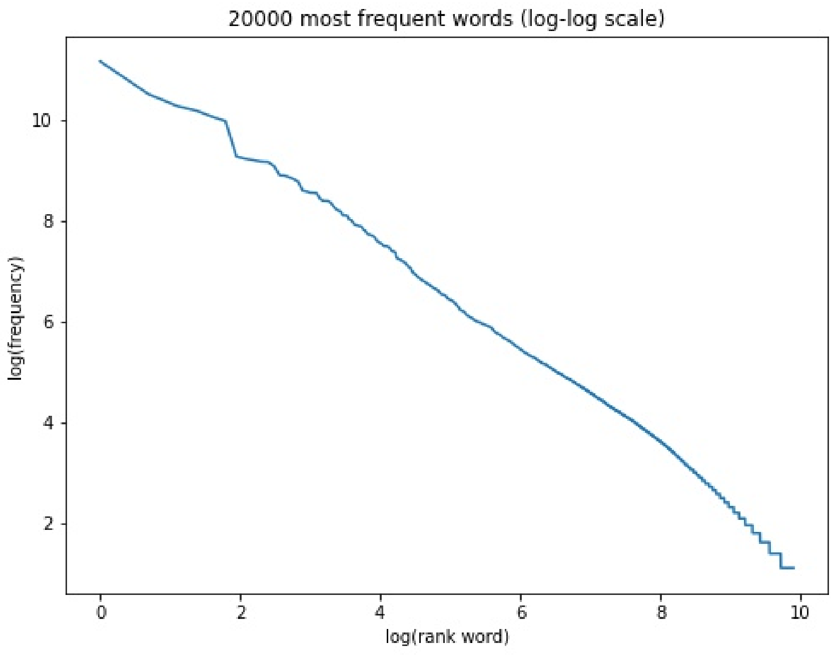


Figure 2.5: Log-log scale plot of the 20000 most frequent words in the Brown Corpus, regarding their rank. Source: Own elaboration.

In conclusion, the analysis of the *Brown Corpus* has provided evidence to follow Zipf's law. On the whole, the line could be considered approximately as a straight line, despite existing some irregularities regarding the least frequent words as well as those words whose log rank is around 2.

2.2.3 Documents representation

In natural language processing, various document representarion methods are used when dealing with a corpus of documents. These methods can be token-based, multitoken-based or character-based, and can be either sparse or dense representations. The motivation for this representations is that providing raw text data to a machine is not enough. It is necessary to convert the text into a numerical format that can be easily understood by the machine. This mathematical representation of a set of documents is known as a **document-term matrix**.

A document-term matrix is a matrix that describes the frequency of terms that occur in a collection of documents. In this case, rows correspond to documents and columns correspond to terms. However, it is also common to use the transpose, which is known as **term-document matrix**. The entries of the matrix can be computed with different methods, which are explained below. These matrices are often stored as a sparse matrix object, which can be treated as though they were matrices, but are stored in a more efficient format.

2.2.3.1 Bag-of-Words

The **bag-of-words** (BoW) set each entry of the document-term matrix equal to the frequency of word occurrence within each document. This process is also known as **count vectorisation**. Therefore, each document (row) can be represented by a high dimensional sparse vector, where each component corresponds to the term frequency of a unique word within a dictionary. In other words, the bag-of-words is basically the absolute frequency f_{ij} of word j in the document i.

As mentioned in section (2.2.2), the most frequent words tend to be the less relevant regarding semantics. To solve this issue, it is enough to remove stop words in a preprocessing step. This also allows to reduce the dimension of the matrix and save memory, but still the matrix would contain many 0s (sparse matrix), which can be computationally challenging. This is why lowercasing, stemming or lemmatisation are also carried out before constructing the BoW to reduce the dimension of the vocabulary.

Another drawback is that it only takes into account the spelling. Therefore, a word appearing with different meanings (polysemic) in the same document will not be distinguised.

Alternative method:
Since absolute frequencies can be influenced by the length of each document, normalisation could also be applied using relative frequencies. These are some possible implementations:

- Respect the maximum frequency per document, denotaded by $\max\limits_{j\in J} f_{ij}$: $\dfrac{f_{ij}}{\max_{j\in J} f_{ij}}$.
- Respect the total number of occurrences per document: $\dfrac{f_{ij}}{\sum_{w\in W} f_{iw}}$.

2.2.3.2 TF-IDF

Term Frequency (TF) - Inverse Document Frequency (IDF) is a numerical statistic whose aim is to reflect the importance a word has in a document within a corpus. It is a document representation where a penalty coefficient is applied to common words in different documents. It is based on the following ideas:

1. The most frequent terms in a given document should have a higher relevance than the rest of the words. This is implemented through the **Term Frequency**

(TF) as the relative frequency explained before

$$TF_{ij} = \frac{f_{ij}}{\sum_{w \in W} f_{iw}} \tag{2.2.2}$$

where f_{ij} denotes the number of ocurrences of term j in the document i. This can also be defined using the logarithmically scaled frequency as:

$$1 + \log(TF_{ij}) \tag{2.2.3}$$

2. The terms occurring multiple times in different documents of the corpus do not help to determine the unambiguous content of a text and should therefore be penalised. To consider this, the **Inverse Document Frequency** (IDF) is calculated

$$IDF_j = \log\left(\frac{N}{n_j}\right) \tag{2.2.4}$$

where N denotes the total number of documents and n_j is the number of documents containing the term j.

Therefore, the result TF-IDF is the combination of both criteria so that the associated weights are directly proportional to the occurrences of the term in a document and at the same time, it reduces the TF-IDF value for the term based on the frequency across all documents. Combining the previous equations 2.2.2, 2.2.3 and 2.2.4, assuming a logarithmic scale, TF-IDF is obtained:

$$TF\text{-}IDF_{ij} = (1 + \log(TF_{ij})) \times IDF_j = \left(1 + \log\left(\frac{f_{ij}}{\sum_{w \in W} f_{iw}}\right)\right) \times \log\left(\frac{N}{n_j}\right) \tag{2.2.5}$$

Suppose there is a term that is present in all the documents of the corpus, that is to say, $n_j = N$. This would imply that IDF_j would be 0 since the fraction of the logarithm would be 1, and therefore, the value of $TF\text{-}IDF$ is 0 too. This is how terms which do not add useful information are completely penalised.

Currently, TF-IDF is one of the most popular weighting method, with well-known libraries using it. Some practical examples are explained in section (7) for better understanding of these methods.

2.2.4 N-Grams

Word level representations have limitations in capturing the relationship with adjacent words, hindering the ability to incorporate grammar and context. The **bag-of-words** model, based on a Markov assumption, disregards word order and focuses solely on token presence and frequency. This model predicts a phrase containing L tokens with probability [1]:

$$P(w_1, w_2 \ldots w_L) = \prod_{i=1}^{L} P(w_i) \tag{2.2.6}$$

However, there are many situations where sequences of words are relevant. For this reason, another approach known as ***n*-gram** is very useful in NLP. An n-gram is a sequence of n consecutive words or tokens in a text. In this case, the probability would be predicted as:

$$P(w_1, w_2 \ldots w_L) = \prod_{i=n}^{L} P(w_i \mid w_{i-1}\ w_{i-2}\ w_{i-n}) \tag{2.2.7}$$

Note that equation (2.2.6) is a particular case for $n = 1$ in equation (2.2.7). This is also known as unigram. Another particular case are bigrams, which considers two consecutive words. They distinguish between “lion king” and “king lion” [1]. Then, a sentence of L tokens would yield $L - 1$ bigrams:

$$P(w_1, w_2 \ldots w_L) = \prod_{i=2}^{L} P(w_i \mid w_{i-1}) \tag{2.2.8}$$

This can be useful to solve the problem of tokens, where United States was considered as two separate tokens and using a bigram we would be able to capture this relationship in a unique token.

Chapter 3

Machine Learning

Machine Learning (ML) has emerged as a powerful discipline that enables computers to learn from data and make predictions or decisions. In this chapter, we dive into the fundamentals of ML, with a primary focus on supervised learning. We will explore the statistical learning theory, the trade-off between bias and variance, model performance evaluation and model validation techniques.

Moving further, we distinguish between discriminative and generative classifiers. Discriminative classifiers focus on learning the decision boundary between different classes, while generative classifiers model the underlying distribution of each class. We provide a comprehensive explanation of logistic regression, support vector machines (SVM), decision trees, gradient descent (GD), stochastic gradient descent (SGD), boosting, gradient boosting, and XGBoost as prominent examples of discriminative classifiers. Additionally, we explore Naive Bayes and Linear Discriminant Analysis (LDA) as examples of generative classifiers.

3.1 Introduction

Learning is the process of converting *information* and *experience* into *knowledge* and *understanding*, which is measured by the ability to perform certain tasks independently [23]. Machine learning is therefore a field dedicated to building methods that let machines *learn* (learning process) so that they could carry out tasks without assistance.

Machine Learning is a subcategory of Artificial Intelligence (see figure 3.1), which refers to the simulation of human intelligence in machines, programmed to execute tasks that would typically require human intelligence. The key difference is that while AI is about creating machines that can reason and make decisions like humans, Machine Learning is about developing algorithms that enable machines to learn from data and improve their performance over time.

Mathematically, Machine Learning encompasses approximation theory, probability and optimization theory. The goal is to learn, which implies finding a function that models the outputs appropiately according to the inputs.

$$h : \mathcal{X} \to \mathcal{Y}$$

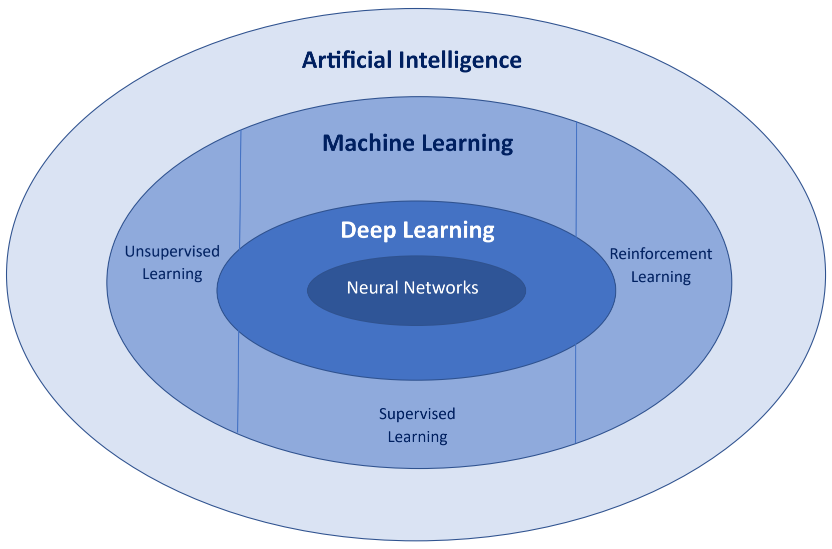


Figure 3.1: Machine Learning as a subset of Artificial Intelligence. Source: [1]

This function h maps the space of inputs or characteristics $\mathcal{X}$ to the space of outputs or responses $\mathcal{Y}$. Such a function is called a **hypothesis**, a **predictor** or a **classifier**. Another common name is **binary classifier** when dealing with problems with two categories, which can be coded as 0 and 1, or -1 and 1.

Machine Learning can be divided into the following areas, which are also shown in figure 3.1:

- **Supervised Learning**: it consists of learning from answers (labels) belonging to a collection of data $\{x^i, y^i\}_{i=1}^n \subset \mathcal{X} \times \mathcal{Y}$. This input data allows the model to adjust its weights until it has been fitted appropriately, considering a balance between overfitting and underfitting. Depending on the type of output, there are two types of supervised learning:
  - **Classification**: labels correspond to a fixed set of categories. For instance, classifying a new as either fake or true.
  - **Regression**: the output would be a real number. For instance, predicting the cost of buying a house in the future.

  Neural networks, Naïve Bayes, linear regression, logistic regression, random forest and support vector machine (SVM) are common algorithms used in these tasks.

- **Unsupervised Learning**: it determines clusters from data where there are no labels present. These algorithms discovers hidden patterns of similarities and differences. Not only is unsupervised learning used for grouping, but it is also used to reduce the number of features in a model through the process of dimensionality reduction.

Principal component analysis (PCA), singular value decomposition (SVD) and k-means clustering are methods used in unsupervised learning.

- **Semi-Supervised Learning**: it is a combination of the previous ones, that is to say, the entire set is not labelled or it is expensive to label all the data.
- **Reinforcement Learning**: it focuses on maximising a reward or penalty associated with an action or environment provided as input. The algorithms are trained to encourage certain sucessive actions or behaviours. It is common to use these techniques for games, where the reward may be winning the game and a number of actions must to be taken before.

Regarding figure 3.1, Deep Learning is a subset of Machine Learning that involves the development of neural networks with multiple layers that can learn and make decisions based on complex data inputs. In essence, it is a type of machine learning algorithm that uses artificial neural networks to simulate the function of the human brain, allowing machines to learn from large amounts of data and improve their performance over time.

3.2 Supervised Learning

As previously stated, supervised machine learning consists of learning from answers (labels). Then, some concepts are introduced to build the basic framework.

Let $\mathcal{X}$ be the population of all possible data for a particular learning problem (e.g., distinguish between fake and true news). From $\mathcal{X}$, samples can be obtained independently with an unknown probability distribution $P(\mathcal{X})$. Therefore, a sample $\boldsymbol{X}$ of n elements or also known as **instances** can be represented as $\boldsymbol{X} = \boldsymbol{x^1}, \boldsymbol{x^2}, \ldots, \boldsymbol{x^n}$. It is clear that $\boldsymbol{X} \subseteq \mathcal{X}$. The input space $\mathcal{X}$ is modelled as a metric space $\mathbb{R}^d$, in which the dimension d refers to the attributes or **features**. These features can be:

- **Categorical** or **qualitative** variables represent characteristics or attributes that can be observed and categorised but not measured numerically. They can also be distinguised between:
  - **Ordinal**: when there are a discrete categories or levels. For example, $quality = \{poor,\ fair,\ good,\ excellent\}$.
  - **Nominal**: describe categories that do not have a specific order to them. For instance, $gender = \{male,\ female\}$.
- **Numeric** or **quantitative** variables represent numerical quantities that can be measured or counted. These variables have meaningful numerical values and allow for mathematical operations such as addition, subtraction, multiplication and division. A further distinction could be made:
  - **Discrete**: takes a finite number of values. Examples of discrete variables include the number of siblings a person has, the number of cars in a parking lot and the number of customers in a store at a given time.

– **Continuous**: can be a scalar in $\mathbb{R}$ and can be measured on a continuous scale. Examples of continuous variables include length, weight, speed and income.

This is a general classification of variables, which also includes the output (labels). As mentioned in the introduction (3.1), when the output is categorial, the problem is to classify, whereas a numerical output involves a regression problem.

Therefore, each instance can be seen as a d-dimensional vector, which together with the whole sample X of n elements, a $n \times d$ matrix can be obtained. Each row would be the vector $\boldsymbol{x^i}$ corresponding to the i-th instance of the sample, and its label y_i could be stored in a vector Y.

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x^1} \\ \boldsymbol{x^2} \\ \vdots \\ \boldsymbol{x^n} \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1d} \\ x_{21} & x_{22} & \cdots & x_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nd} \end{bmatrix}, \qquad \boldsymbol{Y} = \begin{bmatrix} y^1 \\ y^2 \\ \vdots \\ y^n \end{bmatrix} \tag{3.2.1}$$

We can also represent the sample as a labelled dataset $\mathcal{D}$:

$$\mathcal{D} = \{(\boldsymbol{x^1}, y^1), (\boldsymbol{x^2}, y^2), \dots (\boldsymbol{x^n}, y^n)\} \tag{3.2.2}$$

3.2.1 Statistical Learning Theory

In addition to the probability distribution $P(\mathcal{X})$, there is another unknown element referred to as the **target function**. This function $f : \mathcal{X} \to \mathcal{Y}$ maps the input space $\mathcal{X}$ to the desired output space $\mathcal{Y}$. The goal in machine learning is to *approximate* the function f with a function h from a large hypothesis space $\mathcal{H}$, based only on the knowledge of a finite sample as $\mathcal{D}$ in (3.2.2).

The evaluation of the best fit is measured by a loss function or also known as cost function.

Definition 3.2.1. *A **loss function** $L : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$ is a function that measures the mismatch between a prediction on a given input $\boldsymbol{x} \in \mathcal{X}$ and its label $\mathbf{y} \in \mathcal{Y}$, and associate a cost.*

There are different loss functions, but we will focused on two approaches depending on the type of problem. For classification problems, an **indicator loss function** is commonly used:

$$L(h(\boldsymbol{x^i}), y^i) = \mathbf{1}_{\{h(\boldsymbol{x^i}) \neq y^i\}} = \begin{cases} 1 & \text{if } h(\boldsymbol{x^i}) \neq y^i \\ 0 & \text{if } h(\boldsymbol{x^i}) = y^i \end{cases} \tag{3.2.3}$$

<u>Notation</u>: $\mathbf{1}_A(x)$ denotes the indicator function. Then, if $x \in A$, $\mathbf{1}_A(x) = 1$, and $\mathbf{1}_A(x) = 0$ otherwise.

For regression problems, where the output is a real number, the **square error** is used as a loss function:

$$L(h(\boldsymbol{x^i}), y^i) = (h(\boldsymbol{x^i}) - y^i)^2 \tag{3.2.4}$$

Definition 3.2.2. *The* ***empirical*** *or* ***training error*** *of a function $h : \mathcal{X} \to \mathcal{Y}$ is the average loss $L(h(\boldsymbol{x^i}), y^i)$ over the training data.*

$$\hat{R}(h) := \frac{1}{n}\sum_{i=1}^{n} L(h(\boldsymbol{x^i}), y^i) \tag{3.2.5}$$

Therefore, substituting the loss function expression in (3.2.5) with the corresponding forms used for classification and regression problems in equations (3.2.3) and (3.2.4), respectively, it is obtained:

$$\hat{R}(h) = \frac{1}{n}\sum_{i=1}^{n} L(h(\boldsymbol{x^i}), y^i) = \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}_{\{h(\boldsymbol{x^i}) \neq y^i\}} \tag{3.2.6a}$$

$$\hat{R}(h) = \frac{1}{n}\sum_{i=1}^{n} L(h(\boldsymbol{x^i}), y^i) = \frac{1}{n}\sum_{i=1}^{n} (h(\boldsymbol{x^i}) - y^i)^2 \tag{3.2.6b}$$

The equation (3.2.6a) is the final expression of the empirical error for classification problems, while (3.2.6b) is used in the regression case.

All things considered, the main objective was to find an approximation function h among a set $\mathcal{H}$ close to the target function. Using the previous concept introduced, we can formulate an optimisation problem whose aim is to minimise the empirical error:

$$\min_{h \in \mathcal{H}} \hat{R}(h) \tag{3.2.7}$$

However, if this was the only objective, the result would lead to **overfitting** because the optimal solution would be any function h that satisfies

$$h(\boldsymbol{x^i}) = y^i \quad \forall i = \{1, \ldots, n\} \tag{3.2.8}$$

This is known as **memorisation** and it occurs when the function adapts too closely to the seen data. As a result, there is no empirical risk, but the function will perform poorly on unseen data.

$$\hat{R}(h) = \frac{1}{n}\sum_{i=1}^{n} L(h(\boldsymbol{x^i}), y^i) = \frac{1}{n}\sum_{i=1}^{n} L(y^i, y^i) = 0$$

Therefore, it is vital to consider how the classifier generalise the learning process instead of memorising to be able to forecast unseen data. When a model generalises well, it is able to capture underlying patterns and relationships from the training data and apply them to make accurate predictions or classifications on new instances. Nonetheless, generalisation could lead to the opposite side, **underfitting**, when the model has not been trained enough on the data. This also lead to a poor classifier or prediction function since it would not predict accurately neither training nor new data and it would only capture a general trend, which is not enough to fit appropiately. These are the reasons why it is important to find a balance between memorisation and generalisation, or, in other words, between overfitting and underfitting.

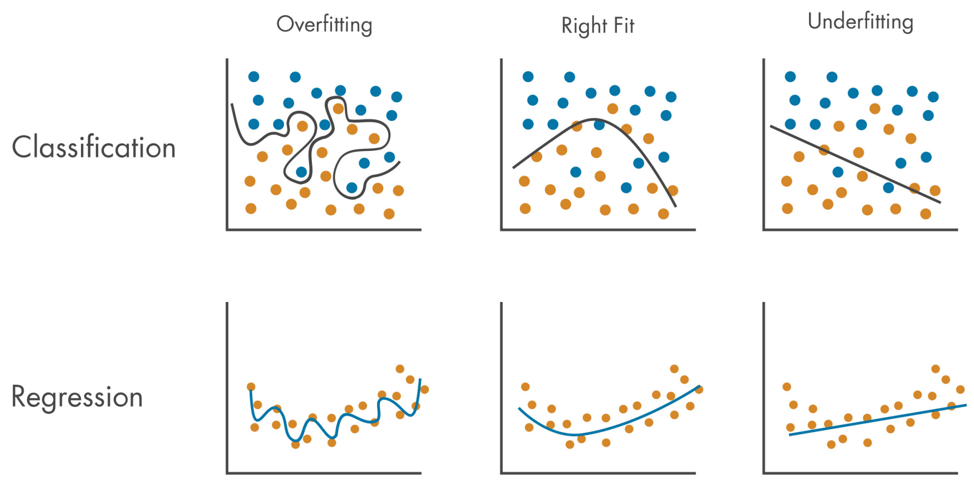


Figure 3.2: Overfitting and underfitting in regression and classification models. Source: [25]

To introduce the concept of generalisation risk, we assume that the training data is considered as samples from a pair of random variables (X, Y) with an unknown probability distribution $X \times Y$.

Definition 3.2.3. *The* ***generalisation risk*** *explains how a classifier $h \in \mathcal{H}$ performs on average on unseen data given a loss function L.*

$$R(h) := \mathbb{E}[L(h(X), Y)] \tag{3.2.9}$$

Note that if we consider the unit loss function, the generalisation risk is the probability of misclassifying a random instance of the distribution $X \times Y$.

$$R(h) := \mathbb{E}[L(h(X), Y)] = 1 \cdot P(\{h(X) \neq Y\}) + 0 \cdot P(\{h(X) = Y\}) = P(\{h(X) \neq Y\})$$

Instead of considering training data to be n instances $\{\boldsymbol{x^i}, y^i\}_{i=1}^n$, the data can be also modelled as sampling from n pairs of random variables (X_i, Y_i), which are identically distributed and independent copies of (X, Y).

Proposition 3.2.1. *The empirical risk $\hat{R}(h)$ is an* ***unbiased estimator*** *of the generalisation risk $R(h)$.*

Proof. For this framework and a given classifier h, the empirical risk is now a random variable, to which the expected value can be applied:

$$\mathbb{E}[\hat{R}(h)] = \mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n} L(h(X_i), Y_i)\right] \overset{lin}{=} \frac{1}{n}\sum_{i=1}^{n} \mathbb{E}[L(h(X_i), Y_i)]$$
$$\overset{iid}{=} \frac{n}{n}\mathbb{E}[L(h(X), Y)] = \mathbb{E}[L(h(X), Y)] = R(h)$$

The linearity property of expectation and the independence and identically distribution of (X_i, Y_i) to (X, Y) $\forall i \in \{1, \ldots, n\}$ help to conclude the proof. □

Definition 3.2.4. *The* ***Bayes classifier*** *is a function that achieves the minimal risk among all possible functions.*

Mathematically, the Bayes classifier can be expressed as a maximum a posteriori (MAP) estimador as:

$$h^*(\boldsymbol{x}) = \arg\max_{y} P(Y = y|X = x) \tag{3.2.10}$$

Definition 3.2.5. *The* ***excess risk*** *of a classifier $h \in \mathcal{H}$ compares the generalisation risk of this classifier to the Bayes classifier.*

$$\mathcal{E}(h) = R(h) - R(h^*) \tag{3.2.11}$$

Note that the excess risk satisfies $\mathcal{E}(h) \geq 0$ by definition.

We can decompose the generalisation risk of a given classifier $\hat{h} \in \mathcal{H}$ into three quantities:

$$R(\hat{h}) = \underbrace{R(\hat{h}) - \inf_{h\in\mathcal{H}} R(h)}_{\text{Estimation error}} + \underbrace{\inf_{h\in\mathcal{H}} R(h) - R(h^*)}_{\text{Approximation error}} + \underbrace{R(h^*)}_{\text{Irreducible error}} \tag{3.2.12}$$

- **Estimation error**: it is the difference in performance between the classifier $\hat{h}$ and the best possible classifier within the class $\mathcal{H}$.
- **Approximation error**: it measures how close the best generalisation error in the space $\mathcal{H}$ is to the one associated with the Bayes classifier (the best among all).
- **Irreducible error**: it is the generalisation error associated to the Bayes classifier, and therefore, it cannot be reduced.

There is a trade-off between estimation and approximation errors: they are inversely related, that is to say, when one increases, the other decreases. Here, there are the two possible extreme situations:

1. The estimation error is reduced when the classifier $\hat{h} \in \mathcal{H}$ is close to the best among the space $\mathcal{H}$. This can be achieved by making $\mathcal{H}$ smaller, but this would lead to an increase in the approximation error since the space $\mathcal{H}$ would be too restrictive and it would be more difficult that the Bayes classifier or a closed approximation could be contained in $\mathcal{H}$.
2. On the contrary, the approximation error is reduced by increasing the size of the space $\mathcal{H}$ so that the best within the class could be closer or even equal to the Bayes classifier. This would imply an increase in the estimation error since it would be complex in some situations to find the minimum within a large space $\mathcal{H}$.

This is the reason why it is vital to find a balance between them.

3.2.2 Generalisation-Approximation Trade-off via the Bias-Variance Analysis

The same analysis as before can be done to evaluate how close our predictions using a function $\hat{h}$ is to the response variable Y. This is known as the **bias-variance** trade-off in statistics.

We assume that there is a function f that relates X and Y such that

$$Y = f(X) + \varepsilon \quad \text{where} \quad \mathbb{E}[\varepsilon] = 0 \tag{3.2.13}$$

This function f is what we called target function, and this is the relation h would like to capture and learn, considering a perturbation factor ε. This can be assumed since in the case X and Y were totally independent, there would not be nothing to learn. There would not be any classifier better than just random guessing.

$$\begin{aligned}\mathbb{E}[(Y - \hat{h}(X))^2] &= \mathbb{E}[(Y - \mathbb{E}[\hat{h}] + \mathbb{E}[\hat{h}] - \hat{h}(X))^2] \\ &= \mathbb{E}[(Y - \mathbb{E}[\hat{h}])^2] + \mathbb{E}[(\mathbb{E}[\hat{h}] - \hat{h}(X))^2] + 2\underbrace{\mathbb{E}[(Y - \mathbb{E}[\hat{h}])(\mathbb{E}[\hat{h}] - \hat{h}(X))]}_{=0}\end{aligned} \tag{3.2.14}$$

The last term of equation (3.2.14) is null by using the linearity of expectation and then, the same term is subtracted, making all the term to be 0. In addition to this, the hypothesis (3.2.13) could be use to develop the first term.

$$\mathbb{E}[(Y - \mathbb{E}[\hat{h}])(\mathbb{E}[\hat{h}] - \hat{h}(X))] = (Y - \mathbb{E}[\hat{h}])\mathbb{E}[(\mathbb{E}[\hat{h}] - \hat{h}(X))] = (Y - \mathbb{E}[\hat{h}])(\underbrace{\mathbb{E}[\hat{h}] - \mathbb{E}[\hat{h}(X)]}_{=0})$$

$$\begin{aligned}\mathbb{E}[(Y - \mathbb{E}[\hat{h}])^2] &= \mathbb{E}[(f(X) + \varepsilon - \mathbb{E}[\hat{h}])^2] \\ &= \mathbb{E}[(f(X) - \mathbb{E}[\hat{h}])^2] + \mathbb{E}[\varepsilon^2] + 2\underbrace{\mathbb{E}[(f(X) - \mathbb{E}[\hat{h}])\varepsilon]}_{=0} \\ &= \mathbb{E}[(f(X) - \mathbb{E}[\hat{h}])^2] + \mathbb{E}[\varepsilon^2]\end{aligned} \tag{3.2.15}$$

Here, we have use the independence regarding the expectation together with the null mean of ε.

$$\mathbb{E}[(f(X) - \mathbb{E}[\hat{h}])\varepsilon] = \mathbb{E}[f(X) - \mathbb{E}[\hat{h}]]\underbrace{\mathbb{E}[\varepsilon]}_{=0}$$

Now, we can substitute the previous expression (3.2.15) in (3.2.14) to obtain the final result.

$$
\begin{aligned}
\mathbb{E}[(Y-\hat{h}(X))^2] &= \mathbb{E}[(Y-\mathbb{E}[\hat{h}]+\mathbb{E}[\hat{h}]-\hat{h}(X))^2] \\
&= \mathbb{E}[(Y-\mathbb{E}[\hat{h}])^2]+\mathbb{E}[(\mathbb{E}[\hat{h}]-\hat{h}(X))^2]+2\underbrace{\mathbb{E}[(Y-\mathbb{E}[\hat{h}])(\mathbb{E}[\hat{h}]-\hat{h}(X))]}_{=0} \\
&= \mathbb{E}[(f(X)-\mathbb{E}[\hat{h}])^2]+\mathbb{E}[(\mathbb{E}[\hat{h}]-\hat{h}(X))^2]+\mathbb{E}[\varepsilon^2] \\
&= \mathbb{E}[\underbrace{(f(X)-\mathbb{E}[\hat{h}])^2}_{\text{Bias}^2}]+\underbrace{\mathbb{E}[(\mathbb{E}[\hat{h}]-\hat{h}(X))^2]}_{\text{Variance}}+\underbrace{\sigma^2}_{\text{Noise}}
\end{aligned}
\tag{3.2.16}
$$

The idea of the bias-variance trade-off is to decompose the mean squared error in terms of the following quantities:

- **Variance**: how much the learning method $\hat{h}$ will move around its average from the entire set $\mathcal{H}$.
- **Bias**: corresponds to the systematic error caused by simplifying assumptions to approximate the target function $f(X)$. An example would be restricting the space of classifiers $\mathcal{H}$ to linear functions to approximate a non-linear target function f.
- **Noise** or also known as **irreducible error** since it is the error associated to the Bayes classifier (the best among all possible classifiers). This noise comes from the data.

From (3.2.16), we can do a similar analysis as before:

1. If our classifier is unbiased, which is usually achieved by complex models, it can suffer from a large error if it is very variable. This is extremely related to overfitting.
2. On the contrary, if our classifier is very stable (low variance), it can suffer from a large bias, which would result in a large error. This is related to underfitting.

3.2.3 Model Performance

This subsection provides an overview of model validation, which plays a vital role in machine learning, whether it entails choosing the most suitable model or evaluating the performance of an existing model. Broadly speaking, numerous metrics are established for supervise learning in both classification and regression domains, which are described above.

3.2.3.1 Classification Metrics

In this work, we will focus on the simple case of binary classification, but the metrics can also be extended to multiclass classification. We can assume that the two classes are positive or negative. The results of a classification task can be simply visualised using the **confusion matrix**, which can be shown in table 3.1.

When dealing with classification, we have to do a previous analysis to know when the data is balanced in terms of the classes. A **balanced** dataset is a dataset where the number of samples in each category is approximately equal, that it to say, the classes have a similar representation. Having a balanced dataset is often desirable because it helps prevent bias towards one class during the training process. When dealing with **imbalanced** datasets, where one class is significantly underrepresented compared to others, machine learning models tend to have difficulty in learning patterns and making accurate predictions for the minority class. This imbalance can lead to biased models that favor the majority class and perform poorly on the minority class. These are the reasons why accuracy is a success informative measure when dealing with balanced sets, but more specific measures such as TPR, TNR and FNR as indicated below are needed in case of imbalance.

	Predicted Class	
True label	Positive	Negative
Positive	TP	FN
Negative	FP	TN

Table 3.1: Confusion matrix for binary classification. Source: [1]

As shown in table 3.1, the confusion matrix divides the test samples into four categories according to the predicted and true label:

- **True Positives (TP)**: both the predicted and the true label are positive.
- **True Negatives (NP)**: both the predicted and the true label are negative.
- **False Positives (FP)**: the true label is negative whereas the predicted label is positive.
- **False Negatives (FN)**: the true label is positive whereas the predicted label is negative.

These categories are also used to construct the following measures for model performance. Regarding the question whether false positives or false negatives are equally problematic, the answer really depends on the application and what we are considering as positive and negative. Once an expert has adviced about the importance and the implications of misclassifying, we could consider to give more relevance to some specific metrics which punish more false or true negatives, according to what had been advised.

Basic metrics

- **True positive rate (TPR)**, **recall** or **sensitivity**: fraction of positive samples actually classified as positive.

$$TPR = \frac{TP}{TP + FN}$$

- **Positive predictive value (PPV)** or **precision**: proportion of correctly predicted positive instances out of all the instances predicted as positive.

$$TPR = \frac{TP}{TP + FP}$$

- **True negative rate (TNR)** or **specificity**: fraction of negative samples actually classified as negative.

$$TNR = \frac{TN}{TN + FP}$$

- **Negative predictive value (NPV)**: proportion of correctly predicted negative instances out of all the instances predicted as negative.

$$NPV = \frac{TN}{TN + FN}$$

- **False negative rate (FNR)**, **miss rate** or **Type II error rate**: proportion of actual positive instances that are incorrectly classified as negative by a classification model.

$$FNR = \frac{FN}{TP + FN}$$

Summary metrics

- **Accuracy**: proportion of the samples correctly classified.

$$Accurary = \frac{TN + TP}{TP + FN + FP + TN}$$

- **F1 score**: is the harmonic mean of precision and recall.

$$F1 = \frac{2}{\frac{1}{Precision} + \frac{1}{Recall}} = \frac{2 \times Precision \times Recall}{Precision + Recall} = \frac{2TP}{2TP + FP + FN}$$

  A higher F1 Score indicates a better balance between precision and recall, with values ranging from 0 to 1. An F1 Score of 1 represents perfect precision and recall, while a score of 0 indicates poor performance.

- **Matthews correlation coefficient (MCC)**: it combines the basic measures to provide a balanced assessment of the model's performance.

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP) \times (TP + FN) \times (TN + FP) \times (TN + FN)}}$$

  MCC ranges from -1 to $+1$, where $+1$ indicates a perfect classification, 0 indicates random performance, and -1 indicates a complete disagreement between predicted and actual labels.

- **Area under the receiver operating characteristic curve (ROC AUC).** The ROC curve is a graphical representation that shows the trade-off between the true positive rate (sensitivity) and the false positive rate (1 − specificity) at various classification thresholds. It effectively demonstrates the model's ability to differentiate between positive and negative instances across varying threshold settings. On the other hand, the ROC AUC (Area Under the Curve) is a metric that quantifies the overall performance of the model by computing the area under the ROC curve. It ranges from 0 to 1, with a higher value indicating stronger discrimination and predictive capability. A score of 0.5 implies a random classifier, while a score of 1 signifies a flawless classifier [26].

In multiclass classification, distinguishing between multiple classes differs from the binary classification scenario. Traditional performance measures such as sensitivity, specificity, F1-score, ROC curves, and precision-recall curves may not hold meaning for the entire dataset. Nonetheless, accuracy remains a relevant and straightforward metric to evaluate performance. In this situation, analysing the confusion matrix in multi-class settings provides a valuable tool for understanding errors and gaining insights into the classification process.

A common approach in multiclass setting is to calculate metrics for each class individually, treating it as the positive class and the rest as the negative class (*one versus all*). There are different methods for averaging the results [26]:

- **Macro average** computes the metric for each class and then averages them, but it may give too much importance to infrequent classes. It is an arithmetic (unweighted) mean.
- **Weighted average** considers the metric weighted by the number of true instances (also known as support).
- **Micro average** combines the results taking into account the number of instances for each class, that is to say, the number of true positives, true negatives, false positives, and false negatives across all classes. It essentially computes the proportion of correctly classified observations out of all observations, which is the definition of accuracy.

3.2.3.2 Regression Metrics

Regression analysis is a statistical predictive modelling technique, which investigates the relationship between a dependent (target) and independent variable(s) (predictor) [29]. This technique is used for forecasting, time series modelling and finding the causal effect relationship between variables [27]. In the regression domain, we have to compare the predicted output $\hat{y}_i$, which is a real number, to the actual real-value y_i [1]. Regression metrics mainly consist of variants of squared errors:

- **Mean Bias Error** (MBE): is the average of the bias error, which is defined as the difference between the predicted values and the real target values.

$$MBE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)$$

Note that $MBE = 0$ is a necessary but not a sufficient condition for a perfect match since the positive and negative errors can cancel each other in cases where the match is not perfect [27].

- **Mean Squared Error** (MSE): is the average squared error. Thus, it can take positive or zero values, giving importance to large errors. Lower MSE values indicate better model performance.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

In this case, note that $MSE = 0$ is a necessary and sufficient condition for a perfect match.

- **Root Mean Squared Error** (RMSE): is the square root of MSE, providing a measure of the average prediction error in the same units as the target variable.

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

- **Mean Absolute Percentage Error** (MAPE): calculates the average percentage absolute difference between the predicted and actual values, relative to the actual values.

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{\hat{y}_i}\right|$$

- **Coefficient of determination** or (R^2): summarises the explanatory power of the regression model. It ranges from 0 to 1, with higher values indicating a better fit of the model to the data.

$$R^2 = 1 - \frac{SSE_{residual}}{SSE_{total}} = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \overline{y})^2}$$

where $\overline{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$. It is also worth mentioning that R^2 can sometimes be misleading, especially in the case of overfitting. A high R^2 value does not necessarily mean that the model will generalise well to new data.

- **Adjusted R^2**: modification of the R^2 coefficient to take into account the number of predictors since overfitting increases as unnecessary predictors are included.

$$R^2 = 1 - \frac{\frac{SSE_{residual}}{df_{residual}}}{\frac{SSE_{total}}{df_{total}}} = 1 - \frac{\frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n-(p+1)}}{\frac{\sum_{i=1}^{n}(y_i - \overline{y})^2}{n-1}}$$

where p is the number of explanatory variables the model has.

3.2.4 Model Validation

In the context of machine learning, model validation refers to the procedure of evaluating and assessing the performance as well as the generalisation capability of a trained model using validation data. It helps to determine the model's performance on unseen data, offering valuable insights into its effectiveness and reliability.

Validation requires selecting a model with the right hyperparameter values. For this, it is crucial that performance should not be measured using the same data that was used for training. Therefore, the first step is to split the data into a training set $\mathcal{D}_{train}$ and a validation set $\mathcal{D}_{val}$. This can be done using the function `model_selection.train_test_split` from the library `sklearn` (section (5.3)). The process consists of training the model and then, evaluating this model in the validation set for further improvement, as shown in Figure 3.3.

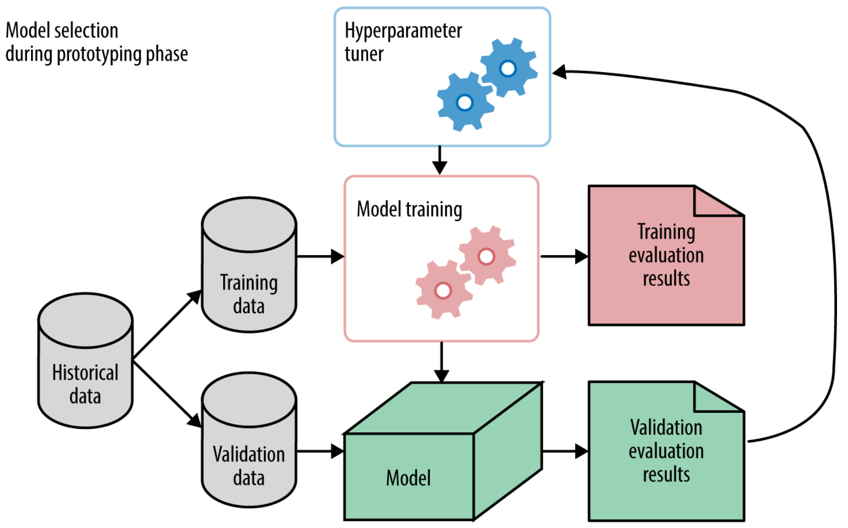


Figure 3.3: Model training and validation process. Source: [30]

The main drawback with this methodology is the requisite to have a large dataset for splitting, which sometimes can be a difficult or expensive task. For this reason, an alternative is k-**fold cross-validation**, in which the dataset is divided into k folds of approximately equal size. The model is trained k times, each time using $k-1$ folds for training and one fold for validation to measure the error E^i_{Val} for a fold $i \in \{1,..k\}$. These metrics are then averaged across all the iterations to obtain a single estimte of the validation error E_{Val}.

$$E_{Val} = \frac{1}{k}\sum_{i=1}^{k} E^i_{Val}$$

This technique provides a more robust estimation of model performance and is less dependent on a single data split. For the smallest datasets, a special case of k-fold cross-validation is used: the **leave-one-out cross validation**, where only one observation is omitted.

The process is illustrated in figure 3.7, which can be used to retrieve information about model performance and also for hyper-parameters search.

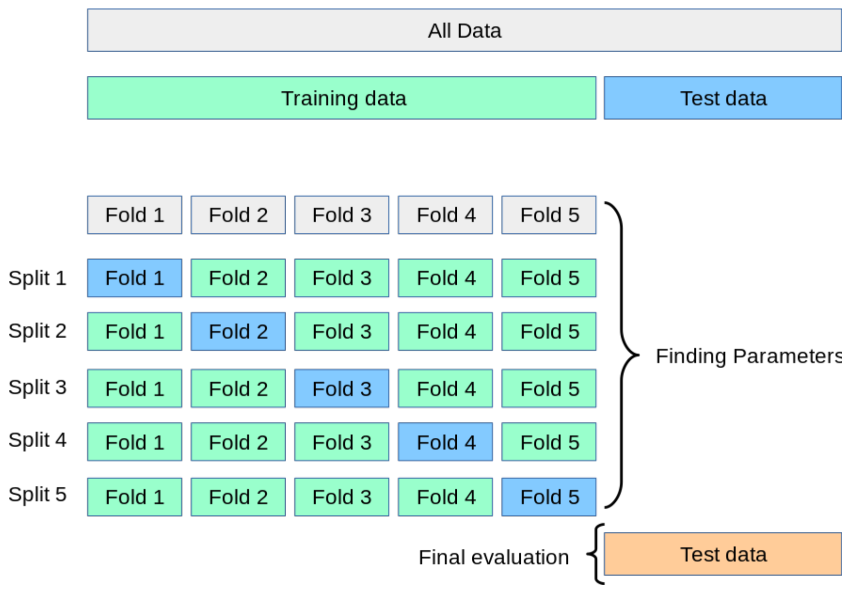


Figure 3.4: Illustration of 5-fold cross-validation. Source: Scikit-learn [83].

As mentioned in the previous section (3.2.3), it is important to have a balanced set. In some cases, we may want also want labels to be equally distributed in the training and testing sets. This is called **stratification**, which can be easily implemented by `sklearn.model_selection.StratifiedShuffleSplit()` or `StratifiedKFold` method for stratified k-fold cross-validation.

3.3 Discriminative Classifiers

In this section, we are introducing discriminative classifiers that are used in the final section (7). **Discriminative classifiers** focuses on modelling the decision boundary or the posterior probability of the class labels given the input features $P(Y|X)$. Rather than explicitly modelling the joint probability distribution of the features and class labels, discriminative classifiers directly learn the mapping function from input features to class labels.

Discriminative classifiers require labelled training data to learn the mapping function that directly predicts the labels from the input features. This learning should consider unseen data when deciding the best parameters for the model. In classification, a decision rule is required to assign instances to classes.

3.3.1 Logistic Regression

Logistic regression extends the ideas of linear regression since it can be seen as a transformation on the linear combination $\boldsymbol{\beta}^\intercal \boldsymbol{x}$, which returns a probability in case of classification.
This transformation is mainly based in the **logit** or **sigmoid function**:

$$\sigma(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}$$

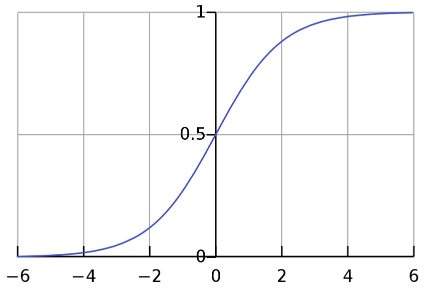


Figure 3.5: Sigmoid function. Source: [32].

Properties of the sigmoid function:

- Bounded in (0,1).
- $\lim_{x\to\infty} \sigma(x) = 1$
- $\lim_{x\to-\infty} \sigma(x) = 0$
- $\sigma(x)' = \sigma(x)\,(1-\sigma(x))$

For a binary classification, then we assume the following framework $y_i \in \{0,1\}$ and explanatory variables $x_{i1},\dots,x_{ip}$, $\boldsymbol{x}_i \in \mathbb{R}^p \quad \forall i \in \{1,\dots,n\}$.

In this case, we want to model $P(y_i = 1|\boldsymbol{x_i})$. Since it is a probability, we cannot model it as a linear function as this would return values outside of (0,1). This is the reason why the logit function was used as well as **odds ratio** to end up modelling their log-odds as a linear function:

$$\frac{P(y_i = 1|\boldsymbol{x_i})}{P(y_i = 0|\boldsymbol{x_i})} = \frac{p_i}{1-p_i} \tag{3.3.1}$$

Taking the logarithm, it can now take any real value, which can be modelled as an instance of a generalised linear model (GLM).

$$\log\left(\frac{P(y_i = 1|\boldsymbol{x_i})}{P(y_i = 0|\boldsymbol{x_i})}\right) = \log\left(\frac{p_i}{1-p_i}\right) = \beta_0 + \boldsymbol{\beta}^\intercal \boldsymbol{x_i} \tag{3.3.2}$$

where $\boldsymbol{\beta} \in \mathbb{R}^p$.

It sufficies to solve for p_i to obtain the value modelled by the logit or sigmoid function as:

$$p_i = P(y_i = 1|\boldsymbol{x_i}) = \frac{e^{\beta_0 + \boldsymbol{\beta}^\intercal \boldsymbol{x_i}}}{1 + e^{\beta_0 + \boldsymbol{\beta}^\intercal \boldsymbol{x_i}}} = \frac{1}{1 + e^{-(\beta_0 + \boldsymbol{\beta}^\intercal \boldsymbol{x_i})}} = \sigma\left(\beta_0 + \boldsymbol{\beta}^\intercal \boldsymbol{x_i}\right) \tag{3.3.3}$$

Now, we can know the value of the odds ratio in equation (3.3.1) :

$$\frac{P(y_i = 1|\boldsymbol{x_i})}{P(y_i = 0|\boldsymbol{x_i})} = \frac{p_i}{1 - p_i} = e^{\beta_0 + \boldsymbol{\beta}^\intercal \boldsymbol{x_i}}$$

The resulting classifier is $\hat{y}_i = 1$ if $p_i > c$ and $\hat{y}_i = 0$ if $p_i < c$ for a threshold c fixed according to classification metrics (section (3.2.3.1)), which is usually $\frac{1}{2}$.

The problem has been reduced to estimating the parameters β_0 and $\boldsymbol{\beta}$ given a dataset.

Some advantages of logistic regression are that it can detect irrelevant variables ($\beta_j = 0$) and it also allows for non-linear terms by substituting with a new variable ($z_k = x_k^2$).

The likelihood given a dataset can be written as:

$$\mathcal{L}(\boldsymbol{x}) = \prod_{i=1}^{n} P(y_i|\boldsymbol{x_i})$$

Taking the logarithm, we obtain the log-likelihood:

$$\log \mathcal{L}(\boldsymbol{x}) = \sum_{i=1}^{n} \log\left(P(y_i|\boldsymbol{x_i})\right) = \sum_{i=1}^{n} \begin{cases} \log P(y_i = 1|\boldsymbol{x_i}) = \log(p_i) & \text{if } y_i = 1 \\ \log P(y_i = 0|\boldsymbol{x_i}) = \log(1 - p_i) & \text{if } y_i = 0 \end{cases}$$

$$\log \mathcal{L}(\boldsymbol{x}) = \sum_{i=1}^{n} \begin{cases} \log\left(\sigma\left(\beta_0 + \boldsymbol{\beta}^\intercal \boldsymbol{x_i}\right)\right) & \text{if } y_i = 1 \\ \log\left(1 - \sigma\left(\beta_0 + \boldsymbol{\beta}^\intercal \boldsymbol{x_i}\right)\right) & \text{if } y_i = 0 \end{cases} \tag{3.3.4}$$

This expression (3.3.4) can be rewritten as:

$$\log \mathcal{L}(\sigma(\boldsymbol{x})) = \sum_{i=1}^{n} \left(y_i \log\left(\sigma\left(\beta_0 + \boldsymbol{\beta}^\intercal \boldsymbol{x_i}\right)\right) + (1 - y_i) \log\left(1 - \sigma\left(\beta_0 + \boldsymbol{\beta}^\intercal \boldsymbol{x_i}\right)\right)\right) \tag{3.3.5}$$

As mentioned in section (3.2.1), the loss or cost function can be used to measure how a model performs. In the case of logistic regression, we use the **logarithmic cost** or **cross-entropy** function (3.3.5). This loss function can be used to optimise the solution through iterative algorithms as **gradient descent** (see section (3.3.4)).

3.3.2 Support Vector Machines

Support Vector Machines (SVM) is considered to be one of the most relevant machine learning algorithm based on statistical learning theory. This model was developed by Vladimir Vapnik and his colleagues (Boser, Guyon and Cortes). SVM is designed to map training examples to points in space with the objective of maximising the margin or width of separation between different categories. This margin represents the gap between the decision boundary of the SVM and the closest data points from each category. Once the mapping is established, new examples can be projected onto the same space and predicted to belong to a specific category based on which side of the margin they fall on. In other words, the SVM classifies new examples by determining which side of the gap they lie on.

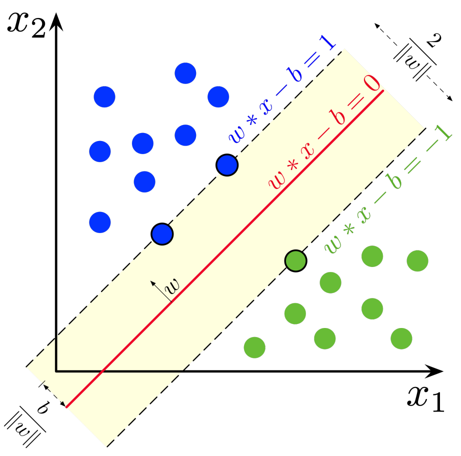


Figure 3.6: SVM finding the maximum margin separation between labelled data. Source: Wikipedia [33].

In the simplest case, given a binary labelled dataset $\mathcal{D}_{labelled}$, which corresponds to $\{(\boldsymbol{x^i}, y^i)\}_{i=1}^N$, with $\boldsymbol{x^i} \in \mathcal{X} \subset \mathbb{R}$ and $y^i \in \{-1, 1\}$, being linearly separable. The goal is to find a hyperplane that separates one class from the other so that we can determine the maximum separation between labels. This problem can be formulated as a linear programming problem:

Find the weights $\boldsymbol{w} \in \mathbb{R}^d$ and a bias term $b \in \mathbb{R}$ such that

$$\boldsymbol{w}^\intercal \boldsymbol{x} - b \geq 1 \quad \text{if } y_i = 1, \qquad \boldsymbol{w}^\intercal \boldsymbol{x} - b \leq -1 \quad \text{if } y_i = -1 \tag{3.3.6}$$

By convention, the quantities 1 and -1 are used, but any other quantity could be accepted by rescaling $\boldsymbol{w}$ and b. In addition to this, the problem (3.3.6) is equivalent to:

$$y_i(\boldsymbol{w}^\intercal \boldsymbol{x} - b) - 1 \geq 0 \quad \forall i \in \{1, \ldots, n\} \tag{3.3.7}$$

When the equality of these inequations is reached, that is to say, a point $\boldsymbol{x}$ that satisfies $\boldsymbol{w}^\intercal \boldsymbol{x} - b$ equal to 1 or -1 is known as **support vector** since it is closest to the **maximum-margin hyperplane**. This hyperplane satisfies $\boldsymbol{w}^\intercal \boldsymbol{x} - b = 0$.

Denote the distance from the maximum-margin hyperplane to the closest positive and closest negative point by δ_+ and δ_-, respectively. Then, the **margin** of a classifier is the sum of these two distances $\delta = \delta_+ + \delta_-$.

Assume $\boldsymbol{x}$ to be the closset positive point to $H = \{\boldsymbol{x} : h(\boldsymbol{x}) = 0\}$. Then,

$$\boldsymbol{w}^\intercal \boldsymbol{x} - b = 1 \text{ and } \delta_+^2 = \min_{\boldsymbol{z}} \|\boldsymbol{x} - \boldsymbol{z}\|^2 \text{ such that } h(\boldsymbol{z}) = 0$$

The latter can be formulated using Lagrange multipliers:

$$\mathcal{L}(\boldsymbol{z}, \lambda) = \|\boldsymbol{x} - \boldsymbol{z}\|^2 + \lambda h(\boldsymbol{z}) = \|\boldsymbol{x} - \boldsymbol{z}\|^2 + \lambda(\boldsymbol{w}^\intercal \boldsymbol{z} - b)$$

To solve the optimisation problem, differenciate as follows:

$$\nabla_{\boldsymbol{z}} \mathcal{L}(\boldsymbol{z}, \lambda) = 2(\boldsymbol{x} - \boldsymbol{z}) + \lambda \boldsymbol{w} = 0 \tag{3.3.8}$$

$$\frac{\partial}{\partial \lambda} \mathcal{L}(\boldsymbol{z}, \lambda) = \boldsymbol{w}^\intercal \boldsymbol{z} - b = 0 \;\Rightarrow\; \boldsymbol{w}^\intercal \boldsymbol{z} = b$$

By multiplying the first equation (3.3.8) by $\boldsymbol{w}^\intercal$ and substituting then:

$$2(\boldsymbol{w}^\intercal \boldsymbol{x} - \underbrace{\boldsymbol{w}^\intercal \boldsymbol{z}}_{=b}) + \lambda \|\boldsymbol{w}\|^2 = 0 \;\Rightarrow\; 2(\underbrace{\boldsymbol{w}^\intercal \boldsymbol{x} - b}_{=h(\boldsymbol{x})=1}) + \lambda \|\boldsymbol{w}\|^2 = 0 \;\Rightarrow\; \lambda = \frac{-2}{\|\boldsymbol{w}\|^2}$$

Once we have λ value, we can substitue it in equation (3.3.8):

$$\boldsymbol{x} - \boldsymbol{z} = \frac{-\lambda \boldsymbol{w}}{2} = \frac{\boldsymbol{w}}{\|\boldsymbol{w}\|^2}$$

Then, we have found the minimum distance to the closest positive label by applying the norm to the previous equation:

$$\delta_+ = \|\boldsymbol{x} - \boldsymbol{z}\| = \frac{1}{\|\boldsymbol{w}\|}$$

The same argument can be applied to the distance closest to the negative label δ_- and then, the margin of this particular hyperplane is:

$$\delta = \delta_+ + \delta_- = \frac{2}{\|\boldsymbol{w}\|}$$

To find the hyperplane with largest margin, we thus have to solve a quadratic optimisation problem:

$$\text{(P)} \quad \min_{\boldsymbol{w}, b} \frac{\|\boldsymbol{w}\|^2}{2} \qquad \text{subject to } \; y_i(\boldsymbol{w}^\intercal \boldsymbol{x} - b) - 1 \geq 0 \quad \forall i \in \{1, ..n\} \tag{3.3.9}$$

The Lagrangian of this problem is

$$\mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\lambda}) = \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_{i=1}^{n} (\lambda_i y_i \boldsymbol{w}^\intercal \boldsymbol{x} - \lambda_i y_i b - \lambda_i) = \frac{1}{2} \boldsymbol{w}^\intercal \boldsymbol{w} - \boldsymbol{\lambda}^\intercal \boldsymbol{X} \boldsymbol{w} + b \boldsymbol{\lambda}^\intercal \boldsymbol{y} + \sum_{i=1}^{n} \lambda_i$$

where $\boldsymbol{X}$ denotes the matrix with the $y_i \boldsymbol{x_i}^\intercal$ as rows. Now, we can derive the conditions on the gradient with respect to $\boldsymbol{w}$ and b of the Lagrangian as

$$\nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\lambda}) = \boldsymbol{w} - \boldsymbol{X}^\intercal \boldsymbol{\lambda} = 0 \tag{3.3.10}$$

$$\frac{\partial}{\partial b} \mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\lambda}) = \boldsymbol{y}^\intercal \boldsymbol{\lambda} = 0 \tag{3.3.11}$$

If (3.3.11) is not satisfied, the problem is unbounded and cannot be solved.

If $\boldsymbol{y}^\intercal\boldsymbol{\lambda} = 0$, (3.3.10) is a necessary and sufficient condition for a minimiser. To obtain the Lagrange dual $g(\boldsymbol{\lambda})$, we use the conditions (3.3.10) and (3.3.11):

$$g(\boldsymbol{\lambda}) = \begin{cases} -\frac{1}{2}\boldsymbol{\lambda}^\intercal\boldsymbol{X}\boldsymbol{X}^\intercal\boldsymbol{\lambda} + \sum_{i=1}^{n}\lambda_i & \text{if } \boldsymbol{y}^\intercal\boldsymbol{\lambda} = 0 \\ -\infty & \text{else} \end{cases}$$

Since, it sufficies to maximise the Lagrange dual problem:

$$\max_{\boldsymbol{\lambda}\geq\boldsymbol{0}} g(\boldsymbol{\lambda}) \Leftrightarrow \min_{\boldsymbol{\lambda}\geq\boldsymbol{0}} -g(\boldsymbol{\lambda})$$

Then, the dual optimisation problem to solve is:

$$\text{(D)} \qquad \min_{\boldsymbol{\lambda}} \frac{1}{2}\boldsymbol{\lambda}^\intercal\boldsymbol{X}\boldsymbol{X}^\intercal\boldsymbol{\lambda} - \boldsymbol{\lambda}^\intercal\boldsymbol{e} \quad \text{subject to } \boldsymbol{\lambda} \geq \boldsymbol{0} \tag{3.3.12}$$

where $\boldsymbol{e}$ is the vector whose components are equal to 1.

Karush-Kuhn-Tucker conditions are introduced to determine b, since the solution of the dual problem only gives the weights $\boldsymbol{w}$ obtained from $\boldsymbol{w} = \boldsymbol{X}^\intercal\boldsymbol{\lambda}$ (equation (3.3.10)).

Theorem 3.3.1 (Karush-Kuhn-Tucker conditions)**.** *Given a primal problem*

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} f(\boldsymbol{x})$$

$$\textit{subject to } \boldsymbol{h}(\boldsymbol{x}) \leq 0$$

$$\boldsymbol{l}(\boldsymbol{x}) = 0$$

Assume $\boldsymbol{x}^*$ and $(\boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$ are the primal and dual solutions to the previous problem, respectively. Then, the following conditions are satisfied:

$$\begin{aligned} \boldsymbol{h}(\boldsymbol{x}^*) &\leq \boldsymbol{0} \\ \boldsymbol{l}(\boldsymbol{x}^*) &= \boldsymbol{0} \\ \boldsymbol{\lambda}^* &\geq \boldsymbol{0} \\ \lambda_i^* h_i(\boldsymbol{x}^*) &= 0, \quad 1 \leq i \leq n \\ \nabla_{\boldsymbol{x}} f(\boldsymbol{x}^*) + \nabla_{\boldsymbol{x}}\boldsymbol{h}(\boldsymbol{x}^*)^\intercal\boldsymbol{\lambda}^* + \nabla_{\boldsymbol{x}}\boldsymbol{l}(\boldsymbol{x}^*)^\intercal\boldsymbol{\mu}^* &= \boldsymbol{0} \end{aligned} \tag{3.3.14}$$

Then, combinaning the KKT conditions of the primal problem (3.3.9) with the conditions of the Lagrangian, we obtain the following:

$$\begin{aligned} \boldsymbol{X}\boldsymbol{w} - b\boldsymbol{y} - \boldsymbol{e} &\geq \boldsymbol{0} \\ \boldsymbol{\lambda} &\geq \boldsymbol{0} \\ \lambda_i(1 - y_i(\boldsymbol{w}^\intercal\boldsymbol{x} - b)) &= 0 \quad \forall i \in \{1, \ldots, n\} \\ \boldsymbol{w} - \boldsymbol{X}^\intercal\boldsymbol{\lambda} &= 0 \\ \boldsymbol{y}^\intercal\boldsymbol{\lambda} &= 0 \end{aligned} \tag{3.3.15}$$

We can know b from the third equation when $\lambda_i \neq 0$. Now, we can find the maximum margin hyperplane solving the previous equations since it provides the weights $\boldsymbol{w}$ and the bias term b.

Nevertheless, this method is not applicable when labels are not linearly separable (our main assumption). To solve this problem, we introduce *slack variables* s_i to allow some observations to be *misclassified.* The resulting problem would be:

$$\text{(P)} \qquad \min_{\boldsymbol{w},b} \frac{\|\boldsymbol{w}\|^2}{2} + \mu \sum_{i=1}^{n} s_i$$
$$\text{subject to} \;\; y_i(\boldsymbol{w}^\intercal \boldsymbol{x} - b) - 1 + s_i \geq 0 \quad \forall i \in \{1, ..n\}$$
$$s_i \geq 0 \quad \forall i = \{1, \dots, n\}$$

where the parameter μ determines how much weight is given to misclassification, that is to say, if μ is large, then s_i will be close to 0 and less observations would be misclassified. The restrictions also allow for the i-th point to land on the wrong side if $s_i > 1$.

Following a similar procedure as before, the Lagrangian of this problem can also be retrieved.

Despite this new methodology, which can solve non-linearity issues in some cases, it can also not be enough. For this reason, non-linear transformations can be performed through the *kernel trick.* Note that the dual problem (3.3.12) depends only on the dot product of the data points, the entries of the matrix $\boldsymbol{X}\boldsymbol{X}^\intercal$. The trick is to perform non-linear separation by embedding data in **higher**-dimensional space $\mathcal{H}$:

$$\Phi : \mathbb{R}^p \to \mathcal{H}$$

This function Φ is known as **feature map**.

Definition 3.3.1. *A function*

$$k(\cdot, \cdot) : \mathbb{R}^p \times \mathbb{R}^p \to \mathbb{R}$$

is a ***kernel function*** *if* $\forall m \geq 1$ *and* $\boldsymbol{x_1}, \dots, \boldsymbol{x_m} \in \mathbb{R}^p$*, the* $m \times m$ *matrix* K *with* $(K)_{ij} := k(\boldsymbol{x_i}, \boldsymbol{x_j})$ *is symmetric and positive semi-definite.*

Proposition 3.3.1. *If* $k(\cdot, \cdot) : \mathbb{R}^p \times \mathbb{R}^p$ *defined as*

$$k(\boldsymbol{x}, \boldsymbol{y}) = \langle \Phi(\boldsymbol{x}), \Phi(\boldsymbol{y}) \rangle_{\mathcal{H}}$$

is a kernel function.

Proof. For the proof, we use the definition of a kernel function (3.3.1) to prove that the matrix K defined by $K_{ij} = k(\boldsymbol{x_i}, \boldsymbol{x_j})$ is:

- Symmetric: use the commutative property of the dot product.

$$K_{ij} = k(\boldsymbol{x_i}, \boldsymbol{x_j}) = \langle \Phi(\boldsymbol{x_i}), \Phi(\boldsymbol{x_j}) \rangle_{\mathcal{H}} = \langle \Phi(\boldsymbol{x_j}), \Phi(\boldsymbol{x_i}) \rangle_{\mathcal{H}} = k(\boldsymbol{x_j}, \boldsymbol{x_i}) = K_{ji}$$

- Positive semi-definite: for any $\boldsymbol{v} \in \mathbb{R}^m$

$$\boldsymbol{v}^\intercal K \boldsymbol{v} = \sum_{i,j=1}^{m} v_i K_{ij} v_j = \sum_{i,j=1}^{m} v_i k(\boldsymbol{x_i}, \boldsymbol{x_j}) v_j = \sum_{i,j=1}^{m} v_i \langle \Phi(\boldsymbol{x_i}), \Phi(\boldsymbol{x_j}) \rangle_{\mathcal{H}} v_j$$
$$= \langle \sum_{i=1}^{m} v_i \Phi(\boldsymbol{x_i}), \sum_{j=1}^{m} v_j \Phi(\boldsymbol{x_j}) \rangle_{\mathcal{H}} = \| \sum_{i=1}^{m} v_i \Phi(\boldsymbol{x_i}) \|_{\mathcal{H}}^2 \geq 0$$

□

Theorem 3.3.2 (Moore-Aronszajn theorem: Every kernel has an associated feature map)**.** *If $k(\cdot,\cdot)$ is a kernel function, then there exists a Hilbert space $\mathcal{H}$ with dot product $\langle\cdot,\cdot\rangle_{\mathcal{H}}$ such that*

$$k(\boldsymbol{x},\boldsymbol{y}) = \langle\Phi(\boldsymbol{x}),\Phi(\boldsymbol{y})\rangle_{\mathcal{H}} \quad \forall \boldsymbol{x},\boldsymbol{y} \in \mathbb{R}^p$$

By theorem (3.3.2), we can know work with a kernel instead of using the feature map. Some useful kernel functions are:

- **Radial basis function** or also known as **Gaussian kernel**:

$$k(\boldsymbol{x},\boldsymbol{y}) = e^{-\frac{\|\boldsymbol{x}-\boldsymbol{y}\|^2}{2\sigma^2}}$$

- **Polynomial kernel of degree** p:

$$k(\boldsymbol{x},\boldsymbol{y}) = (1+\boldsymbol{x}\boldsymbol{x}^\intercal)^p$$

Now, it sufficies to implement this idea into the (3.3.12) formulation:

$$\text{(D)} \qquad \min_{\boldsymbol{\lambda}} \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}(\lambda_i y_i k(\boldsymbol{x_i},\boldsymbol{x_j}) y_j \lambda_j) + \sum_{i=1}^{n}\lambda_i \quad \text{subject to } \boldsymbol{\lambda} \geq \boldsymbol{0} \qquad (3.3.17)$$

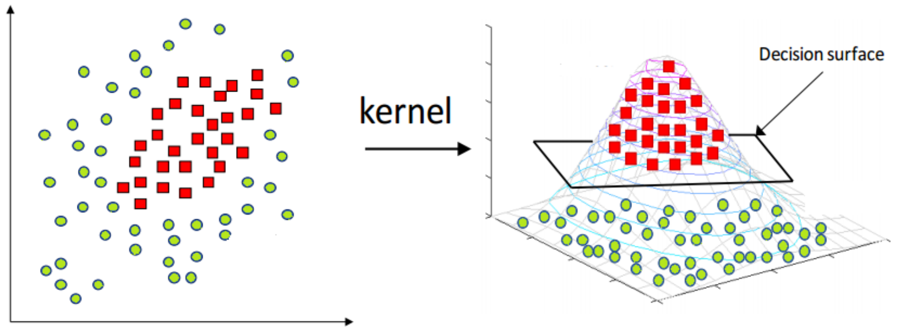


Figure 3.7: Non-linear transformation and finding of a separating hyperplane in the transformed space. Source: [36]

In summary, the key is to transform the data to a higher dimensional space so that linearity separation could be achieved to classify.

3.3.3 Decision Tree

Decision tree is a hierarchical data structure that represents data through a divide and conquer strategy ([37] pp 2) and it is used for both classification and regression tasks.

As depicted in table 3.8, the structure of a decision tree begins with a **root node** that lacks incoming branches. From the root node, **branches** extend to internal nodes, also referred to as **decision nodes**. These nodes evaluate the available features to create subsets of data that exhibit homogeneity, which are represented by leaf nodes or terminal nodes. Each **leaf node** corresponds to a distinct outcome within the dataset, encompassing all the possible results. The **maximum depth** is the maximum number of nodes between the root note and the lower end.

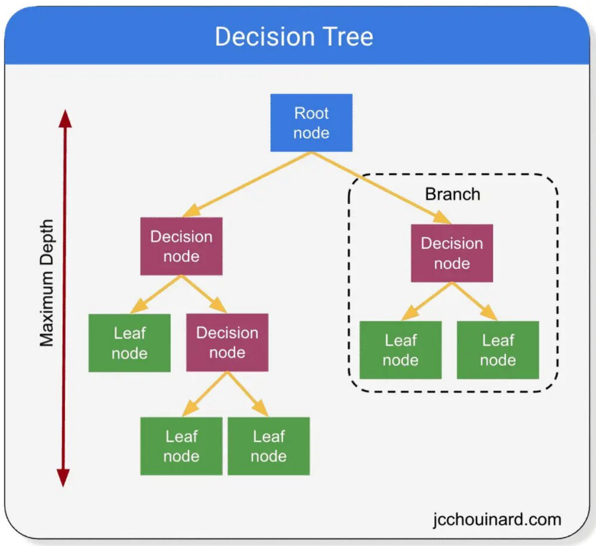


Figure 3.8: Decision Tree parts. Source: [39]

The tree structure is constructed in a recursive manner, where each node splits the data into subsets based on a selected feature. This process continues until a stopping criterion is met, such as reaching a maximum depth or when all the instances in a subset belong to the same class.

Once the decision tree is constructed, it can be used for classification or regression tasks. For classification, each instance follows the decision path from the root to a leaf node, and the class associated with that leaf node is assigned as the predicted class for the instance. For regression, the predicted value is determined by averaging the target values of the instances in the leaf node.

Decision trees have several advantages. They are easy to interpret and visualise, providing insights into the decision-making process. Decision trees can handle both numerical and categorical features and can capture non-linear relationships in the data. They can also handle missing values and outliers effectively.

However, decision trees are prone to overfitting, especially when the tree grows too deep. Techniques like pruning, setting a maximum depth, or using ensemble methods like random forests can help mitigate overfitting.

3.3.4 Gradient Descent

Gradient descent is an iterative algorithm usually applied with a view to minimising the loss or cost function.

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \, \boldsymbol{d}_k \quad \text{such that} \quad f(\boldsymbol{x}_{k+1}) \leq f(\boldsymbol{x}_k)$$

The parameter α_k is the **learning rate** and $\boldsymbol{d}_k$ is the **search direction**. Gradient descent is the method that takes the steepest descent:

$$\boldsymbol{d}_k = -\nabla f(\boldsymbol{x}_k)$$

Considering the Taylor expansion around $\boldsymbol{x}_k$:

$$f(\boldsymbol{x}_k + \alpha_k \, \boldsymbol{d}_k) = f(\boldsymbol{x}_k) + (\boldsymbol{x}_k + \alpha_k \, \boldsymbol{d}_k - \boldsymbol{x}_k)^\intercal \nabla f(\boldsymbol{x}_k) + O(\alpha^2)$$

The rate of change in direction $\boldsymbol{d}_k$ at $\boldsymbol{x}_k$ is the derivative of this function at $\alpha = 0$.

$$\frac{\mathrm{d}f(\boldsymbol{x}_k + \alpha_k \, \boldsymbol{d}_k)}{\mathrm{d}\alpha_k}\Big|_{\alpha=0} = \boldsymbol{d}_k^\intercal \nabla f(\boldsymbol{x}_k)$$

Then, we have a descent direction if $\boldsymbol{d}_k^\intercal \nabla f(\boldsymbol{x}_k) < 0$. Gradient descent satisfies this condition since $-\|\nabla f(\boldsymbol{x}_k)\|^2 < 0$.

It is important to consider a good learning rate parameter so that the algorithm do not progress very slow neither end up at a point with larger function value. Armijo and minimisation rules are often used to determine this parameter.

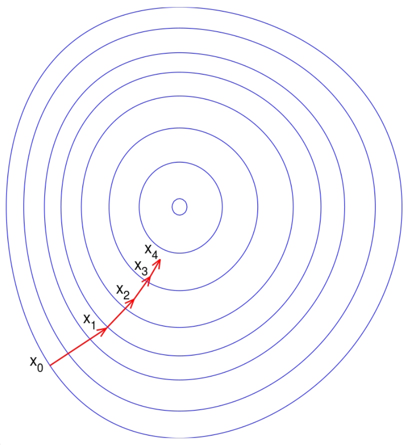


Figure 3.9: Gradient descent on a series of level sets. Source: [41]

Another relevant decision is the termination criterion, which can be set at a maximum number of iterations or when the value of the gradient reaches a threshold.

3.3.4.1 Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) is a variant of the traditional gradient descent algorithm that performs updates on the model parameters based on a random subset of the training data at each iteration. This implies a decrease in the use of the memory and an increase in computational speed.

Let $\{(\boldsymbol{x_i}, y_i)\}_{i=1}^n \in \mathcal{X} \times \mathcal{Y}$ and a set of functions $\mathcal{H} = \{h_{\boldsymbol{w}} : \boldsymbol{w} \in \mathbb{R}^d\}$, we want to minimize the empirical risk

$$\min_{\boldsymbol{w}\in\mathbb{R}^d} \frac{1}{n}\sum_{i=1}^{n} f_i(\boldsymbol{w}) = \min_{\boldsymbol{w}\in\mathbb{R}^d} \frac{1}{n}\sum_{i=1}^{n} L(h_{\boldsymbol{w}}(\boldsymbol{x}_i, y_i)) = \min_{\boldsymbol{w}\in\mathbb{R}^d} f(\boldsymbol{w}) \tag{3.3.18}$$

Note that $f(\boldsymbol{w})$ is an estimator of the generalisation risk $E_{\boldsymbol{\xi}}[f_{\boldsymbol{\xi}}(\boldsymbol{w})]$.

$$E_U[f_U(\boldsymbol{w})] = \sum_{i=1}^{n} P(U = i) f_i(\boldsymbol{w}) = \frac{1}{n}\sum_{i=1}^{n} f_i(\boldsymbol{w}) = f(\boldsymbol{w})$$

$f_U(\boldsymbol{w})$ is an unbiased estimator of $f(\boldsymbol{w})$.
Then, SGD procedure focus on finding an unbiased estimator of the gradient.

The simplest SGD Algorithm 1, **SGD with uniform sampling**, consits of picking uniformly at random one of the gradients $\nabla f_i(\boldsymbol{w})$.

```
Data: Data $\{(\boldsymbol{x_i}, y_i)\}_{i=1}^n$, $\boldsymbol{x_i} \in \mathbb{R}^d$
begin
    Initialise weights: $\boldsymbol{w}^0 \in \mathbb{R}^d$.;
    for $k = 1, 2, \ldots$ do
        Compute an unbiased estimator $\boldsymbol{g}^k$ of the gradient $\boldsymbol{w}^k$:
            $E[\ \boldsymbol{g}^k | \boldsymbol{w}^k\ ] = \nabla f(\boldsymbol{w}^k)$
        Take a step in the direction $\boldsymbol{g}^k$:
            $\boldsymbol{w}^{k+1} = \boldsymbol{w}^k - \eta_k \boldsymbol{g}^k$
    end
end
```

Algorithm 1: Stochastic Gradient Descent Algorithm. Source: [23]

This algorithm 1 can be extented to **mini-batch** sampling, which selects a small random subset of data to compute the gradient and update the model parameters. Then, we can formulate a new algorithm 2 for this particular case.

Following a similar reasoning as before, it is shown that ∇f_U is an unbiased estimator of ∇f.

$$E_U[\nabla f_U(\boldsymbol{w})] = \sum_{i=1}^{n} P(U = i) \nabla f_i(\boldsymbol{w}) = \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(\boldsymbol{w}) = \nabla f(\boldsymbol{w})$$

```
Data: Data $\{(\boldsymbol{x_i}, y_i)\}_{i=1}^{n}$, $\boldsymbol{x_i} \in \mathbb{R}^d$
begin
    Initialise weights: $\boldsymbol{w}^0 \in \mathbb{R}^d$.;
    for $k = 1, 2, \dots$ do
        Draw a random subset $I \in \{1, \dots, n\}$;
        Update
            $$\boldsymbol{w}^{k+1} = \boldsymbol{w}^k - \eta_k \frac{1}{|I|} \sum_{i \in I} \nabla f_i(\boldsymbol{w}^k)$$
    end
end
```

Algorithm 2: Stochastic Gradient Descent Algorithm. Mini-batch sampling. Source: [43]

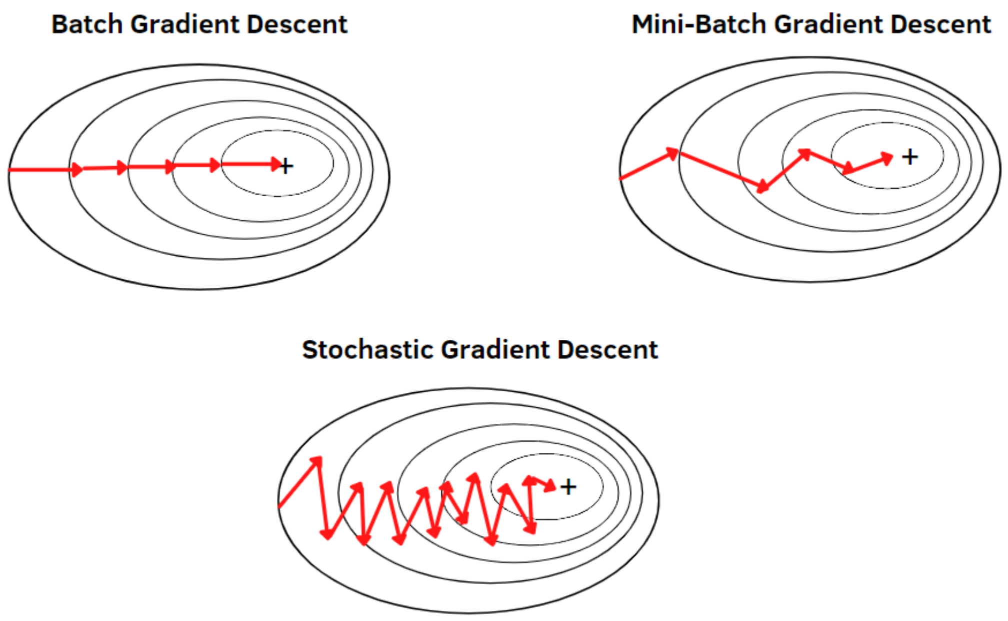


Figure 3.10: Gradient Descent and Stochastic Gradient Descent on a series of level sets comparison. Source: [47]

3.3.5 Boosting

Boosting (Freund and Shapire, 1997) is a machine learning emsemble technique[1] whose main idea is to iteratively train a sequence of weak learners, where each subsequent weak learner focuses on correcting the mistakes made by its predecessors. A **weak learner** refers to a model that performs slightly better than random guessing but is not highly accurate. It is typically characterised by having a limited predictive capacity and making relatively simple predictions. They are often combined in ensemble methods as boosting to create a stronger learner. The training set for constructing each classifier is obtained through weighted sampling, which allows instances that are misclassified in one iteration to be more likely to be selected in the following iteration.

The original gradient boosting procedure consits of training a classifier $C_1(\boldsymbol{x})$ with a subset E_1 from the training set. To improve performance, it would have to be forced to learn more in the difficult parts of the given feature space. To do so, we create a new training dataset E_2 by taking another subset of the training set with the peculiarity that half of them must have been misclassified by $C_1(\boldsymbol{x})$. Now, we use E_2 to train the $C_2(\boldsymbol{x})$ classifier. Finally, we take all the elements of the training set for which $C_1(\boldsymbol{x}) \neq C_2(\boldsymbol{x})$, and train on them the last model C_3. Finally, we construct the assembly the final classifier $C(\boldsymbol{x})$ using the majority voting criterion between C_1, C_2 and C_3.

The first successful boosting algorithm was the **Ada**ptive **Boos**ting, which is summarised in Algorithm 3. It is mainly presented for binary classification.

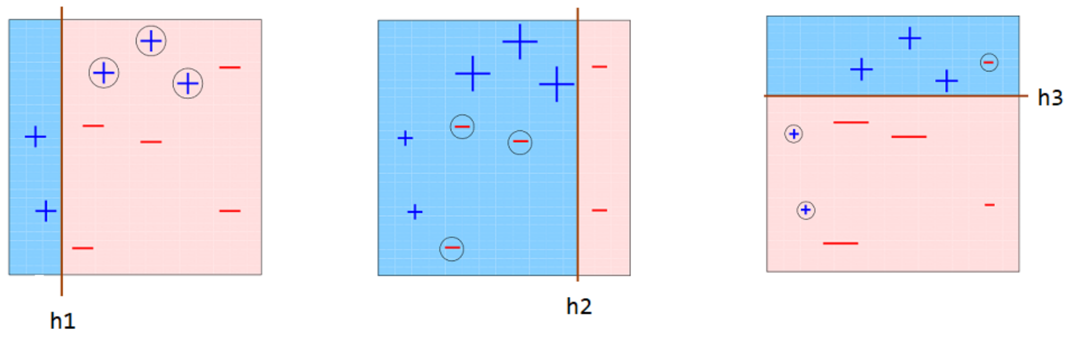


Figure 3.11: Boosting algorithm classifiers. Source: [50]

[1]Ensemble methods leverage the power of multiple learning algorithms to achieve enhanced predictive performance that surpasses what could be achieved by any individual algorithm alone.

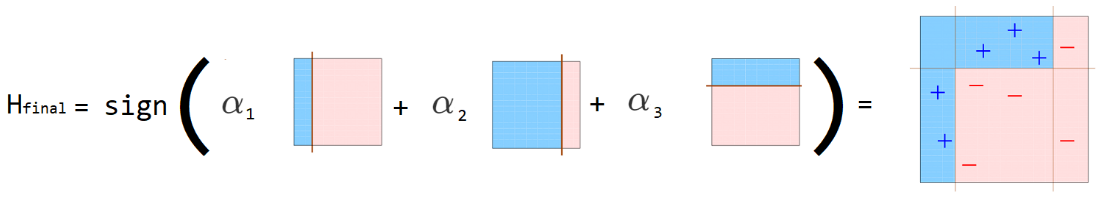


Figure 3.12: Boosting algorithm final classifier. Source: [50]

```
**Data:** Data $\{(\boldsymbol{x_i}, y_i)\}_{i=1}^{n}$, $\boldsymbol{x_i} \in \mathbb{R}^d$, $y_i \in \{-1, +1\}$, $M$ be the number of
          iterations
**Result:** A final classifier $C(\boldsymbol{x})$
**begin**
    Initialise weights for each observation: $w_i = \frac{1}{n}$, $i = 1, \ldots, n$;
    **for** $m = 1$ *to* $M$ **do**
        Obtain a Classifier $C_m(\boldsymbol{x})$, that minimises the weighted error $\varepsilon_m$
            $$\varepsilon_m = \sum_{i:y_i \neq C_m(\boldsymbol{x_i})} w_i$$
        If $\varepsilon_m \geq \frac{1}{2}$. END $(\alpha_m < 0)$;
        Calculate the weight of the $C_m$ classifier:
            $$\alpha_m = \frac{1}{2} \log \left( \frac{1 - \varepsilon_m}{\varepsilon_m} \right)$$
        Update the weights of the observations
            $$w_i := w_i e^{-\alpha_m C_m(\boldsymbol{x_i}) y_i} \quad \forall i = \{1, \ldots, n\}$$
        Rescale the weights: $w_i = \frac{w_i}{\sum_{j=1}^{n} w_j}$
    **end**
    Final classifier:
        $$C(\boldsymbol{x}) = sign(\sum_{m=1}^{M} \alpha_m C_m(\boldsymbol{x}))$$
**end**
```

Algorithm 3: AdaBoost Algorithm. Source: [48]

- A classifier as random guessing ($\varepsilon_m = 0.5$), would have weight 0.
- A perfect classifier ($\varepsilon_m = 0$) would have weight $+\infty$.
- A wrong classifier, always misclassifying, ($\varepsilon_m = 1$) would have weight $-\infty$.

The updated of the weights in Algorithm 3 can be interpreted as:

- If $y_i = C_m(\boldsymbol{x_i})$, the weight associated w_i decreases:

$$w_i := w_i e^{-\alpha_m} = w_i \sqrt{\frac{\varepsilon_m}{1-\varepsilon_m}}$$

- If $y_i \neq C_m(\boldsymbol{x_i})$, the weight associated w_i increases:

$$w_i := w_i e^{\alpha_m} = w_i \sqrt{\frac{1-\varepsilon_m}{\varepsilon_m}}$$

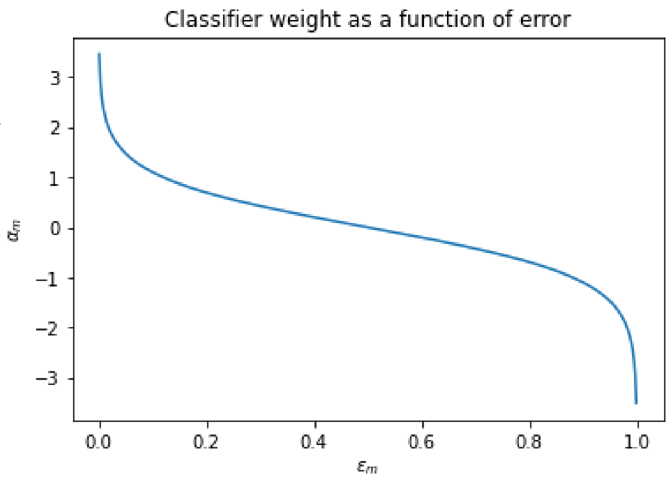


Figure 3.13: Classifier weight as a function of error. Source: Own elaboration extracted from [48]

It is observed that AdaBoost tends to overfit in the presence of noise/outliers. This can be attributed to the exponential loss function used. The costs of misclassification are exponential. A possible solution could be to reduce the cost of misclassification by using a different loss function such as the mean squared error or logit function.

3.3.5.1 Gradient Boosting

Gradient Boosting produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. It is an ensemble learning method that sequentially builds a series of models, each one correcting the mistakes of the previous models through a loss function and gradient descent. This allows to find better parameters in each iteration [42].

Gradient Boosting encompasses different boosting algorithms, including AdaBoost, as specific cases. It is a versatile technique that can be employed for both regression and classification tasks, with AdaBoost designed for classification.

Let $y \in \mathbb{R}$ be the target variable and $\boldsymbol{x}$ be the vector of explanatory variables with a joint probability distribution $P(x, y)$.

The objective is to find a function F^* that minimises the expected value of a given loss function $L(y, F(x))$, that is to say:

$$F^* = \arg\min_F E[L(y, F(\boldsymbol{x}))] \tag{3.3.19}$$

Gradient Boosting searchs for an approximation of (3.3.19) as:

$$\hat{F}(\boldsymbol{x}) = \sum_{i=1}^{M} \alpha_i h_i(\boldsymbol{x}) \tag{3.3.20}$$

where $h_i(\boldsymbol{x})$ are functions belonging to the class $\mathcal{H}$, which are usually trees.

To determine $\hat{F}$, a training sample $\{(\boldsymbol{x_1}, y_1), \ldots, (\boldsymbol{x_n}, y_n)\}$ is used.

A recursive *greedy*[2] procedure is used, starting from a constant function F_0:

$$F_0(\boldsymbol{x}) = \arg\min_{\alpha} \sum_{i=1}^{n} L(y_i, \alpha)$$

$$F_m(\boldsymbol{x}) = F_{m-1}(\boldsymbol{x}) + \arg\min_{h \in \mathcal{H}} \sum_{i=1}^{n} L(y_i, F_{m-1}(\boldsymbol{x_i}) + h(\boldsymbol{x_i}))$$

To solve the previous problem, a maximum descent algorithm is used (see section (3.3.4)), which moves in the direction of the gradient vector of the loss function L but in the opposite sense.

$$F_m(\boldsymbol{x}) = F_{m-1}(\boldsymbol{x}) - \alpha_m \sum_{i=1}^{n} \nabla_h L(y_i, F_{m-1}(\boldsymbol{x_i}))$$

$$\alpha_m = \arg\min_{\alpha} \sum_{i=1}^{n} L\left(y_i, F_{m-1}(\boldsymbol{x_i}) - \alpha \left[\frac{\partial L(y_i, F(\boldsymbol{x_i}))}{\partial F(\boldsymbol{x_i})} \right]_{F_m(\boldsymbol{x}) = F_{m-1}(\boldsymbol{x})} \right)$$

This procedure is summarised in Algorithm 4.

Gradient boosting can be prone to overfitting because of the model capacity, learning rate or the weak learner complexity. To mitigate overfitting, **regularisation** can be employed.

- Modify the model update rule.:

  $$F_m(\boldsymbol{x}) = F_{m-1}(\boldsymbol{x}) + \nu\ \alpha_m\ h_m(\boldsymbol{x})$$

  where ν is an adjustable parameter known as **learning rate**.

[2]A greedy algorithm makes locally optimal choices at each step with the aim of finding a global optimum. It makes the best choice at each stage without considering the larger context or potential consequences of that decision. The algorithm greedily selects the option that appears to be the most advantageous at the current moment, without reconsidering or revising previous choices. While this approach can often yield efficient solutions, it does not guarantee finding the optimal solution in all cases.

```
Data: Data $\{(\boldsymbol{x_i}, y_i)\}_{i=1}^n$, $\boldsymbol{x_i} \in \mathbb{R}^d$, $y_i \in \mathbb{R}$, $M$ be the number of iterations
Result: A final model $F(\boldsymbol{x})$
begin
    Initialise the model: $F_0(\boldsymbol{x}) = \arg\min_\alpha \sum_{i=1}^n L(y_i, \alpha)$;
    for $m = 1$ to $M$ do
        Calculate the pseudo-residual:
            $w_i = -\left[\frac{\partial L(y_i, F(\boldsymbol{x_i}))}{\partial F(\boldsymbol{x_i})}\right]_{F_m(\boldsymbol{x})=F_{m-1}(\boldsymbol{x})} \quad \forall i = 1, \ldots, n$
        Train a base classifier $h_m(\boldsymbol{x})$ to the pseudo-residuals by using as
          training sample $\{(\boldsymbol{x_i}, w_i)\}_{i=1}^n$.;
        Calculate the search step $\alpha$ by solving the following one-dimensional
          optimization problem:
            $\alpha_m = \arg\min_\alpha \sum_{i=1}^n L(y_i, F_{m-1}(\boldsymbol{x_i}) + \alpha h_m(\boldsymbol{x_i}))$
        Update the model:
            $F_m(\boldsymbol{x}) = F_{m-1}(\boldsymbol{x}) + \alpha_m h_m(\boldsymbol{x})$
    end
end
```

Algorithm 4: Gradient Boosting Algorithm. Source: [48]

The model's ability to generalise is greatly enhanced by using small values of ν ($\nu < 0.1$), despite the significant increase in computational effort associated with them.

- **Stochastic Gradient Boosting** introduces randomness into the algorithm to improve its performance and reduce overfitting. Unlike traditional gradient boosting, which uses the full training set to grow each weak learner, stochastic gradient boosting samples a subset of the training data at each iteration as it is mainly based on SGD (see section (3.3.4.1)). This reduces the computational cost as well as decreases overfitting.

  A specific fraction f of the learning sample is chosen, where values of f ranging from 0.5 to 0.8 yield satisfactory outcomes. Generally, a value of $f = 0.5$ is commonly employed.

3.3.5.1.1 XGBoost

XGBoost (eXtreme Gradient Boosting) is a high-performance, adaptable and portable distributed gradient boosting library that has been optimised for efficiency. It implements machine learning algorithms within the Gradient Boosting framework [51]. XGBoost is is an open-source software library for C++, Java, Python, R, Julia, Perl and Scala. Therefore, it has a large and active community, which has contributed to its continuous development and improvement.

XGBoost incorporates regularisation techniques to control overfitting and improve generalisation. It includes L_1 and L_2 regularisation terms which penalise the complexity of the model. Regularisation can be defined as an application of *Occam's Razor* (also known as the **principle of parsimony**) in that the goal is to choose a simpler hypothesis [1].

- **Lasso Regularisation** (L_1 Norm): adds a penalty term to the loss function that is proportional to the absolute value of the model's coefficients. It encourages some of the coefficients to become exactly zero, effectively performing feature selection. Due to this, the resulting models tend to be more interpretable, indicating the most important features in the model.

$$\boldsymbol{w_{opt}} = \arg \min_{\boldsymbol{w} \in \mathbb{R}^{d+1}} (L_{train}(h) + \lambda |\boldsymbol{w}|) \tag{3.3.21}$$

- **Ridge Regularisation** (L_2 Norm): adds a penalty term proportional to the squared value of the coefficients. By contrary, the coefficients tend to 0 , but they are not equal to 0. This can be advantageous when dealing with highly correlated features.

$$\boldsymbol{w_{opt}} = \arg \min_{\boldsymbol{w} \in \mathbb{R}^{d+1}} (L_{train}(h) + \lambda \boldsymbol{w}^\intercal \boldsymbol{w}) \tag{3.3.22}$$

λ is known as the **regularisation constant**, which is normally selected using the validation technique.

What made XGboost so popular was the remarkable success in various machine learning competitions [52] and real-world applications. Its excellent performance, scalability and feature-rich nature make it a popular choice for data scientists and practitioners working on predictive modelling tasks.

3.4 Generative Classifier

Generative Classifier learn a model of the join probability distribution $P(X, Y)$ on given input data X and their labels Y. Then, algorithms such as naive Bayes or Linear Discriminant Analysis are used to calculate conditional probability $P(Y|X)$ and predict the most likely label.

Generative classifiers offer more than just predicting class labels: they also have the capability to generate new samples based on the learned data distribution. This feature makes generative classifiers valuable for tasks such as data synthesis or generating new instances that exhibit similarities to the original data.

3.4.1 Naive Bayes

Naive Bayes classifier is a simple probabilistic classifier based on **Bayes's rule** and an assumption of *independence* between the features or attributes.

Theorem 3.4.1 (Baye's rule)**.**

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

where A and B are events with $P(B) \neq 0$.
*$P(A|B)$ is the **posterior probability**.*
*$P(B|A)$ is known as the **likelihood**.*
*$P(A)$ and $P(B)$ are **prior probabilities**.*

The aim is to model $P(Y = y|\boldsymbol{X} = \boldsymbol{x})$, which using Bayes' rule (3.4.1):

$$P(Y = y|\boldsymbol{X} = \boldsymbol{x}) = \frac{P(\boldsymbol{X} = \boldsymbol{x}|Y = y)P(Y = y)}{P(\boldsymbol{X} = \boldsymbol{x})}$$

which under Naive Bayes assumption,

$$= \frac{\prod_{j=1}^{m} P(X_j = x_j|Y = y)P(Y = y)}{P(\boldsymbol{X} = \boldsymbol{x})} \propto \prod_{j=1}^{m} P(X_j = x_j|Y = y)P(Y = y)$$

We want the maximum a-posteori to know how to classify, that is to say:

$$h_{Bayes}(\boldsymbol{x}) = \arg\max_{y\in\{0,1\}} P(Y = y) \prod_{j=1}^{m} P(X_j = x_j|Y = y) \qquad (3.4.1)$$

The key is estimating the parameters of Naive Bayes: the prior class $P(Y = y)$ and the likelihood of each feature given the class $P(X_j = x_j|Y = y)$.

The maximum likelihood of these estimates are:

$$P(Y = y_i) = \frac{count(y_i)}{n}$$

$$P(X_j = k|Y = y_i) = \frac{count(X_{ji} = k)}{count(y_i)}$$

where $count(X_{ji} = k)$ refers to the number of instances that the attribute X_j takes the value k and belong to the class y_i. The problem is that this goes to 0 if it does not exist such an instance. In this case, it would assign a probability 0, ignoring the other attributes. To solve this issue, **Laplace smoothing** adds a pseudocount parameter $\alpha > 0$:

$$P(X_j = k|Y = y_i) = \frac{count(X_{ji} = k) + \alpha}{count(y_i) + \alpha \ values(X_j)}$$

where $values(X_j)$ is the number of categories the attribute X_j has.

An alternative is to assume that continues values are distributed according to a Gaussian distribution, and therefore, the probability can be computed plugging k into normal density function:

$$P(X_j = k|Y = y_i) = \frac{1}{\sqrt{2\pi}\sigma_{ji}} e^{\frac{(k-\mu_{ji})^2}{2\sigma_{ji}^2}} \tag{3.4.2}$$

where μ_{ji} is the mean of the values in class i of attribute X_j and σ_{ji} refers to the variance.

3.4.2 Linear Discriminant Analysis

Linear Discriminant Analysis (LDA) was developed by Ronald Fisher to classify data. The key idea is to project data such that separation between classes is achieved.

Assume binary classification with labels 1 and 2 and that there are p predictor variables for each group:

$$\boldsymbol{x}_{ji} \in \mathbb{R}^p \quad i \in \{1, \ldots, n_j\}, \ j \in \{1, 2\}$$

Then, the aim is to find a linear combination which maximises the ratio of variability between groups relative to the total variability.

Let $\boldsymbol{v} \in \mathbb{R}^p$. Note that $\boldsymbol{v}^\intercal \boldsymbol{x}_{ji}$ is the projection of $\boldsymbol{x}_{ji}$ into a one dimensional subspace, a line in direction $\boldsymbol{v}$.

Now, we define some measure to proceed with the reasoning. Let $\overline{\boldsymbol{x}}_{\boldsymbol{j}}$ be the mean of class j and $\overline{z}_j$ be the projected mean of class j. The relationship between them is:

$$\overline{z}_j = \frac{1}{n_j}\sum_{i=1}^{n_j} \boldsymbol{v}^\intercal \boldsymbol{x}_{\boldsymbol{ji}} = \boldsymbol{v}^\intercal \left(\frac{1}{n_j}\sum_{i=1}^{n_j} \boldsymbol{x}_{\boldsymbol{ji}}\right) = \boldsymbol{v}^\intercal \overline{\boldsymbol{x}}_{\boldsymbol{j}}$$

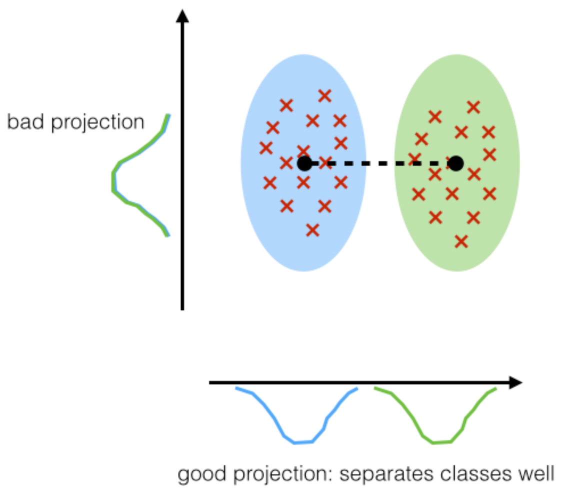


Figure 3.14: LDA maximising the component axes for class-separation. Source: [56]

One could think that $|z_1 - z_2|$ is a good measure of separation, but the problem relies that it does not consider the variance of the classes. This is the reason why we consider the maximisation relative to the total variability.

To measure the spread around the mean, we consider the **scatter** or **pooled covariance matrix**. The scatter for projected samples of group j is:

$$s_{z_j}^2 = \sum_{i=1}^{n_j} (z_{ji} - \overline{z}_j)^2$$

Small $s_{z_j}^2$ means that projections of group j are closed to their projected mean of the group. In the same way, we can also define the scatter matrices S_j for group j before the projection and find the relationship with the projected one:

$$S_j^2 = \sum_{i=1}^{n_j} (\boldsymbol{x_{ji}} - \overline{\boldsymbol{x}}_{\boldsymbol{j}})(\boldsymbol{x_{ji}} - \overline{\boldsymbol{x}}_{\boldsymbol{j}})^\intercal$$

$$\begin{aligned} s_{z_j}^2 &= \sum_{i=1}^{n_j} (z_{ji} - \overline{z}_j)^2 = \sum_{i=1}^{n_j} (\boldsymbol{v}^\intercal(\boldsymbol{x_{ji}} - \overline{\boldsymbol{x}}_{\boldsymbol{j}}))^\intercal(\boldsymbol{v}^\intercal(\boldsymbol{x_{ji}} - \overline{\boldsymbol{x}}_{\boldsymbol{j}})) \\ &= \sum_{i=1}^{n_j} ((\boldsymbol{x_{ji}} - \overline{\boldsymbol{x}}_{\boldsymbol{j}})^\intercal \boldsymbol{v})(\boldsymbol{v}^\intercal(\boldsymbol{x_{ji}} - \overline{\boldsymbol{x}}_{\boldsymbol{j}})) \\ &= \sum_{i=1}^{n_j} \boldsymbol{v}^\intercal(\boldsymbol{x_{ji}} - \overline{\boldsymbol{x}}_{\boldsymbol{j}})(\boldsymbol{x_{ji}} - \overline{\boldsymbol{x}}_{\boldsymbol{j}})^\intercal)\boldsymbol{v} = \boldsymbol{v}^\intercal S_j^2 \boldsymbol{v} \end{aligned}$$

The **within-class scatter matrix** is defined as:

$$S_W^2 = \frac{1}{N-K} \sum_{j=1}^{K} S_j^2$$

where $N = \sum_{j=1}^{K} n_j$ and K is the total number of groups that exists. For binary classification, it would be:

$$S_W^2 = \frac{1}{n_1 + n_2 - 2}(S_1^2 + S_2^2)$$

Then, the relationship with projection is:

$$s_z^2 = \frac{1}{N-K}\sum_{j=1}^{K} s_{z_j}^2 = \frac{1}{N-K}\sum_{j=1}^{K} \boldsymbol{v}^\intercal S_j^2 \boldsymbol{v} = \boldsymbol{v}^\intercal S_W^2 \boldsymbol{v}$$

In the same way, the **between-class** scatter matrix is:

$$S_B^2 = (\overline{\boldsymbol{x}}_{\boldsymbol{1}} - \overline{\boldsymbol{x}}_{\boldsymbol{2}})(\overline{\boldsymbol{x}}_{\boldsymbol{1}} - \overline{\boldsymbol{x}}_{\boldsymbol{2}})^\intercal$$

which measures the separation bwteen the means of two classes before projection.

The separation between the projected means can be rewritten as:

$$(\overline{z}_1 - \overline{z}_2)^2 = (\boldsymbol{v}^\intercal \overline{\boldsymbol{x}}_{\boldsymbol{1}} - \boldsymbol{v}^\intercal \overline{\boldsymbol{x}}_{\boldsymbol{2}})^2 = \boldsymbol{v}^\intercal (\overline{\boldsymbol{x}}_{\boldsymbol{1}} - \overline{\boldsymbol{x}}_{\boldsymbol{2}})(\overline{\boldsymbol{x}}_{\boldsymbol{1}} - \overline{\boldsymbol{x}}_{\boldsymbol{2}})^\intercal \boldsymbol{v} = \boldsymbol{v}^\intercal S_B^2 \boldsymbol{v}$$

Thus, our objective is to find $\boldsymbol{v} \in \mathbb{R}^p$ such that $z_{ji} = \boldsymbol{v}^\intercal \boldsymbol{x}_{ji} \in \mathbb{R}$ maximises the separation between projected classes relative to the the variance within each projected class:

$$\max_{\boldsymbol{v}\in\mathbb{R}^p} \frac{(\overline{z}_1 - \overline{z}_2)^2}{s_z^2} = \max_{\boldsymbol{v}\in\mathbb{R}^p} \frac{(\boldsymbol{v}^\intercal(\overline{\boldsymbol{x}}_{\boldsymbol{1}} - \overline{\boldsymbol{x}}_{\boldsymbol{2}}))^2}{\boldsymbol{v}^\intercal S_W^2 \boldsymbol{v}} = \max_{\boldsymbol{v}\in\mathbb{R}^p} \frac{\boldsymbol{v}^\intercal S_B^2 \boldsymbol{v}}{\boldsymbol{v}^\intercal S_W^2 \boldsymbol{v}} \tag{3.4.3}$$

Note that we need to assume that S_W^2 is symmetric positive definite or we cannot apply LDA. This also implies that we need to have enough data points.

The rule for classification consists of assigning a new observation $\boldsymbol{x}^*$ to class 1 when its projection is closer to the projected mean of class 1 $\overline{z}_1$ than to $\overline{z}_2$:

$$\begin{aligned} & (\boldsymbol{v}^\intercal \boldsymbol{x}^* - \overline{z}_1)^2 < (\boldsymbol{v}^\intercal \boldsymbol{x}^* - \overline{z}_2)^2 \\ \Leftrightarrow\ & \overline{z}_1^2 - 2\overline{z}_1 \boldsymbol{v}^\intercal \boldsymbol{x}^* < \overline{z}_2^2 - 2\overline{z}_2 \boldsymbol{v}^\intercal \boldsymbol{x}^* \\ \Leftrightarrow\ & \overline{z}_1^2 - \overline{z}_2^2 < 2(\overline{z}_1 - \overline{z}_2)\boldsymbol{v}^\intercal \boldsymbol{x}^* \end{aligned} \tag{3.4.4}$$

Proposition 3.4.1. *For any symmetric positive definite $p \times p$ matrix S and non-zero $\boldsymbol{u} \in \mathbb{R}^p$. Then, for any non-zero $\boldsymbol{v} \in \mathbb{R}^p$*

$$\max_{\boldsymbol{v}\neq\boldsymbol{0}} \frac{(\boldsymbol{v}^\intercal \boldsymbol{u})^2}{\boldsymbol{v}^\intercal S \boldsymbol{v}} = \boldsymbol{u}^\intercal S^{-1} \boldsymbol{u}$$

and the maximum is achieved for $\boldsymbol{v} = S^{-1}\boldsymbol{u}$

Proof.

$$(\boldsymbol{v}^\intercal \boldsymbol{u})^2 = (\boldsymbol{v}^\intercal S^{\frac{1}{2}} S^{-\frac{1}{2}} \boldsymbol{u})^2 = ((S^{\frac{1}{2}}\boldsymbol{v})^\intercal (S^{-\frac{1}{2}}\boldsymbol{u}))^2 \overset{\text{C-S}}{\leq} (\boldsymbol{v}^\intercal S \boldsymbol{v})(\boldsymbol{u}^\intercal S^{-1} \boldsymbol{u})$$

By Cauchy-Schwartz (C-S), we also know that the equality is attained for $\boldsymbol{v} = S^{-1}\boldsymbol{u}$. □

Theorem 3.4.2 (Fisher's Linear Discriminant Analysis)**.** *Assuming* $\boldsymbol{x}_1 \neq \boldsymbol{x}_2$ *and* S_W^2 *is symmetric positive definite. The maximiser of (3.4.3) is*

$$\boldsymbol{v} := S_W^{-1}(\overline{\boldsymbol{x}}_{\boldsymbol{1}} - \overline{\boldsymbol{x}}_{\boldsymbol{2}})$$

Proof. The proof relies on applying the proposition (3.4.1) setting $\boldsymbol{u} = \overline{\boldsymbol{x}}_1 - \overline{\boldsymbol{x}}_2$ so that the maximiser is $\boldsymbol{v} = S_W^{-1}(\overline{\boldsymbol{x}}_{\boldsymbol{1}} - \overline{\boldsymbol{x}}_{\boldsymbol{2}})$ ☐

Since S_W is positive definite, we can simplify the classification rule (3.4.4):

$$\overline{z}_1 - \overline{z}_2 = \boldsymbol{v}^\intercal(\overline{\boldsymbol{x}}_{\boldsymbol{1}} - \overline{\boldsymbol{x}}_{\boldsymbol{2}}) = (\overline{\boldsymbol{x}}_{\boldsymbol{1}} - \overline{\boldsymbol{x}}_{\boldsymbol{2}})^\intercal S_W^{-1}(\overline{\boldsymbol{x}}_{\boldsymbol{1}} - \overline{\boldsymbol{x}}_{\boldsymbol{2}}) > 0$$

Then, the classification rule is simplified to

$$\text{Assign to class 1 if} \quad \frac{1}{2}(\overline{z}_1 + \overline{z}_2) < \boldsymbol{v}^\intercal \boldsymbol{x}^*$$

Chapter 4

Applications

Communication is an essential aspect of human life, as it allows us to convey information, express emotions and ideas. Language serves as a fundamental tool for communication, providing a common ground for understanding between individuals. However, when it comes to interacting with computer systems, the challenge of language comprehension arises.

Fortunately, Artificial Intelligence (AI) provides a solution, specifically in the form of Natural Language Processing (NLP), which empowers computer systems to understand and interpret this information. NLP enables systems to grasp not only the literal meaning of text but also to recognise sentiments, tones, opinions and other elements that contribute to meaningful conversations, reducing the gap between humans and machines.

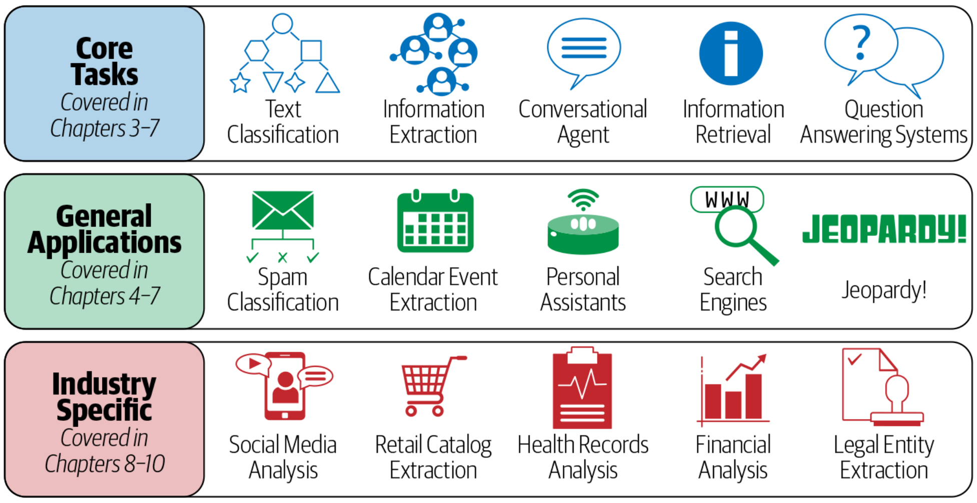


Figure 4.1: NLP taks and applications. Source: [57].

This chapter aims to provide a comprehensive overview of the applications of NLP. By leveraging NLP techniques, organizations can enhance communication, improve information retrieval, conduct sentiment analysis and augment decision-making processes across diverse domains. As NLP continues to advance, it promises to reshape industries, transform customer experiences and empower individuals and organizations with valuable insights derived from the vast amount of textual data available today. We will discuss the main core applications NLP has, but these have a wide variety of purposes through different industries. Some of these applications are shown in figure 4.2.

While the applications of NLP are vast and promising, the subsection (4.7) also acknowledges the challenges and limitations associated with its implementation. Ethical considerations such as privacy, bias and fairness in language processing systems need to be carefully addressed. Additionally, the scalability, accuracy and robustness of NLP algorithms pose ongoing research and development challenges.

4.1 Text Classification

Text Classification (TC) is a core task of text-mining whose main goal is to assign documents to one or even more categories in some cases. The most common approach to building classifiers is through supervised machine learning whereby classification rules are learned from examples [1].

Text classification involves several steps, including document representation, feature selection, the application of a machine learning classifier and the evaluation of classifier performance. Feature selection can utilise various representations, such as morphological, lexical, syntactic or semantic features discussed earlier.

Given a set of n labelled documents, $\mathcal{D}_{labelled}$, the initial step is to construct representations of these documents in a feature space. The typical approach is to employ a bag-of-words technique with TF-IDF (Term Frequency-Inverse Document Frequency) to create document vectors, denoted as $\boldsymbol{x_i}$, along with their corresponding labelled categories, y_i.

Several supervised machine models are trained and evaluated their performance on unseen data following the procedure indicated in section (3.2.4). Some of these machine learning algorithms used have been explained in section (3), but there are also many others such as K-nearest neighbour, random forest and neural networks.

This process is summarised in Algorithm 5.

According to the number of labels, we can distinguish:

- **Binary classification**: there are only two distinct categories or labels taken into account, where each element can be assigned exclusively to one category or potentially belong to both categories simultaneously (**overlapping** or **fuzzy binary classification**).
- **Multiple classification**: each item is classified into one of at least three categories. It may be decomposed into binary classification tasks as many as

```
Data: A set of documents $\mathcal{D}_{labelled}$
Result: A trained model $h(x)$
begin
    Preprocess documents;
    Create document representations $\boldsymbol{x_i}$;
    Split into train, validation, test sets;
    for $\boldsymbol{x_i} \in \boldsymbol{X}$ do
        Train machine learning classifier model on train set;
        Tune model on validation set;
    end
    Evaluate tuned model on test set;
end
```

Algorithm 5: Text Classification Pipeline. Source: [1]

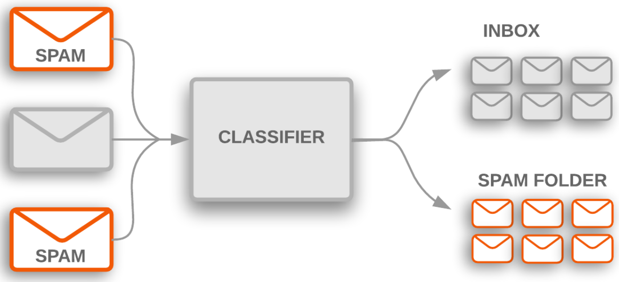


Figure 4.2: Identification of spam emails by text classification. Source: [60].

categories [59].

According to the number of labels an item can be assigned, there are two types of classification, which have been mentioned previously:

- **Hard classification**. There is no overlapping between the categories, meaning that each element belongs exclusively to one category and does not share any characteristics with other categories.

- **Soft classification**. There is overlapping between the categories, that is to say, at least one item is classified into more than one category.

Depending on whether any nested category is allowed, we can distinguised:

- **Flat classification**: There is no inherent hierarchy among the potential categories to which the data can be assigned.

- **Hierarchical classification**. Documents are organised in a hierarchical structure with multiple levels of categories. Due to nesting, the resulting final classification will depend on the path and decisions made beforehand, which makes this process more challenging.

4.1.1 Sentiment Analysis

Textual data presents an opportunity to distinguish between objective and subjective documents. Objective documents primarily focus on describing specific facts or situations, while subjective documents are driven by the feelings, opinions and emotions expressed by the author regarding a particular issue.

The presence of subjective documents sparked the idea and subsequent necessity to analyse the population's reactions to a certain entity, subject or factor. This analysis aims to gather valuable insights and inform decision-making processes. By understanding the sentiments and emotions expressed by the population, it becomes possible to make informed and appropriate decisions.

Sentiment analysis involves evaluating language to determine if expressions are favorable, unfavorable or neutral, and to what degree. For this, both objective and subjective aspects are considered, which include the impact emotions have on our communication.

Sentiment analysis is heavily influenced by the predefined degrees or categories. As a result, subjectivity and the multitude of connotations associated with these categories make this task highly complex. For instance, the **Mehrabian and Rusell model** decomposes human emotions into 3 dimensions, which are part of the PAD model:

- *Pleasure-Displeasure Scale*: refers to the level of enjoyment or positive experience. High pleasure signifies a state of positive emotions, contentment or joy, while low pleasure indicates a lack of enjoyment or negative emotions.
- *Arousal-Nonarousal Scale*: represents the intensity of an emotional response or the degree of alertness and excitement one feels. High arousal is associated with heightened physiological responses, increased heart rate and a sense of being energised or activated, while low arousal indicates a state of calmness or relaxation.
- *Dominance-Submissiveness Scale*: measures the dominion of an emotion over others. High dominance indicates a feeling of being in charge, while low dominance suggests a sense of being vulnerable or controlled by others or circumstances.

The Affective Norms for English Words (ANEW) by Bradley and Lang are a commonly used set of 1,034 words characterised on these dimensions.

By contrast, **Robert Plutchik** designed a wheel of emotions (see figure 4.3) considering 8 primary bipolar emotions: joy versus sadness; anger versus fear; trust versus disgust; and surprise versus anticipation. Like colors, primary emotions can be expressed at different intensities and can mix with one another to form different emotions [62]. The Plutchik Wheel also includes secondary emotions, which are formed by combining the primary emotions. For example, blending joy and trust can result in the secondary emotion of love. Similarly, blending fear and surprise

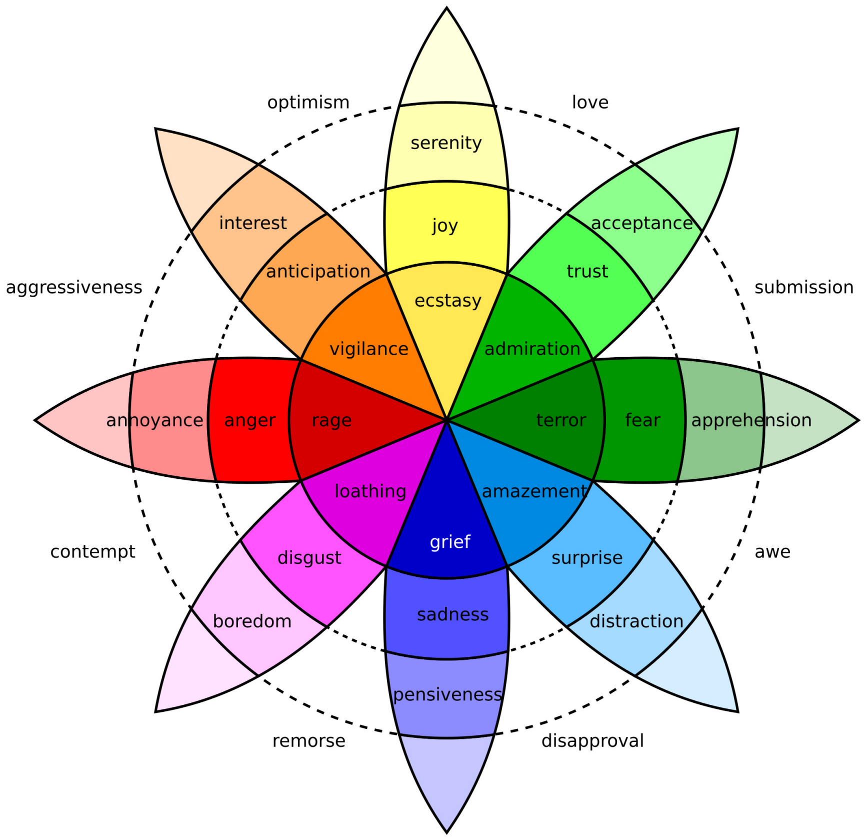


Figure 4.3: Plutchik's wheel of emotions. Source: [62]

can lead to the secondary emotion of awe.

Sentiment analysis procedure is mainly based on a set of words that describe emotional states and vectorise them with the dimensional values of the emotional state model. When analysing a document, the frequency of these words associated with specific emotional states are taken into account. By summing up the scores of these words, an overall sentiment score is obtained, indicating the sentiment expressed in the document.

4.2 Text Clustering

Text clustering is the most common application of unsupervised learning and it is used to group similar textual documents together based on their content or other features.

The procedure involves a data preprocessing step and text is vectorised as explained in section (2). Then, a selected clustering algorithm is applied to group similar documents based on similarity or distance metrics. These algorithms can be: Latent Dirichlet Allocation (LDA), k-means and hierarchical clustering.

In case of using k-means clustering, it is important to determine the vale of k, that is to say, the number of clusters exists within a corpus. For this purpose, the elbow method is frequently used. In addition to this, the metric used is another decision to make. Euclidean, Manhattan or Chebyshev distance are some posibilities together with cosine similarity.

This methodology is summarised in Algorithm 6.

```
Data: A set of documents $\mathcal{D}_{unlabelled}$
Result: $k$ text clusters
begin
    Preprocess documents;
    Create document representations $\boldsymbol{x_i}$;
    for values of k do
        Apply $k$-means algorithm;
    end
    Choose best $k$-value;
end
```

Algorithm 6: k-means text clustering pipeline. Source: [1]

4.2.1 Topic Modelling

Topic modelling is a technique used to uncover latent topics or themes present in a collection of documents. Intuitively, given a document about a specific topic, one could expect particular words to appear in that document more or less frequently. In the age of information, it is very helpful for organising, summarising and understanding large collections of text.

One of the most commonly used algorithms for topic modelling is **Latent Dirichlet Allocation** (LDA), which assumes two principles:

- Every document is a mixture of topics.
- Every topic is a mixture of words. Note that some words may be shared between topics due to polisemy.

LDA is a mathematical method for estimating both of these at the same time: finding the mixture of words that is associated with each topic, while also determining the mixture of topics that describes each document (see [21], Chapter 6). For this purpose, LDA decomposes a document-term matrix into a lower-order document-topic matrix and topic-word matrix. The reason why LDA does better disambiguation of words and identification of topics is a stochastic, generative model approach, which assumes topics to have a sparse Dirichlet prior.

A k-dimensional Dirichlet random variable $\boldsymbol{\theta}$ has the following probability density:

$$P(\boldsymbol{\theta}|\boldsymbol{\alpha}) = \frac{\Gamma(\sum_{i=1}^{k}\alpha_i)}{\prod_{i=1}^{k}\Gamma(\alpha_i)}\theta_1^{\alpha_1-1}\cdots\theta_k^{\alpha_k-1}$$

where $\boldsymbol{\alpha} \geq \mathbf{0}$.

Another older technique is **Latent Semantic Analysis** (LSA). It is a mathematical approach that applies **Singular Value Decomposition** (SVD) to a matrix representing the term-document relationships to uncover the underlying semantic structure. This dimensionality reduction helps to identify patterns, and similarity is measured by the cosine distance. It is vital that SVD transforms the document-term matrix into a document-topic matrix and a topic-word matrix.

4.3 Machine Translation

Machine translation (MT) refers to the process of translating text from a source language to a different target language [1]. It aims to bridge language barriers and facilitate communication between different language speakers. Machine translation focus on preserving both the meaning of text and the natural sounding.

The evaluation of Machine Translation is commonly conducted using the BLEU (BiLingual Evaluation Understudy) score. BLEU is a metric designed to automatically assess the quality of machine-translated text. The score ranges from 0 to 1, with a higher score indicating a better quality of machine translation. The BLEU score is obtained by comparing machine-translated output against human-created translations, enabling researchers and practitioners to gauge the accuracy and fluency of the machine translation system.

As mentioned in section (1.2), the need for machine translation played a significant role during World War II, which made NLP research arise. We can also distinguish three periods, according to the history of NLP:

- Rule-based Machine Translation (RBMT): 1970-1990
- Statistical Machine Translation (SMT): 1990-2010
- Neural Machine Translation (NMT): 2010-Present

4.3.1 Rule-based Machine Translation

Rule-based Machine Translation stands out from other methods as it incorporates extensive linguistic information about both the source and target languages. One of the first types RBMT was **dictionary-based**, which is a direct translation of each word by a bilingual dictionary. This method could be improved by translating whole sentences instead of word-by-word to take into account the syntactic or semantic context.

However, RBMT was improved by leveraging morphological and syntactic rules, as well as semantic analysis, in order to achieve accurate translations. The fundamental approach of RBMT involves establishing a connection between the structure of the input sentence and the structure of the output sentence. This type of translation is used mostly in the creation of dictionaries and grammar programs

To accomplish this, RBMT utilises several components, including a parser and analyser for the source language. These components analyse the grammatical structure of the input sentence, extracting relevant linguistic information. Additionally, a generator for the target language is employed to construct the desired output sentence based on the analysed structure. The transfer lexicon plays a crucial role in the translation process, serving as a resource that maps words and phrases from the source language to their appropriate translations in the target language.

4.3.2 Statistical Machine Translation

Statistical Machine Translation (SMT) is characterised by its adoption of a probabilistic approach to facilitate the mapping from one language (source) to another (target). The underlying assumption in SMT is that each word in the target language corresponds to a translation of the source language words, with certain probabilities assigned to these mappings. By selecting the words with the highest probabilities, the system aims to generate the most optimal translation.

Word alignment plays a crucial role in these statistical models and refers to the task of mapping words between a source language sentence and its corresponding translation in the target language.

The model can be formulated as an optimisation problem [65] applying Bayes's Rule (theorem (3.4.1)):

$$\hat{Y} = \arg\max_{Y} P(Y \mid X) = \arg\max_{Y} P(X \mid Y)P(Y) \tag{4.3.1}$$

where X refers to the source language and Y to the target language.

This decomposition allows to split the problem into subproblems:

- $P(X \mid Y)$ is the **translation model** (faithfulness), which defines a range of potential translations for a given target sentence. Additionally, the model assigns probabilities to these translations, reflecting their relative accuracy or correctness compared to each other.
- $P(Y)$ is the **language model** (fluency), which reflects the model's estimation of how well each sentence captures the characteristics and fluency of the target language.
- The argmax search process through the space of possible target translations is known as **decoding**. Decoding for SMT is NP-hard[1].

[1]NP-hard (nondeterministic polynomial-time hard) is a classification used in computational complexity theory to describe problems that are at least as hard as the hardest problems in the complexity class NP (nondeterministic polynomial-time). A problem is considered NP-hard if any problem in NP can be reduced to it in polynomial time.

Online translation tools as Google Translate was primarily driven by the statistical method, but now, it has transitioned to neural machine translation (see figure 4.4).

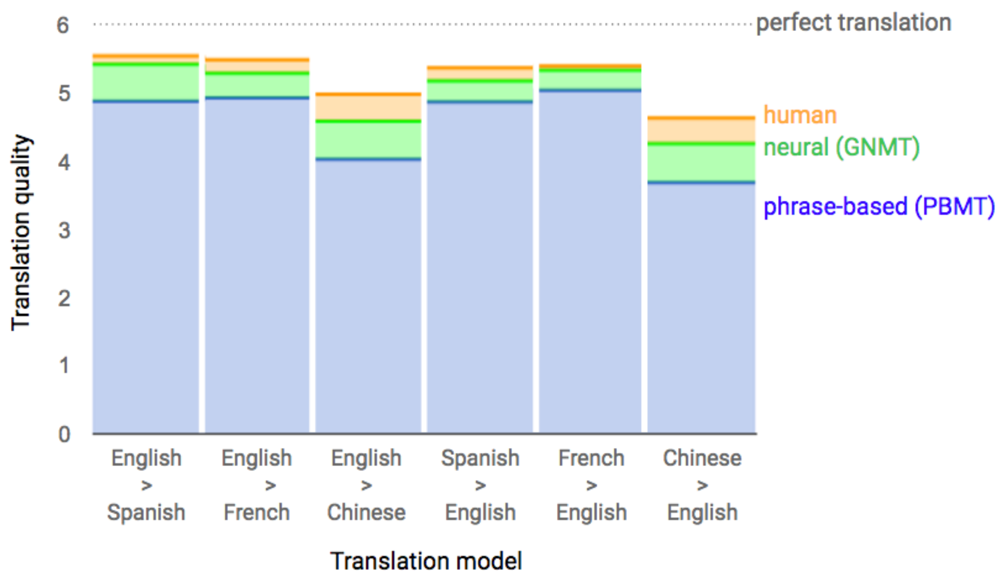


Figure 4.4: Human raters assess the quality of translations for a given source sentence by comparing them. The raters assign scores ranging from 0 (nonsense) to 6 (perfect translation). Source: [66]

4.3.3 Neural Machine Translation

Despite these challenges, machine translation has made significant progress in recent years, driven by advancements in deep learning and neural network encoder-decoder architectures. Thus, **Neural Machine Translation** (NMT) employs artificial neural networks to predict the probability of a sequence of words. Unlike traditional methods that focus on individual words or phrases, NMT models consider entire sentences as a whole in a unified framework. By utilising neural networks, NMT systems aim to capture the context, grammar and semantic information of the source language to generate accurate and fluent translations in the target language. This integrated modelling approach enables NMT to produce more coherent and contextually appropriate translations compared to previous machine translation techniques.

Analysing sentence-level translation, NMT model can be viewed as a **sequence-to-sequence** model. Let $\boldsymbol{x} = \{x_1, \ldots, x_S\}$ be a source sentence and $\boldsymbol{y} = \{y_1, \ldots, y_T\}$ be a target sentence. By using the chain rule, the conditional distribution can be factorised as [67]:

$$P_{\boldsymbol{w}}(\boldsymbol{y} = \boldsymbol{y} | \boldsymbol{x} = \boldsymbol{x}) = \prod_{t=1}^{T} P_{\boldsymbol{w}}(y_t | y_0, \ldots, y_{t-1}, x_1, \ldots, x_S) \tag{4.3.2}$$

where each translated word y_t depends on previous translated words $y_1, \ldots, y_{t-1}$ from the source sentence $\boldsymbol{x}$ as well as the parameters $\boldsymbol{w}$ of the model.

As mentioned before, the encoder-decoder framework consits of four fundamental elements:

- Embedding layers: transform input data, such as words, into continuous vector representations called embeddings, which can be effectively processed by the neural networks.
- Encoder networks: takes the source embeddings and encodes it into hidden continuous representations or states. The encoder typically consists of recurrent neural network (RNN) layers:

  $$\boldsymbol{h}_t = RNN_{ENC}(\boldsymbol{x}_t, \boldsymbol{h}_{t-1})$$

  which is iteratively applied until we obtain the final stage $\boldsymbol{h}_S$.
- Decoder networks: takes the encoded representation (final stage $\boldsymbol{h}_S$) from the encoder and generates the translated target language sentence. It operates in a sequential manner, generating one word at a time based on the previously generated words and the encoded source sentence.
- Classification layer: predicts the distribution of target tokens. It is usually a linear layer with softmax activation function to ensure that we obtain a probability.

  $$softmax(\boldsymbol{z}) = \frac{e^{\boldsymbol{z}}}{\sum_{i=1}^{V} z_i}$$

  where V is the size of the vocabulary of the target language.

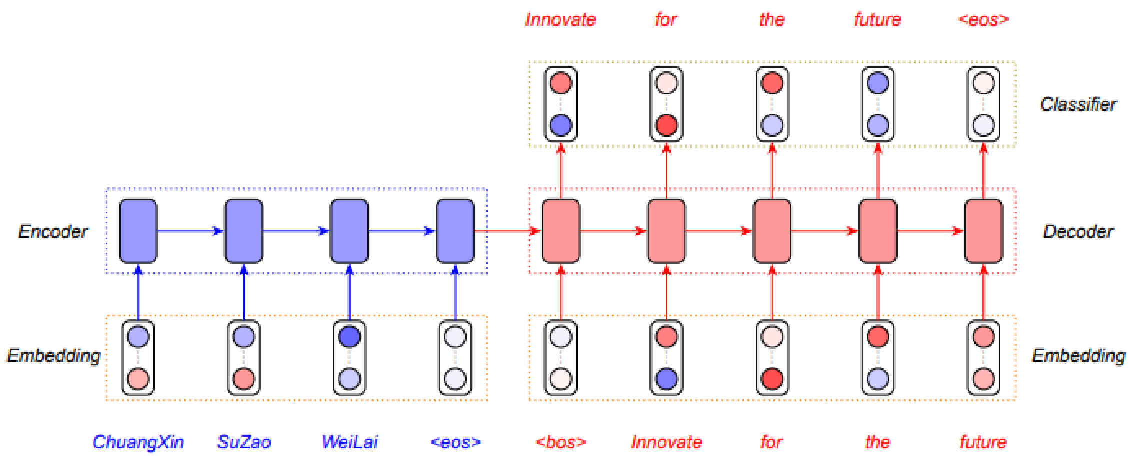


Figure 4.5: An overview of the NMT architecture (encoder-decoder framework). Source: [67]

An example of NMT is DeepL Translator, which is mainly based on Convolutional Neural Networks (CNN).

4.4 Question Answering

Question answering (QA) is the NLP task that focuses on developing systems capable of automatically answering questions posed by users in natural language. QA systems aim to understand the meaning and intent of the questions and provide accurate and relevant answers.

Question answering is a complex procedure involving several steps. It begins with retrieving relevant documents, followed by extracting valuable information from these documents. Next, potential answers are suggested and evaluated based on supporting evidence. Finally, a concise text response in natural language is generated as the answer.

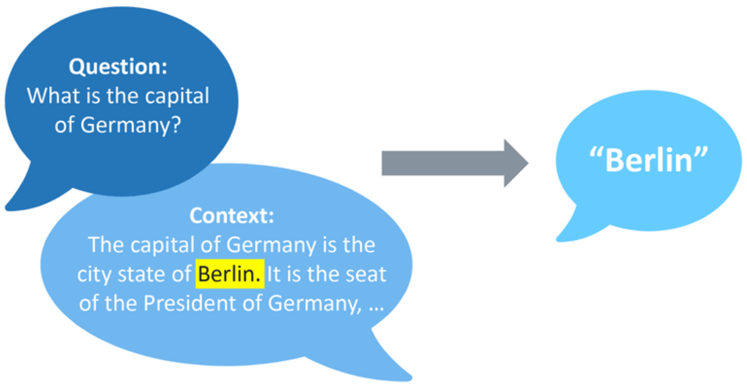

Question:
What is the capital of Germany?
Context:
The capital of Germany is the city state of Berlin. It is the seat of the President of Germany, ...
"Berlin"

Figure 4.6: A natural language question answering system takes questions in natural language and returns a concrete answer. Source: [69]

There are different approaches to question answering:

- **Closed-domain QA**: focuses on answering questions within a specific topic. The system is designed to have in-depth knowledge about a limited set of topics.
- **Open-domain QA**: aims to answer questions across a wide range of topics and does not have a specific limitation to a predefined domain. These systems are designed to understand and generate answers based on general knowledge or information available on the web.

Regarding QA methodology, the first step corresponds to question decomposition to form a query. This **query formulation** is a list of tokens or keywords that are send to an expert system to produce answers. Another approach is **query reformulation**, which consists of applying certain rules to rephrase the query so that it may look like a substring of possible answers. Here are some examples from [70]:

$$\textit{Wh-word} \text{ did A } \textit{verb} \text{ B} \longrightarrow \text{ A } \textit{verb}\text{+ed B}$$

$$\text{Where is A} \longrightarrow \text{ A is located in}$$

4.4.1 Information Retrieval Based

The objective of question answering using **information retrieval** is to respond to a user's question by locating brief text sections from the web or another large collection of documents. Web-based QA systems as Google Search use this methodology.

After having formulated the query, it is mapped into a bag-of-words or TF-IDF to *retrieve* relevant documents and sections within them. Then, the system generates the final answer which can be the most relevant sentence or a combination between multiple passages.

4.4.2 Knowledge-Based QA

Knowledge-Based QA is an approach that relies on structured knowledge bases to provide answers to user questions. Semantic parsing is employed to transform questions into relational queries that can be executed on a comprehensive database. These systems are known as **semantic parsers** and the database can be a full relational databse or knowledge base of RDF (Resource Description Framework) triples. A semantic triple or RDF triple is a set of three entities that codifies data into the form of subject-predicate-object.

DBpedia is an example of a free semantic relation database that contains over 9.5 billion RDF in multiple languages. The extraction of this content is made from Wikipedia.

4.4.3 Automated Reasoning

Automatic reasoning question answering (QA) is an approach that focuses on answering questions by performing logical inference and reasoning on structured knowledge or information. Automatic reasoning is a field of artificial intelligence that involves applying logical rules and inference mechanisms to derive answers based on the given question and the available knowledge. QA systems can improve their answer hypotheses by generating a collection of first-order logic clauses that augment semantic relations and the retrieved evidence.

IBM Watson's DeepQA is a QA system developed by IBM that incorporates all these three methodologies: IR-based, knowledge-based and automated reasoning. It gained significant attention when it competed and won against human contestants on the quiz show Jeopardy! in 2011. DeepQA continues to evolve, with ongoing research and development to enhance its capabilities in understanding complex questions, handling ambiguity, and providing accurate and insightful answers.

Mean Reciprocal Rank (MRR) is a measure to evaluate question answering performance. The MRR is calculated as the average reciprocal rank (RR) of the first correct answer. The reciprocal rank of a query response is the multiplicative inverse of the rank of the first correct answer [71].

$$MRR = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{rank_i} \tag{4.4.1}$$

where $rank_i$ refers to the rank position of the *first* relevant document for the i-th query. If no correct answer was returned in the query, the reciprocal rank is 0. An outstanding performance is set at exceeding $MRR = 0.83$.

4.5 Summarisation

Text Summarisation consists of identifying the most relevant information in a document or group of documents and generating a concise and coherent summary. Due to the exponential growth of textual content on the Internet and in various archives such as news articles, scientific papers, and legal documents, Automatic Text Summarisation (ATS) is gaining significant importance. The high volume of textual content makes manual text summarisation a time-consuming, labor-intensive and costly process. In fact, with such a vast amount of text, manual summarisation often becomes impractical. Therefore, the need for ATS arises as an efficient and effective solution to automatically summarise text, enabling users to extract key information without the need for extensive human effort [72].

4.5.1 Extraction Based

Extraction-based summarisation involves selecting and extracting the most important sentences or phrases from a document to create a summary. It relies on identifying key information and preserving the original wording of the source text.

Sentences are selected based on their relevance to the overall meaning of the document. Various techniques can be used, such as ranking sentences based on their frequency, presence of keywords, or similarity to the document's main topic. One approach to assessing significance is by quantifying informative words using lexical measures such as TF-IDF. Another approach is TextRank algorithm [73], which is a graph-based ranking model by considering lexical similarity between words. It is based on the concept of PageRank, which is used by Google to rank web pages. Once we get the scores, the sentences with the highest scores are reordered to form a coherent and concise final summary.

4.5.2 Abstraction Based

Abstraction-based summarisation generates a summary by paraphrasing and rephrasing the content of the source text, rather than directly copying sentences as extraction-based. It takes a semantic approach to identify relations.

It can be more challenging than extraction-based summarisation since it requires advanced NLP techniques and a deep understanding of the source text to accurately paraphrase the content while preserving its original meaning. It can also be computationally more demanding due to the need for sentence generation and linguistic analysis.

4.6 Industrial applications

The progress in technology has brought a significant transformation to how humans communicate and interact with computers. This revolution has not only impacted individuals but also societies worldwide, providing them with communicative tools. Communication and interaction play a vital role in various aspects of daily life, including business, education, commerce, healthcare, politics, finance and socialising [76].

4.6.1 Finance

Data has become crucial in the field of finance, with valuable information residing in written documents and websites, among others. Finance professionals spend considerable time reviewing financial reports. To enhance financial decision-making, NLP and ML provide great solutions in the following areas:

- **Risk assessment**. It is based on assessing credit risk through credit scoring models, which estimates payment capacity based on previous spendings and history data. In addition to this, NLP can help fraud detection by identifying patterns or anomalies.
- **Acounting and Auditing**. NLP analyses financial statements and audit reports, extracting relevant information, identifying anomalies or errors, and facilitating data analysis.
- **Stock market predictions**. By extracting sentiment and combining it with other quantitative indicators, NLP can help investors and traders make more informed decisions and predict potential market movements. Furthermore, it can be really useful in constructing optimal investment portfolios that aim to maximise returns and minimise risk based on textual and numerical data.

4.6.2 Business

Companies have the opportunity to expand their operations globally, leading to improved trade relations and ultimately boosting international commerce. There is a wide variety of NLP applications in this field, which can be summarised in the following keys with special focus on the latter:

- **Cibersecurity**. A clear example would be spam detection, which allow to mitigate risks and protect data.
- **Recruiting and Hiring**. Information extraction and named entity recognition are used to extract information about candidates, making resume screening easier and identifying the most suitable candidate for a position. Interview

assessments and virtual interviews use NLP to summarise the information and make a candidate report. Moreover, employees' job satisfaction can be measured through sentiment analysis techniques.

- **Market Intelligence**[2] (MI). Online assistants or chatbots can solve customers' queries as well as to measure customer satisfaction and further areas of improvement. They can also be an interative way of communicating important information about the company to the customers such as new product release or delays. Competitors websites and industry news can provide useful insights so that NLP can identify new opportunities and strategies for the company to stay competitive.

4.6.3 Healthcare

The applications of NLP in healthcare is one of the fastest growing and it is revolutionasing the way we process and analyse text in the medical field. This subsection explores these diverse applications:

- **Clinical documentation**. The vast amount of data collected in healthcare through electronic processes, particularly from Electronic Health Records (EHRs), is predominantly unstructured and of poor quality, making it largely unusable. Approximately 80% of healthcare data falls into this category. However, NLP offers a solution by extracting essential data elements from unstructured sources, restructuring it and making it more accessible and meaningful for analysis and decision-making purposes [76].

- **Clinical decision support**. NLP techniques extract and analyse relevant information from diverse sources, aiding in identifying potential diagnoses, recommending appropriate treatment options and predicting patient outcomes. In addition to this, chatbots can provide immediate responses to patient inquiries, appointment scheduling, and basic healthcare information.

- **Computer-Assisted Coding** (CAC). NLP automates the process of assigning medical codes to medical records. Although it has significantly expedited the coding process, it has faced challenges in achieving optimal accuracy levels.

- **Clinical trial matching**. This is very similar to the HR process explained in section (4.6.2) since it basically consists of a selection of suitable candidates who meet the research criteria for the drug development process.

- **Biological sequence analysis**. This task aims to uncover patterns, relationships and functional insights within DNA, RNA and protein sequences. For instance, in the latter, n-grams can help identify functional domains, predict protein structures or classify proteins into families.

[2]Market intelligence refers to the process of gathering and analysing relevant information about market trends, customer behavior and competitor activities to make informed business decisions and gain a competitive edge.

4.6.4 Legal and Compliance

NLP applications in the field of legal and compliance play a crucial role in various aspects:

- **Criminal records**. It becomes possible to extract key information from legal documents, such as court records or criminal databases. This increases efficiency search about individuals' criminal histories, aiding in investigations and the administration of justice.
- **Identification of patterns and trends**. By analysing large volumes of legal texts, NLP can uncover patterns, anomalies and correlations that may go unnoticed by manual analysis. This allows to gain insights into emerging trends, potential risks and compliance violations.
- **Extracting information from policy reports**, which makes it easier to keep up to date and avoids potential risks or compliance violations.

4.6.5 Education

NLP can enhance education by supporting the needs of students, teachers and researchers. Appart from information extraction, these are some useful applications in the field of education:

- **Translation**. NLP with machine translation help people to learn other languages together with a correct spelling and pronunciation. In addition to this, there are some specific tools that allow to improve academic writing since they provide immediate feedback.
- **E-learning**. It enables the development of intelligent virtual tutors and chatbots that can engage with learners providing personalised assistance, answering questions and offering real-time feedback. Some e-learning platforms include interactive games, making learning more enjoyable.
- **Assessment**. NLP can provide meaningful feedback to students and teachers assessing essays and answers, even with automate scoring system. Moreover, it is also able to generate quizzes to assess performance and identify areas of improvement.

4.7 Ethical Considerations

The widespread adoption of NLP technologies has raised important ethical considerations that demand careful attention. One of the primary concerns is the privacy and security of user data. NLP systems often rely on collecting and analysing substantial amounts of personal information, such as text messages, emails and social media posts. It is essential to protect individuals' privacy and handle their data responsibly and transparently. Organizations should establish robust data protection measures, employ anonymisation techniques and secure storage systems while adhering to relevant data protection regulations. Obtaining clear and informed consent from users regarding the collection and use of their personal data is crucial to build trust and uphold ethical standards.

Another critical ethical consideration in NLP is the presence of bias and fairness issues in language processing systems. NLP algorithms are trained on vast amounts of textual data, which may contain societal biases and prejudices. If left unaddressed, these biases can perpetuate discrimination and inequalities in automated decision-making. It is crucial to mitigate bias in both the training data and the algorithms themselves to ensure fairness and equitable treatment. Ethical guidelines and practices should be implemented to promote diversity and inclusivity in NLP systems. Ongoing monitoring and evaluation should be conducted to identify and rectify biases that may emerge during the system's usage. Furthermore, transparency in the decision-making process of NLP algorithms is essential, enabling users and stakeholders to understand how decisions are made and holding the technology accountable for any biases that may arise.

In conclusion, as NLP technologies continue to advance, it is vital to address the ethical considerations associated with their use. Safeguarding privacy and data security, tackling bias and fairness concerns, and promoting transparency and accountability are crucial for responsible and ethical deployment of NLP systems. By proactively addressing these ethical considerations, we can maximise the benefits of NLP while protecting individuals' rights and well-being and fostering trust in the technology.

Chapter 5

Python

Python is one of the youngest, fast-changing programming language with a continuous and rapid increase in the numbers of users. It was created by Guido van Rossum and the first version, version 0.9.0, was realeased in 1991. Guido's objectives for Python were to create an intuitive, powerful language which was suitable for solving efficiently everyday tasks. However, this first version was too simple, considering that it was one person's work. Over the years, Python started to evolve including many new features and combining different paradigms[1]. Then, Python 2.0 was released in 2000 with these important changes, and, in 2008, Python 3.0 implemented some corrections and improvements to make this language even easier. In twenty years, we can confirm that Guido's goals have been fulfilled.

Figure 5.1: Python logo. Source: [78]

Today, Python is one of the world's most used programming languages, known for its simplicity, readability and versatility. Developers use it for a wide range of applications, including web development, data analysis, scientific computing, machine learning and artificial intelligence.

Python is an open-source programming language characterised by being:

- **Interpreted**. It requires to be executed directly by an interpreter program, which reads the code line-by-line. This allows to see the results at once, although the performance is slower than in compiled languages.
- **Multi-paradigm**. It supports different programming styles and techniques depending on the requirements to solve a problem. The most remarkable Python paradigms are:
  - **Object-oriented**. It is organised as collections of cooperative, dynamic "objects", which can contain data and code to manipulate that data.

[1]Paradigms is a clasification of programming languages according to their features.

Classes define the characteristics of an object, such as its attibutes and methods. It is relevant to understand the latter concepts in Python:

  * **Attributes** are used to store data within and object and can be accessed using a dot notation, such as `object.attribute`
  * **Methods** are functions that are defined within a class and can be called on an object belonging to that class. They usually perform actions using the data stored in the object as attributes. As attributes, they also use the dot notation, but it is followd by brackets: `object.method()` .

- **Functional**. It is mainly based on functions and its composition, being similar to mathematical notation. For this, Python has implemented:
  * Higher-order functions that take other functions as arguments or return functions as their result, such as `map` and `filter`.
  * Pure functions return the same output given the same input and they can be constructed using `lambda`.
- **Procedural**. It focuses on dividing a program into more manageable procedures, which are executed line-by-line.

• **High-level**. Python is close to human natural language, which eases reading and writing code without dealing with machine code. A high-level language allows the programmer to focus on planning the approach for solving a problem using code instead of focusing on knowing how the machine works to understand the code. In a nutshell, it is problem-oriented rather than machine-oriented.

Although there are other suitable programming languages for natural language processing, Python is the most used thanks to these properties and the diversity of its libraries. The next section is focused on one of the most world-known NLP libraries.

5.1 Natural Language Toolkit

The **Natural Language Toolkit**, or commonly **NLTK**, is a set of libraries and programmes to work with human language data in Python. It also contains a comprehensive collection of text corpora and lexical resources. Steven Bird and Edward Loper developed this powerful tool at the University of Pennsylvania, and it was realeased in 2001. They wrote a book, *Natural Language Processing with Python* [79], to explain the concepts and how this tool supported NLP tasks. The toolkit was implemented as a collection of independent *modules*, each of which defines a specific data structure or task. It started linked to research and education, but its extensive features and functionality made it one of the libraries with a large number of learning resources.

One notable feature of NLTK is that its functions are executed in a sequential manner. This implies that, in many instances, it is necessary to execute a series of preceding functions before running a particular function (figure 5.2).

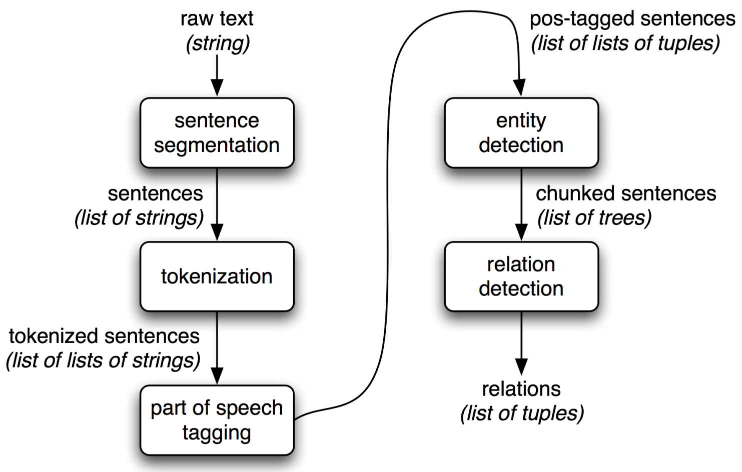


Figure 5.2: NLTK Architecture Pipeline. Source: [79], Chapter 7

5.2 SpaCy

SpaCy is another popular powerful and open-source library for NLP in Python that was developed by Ines Montani and Matthew Honnibal. SpaCy utilises Cython, a programming language that combines the ease of Python with the speed of C, to achieve excellent performance. This makes SpaCy suitable for real-time applications where processing time is a critical factor.

Another notable feature of SpaCy is its pre-trained models in multiple languages that have been trained on large corpora and can be directly used for various NLP tasks. Furthermore, it also offers seamless integration with other popular Python libraries such as NumPy, Pandas and scikit-learn for advanced data analysis and machine learning tasks.

In contrast to NLTK's sequential function execution, Spacy utilises an object called `nlp()`, which leverages a pre-trained model to instantly retrieve a range of attributes when texts are provided. This offers advantages such as simplified result retrieval, but it implies a slower performance due to the computation of numerous attributes that may not be used (figure 5.3).

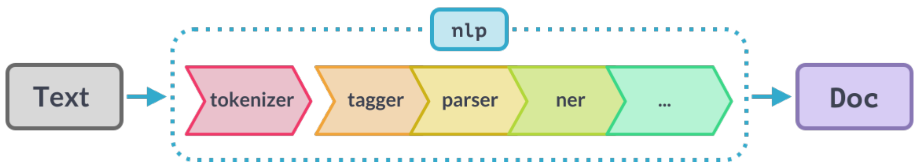


Figure 5.3: SpaCy Processing Pipeline. Source: spacy.io/api

SpaCy's architecture consists of three main components (see figure 5.4):

- The **Language** class initialises the processing pipeline and coordinates the various processing steps such as tokenisation. It allows users to add or remove pipeline components as needed, making it flexible and adaptable to different requirements (see figure 5.3).
- The **Vocab** class contains the mapping of words to unique integer IDs and provides access to word vectors, word frequencies and other lexical attributes.
- The **Doc** class represents a processed text document. It provides easy access to the linguistic annotations and supports various operations and transformations, allowing users to manipulate and analyse the text data efficiently.

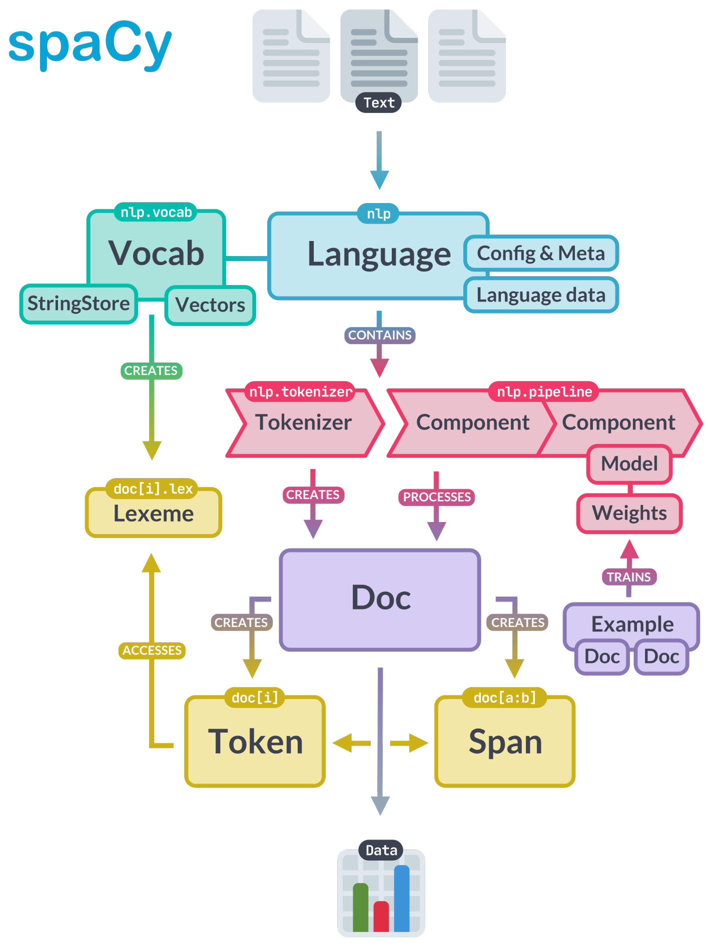


Figure 5.4: SpaCy Architecture. Source: spacy.io/api

5.3 Scikit-learn

As stated in the paper *Scikit-learn: Machine Learning in Python* [83]:

> "**Scikit-learn** is a Python module integrating a wide range of state-of-the-art machine learning algorithms for medium-scale supervised and unsupervised problems. This package focuses on bringing machine learning to non-specialists using a general-purpose high-level language. Emphasis is put on ease of use, performance, documentation, and API consistency."

The history of scikit-learn traces back to the early 2000s. It originated as a Google Summer of Code project in 2007 and was initially developed by David Cournapeau. In 2010, contributors Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort and Vincent Michel took leadership of the project and released the first public version of this library.

Scikit-learn quickly gained traction within the Python community due to its simplicity, well-documented API and extensive set of features. This API took a vital role in this success since not only does it include an extensive documentation, but it also has examples, tutorials and some featuring real-word applications. In addition to this, they keep the balance between mathematical precision and easy-understanding of the concepts explained, which makes it accessible to a wider audience and foster a collaborative community.

Scikit-learn uses mainly Numpy and Scipy Python libraries to build their algorithms, which some are written in Cython[2] to improve performance. Numpy is used for multi-dimensional arrays and matrices, together with functions for array operations. In scikit-learn, numpy is used for the data structure and model parameters. On the other hand, Scipy allows sickit-learn to implement efficient algorithms for linear algebra, sparse matrix representation, special functions and basic statistical functions.

The scikit-learn library provides powerful tools for model selection, which is a crucial aspect of Machine Learning. Model selection involves choosing the best model or algorithm and its hyperparameters for a given task. For this purpose, scikit-learn offers several modules and functions which have been specifically designed.

One of the primary tools for model selection is the `model_selection` module. This module provides functions for evaluating and comparing different models using various techniques, such as cross-validation and grid search. For this, we use the `train_test_split` to obtain train and test subsets to validate our models.

Cross-validation is a common technique for assessing the performance of a model on unseen data. Scikit-learn offers functions like `cross_val_score` and `cross_validate` that facilitate cross-validation by splitting the data into multiple folds and evaluating the model on each fold.

Grid search is another essential technique in model selection, which involves systematically searching through a specified hyperparameter grid to find the best combination of hyperparameters for a model. Scikit-learn provides the `GridSearchCV`

[2]Cython is a language for combining C in Python.

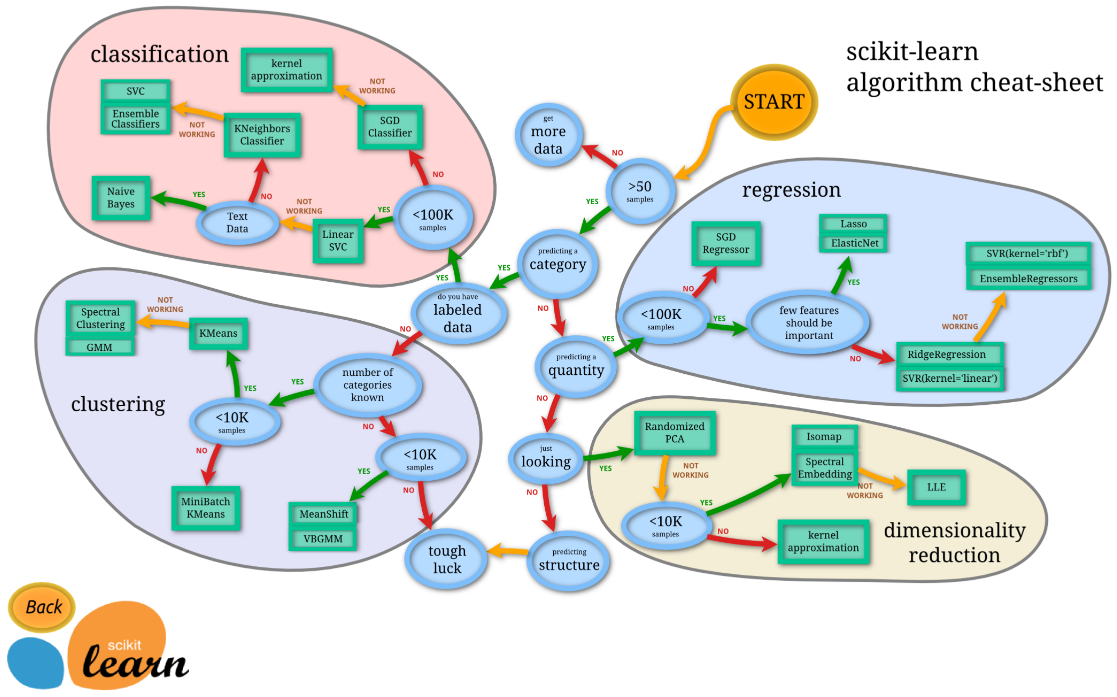


Figure 5.5: Scikit-learn algorithm cheat-sheet. Source: Scikit-learn

class, which automates the process of grid search and cross-validation, enabling users to efficiently explore different hyperparameter settings and identify the optimal configuration.

In addition to grid search, scikit-learn also supports randomised search with the `RandomizedSearchCV` class. Randomised search is useful when the hyperparameter search space is large, as it allows for a more efficient exploration by sampling a subset of the parameter grid.

Both the model and the hyperparameters are selected based on different evaluation metrics. The `metrics` module in scikit-learn offers more than 40 metrics available to analyse the results of the models. Some metrics used for regression models are mean square error (MSE), mean absolute error (MAE) or the coefficient of determination R^2, which correspond to funtions `mean_squared_error`, `mean_absolute_error` and `r2_score`, respectively. Others like `confusion_matrix`, `accuracy_score` and `classification_report` are used to evaluate and compare the results of classification models.

Furthermore, scikit-learn pipeline is a powerful tool which makes it easier to organise the code since it allows to combine data normalisation and machine learning models into a single entity. Pipelines promote good coding practices, improve the reproducibility of results, and allow for efficient preprocessing, model training, cross-validation and hyperparameter tuning.

Chapter 6

Results

In this chapter, we will provide an overview of the Python code implemented for machine learning techniques that gather the concepts and procedures that have been explained along this work, together with a previous preprocessing step.

To analyse the different machine learning algorithms explained (section (3)) and their performance, a binary text classification task is implemented. For this, a dataset [84] from Kaggle has been used, whose main aim is to find the best classifier (model) that distinguishes fake from real news. Kaggle is an online platform and community that hosts data science competitions, provides datasets for practice and offers a collaborative environment for data scientists, machine learning engineers and researchers.

In addition to this, we are also going to perform a special case of text classification, sentiment analysis, which is based on distinguishing positive from negative film reviews. For this, another dataset from Kaggle about IMDb (Internet Movie Database) reviews has been used [85].

Fake News Classifier

Import Libraries

```
[1]: import matplotlib.pyplot as plt
     import warnings
     warnings.filterwarnings('ignore')
     import numpy as np
     import pandas as pd
     import seaborn as sns
     from nltk.corpus import stopwords
     from sklearn.model_selection import StratifiedKFold, cross_val_score,␣
      ↪cross_val_predict
     from sklearn.metrics import confusion_matrix, classification_report,␣
      ↪ConfusionMatrixDisplay
     from sklearn.feature_extraction.text import TfidfVectorizer
     from sklearn.model_selection import train_test_split, GridSearchCV, KFold
```

Data Load

```
[2]: true = pd.read_csv("C:/Users/Amparo/OneDrive/Desktop/data/true.csv")
     fake = pd.read_csv("C:/Users/Amparo/OneDrive/Desktop/data/fake.csv")
```

Preprocessing and Exploratory Data Analysis (EDA)

In this section, it is vital to deal with NA values and duplicates as a first step.

Remove NA values and add a target column indicating whether the news is true or fake.

```
[3]: true.dropna(inplace = True)
     fake.dropna(inplace = True)

     true['Labels'] = 0
     fake['Labels'] = 1
```

Check for duplicated registers in the fake news dataset, and then, drop the occurrences except for the first one.

```
[4]: fake.duplicated().sum()
```

```
[4]: 3
```

```
[5]: fake.drop_duplicates(inplace=True)
```

The same procedure is applied to the real news dataset.

```
[6]: true.duplicated().sum()
```

```
[6]: 206
```

```
[7]: true.drop_duplicates(inplace=True)
```

Now, we can merge both datasets into a single labelled dataset to work with them.

```
[8]: train_data = pd.concat([true, fake], ignore_index = True)
```

```
[9]: print("We have a dataset with {} news".format(len(train_data)))
```

```
We have a dataset with 44689 news
```

Now, we count the number of documents per label to know if we have a balanced dataset.

```
[10]: train_data["Labels"].value_counts()
```

```
[10]: 1    23478
      0    21211
      Name: Labels, dtype: int64
```

```
[11]: train_data["subject"].value_counts()
```

```
[11]: politicsNews       11220
      worldnews           9991
      News                9050
      politics            6838
      left-news           4459
      Government News     1570
      US_News              783
      Middle-east          778
      Name: subject, dtype: int64
```

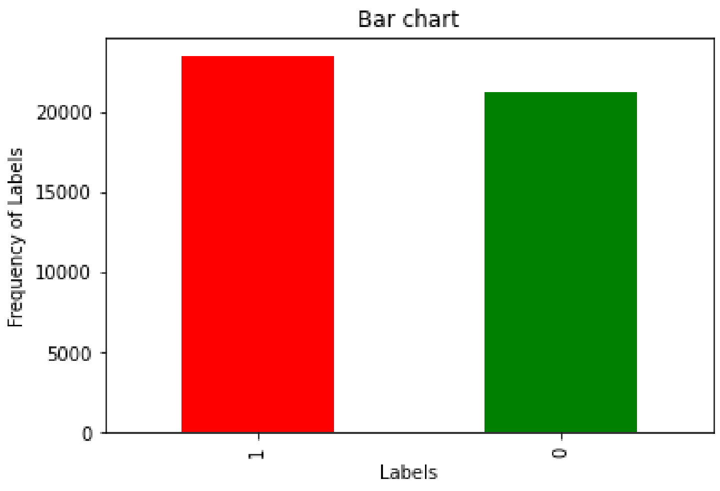
```
[12]: train_data.groupby(['Labels']).count()
```

```
[12]:         title   text  subject   date
      Labels
      0       21211  21211    21211  21211
      1       23478  23478    23478  23478
```

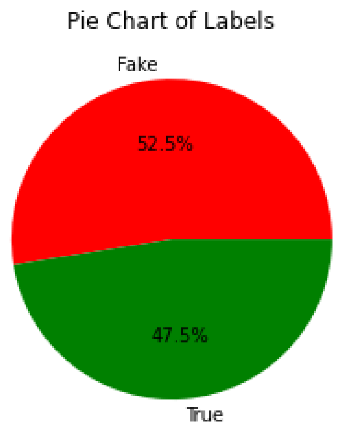
```
[13]: np.sum(train_data.isnull())[2]
```

```
[13]: 0
```

[14]:
```
ax, fig = plt.subplots()
etiquetas = train_data.Labels.value_counts()
etiquetas.plot(kind= 'bar', color= ["red", "green"])
plt.title('Bar chart')
plt.ylabel("Frequency of Labels")
plt.xlabel("Labels")
plt.show()
```



[15]:
```
import matplotlib.pyplot as plt
# Count the frequency of each sentiment category
etiquetas = train_data['Labels'].value_counts()

# Create a pie chart
fig, ax = plt.subplots()
ax.pie(etiquetas, labels=["Fake", "True"], colors=[ "red", "green"], autopct='%1.
 ↪1f%%')
ax.set_title('Pie Chart of Labels')

# Display the pie chart
plt.show()
```



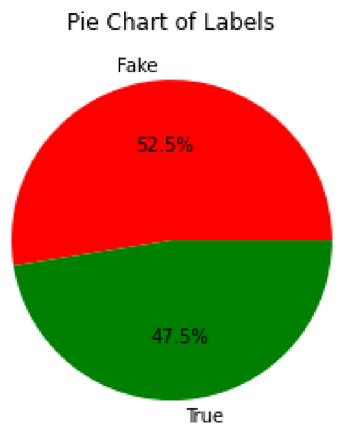
Then, we can conclude that the dataset with 44689 documents is approximately balanced. Therefore, there are approximately the same number of fake and true news.

To reduce the computational cost of this task, we are going to select 35000 news preserving this representation of labels.

```
[16]: X = train_data[['title', 'text', 'subject']]
y = train_data['Labels']

# Use train_test_split for stratified sampling
X_sampled, _, y_sampled, _ = train_test_split(X, y, train_size=35000,␣
 ↪stratify=y, random_state=10)

# Combine the sampled features and target into a dataframe
data = pd.concat([X_sampled, y_sampled], axis=1)

data.head()
```

```
[16]:                                                   title  \
33440  DAY 2 RESULTS Of Wisconsin Recount Are In...And ...
32308  CNN ANCHOR Piles On Bill O'Reilly With Silly S...
9966   Leak of Senate encryption bill prompts swift b...
15770  German railway under fire for proposal to name...
22037   Here Are 12 Tweets Trump DEFINITELY Regrets S...

                                                    text       subject  Labels
33440  The Wisconsin Election Commission posts recoun...      politics       1
32308  The accusations surrounding Bill O Reilly have...      politics       1
9966   WASHINGTON (Reuters) - Security researchers an...  politicsNews       0
15770  BERLIN (Reuters) - German rail operator Deutsc...     worldnews       0
22037  As you are probably aware, Donald Trump loves ...          News       1
```

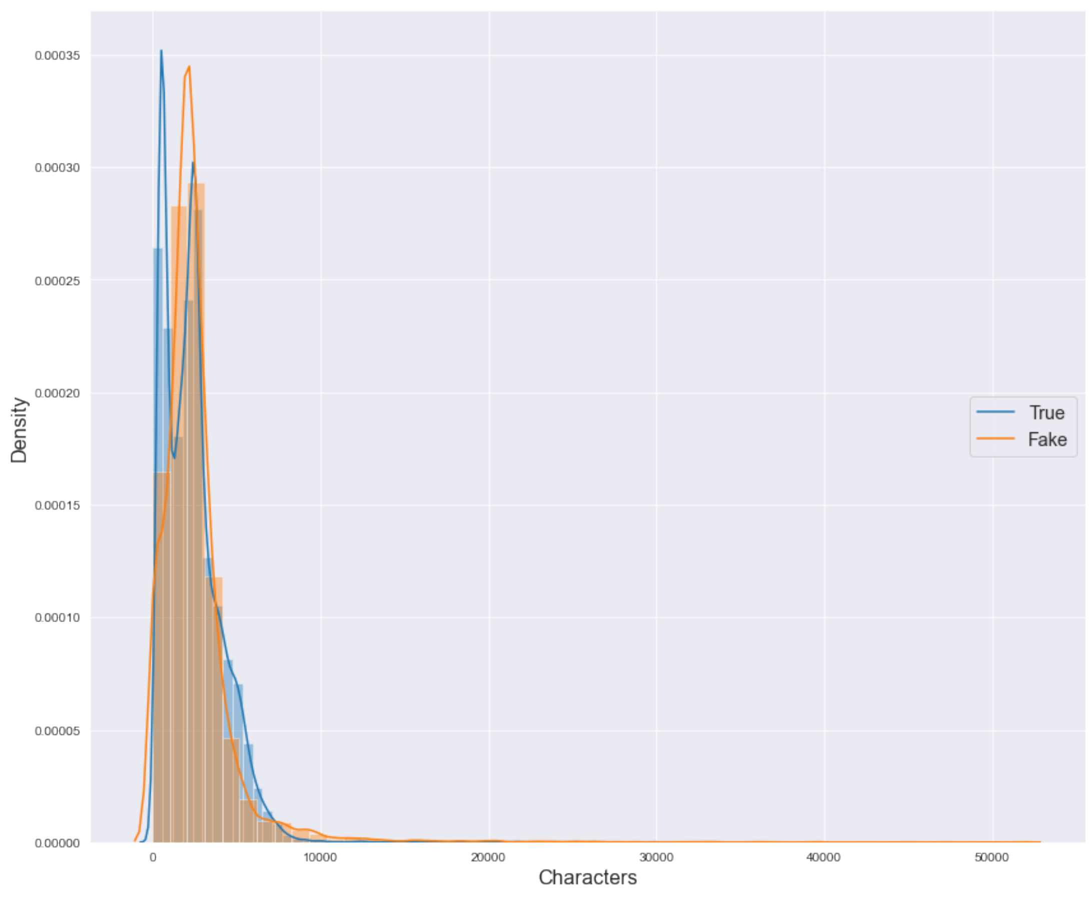
[17]:
```
import matplotlib.pyplot as plt
# Count the frequency of each sentiment category
etiquetas = data['Labels'].value_counts()

# Create a pie chart
fig, ax = plt.subplots()
ax.pie(etiquetas, labels=["Fake", "True"], colors=[ "red", "green"], autopct='%1.
 ↪1f%%')
ax.set_title('Pie Chart of Labels')

# Display the pie chart
plt.show()
```



Make an exploratory data analysis about the length of the characters for each class to know if this can be used as a determining feature.

[18]:
```
data["char_len"] = data["text"].apply(lambda x: len(x))
```

[19]:
```
fig = plt.figure(figsize=(14,12))
sns.set_style("darkgrid")

plt0 = sns.distplot(data[data.Labels==0].char_len, hist=True, label="True")
plt1 = sns.distplot(data[data.Labels==1].char_len, hist=True, label="Fake")
plt.legend(labels=["True","Fake"], loc = 'center right', fontsize = "x-large")

# Axis labels
plt.xlabel('Characters', fontsize=16)
plt.ylabel('Density', fontsize=16)

plt.show()
```



We can see that true news have a lower length of characters, although they are really similar and this cannot be considered a determining factor to classify news.

Now, we will define functions that will be used to preprocess the text as explained in this work.

[20]:
```
# Delete spaces
def delete_spaces(text):
    return  " ".join(text.split())

# To lower
def texto_to_lower(text):
    return text.lower()

# Tokenisation
from nltk import word_tokenize
def tokenisation(text):
    tokens = word_tokenize(text)
    return tokens
```

```
# Once text has been tokenised, we can create functions that take tokens as␣
 ↪inputs and apply further preprocessing.
# Remove stop words
def stopwords_removal(tokens):
    stop_words = set(stopwords.words('english'))
    filtered_sentence = [w for w in tokens if not w in stop_words]
    return filtered_sentence

# Remove punctuation (only alphanumeric values are allowed)
def punctuation_removal(tokens):
    words=[word for word in tokens if word.isalnum()]
    return words

# Stemming
from nltk.stem import PorterStemmer
stemmer = PorterStemmer()
def stem(tokens):
    tokens = [ stemmer.stem(token) for token in tokens]
    return tokens
```

We create a function that implements all these functions that have been created so that we can preprocess the text in just one step.

[21]:
```
def preprocess(sentence):
    sentence = delete_spaces(sentence)
    sentence = texto_to_lower(sentence)
    sentence = tokenisation(sentence)
    sentence = stopwords_removal(sentence)
    sentence = punctuation_removal(sentence)
    sentence = stem(sentence)
    return sentence

# Store in a new column "normalise" of the dataframe to keep the original data.
data["tokens_norm"] = data["text"].apply(lambda x: preprocess(x))
```

[22]:
```
data.head()
```

[22]:
```
                                                   title  \
33440  DAY 2 RESULTS Of Wisconsin Recount Are In...And ...
32308  CNN ANCHOR Piles On Bill O'Reilly With Silly S...
9966   Leak of Senate encryption bill prompts swift b...
15770  German railway under fire for proposal to name...
22037   Here Are 12 Tweets Trump DEFINITELY Regrets S...

                                                    text       subject  \
33440  The Wisconsin Election Commission posts recoun...      politics
32308  The accusations surrounding Bill O Reilly have...      politics
9966   WASHINGTON (Reuters) - Security researchers an...  politicsNews
```

```
15770  BERLIN (Reuters) - German rail operator Deutsc...     worldnews
22037  As you are probably aware, Donald Trump loves ...          News

       Labels  char_len                                        tokens_norm
33440       1      1980  [wisconsin, elect, commiss, post, recount, dat...
32308       1      2104  [accus, surround, bill, reilli, relentless, no...
9966        0      4158  [washington, reuter, secur, research, civil, l...
15770       0      1460  [berlin, reuter, german, rail, oper, deutsch, ...
22037       1      5205  [probabl, awar, donald, trump, love, tweet, lo...
```

Since we have obtained a list of normalised tokens, we just have to join them with a whitespace to obtain the normalised text.

```
[23]: data["text_normalised"] = data["tokens_norm"].apply(lambda x: " ".join(x))
      data.head()
```

```
[23]:                                                   title  \
33440  DAY 2 RESULTS Of Wisconsin Recount Are In...And ...
32308  CNN ANCHOR Piles On Bill O'Reilly With Silly S...
9966   Leak of Senate encryption bill prompts swift b...
15770  German railway under fire for proposal to name...
22037   Here Are 12 Tweets Trump DEFINITELY Regrets S...

                                                    text       subject  \
33440  The Wisconsin Election Commission posts recoun...      politics
32308  The accusations surrounding Bill O Reilly have...      politics
9966   WASHINGTON (Reuters) - Security researchers an...  politicsNews
15770  BERLIN (Reuters) - German rail operator Deutsc...     worldnews
22037  As you are probably aware, Donald Trump loves ...          News

       Labels  char_len                                        tokens_norm  \
33440       1      1980  [wisconsin, elect, commiss, post, recount, dat...
32308       1      2104  [accus, surround, bill, reilli, relentless, no...
9966        0      4158  [washington, reuter, secur, research, civil, l...
15770       0      1460  [berlin, reuter, german, rail, oper, deutsch, ...
22037       1      5205  [probabl, awar, donald, trump, love, tweet, lo...

                                         text_normalised
33440  wisconsin elect commiss post recount data spre...
32308  accus surround bill reilli relentless noth pro...
9966   washington reuter secur research civil liberti...
15770  berlin reuter german rail oper deutsch bahn fa...
22037  probabl awar donald trump love tweet love love...
```

Vectorisation of text

Given a collection of documents, the TfidfVectorizer() method computes the TF-IDF scores for all terms in the corpus. Then, the selection of the features to include in the vocabulary is based on their importance, which is determined by their TF-IDF scores.

The TF-IDF scores reflect the relevance of a term or n-gram in a document relative to the entire corpus. The higher the TF-IDF score, the more important the term is considered. The selection of the most frequent n-grams is also influenced by other parameters as well, such as min_df (minimum document frequency) and ngram_range, whose lower and upper boundary indicates the range of n for different n-grams.

In this case, we consider 25 features and unigrams, bigrams and trigrams.

[24]:
```
vectoriser = TfidfVectorizer(max_features=25,
                             ngram_range=(1,3),
                             min_df=0.05,
                             lowercase=False)

matrix_data = vectoriser.fit_transform(data["text_normalised"]) #matriz sparse
print(matrix_data)
```

```
  (0, 18)	0.07724912803551612
  (0, 7)	0.13068983007332996
  (0, 23)	0.09827319441059361
  (0, 19)	0.10609218237363716
  (0, 1)	0.49374962028672537
  (0, 22)	0.6995224644593263
  (0, 15)	0.12297343518398293
  (0, 20)	0.20338226618527075
  (0, 5)	0.4079612614655374
  (1, 0)	0.11742321363250119
  (1, 24)	0.24389823948901312
  (1, 12)	0.1127623909559013
  (1, 8)	0.4996658918934008
  (1, 18)	0.7411060178571649
  (1, 19)	0.33927267445898496
  (2, 21)	0.11826806424907833
  (2, 2)	0.27047024568571026
  (2, 16)	0.12923509778257514
  (2, 10)	0.12029481812184485
  (2, 6)	0.26005609361758514
  (2, 11)	0.14710864447705083
  (2, 14)	0.1933634827065666
  (2, 17)	0.10167117541472286
  (2, 0)	0.1103051557960119
  (2, 24)	0.11455670677438416
  :	:
  (34998, 9)	0.08910903765375215
  (34998, 3)	0.7234644136270463
  (34998, 13)	0.07880099111087438
  (34998, 21)	0.1718323711997319
  (34998, 10)	0.17477705390851672
  (34998, 6)    0.18891851953008254
  (34998, 0)    0.24039452150074767
  (34998, 24)   0.16644011793765556
  (34998, 12)   0.2308526583334894
  (34998, 8)    0.4262251449128839
  (34998, 18)   0.05619364192162943
  (34998, 23)   0.07148726254443444
  (34998, 19)   0.07717506020582411
  (34998, 15)   0.1789101148021851
  (34999, 9)    0.20233588907103997
  (34999, 13)   0.17892987081795192
  (34999, 10)   0.19842920271010098
  (34999, 6)    0.21448439809008712
  (34999, 11)   0.4853185114886348
  (34999, 14)   0.47843658986020143
  (34999, 0)    0.18195101386035034
  (34999, 24)   0.1889640497008905
  (34999, 12)   0.17472892050081437
  (34999, 15)   0.4062431611366153
  (34999, 20)   0.335937003835908
```

[25]:
```
matrix_data
```

[25]:
```
<35000x25 sparse matrix of type '<class 'numpy.float64'>'
        with 339229 stored elements in Compressed Sparse Row format>
```

We have obtained a 35000×25 matrix, which corresponds to the number of documents and the number of features allowed in the vectoriser. In addition to this, the element (i, j) of this matrix corresponds to the TD-IDF score of the j-th feature in the i-th document. As we can appreciate, it is an sparse matrix since there are only 339229 stored elements out of 875000 elements.

We can also obtain the normalised tokens or n-grams which have been selected as features by the TD-IDF.

[26]:
```
feature_names = vectoriser.get_feature_names_out()
print(feature_names)
```

```
['also' 'clinton' 'democrat' 'donald' 'donald trump' 'elect' 'govern'
 'hous' 'like' 'nation' 'new' 'obama' 'one' 'peopl' 'presid' 'report'
 'republican' 'reuter' 'said' 'say' 'state' 'time' 'trump' 'would' 'year']
```

[27]:
```
# Define categorical variable of the labels
y = data["Labels"].values.astype(np.float32)
print(y.shape)
```

```
(35000,)
```

Machine Learning

Split the dataset into a training and test sets (70%-30% respectively).

```
[28]: X_train, X_test, y_train, y_test = train_test_split(matrix_data, y, test_size =␣
      ↪0.3, random_state=10)
      print(X_train.shape)
      print(X_test.shape)
```

```
(24500, 25)
(10500, 25)
```

We have obtained a training sample of 24500 news and a test set of 10500 news.

We define a function to perform cross validation with 10 folds across different models, which returns the final confusion matrix and the results for the given metrics. In this case, we have considered StratifiedKFold to preserve class representation within each sample, although it would not be strictly necessary since we have a balanced dataset.

```
[29]: def model_evaluation(models, scores, X, y):
          global results
          global names
          results = {}
          names = []

          # For each model
          for name, model in models:

              SKF = StratifiedKFold(n_splits=10, shuffle=True, random_state=98)
              model_results = {}

              # Cross-validation
              for score in scores:
                  cv_results = cross_val_score(model, X, y, cv=SKF, scoring=score,␣
      ↪verbose=False)
                  model_results[score] = cv_results

              y_pred = cross_val_predict(model, X, y, cv=SKF)
              print(name)
              print(confusion_matrix(y, y_pred))
              print(classification_report(y, y_pred, digits=4))

              # Store results
              results[name] = model_results
              names.append(name)
```

```
[30]: # Load the models
      from sklearn.linear_model import LogisticRegression
      from sklearn.tree import DecisionTreeClassifier
      from sklearn.naive_bayes import GaussianNB
      from sklearn.ensemble import GradientBoostingClassifier
      from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.svm import SVC
from xgboost import XGBClassifier
```

The toarray() method converts this sparse matrix into a dense array, where all elements are explicitly stored in memory.

[31]:
```
# Define the models
models = [
         ("Logistic", LogisticRegression()),
         ("GaussianNB",GaussianNB()),
         ("Tree", DecisionTreeClassifier()),
         ("XGBoost", XGBClassifier()),
         ("Gradient Boosting", GradientBoostingClassifier()),
         ("LDA", LinearDiscriminantAnalysis()),
         ("SVC", SVC())
]

scores = ["accuracy", "roc_auc"]

model_evaluation(models,  scores, X_train.toarray(), y_train)
```

```
Logistic
[[11308   322]
 [  215 12655]]
              precision    recall  f1-score   support

         0.0     0.9813    0.9723    0.9768     11630
         1.0     0.9752    0.9833    0.9792     12870

    accuracy                         0.9781     24500
   macro avg     0.9783    0.9778    0.9780     24500
weighted avg     0.9781    0.9781    0.9781     24500

GaussianNB
[[11071   559]
 [ 1434 11436]]
              precision    recall  f1-score   support

         0.0     0.8853    0.9519    0.9174     11630
         1.0     0.9534    0.8886    0.9198     12870

    accuracy                         0.9187     24500
   macro avg     0.9194    0.9203    0.9186     24500
weighted avg     0.9211    0.9187    0.9187     24500

Tree
[[11508   122]
 [  109 12761]]
```

```
              precision    recall  f1-score   support

         0.0     0.9906    0.9895    0.9901     11630
         1.0     0.9905    0.9915    0.9910     12870

    accuracy                         0.9906     24500
   macro avg     0.9906    0.9905    0.9905     24500
weighted avg     0.9906    0.9906    0.9906     24500

XGBoost
[[11596    34]
 [   91 12779]]
              precision    recall  f1-score   support

         0.0     0.9922    0.9971    0.9946     11630
         1.0     0.9973    0.9929    0.9951     12870

    accuracy                         0.9949     24500
   macro avg     0.9948    0.9950    0.9949     24500
weighted avg     0.9949    0.9949    0.9949     24500

Gradient Boosting
[[11592    38]
 [  133 12737]]
              precision    recall  f1-score   support

         0.0     0.9887    0.9967    0.9927     11630
         1.0     0.9970    0.9897    0.9933     12870

    accuracy                         0.9930     24500
   macro avg     0.9928    0.9932    0.9930     24500
weighted avg     0.9931    0.9930    0.9930     24500

LDA
[[10842   788]
 [  512 12358]]
              precision    recall  f1-score   support

         0.0     0.9549    0.9322    0.9434     11630
         1.0     0.9401    0.9602    0.9500     12870

    accuracy                         0.9469     24500
   macro avg     0.9475    0.9462    0.9467     24500
weighted avg     0.9471    0.9469    0.9469     24500

SVC
[[11534    96]
 [  136 12734]]
```

```
             precision    recall  f1-score   support

         0.0     0.9883    0.9917    0.9900     11630
         1.0     0.9925    0.9894    0.9910     12870

    accuracy                         0.9905     24500
   macro avg     0.9904    0.9906    0.9905     24500
weighted avg     0.9905    0.9905    0.9905     24500
```

By definition, a confusion matrix C is such that $C_{i,j}$ is equal to the number of observations known to be in group i and predicted to be in group j.

Thus, in binary classification, the count of true positives is $C_{0,0}$, false positives is $C_{1,0}$, true negatives is $C_{1,1}$ and false negatives is $C_{0,1}$.

In this case, the positive class refers to true news and the negative class are the fake news.

For a better understanding, we are going to check the classification report results to know how they have been calculated and put the theoretical concepts preivously explained into practice.

Let select GaussianNB confussion matrix to perform these calculations.

First, we calculate the precision or positive predictive value (PPV).

$$Precision^0 = \frac{TP}{TP + FP} = \frac{11071}{11071 + 1434} = 0.885325$$

Since we are calculating precision for each class, we have to consider positive to be the class Python is calculating. This is equivalent to the negative predictive value (NPV).

$$Precision^1 = \frac{TN}{TN + FN} = \frac{11436}{11436 + 559} = 0.9533972$$

Another important metric is the recall, which is also known as sensitivity or true positive rate (TPR).

$$Recall^0 = \frac{TP}{TP + FN} = \frac{11071}{11071 + 559} = 0.95193465$$

The same reasoning as before is applied for the other class, where we would calculate the specificity or true negative rate (TNR).

$$Recall^1 = \frac{TN}{TN + FP} = \frac{11436}{11436 + 1434} = 0.888578$$

Finally, F_1 score is a harmonic mean of precision and recall:

$$F_1 = \frac{2Precision \times Recall}{Precision + Recall}$$

$$F_1^0 = \frac{2 \times 0.8853 \times 0.9519}{0.8853 + 0.9519} = 0.9173928$$

$$F_1^1 = \frac{2 \times 0.9534 \times 0.8886}{0.9534 + 0.8886} = 0.9198$$

Support represents the number of instances in each class: 11630 true and 12870 fake news.

The accuracy is proportion of the samples correctly classified.

$$Accurary = \frac{TN + TP}{TP + FN + FP + TN} = \frac{11071 + 11436}{24500} = 0.918653$$

Macro average for a given metric is an arithmetic (unweighted) mean.

$$Macro\ avg\ metric = \frac{1}{N} \sum_i metric^i$$

where N is the number of classes.

For instance, we can calculate the macro average precision:

$$Macro\ avg\ precision = \frac{precision^0 + precision^1}{2} = \frac{0.8853 + 0.9534}{2} = 0.91935$$

On the other hand, weighted average for a given metric is a weighted mean with support as weights.

$$Weighted\ avg\ metric = \frac{support^0 \times metric^0 + \cdots + support^{N-1} \times metric^{N-1}}{support^0 + \cdots + support^{N-1}}$$

where the denominator is the number of total instances.

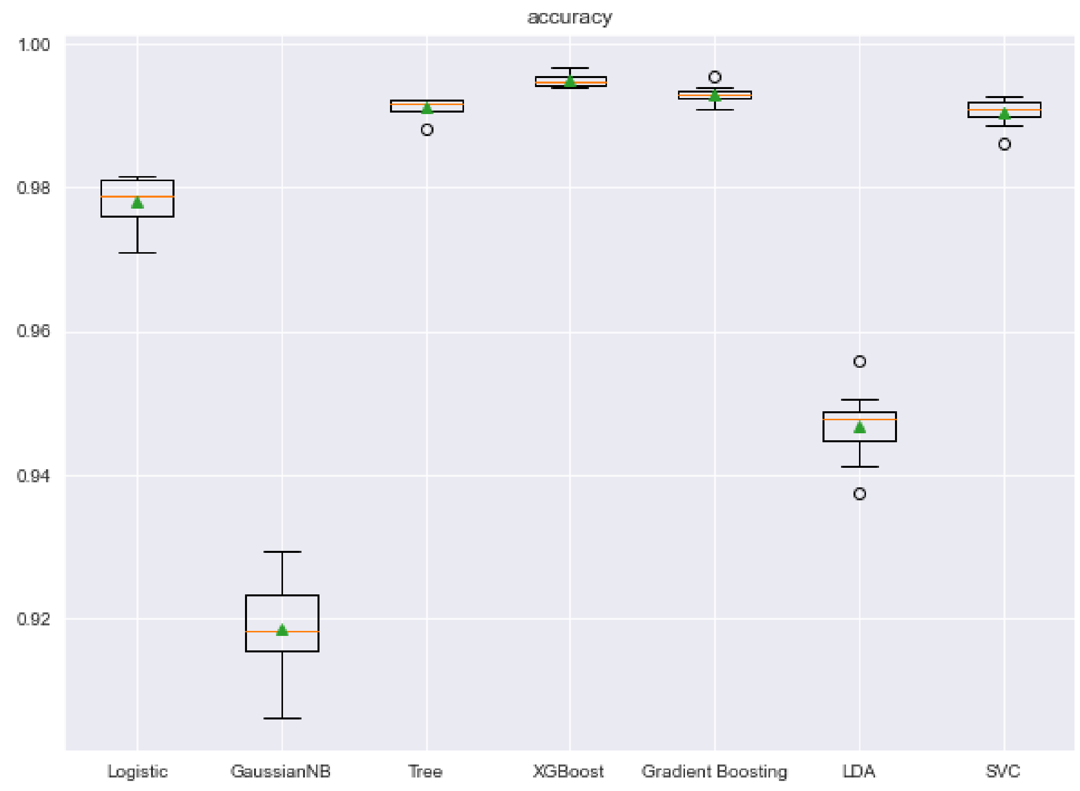
For the precision, we would obtain:

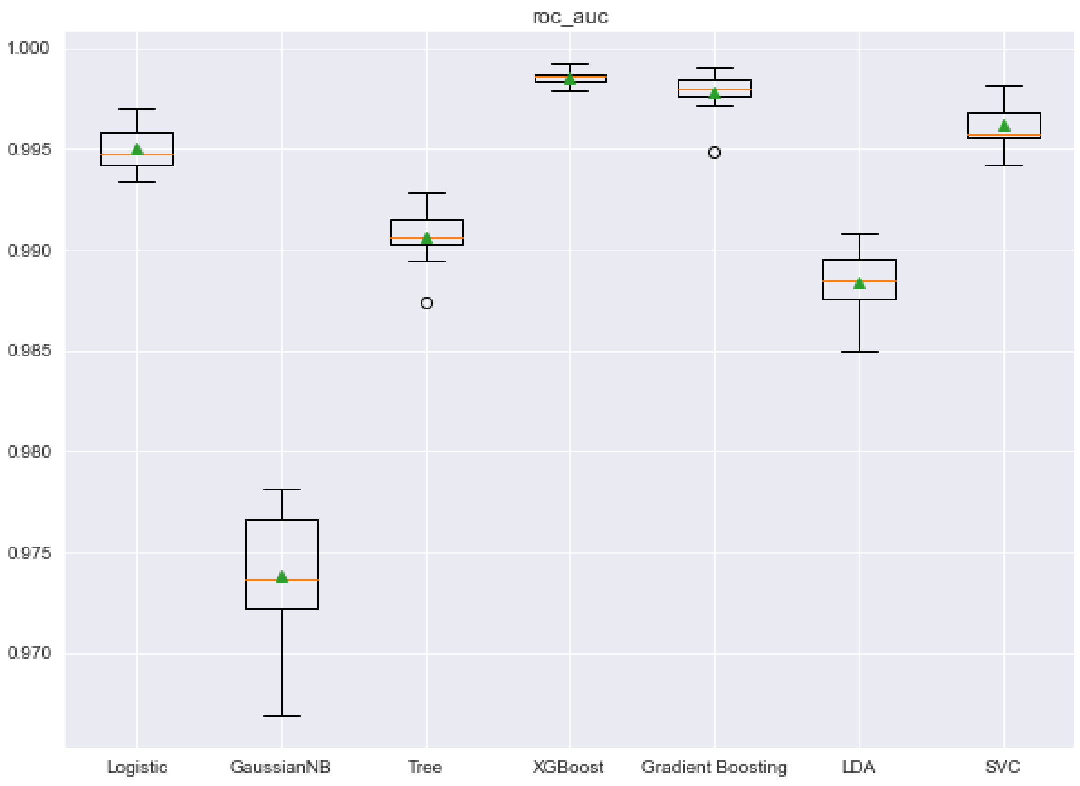
$$Weighted\ avg\ precision = \frac{11630 \times 0.8853 + 12870 \times 0.9534}{24500} = 0.921073$$

For each model, we would obtain the mean and standard deviation of the 10 folds for each metric. In addition to this, the results would be visualised as a boxplot.

[32]:
```
def display_results(results, names, scores):
    # Create a DataFrame with mean and std for each score
    df_results = pd.DataFrame()
    df_results['Model'] = names
    for score in scores:
        df_results['Mean ' + score] = [result[score].mean() for result in␣
↪results.values()]
        df_results['Std ' + score] = [result[score].std() for result in results.
↪values()]

    display(df_results)
    # Plot boxplot with means
    fig, axes = plt.subplots(len(scores), figsize=(10, 8 * len(scores)))
    for i, score in enumerate(scores):
        axes[i].set_title(score)
        axes[i].boxplot([result[score] for result in results.values()],␣
 ↪labels=names, showmeans=True)

    plt.show()

# Example usage
display_results(results, names, ['accuracy', 'roc_auc'])
```

```
              Model  Mean accuracy  Std accuracy  Mean roc_auc  Std roc_auc
0          Logistic       0.978082      0.003401      0.995008     0.001195
1        GaussianNB       0.918653      0.006893      0.973839     0.003292
2              Tree       0.991224      0.001186      0.990618     0.001406
3           XGBoost       0.994898      0.000899      0.998566     0.000335
4  Gradient Boosting       0.992939      0.001211      0.997783     0.001095
5               LDA       0.946939      0.004829      0.988385     0.001780
6               SVC       0.990531      0.001869      0.996192     0.001168
```


accuracy
1.00
0.98
0.96
0.94
0.92
Logistic
GaussianNB
Tree
XGBoost
Gradient Boosting
LDA
SVC


roc_auc
1.000
0.995
0.990
0.985
0.980
0.975
0.970
Logistic
GaussianNB
Tree
XGBoost
Gradient Boosting
LDA
SVC

From each fold, we obtain the accuracy, which is the percentage of well-classified news. Thus, we obtain the accuracy calculated 10 times and the accuracy associated with the model is the mean of these values, together with a measure of standard deviation. The latter allow us to know if the metric, accuracy in this case, has variability and relies on the sample.

In general, all the models have a great performance in terms of accuracy (more than 90%). The "worst" performance is related with the generative classifiers: LDA and GaussianNB. As a winner in many Kaggle competitions, XGBoost has the best performance with almost 100% of accuracy, followed closely by Gradient Boosting. In addition to this, the accuracy across cross-validation has the lowest variance.

It is true that XGBoost can be difficult to apply in real practice due to the high computational cost. This is why it can be better sometimes to consider other models that perform well, with less accuracy than XGBoost, but not so expensive computationally.

Regarding the area under the characteristic curve, we still have very similar results to the accuracy with XGBoost and Gradient Boosting as best models. However, there are some differences in the ranking regarding Tree, SVC and Logistic Regression compared to accuracy. There are different explanations such as dealing with imbalanced dataset (not this case), predicting the majority class correctly, or model's difficulty to separate labels.

In this case, Tree has a higher accuracy but a lower ROC AUC than Logistic, which indicates that Tree performs better in terms of overall correctness or accuracy in predicting the class labels. However, when it comes to the ability to distinguish between the positive and negative classes, Logistic outperforms Tree.

Hyperparameters search

Once we have selected the best model, XGBoost, for our machine learning task, it becomes crucial to fine-tune its performance by optimising its hyperparameters. Performing a hyperparameter search involves systematically exploring different combinations of hyperparameters to find the optimal configuration that maximises the model's performance on the given dataset.

```
[33]: from sklearn.model_selection import KFold
      from sklearn.model_selection import GridSearchCV
      # define models and parameters
      model = XGBClassifier()
      lr = [0.1, 0.01]
      md = [2, 3, 5]
      mcw = [1, 5]
      g = [0.5, 0.1, 0.01]
      l = ["reg:squarederror", "reg:squaredlogerror"]
```

Some XGBoost() hyperparameters we can tune are:

- learning_rate: It controls the step size shrinkage during each boosting iteration.
- max_depth: It specifies the maximum depth of a tree.

- min_child_weight: It defines the minimum sum of instance weight needed in a child node.
- gamma: It specifies the minimum loss reduction required to split a node further.
- objective: It determines the loss function to be optimised during training.

```
[34]: grid = dict(min_child_weight= mcw, max_depth = md, learning_rate=lr, gamma = g,␣
      ↪objective = l)
      cv = KFold(n_splits=5)
      grid_search = GridSearchCV(estimator=model, param_grid=grid, n_jobs=-1, cv=cv,␣
      ↪scoring='roc_auc',error_score=0)
```

```
[35]: grid_result = grid_search.fit(X_train, y_train)
      # summarise results
      print("Best ROC AUC: %f with parameters %s" % (grid_result.best_score_,␣
      ↪grid_result.best_params_))
```

```
Best ROC AUC: 0.998508 with parameters {'gamma': 0.1, 'learning_rate': 0.1,
'max_depth': 5, 'min_child_weight': 1, 'objective': 'reg:squaredlogerror'}
```

The probability that the model will assign a larger probability to a random true new than a random fake new is 0.998508, which is as an extraordinary value.

```
[36]: model=XGBClassifier(min_child_weight= 1, max_depth = 5, learning_rate=0.1, gamma␣
      ↪= 0.1, objective = "reg:squaredlogerror")
      model.fit(X_train,y_train)
```

```
[36]: XGBClassifier(base_score=None, booster=None, callbacks=None,
                    colsample_bylevel=None, colsample_bynode=None,
                    colsample_bytree=None, early_stopping_rounds=None,
                    enable_categorical=False, eval_metric=None, feature_types=None,
                    gamma=0.1, gpu_id=None, grow_policy=None, importance_type=None,
                    interaction_constraints=None, learning_rate=0.1, max_bin=None,
                    max_cat_threshold=None, max_cat_to_onehot=None,
                    max_delta_step=None, max_depth=5, max_leaves=None,
                    min_child_weight=1, missing=nan, monotone_constraints=None,
                    n_estimators=100, n_jobs=None, num_parallel_tree=None,
                    objective='reg:squaredlogerror', predictor=None, ...)
```

```
[37]: y_pred = model.predict(X_test)
```

```
[38]: cm=confusion_matrix(y_test,y_pred)
      print(cm)
      print(classification_report(y_test,y_pred,  digits=4))
```

```
[[4978    4]
 [  52 5466]]
              precision    recall  f1-score   support

         0.0     0.9897    0.9992    0.9944      4982
         1.0     0.9993    0.9906    0.9949      5518

    accuracy                         0.9947     10500
   macro avg     0.9945    0.9949    0.9947     10500
weighted avg     0.9947    0.9947    0.9947     10500
```

Interpretation

Precision refers to the proportion of correctly predicted positive instances among all the positive predictions. It can be seen as a normalisation of the confusion matrix by columns.

- Out of all the news that the model predicted to be true, "only" 98.97% actually were.
- Out of all the news that the model predicted to be fake, "only" 99.93% actually were.

Recall refers to the proportion of correctly predicted positive instances among all the actual positive instances. It can be seen as a normalisation of the confusion matrix by rows.

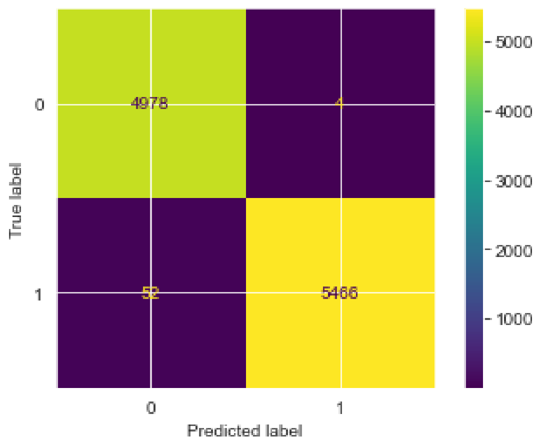
- Out of all the news that actually were true, the model predicted this outcome correctly for 99.92% of those news.
- Out of all the news that actually were fake, the model predicted this outcome correctly for 99.06% of those news.

Accuracy refers to the proportion of correct predictions compared to the total number of predictions.

We can conclude that the model correctly predicted the outcome for approximately 99.47% of the instances.

[39]:
```python
plt = ConfusionMatrixDisplay(cm)

# Show the plot
plt.plot()
```

[39]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2187b0d18b0>
```

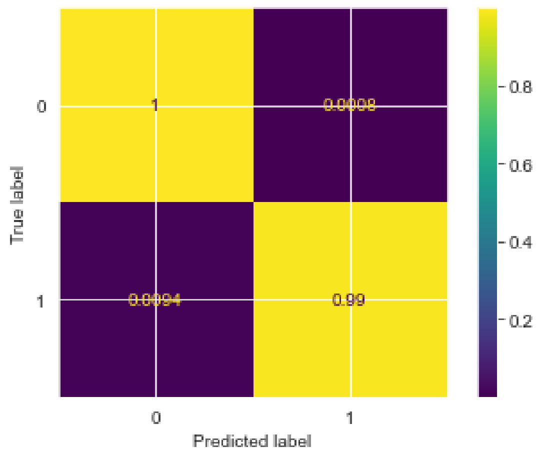


We can visualise that 4978 were correctly classified as true out of 4982 actual true news and that 5466 were correctly classified as fake news out of 5518 actual fake news.

Another insight is that 4978 were actual true news out of 5030 news classified as true and that 5466 were actual fake news out of 5470 reviews classified as fake.

```
[40]: cmn = confusion_matrix(y_test, y_pred, normalize = "true")
      plt = ConfusionMatrixDisplay(cmn)
      # if 'true', the confusion matrix is normalized over the true conditions (e.g.␣
       ↪rows);

      # Show the plot
      plt.plot()
```

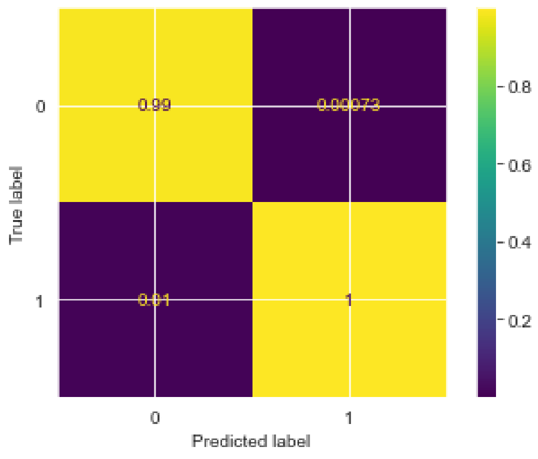
```
[40]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2181feaaaf0>
```



This matrix shows the recall for each class since the confusion matrix has been normalised by rows (true).

```
[41]: cmn = confusion_matrix(y_test, y_pred, normalize = "pred")
      plt = ConfusionMatrixDisplay(cmn)
      # if 'pred', the confusion matrix is normalized over the predicted conditions (e.
       ↪g. columns);

      # Show the plot
      plt.plot()
```

```
[41]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2181feaa0d0>
```



This matrix shows the precision for each class since the confusion matrix has been normalised by columns (predictions).

We obtained a classifier really accurate for both classes and thus, it can distinguish fake from real news.

Alternative

As mentioned previously, in practice, it can be difficult to use XGBoost due to the high computational cost. In this case, we could choose Logisitic Regression since the performance is really good and it is an easy algorithm to interpret the data.

[42]:
```
# define models and parameters
model = LogisticRegression()
solvers = ['newton-cg', 'liblinear']
penalty = ['l1','l2']
c_values = [100, 10, 1, 0.1]
```

Some LogisticRegression() hyperparameters we can tune are:

- solver: The solver determines the optimisation algorithm to be used in fitting the logistic regression model.
- penalty: This hyperparameter specifies the type of regularisation to be applied to the mode.
- C: controls the amount of regularisation applied.

[43]:
```
grid = dict(solver=solvers,penalty=penalty,C=c_values)
cv = KFold(n_splits=5)
grid_search = GridSearchCV(estimator=model, param_grid=grid, n_jobs=-1, cv=cv,␣
 ↪scoring='roc_auc',error_score=0)
```

[44]:
```
grid_result = grid_search.fit(X_train, y_train)
# summarize results
```

```
print("Best ROC AUC: %f with parameters %s" % (grid_result.best_score_,␣
↪grid_result.best_params_))
```

```
Best ROC AUC: 0.995253 with parameters {'C': 10, 'penalty': 'l2', 'solver':
'liblinear'}
```

The probability that the model will assign a larger probability to a random true new than a random fake new is 0.995253, which is as an extraordinary value.

```
[45]: model=LogisticRegression(C=10,penalty="l2",solver="liblinear")
      model.fit(X_train,y_train)
```

```
[45]: LogisticRegression(C=10, solver='liblinear')
```

```
[46]: y_pred = model.predict(X_test)
```

```
[47]: cm = confusion_matrix(y_test,y_pred)
      print(cm)
      print(classification_report(y_test,y_pred, digits = 4))
```

```
[[4909   73]
 [  55 5463]]
              precision    recall  f1-score   support

         0.0     0.9889    0.9853    0.9871      4982
         1.0     0.9868    0.9900    0.9884      5518

    accuracy                         0.9878     10500
   macro avg     0.9879    0.9877    0.9878     10500
weighted avg     0.9878    0.9878    0.9878     10500
```

Interpretations

Precision

- Out of all the news that the model predicted to be true, “only” 98.89% actually were.
- Out of all the news that the model predicted to be fake, “only” 98.68% actually were.

Recall

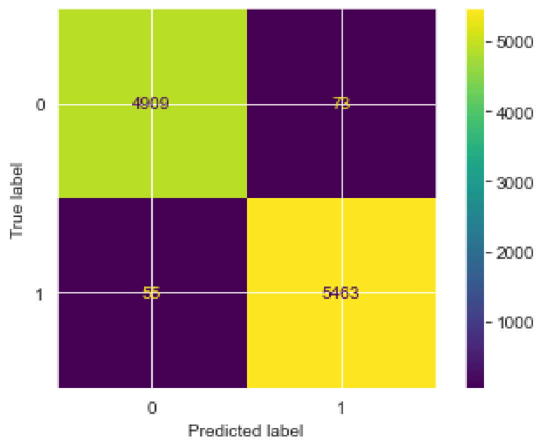
- Out of all the news that actually were positive, the model predicted this outcome correctly for 98.53% of those news.
- Out of all the news that actually were fake, the model predicted this outcome correctly for 99% of those news.

Accuracy

- We can conclude that the model correctly predicted the outcome for approximately 98.78% of the reviews.

```
[48]: plt = ConfusionMatrixDisplay(cm)

      # Show the plot
      plt.plot()
```

```
[48]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x21864a9e5e0>
```

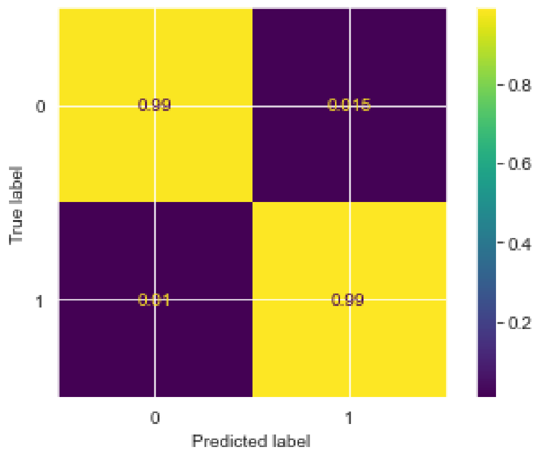


We can visualise that 4909 were correctly classified as true out of 4982 actual true news and that 5463 were correctly classified as fake news out of 5518 actual fake news.

Another insight is that 4909 were actual true news out of 4964 news classified as true and that 5463 were actual fake news out of 5536 reviews classified as fake.

```
[49]: cmn = confusion_matrix(y_test, y_pred, normalize = "true")
      plt = ConfusionMatrixDisplay(cmn)
      # if 'true', the confusion matrix is normalized over the true conditions (e.g.␣
       ↪rows);

      # Show the plot
      plt.plot()
```

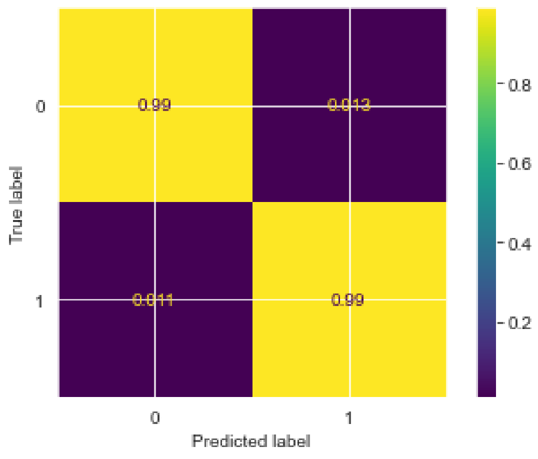
```
[49]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x21812d081f0>
```



This matrix shows the recall for each class since the confusion matrix has been normalised by rows (true).

```
[50]: cmn = confusion_matrix(y_test, y_pred, normalize = "pred")
plt = ConfusionMatrixDisplay(cmn)
# if 'true', the confusion matrix is normalized over the true conditions (e.g.␣
↪rows);

# Show the plot
plt.plot()
```

```
[50]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x218641313a0>
```



This matrix shows the precision for each class since the confusion matrix has been normalised by columns (predictions).

Conclusion

This Logistic Regression model also achieves extraordinary results and it is not as computationally expensive as XGBoost. Therefore, it can be selected as a final model to perform big data analysis.

Logistic Regression model performed exceptionally well and yielded highly accurate results (98.78%) on unseen data. In addition to this, both precision and recall are 0.99 for each class. Thereore, F1 score shows a robust classification performance.

Furthermore, ROC AUC value of 0.995253 indicates that the model exhibited excellent discriminative ability, effectively distinguishing between true and fake news with high confidence.

Note that in other practical examples, it can be more important to be more accurate for a certain label, and this can influence the metric to use and thus, the choice of the final model. It would depend on the objective of the task we are performing.

IMDb Reviews: Sentiment Analysis

Import Libraries

```
[1]: import matplotlib.pyplot as plt
     import warnings
     warnings.filterwarnings('ignore')
     import numpy as np
     import pandas as pd
     import seaborn as sns
     import scipy as sp
     from nltk.corpus import stopwords, wordnet
     from nltk.stem import WordNetLemmatizer
     from nltk.tokenize import word_tokenize, sent_tokenize
     from wordcloud import WordCloud
     from sklearn.model_selection import train_test_split, KFold, StratifiedKFold
     from sklearn.linear_model import LogisticRegression
     from sklearn.tree import DecisionTreeClassifier
     from sklearn.naive_bayes import GaussianNB
     from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
     from sklearn.svm import SVC
     from xgboost import XGBClassifier
     from sklearn.ensemble import GradientBoostingClassifier
     from sklearn.model_selection import cross_val_score, cross_val_predict,␣
      ↪GridSearchCV
     from sklearn.metrics import confusion_matrix, classification_report,␣
      ↪ConfusionMatrixDisplay
```

Load data

```
[2]: data0 = pd.read_csv('C:/Users/Amparo/OneDrive/Desktop/data/IMDB.csv')
     data0.head()
```

```
[2]:                                               review sentiment
     0  One of the other reviewers has mentioned that ...  positive
     1  A wonderful little production. <br /><br />The...  positive
     2  I thought this was a wonderful way to spend ti...  positive
     3  Basically there's a family where a little boy ...  negative
     4  Petter Mattei's "Love in the Time of Money" is...  positive
```

Preprocessing and Exploratory Data Analysis (EDA)

Remove NA values and check for duplicated news.

```
[3]: data0.dropna(inplace = True)
```

```
[4]: data0.duplicated().sum()
```

```
[4]: 418
```

There are 418 duplicated registers. Thus, delete all occurrences except the first.

```
[5]: data0.drop_duplicates(inplace=True)
```

```
[6]: print("We have a dataset with {} films reviews.".format(len(data0)))
```

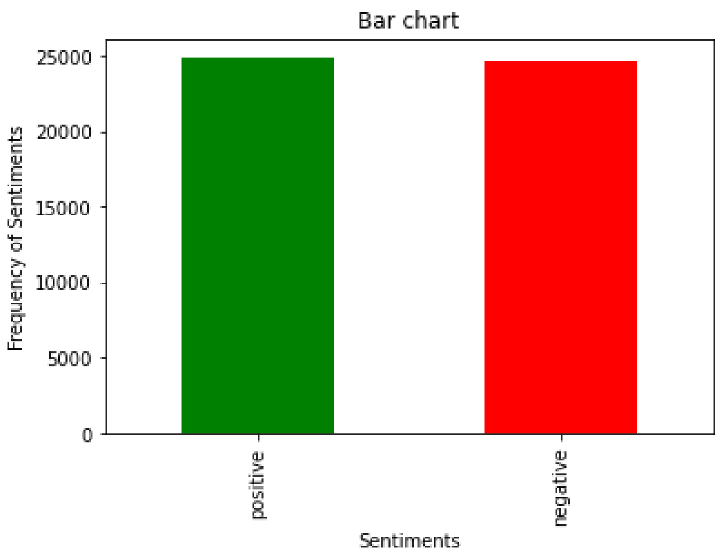
```
We have a dataset with 49582 films reviews.
```

```
[7]: print(data0["sentiment"].value_counts())
```

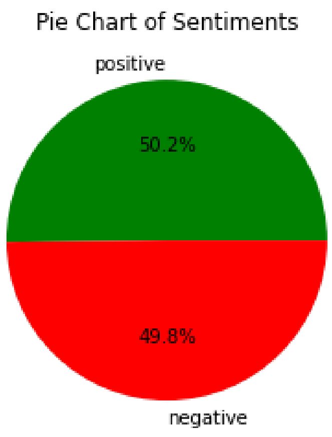
```
positive    24884
negative    24698
Name: sentiment, dtype: int64
```

We have a dataset with 49582 films reviews, whose sentiments are 24884 positive and 24698 negative.

```
[8]: ax, fig = plt.subplots()
     etiquetas = data0.sentiment.value_counts()
     etiquetas.plot(kind= 'bar', color= ["green", "red"])
     plt.title('Bar chart')
     plt.ylabel("Frequency of Sentiments")
     plt.xlabel("Sentiments")
     plt.show()
```



```
[9]: import matplotlib.pyplot as plt

     # Count the frequency of each sentiment category
     etiquetas = data0['sentiment'].value_counts()

     # Create a pie chart
     fig, ax = plt.subplots()
     ax.pie(etiquetas, labels=etiquetas.index, colors=[ "green", "red"], autopct='%1.
      ↪1f%%')
     ax.set_title('Pie Chart of Sentiments')

     # Display the pie chart
     plt.show()
```



We are working with a balanced dataset since there are approximately the same number of positive and negative reviews.

To reduce the computational cost of this task, we are going to select 40000 films reviews preserving this representation to work with a balanced dataset.

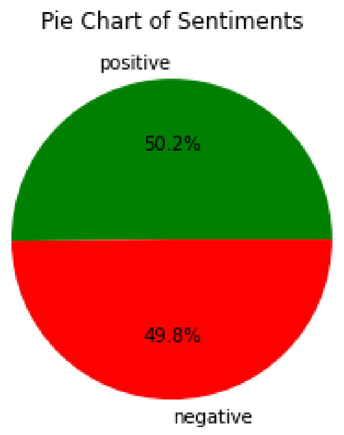
```
[10]: X = data0['review']
      y = data0['sentiment']

      # Use train_test_split for stratified sampling
      X_sampled, _, y_sampled, _ = train_test_split(X, y, train_size=40000,␣
       ↪stratify=y, random_state=42)

      # Combine the sampled features and target into a dataframe
      data = pd.DataFrame({'review': X_sampled, 'sentiment': y_sampled})

      data.head()
```

```
[10]:                                                    review sentiment
      512    If I had known this movie was filmed in the ex...  negative
      2729   The cinema of the 60s was as much as time of r...  positive
      40138  Las Vegas is very funny and focuses on the sub...  positive
      6465   This musical has a deep meaning which is appre...  positive
      12190  Delightful Disney film with Angela Lansbury in...  positive
```

```
[11]: print(data["sentiment"].value_counts())
```

```
positive    20075
negative    19925
Name: sentiment, dtype: int64
```

```
[12]: import matplotlib.pyplot as plt

      # Count the frequency of each sentiment category
      etiquetas = data['sentiment'].value_counts()

      # Create a pie chart
      fig, ax = plt.subplots()
      ax.pie(etiquetas, labels=etiquetas.index, colors=[ "green", "red"], autopct='%1.
       ↪1f%%')
      ax.set_title('Pie Chart of Sentiments')

      # Display the pie chart
      plt.show()
```



In this practical example, we are working with subjective opinions which have been extracted from IMDb website. This is the reason why it is vital to add some preprocessing step first. Here we have a list of abbreviatios that are commonly used in an informal text.

[13]:
```
# Thanks to https://www.kaggle.com/rftexas/text-only-kfold-bert
abbreviations = {
    "$" : " dollar ",
    "€" : " euro ",
    "4ao" : "for adults only",
    "a.m" : "before midday",
    "a3" : "anytime anywhere anyplace",
    "aamof" : "as a matter of fact",
    "acct" : "account",
    "adih" : "another day in hell",
    "afaic" : "as far as i am concerned",
    "afaict" : "as far as i can tell",
    "afaik" : "as far as i know",
    "afair" : "as far as i remember",
    "afk" : "away from keyboard",
    "app" : "application",
    "approx" : "approximately",
    "apps" : "applications",
    "asap" : "as soon as possible",
    "asl" : "age, sex, location",
    "atk" : "at the keyboard",
    "ave." : "avenue",
    "aymm" : "are you my mother",
    "ayor" : "at your own risk",
    "b&b" : "bed and breakfast",
    "b+b" : "bed and breakfast",
    "b.c" : "before christ",
    "b2b" : "business to business",
    "b2c" : "business to customer",
    "b4" : "before",
    "b4n" : "bye for now",
    "b@u" : "back at you",
    "bae" : "before anyone else",
    "bak" : "back at keyboard",
    "bbbg" : "bye bye be good",
    "bbc" : "british broadcasting corporation",
    "bbias" : "be back in a second",
    "bbl" : "be back later",
    "bbs" : "be back soon",
    "be4" : "before",
    "bfn" : "bye for now",
    "blvd" : "boulevard",
    "bout" : "about",
    "brb" : "be right back",
    "bros" : "brothers",
    "brt" : "be right there",
    "bsaaw" : "big smile and a wink",
    "btw" : "by the way",
    "bwl" : "bursting with laughter",
    "c/o" : "care of",
    "cet" : "central european time",
    "cf" : "compare",
    "cia" : "central intelligence agency",
    "csl" : "can not stop laughing",
    "cu" : "see you",
    "cul8r" : "see you later",
    "cv" : "curriculum vitae",
    "cwot" : "complete waste of time",
    "cya" : "see you",
    "cyt" : "see you tomorrow",
    "dae" : "does anyone else",
    "dbmib" : "do not bother me i am busy",
    "diy" : "do it yourself",
    "dm" : "direct message",
    "dwh" : "during work hours",
    "e123" : "easy as one two three",
    "eet" : "eastern european time",
    "eg" : "example",
    "embm" : "early morning business meeting",
    "encl" : "enclosed",
    "encl." : "enclosed",
    "etc" : "and so on",
    "faq" : "frequently asked questions",
    "fawc" : "for anyone who cares",
    "fb" : "facebook",
    "fc" : "fingers crossed",
    "fig" : "figure",
    "fimh" : "forever in my heart",
    "ft." : "feet",
    "ft" : "featuring",
    "ftl" : "for the loss",
    "ftw" : "for the win",
    "fwiw" : "for what it is worth",
    "fyi" : "for your information",
    "g9" : "genius",
    "gahoy" : "get a hold of yourself",
    "gal" : "get a life",
    "gcse" : "general certificate of secondary education",
    "gfn" : "gone for now",
    "gg" : "good game",
    "gl" : "good luck",
    "glhf" : "good luck have fun",
    "gmt" : "greenwich mean time",
    "gmta" : "great minds think alike",
    "gn" : "good night",
    "g.o.a.t" : "greatest of all time",
    "goat" : "greatest of all time",
    "goi" : "get over it",
    "gps" : "global positioning system",
    "gr8" : "great",
    "gratz" : "congratulations",
    "gyal" : "girl",
    "h&c" : "hot and cold",
    "hp" : "horsepower",
    "hr" : "hour",
    "hrh" : "his royal highness",
    "ht" : "height",
    "ibrb" : "i will be right back",
    "ic" : "i see",
    "icq" : "i seek you",
    "icymi" : "in case you missed it",
    "idc" : "i do not care",
    "idgadf" : "i do not give a damn fuck",
    "idgaf" : "i do not give a fuck",
    "idk" : "i do not know",
    "ie" : "that is",
    "i.e" : "that is",
    "ifyp" : "i feel your pain",
    "IG" : "instagram",
    "iirc" : "if i remember correctly",
    "ilu" : "i love you",
    "ily" : "i love you",
    "imho" : "in my humble opinion",
    "imo" : "in my opinion",
    "imu" : "i miss you",
    "iow" : "in other words",
    "irl" : "in real life",
    "j4f" : "just for fun",
    "jic" : "just in case",
    "jk" : "just kidding",
    "jsyk" : "just so you know",
    "l8r" : "later",
    "lb" : "pound",
    "lbs" : "pounds",
    "ldr" : "long distance relationship",
    "lmao" : "laugh my ass off",
    "lmfao" : "laugh my fucking ass off",
    "lol" : "laughing out loud",
    "ltd" : "limited",
    "ltns" : "long time no see",
    "m8" : "mate",
    "mf" : "motherfucker",
    "mfs" : "motherfuckers",
    "mfw" : "my face when",
    "mofo" : "motherfucker",
    "mph" : "miles per hour",
    "mr" : "mister",
    "mrw" : "my reaction when",
    "ms" : "miss",
    "mte" : "my thoughts exactly",
    "nagi" : "not a good idea",
    "nbc" : "national broadcasting company",
    "nbd" : "not big deal",
    "nfs" : "not for sale",
    "ngl" : "not going to lie",
    "nhs" : "national health service",
    "nrn" : "no reply necessary",
    "nsfl" : "not safe for life",
    "nsfw" : "not safe for work",
    "nth" : "nice to have",
    "nvr" : "never",
    "nyc" : "new york city",
    "oc" : "original content",
    "og" : "original",
    "ohp" : "overhead projector",
    "oic" : "oh i see",
    "omdb" : "over my dead body",
    "omg" : "oh my god",
    "omw" : "on my way",
    "p.a" : "per annum",
    "p.m" : "after midday",
    "pm" : "prime minister",
    "poc" : "people of color",
    "pov" : "point of view",
    "pp" : "pages",
    "ppl" : "people",
    "prw" : "parents are watching",
    "ps" : "postscript",
    "pt" : "point",
    "ptb" : "please text back",
    "pto" : "please turn over",
    "qpsa" : "what happens",
    "ratchet" : "rude",
    "rbtl" : "read between the lines",
    "rlrt" : "real life retweet",
    "rofl" : "rolling on the floor laughing",
    "roflol" : "rolling on the floor laughing out loud",
    "rotflmao" : "rolling on the floor laughing my ass off",
    "rt" : "retweet",
    "ruok" : "are you ok",
    "sfw" : "safe for work",
    "sk8" : "skate",
    "smh" : "shake my head",
    "sq" : "square",
    "srsly" : "seriously",
    "ssdd" : "same stuff different day",
    "tbh" : "to be honest",
    "tbs" : "tablespooful",
    "tbsp" : "tablespooful",
    "tfw" : "that feeling when",
    "thks" : "thank you",
    "tho" : "though",
    "thx" : "thank you",
    "tia" : "thanks in advance",
    "til" : "today i learned",
    "tl;dr" : "too long i did not read",
    "tldr" : "too long i did not read",
    "tmb" : "tweet me back",
    "tntl" : "trying not to laugh",
    "ttyl" : "talk to you later",
    "u" : "you",
    "u2" : "you too",
    "u4e" : "yours for ever",
    "utc" : "coordinated universal time",
    "w/" : "with",
    "w/o" : "without",
    "w8" : "wait",
    "wassup" : "what is up",
    "wb" : "welcome back",
    "wtf" : "what the fuck",
    "wtg" : "way to go",
    "wtpa" : "where the party at",
    "wuf" : "where are you from",
    "wuzup" : "what is up",
    "wywh" : "wish you were here",
    "yd" : "yard",
    "ygtr" : "you got that right",
    "ynk" : "you never know",
    "zzz" : "sleeping bored and tired"
}
```

[14]:
```
# Thanks to https://www.kaggle.com/rftexas/text-only-kfold-bert
def convert_abbrev(word):
    return abbreviations[word.lower()] if word.lower() in abbreviations.keys()␣
 ↪else word

def convert_abbrev_in_text(text):
    tokens = word_tokenize(text)
    tokens = [convert_abbrev(word) for word in tokens]
    text = ' '.join(tokens)
    return text

# convert abbreviations
data['review'] = data['review'].apply(lambda x: convert_abbrev_in_text(x))
```

In addition to this abbreviation conversion, we have noticed visualising the dataframe that there are html break tags since these reviews have been extracted from a website. Therefore, it is convenient to remove them.

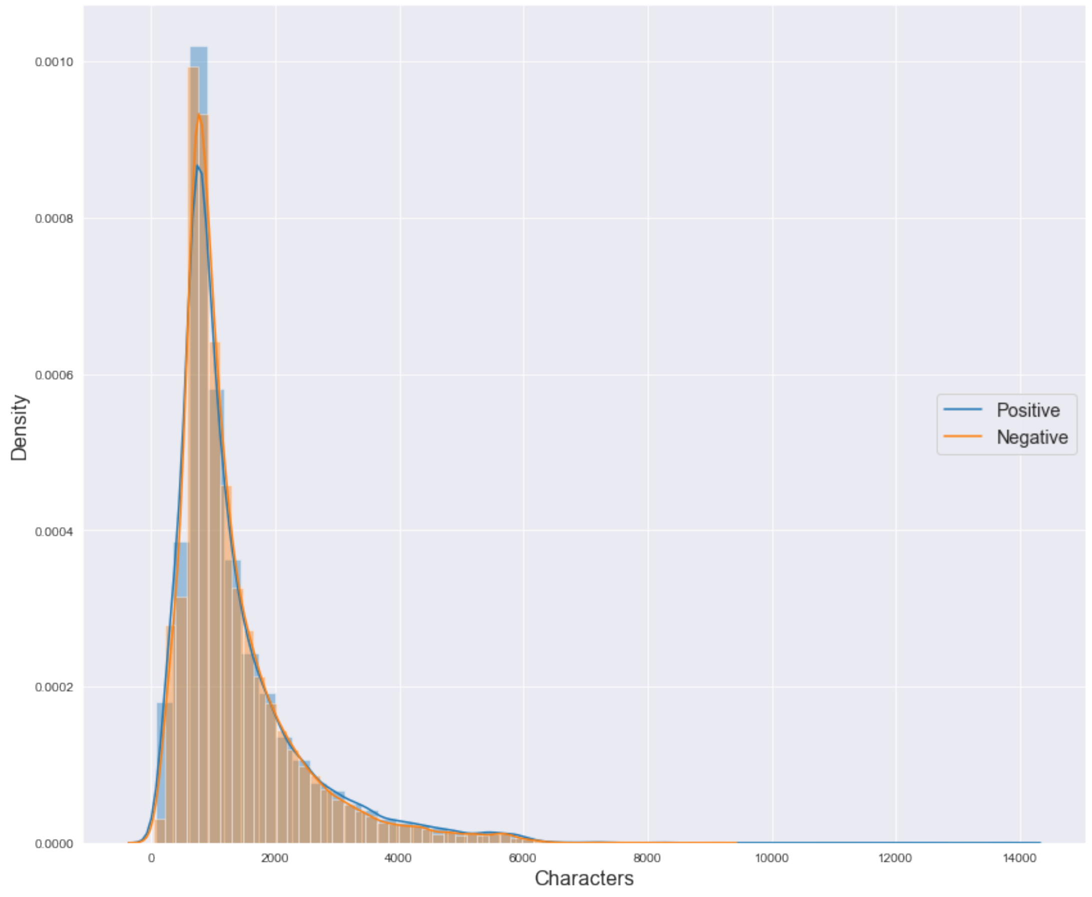
[15]:
```
# remove html tags
data['review'] = data['review'].apply(lambda x: x.replace('< br / >', ''))
```

We can calculate some extra features such as the number of words and number of characters per review to do some exploratory data analysis.

[16]:
```
data['word_count'] = data['review'].apply(lambda x: len(str(x).split()))
data["char_len"] = data["review"].apply(lambda x: len(x))
```

[17]:
```
fig = plt.figure(figsize=(14,12))
sns.set_style("darkgrid")

plt0 = sns.distplot(data[data.sentiment=="positive"].char_len, hist=True,␣
 ↪label="Positive")
```

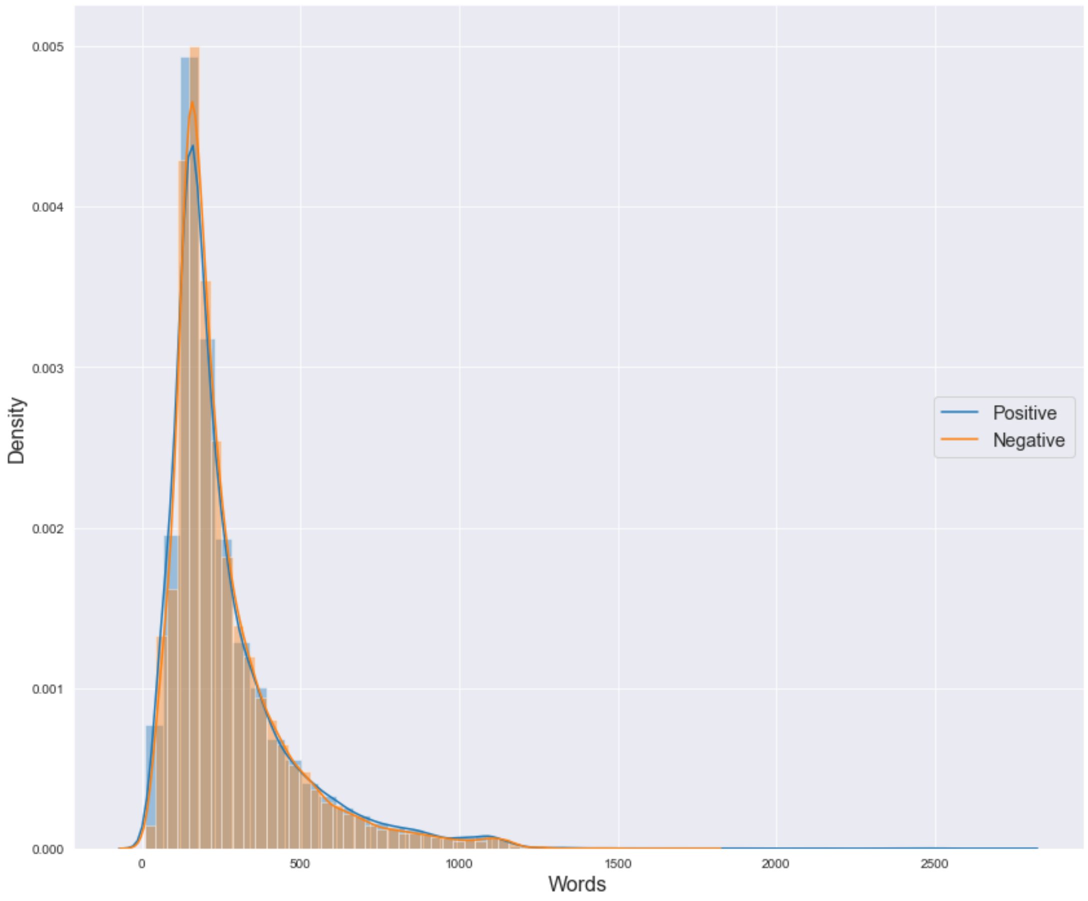
```
plt1 = sns.distplot(data[data.sentiment=="negative"].char_len, hist=True,␣
 ↪label="Negative")
plt.legend(labels=["Positive","Negative"], loc = 'center right', fontsize =␣
 ↪"x-large")


# Axis labels
plt.xlabel('Characters', fontsize=16)
plt.ylabel('Density', fontsize=16)

plt.show()
```



All the classes are right skewed around 1000 characters. Since they are very similar, we have to consider extra features.

[18]:
```
fig = plt.figure(figsize=(14,12))
sns.set_style("darkgrid")
```

```
plt0 = sns.distplot(data[data.sentiment=="positive"].word_count, hist=True,␣
 ↪label="Positive")
plt1 = sns.distplot(data[data.sentiment=="negative"].word_count, hist=True,␣
 ↪label="Negative")
plt.legend(labels=["Positive","Negative"], loc = 'center right', fontsize =␣
 ↪"x-large")


# Axis labels
plt.xlabel('Words', fontsize=16)
plt.ylabel('Density', fontsize=16)

plt.show()
```



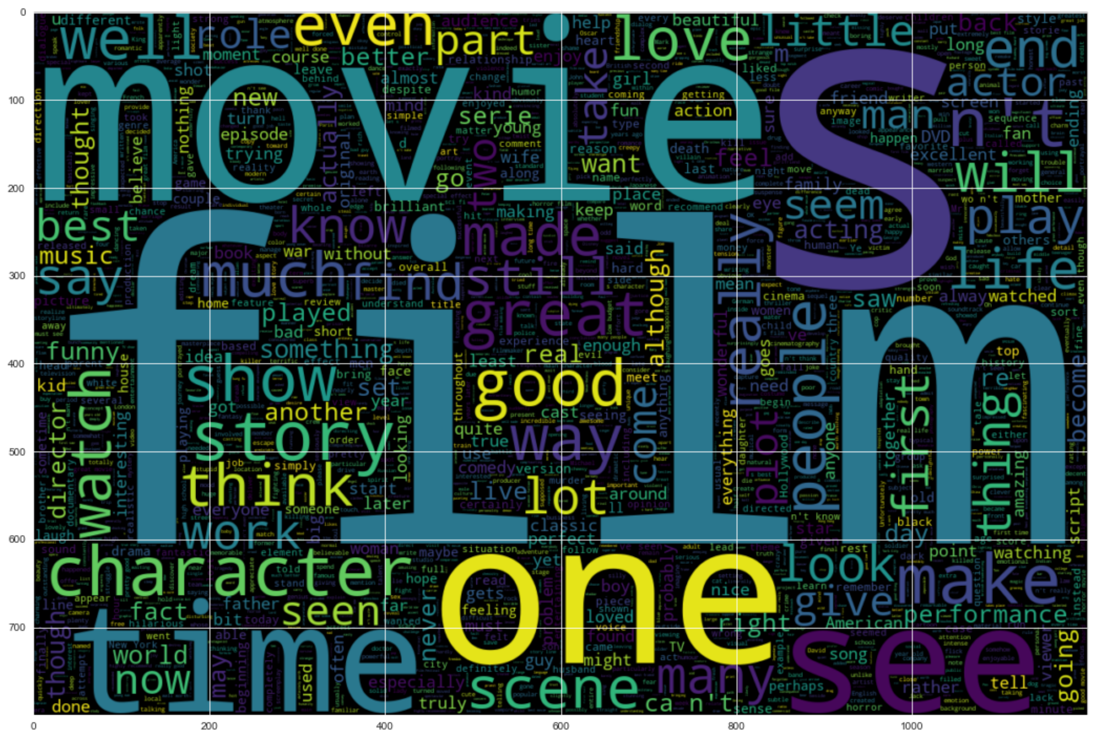
Both the number of words and the number of characters are distributed approximately equal across the positive and negative reviews. Therefore, these cannot be considered a determining feature.

WordCloud provides a quick and intuitive way to identify the most frequent terms within a text.

In this case, we are going to construct a WordCloud for both the positive and negative reviews to visualise simmilarities and differences.

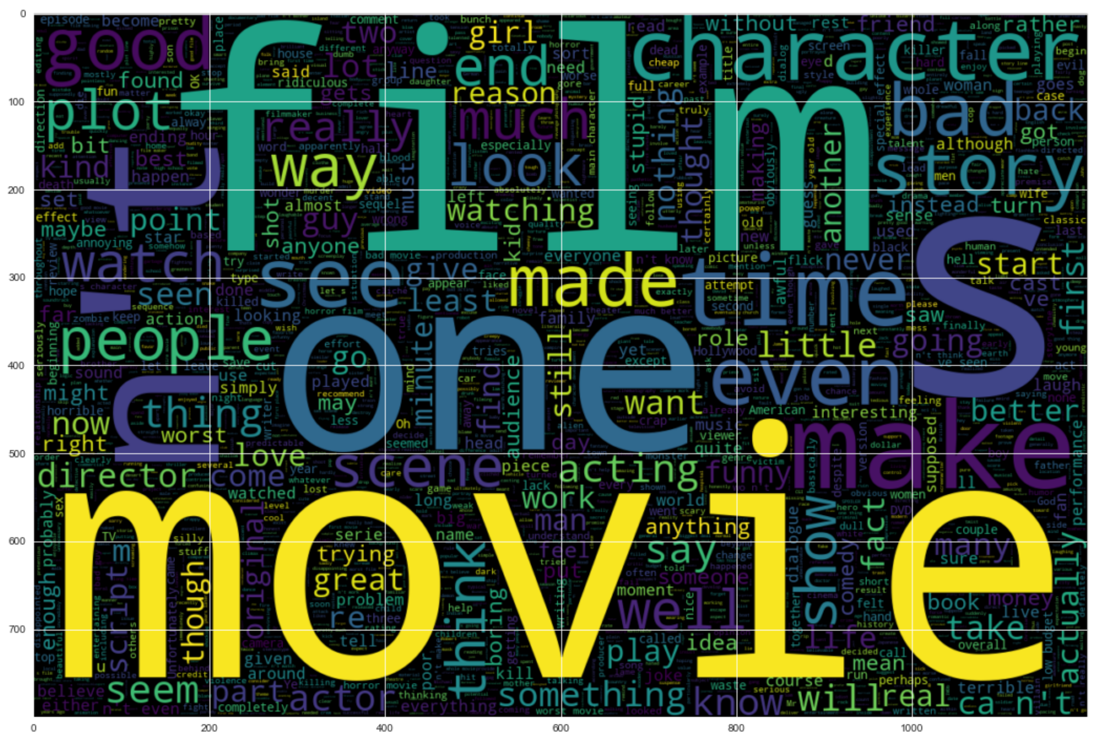
```
[19]: plt.figure(figsize = (18,24)) # Text Reviews with positive sentiment
      wordcloud = WordCloud(min_font_size = 3,  max_words = 2500 , width = 1200 ,
                            height = 800).generate(" ".join(data[data['sentiment'] ==␣
        ↪'positive']['review']))
      plt.imshow(wordcloud,interpolation = 'bilinear')
```

```
[19]: <matplotlib.image.AxesImage at 0x1bb9be0d910>
```



```
[20]: plt.figure(figsize = (18,24)) # Text Reviews with negative sentiment
      wordcloud = WordCloud(min_font_size = 3,  max_words = 2500 , width = 1200 ,
                            height = 800).generate(" ".join(data[data['sentiment'] ==␣
        ↪'negative']['review']))
      plt.imshow(wordcloud,interpolation = 'bilinear')
```

```
[20]: <matplotlib.image.AxesImage at 0x1bb9bde3430>
```



Both WordCloud share common words such as film, movie, time and character among others, which are frequent words in the cinema world. Regarding subjective connotations, we can appreciate funny, good and great, in contrast to bad and terrible in the negative side.

Next, we continue with the preprocessing step: space, punctuation and stopwords removal, lowercasing, tokenisation and stemming.

```
[21]: # Delete spaces
def delete_spaces(text):
    return  " ".join(text.split())

# To lower
def texto_to_lower(text):
    return text.lower()

# Tokenisation
from nltk import word_tokenize
def tokenisation(text):
    tokens = word_tokenize(text)
    return tokens

# Once text has been tokenised, we can create functions that take tokens as␣
 ↪inputs and apply further preprocessing.
```

```
# Remove stop words
from nltk.corpus import stopwords
def stopwords_removal(tokens):
    stop_words = set(stopwords.words('english'))
    filtered_sentence = [w for w in tokens if not w in stop_words]
    return filtered_sentence

# Remove punctuation (only alphanumeric values are allowed)
def punctuation_removal(tokens):
    words=[word for word in tokens if word.isalnum()]
    return words

# Stemming
import string
from nltk.stem import PorterStemmer
stemmer = PorterStemmer()
def stem(tokens):
    tokens = [stemmer.stem(token) for token in tokens]
    return tokens
```

We define preprocessing functions to be applied to the text: lowercasing, tokenisation, stopwords and punctuation removal, and stemming.

```
[22]: def preprocess(sentence):
          sentence = delete_spaces(sentence)
          sentence = texto_to_lower(sentence)
          sentence = tokenisation(sentence)
          sentence = stopwords_removal(sentence)
          sentence = punctuation_removal(sentence)
          sentence = stem(sentence)
          return sentence

      # Store in a new column "normalise" of the dataframe to keep the original data␣
       ↪too.
      data["tokens_norm"] = data["review"].apply(lambda x: preprocess(x))
```

```
[23]: data.head()
```

```
[23]:                                                   review sentiment  \
      512    If I had known this movie was filmed in the ex...  negative
      2729   The cinema of the 60s was as much as time of r...  positive
      40138  Las Vegas is very funny and focuses on the sub...  positive
      6465   This musical has a deep meaning which is appre...  positive
      12190  Delightful Disney film with Angela Lansbury in...  positive

             word_count  char_len                                        tokens_norm
      512           154       759  [known, movi, film, exasper, dogm, 95, style, ...
      2729          327      1737  [cinema, 60, much, time, revolut, polit, music...
```

```
40138          117       614  [la, vega, funni, focus, substanc, set, amaz, ...
6465           214      1089  [music, deep, mean, appreci, wise, see, wisdom...
12190          187       986  [delight, disney, film, angela, lansburi, fine...
```

[24]:
```
data["text_normalised"] = data["tokens_norm"].apply(lambda x: " ".join(x))
data.head()
```

[24]:
```
                                                   review sentiment  \
512    If I had known this movie was filmed in the ex...  negative
2729   The cinema of the 60s was as much as time of r...  positive
40138  Las Vegas is very funny and focuses on the sub...  positive
6465   This musical has a deep meaning which is appre...  positive
12190  Delightful Disney film with Angela Lansbury in...  positive

       word_count  char_len  \
512           154       759
2729          327      1737
40138         117       614
6465          214      1089
12190         187       986

                                             tokens_norm  \
512    [known, movi, film, exasper, dogm, 95, style, ...
2729   [cinema, 60, much, time, revolut, polit, music...
40138  [la, vega, funni, focus, substanc, set, amaz, ...
6465   [music, deep, mean, appreci, wise, see, wisdom...
12190  [delight, disney, film, angela, lansburi, fine...

                                         text_normalised
512    known movi film exasper dogm 95 style would ne...
2729   cinema 60 much time revolut polit music filmma...
40138  la vega funni focus substanc set amaz scene ou...
6465   music deep mean appreci wise see wisdom contai...
12190  delight disney film angela lansburi fine form ...
```

TextBlob is a library that provides a simple and intuitive API for NLP tasks built on top of NLTK. One of its key features is sentiment analysis capability.

The sentiment property returns a namedtuple of the form Sentiment(polarity, subjectivity). The polarity score ranges from -1 to 1, where -1 indicates a highly negative sentiment, 1 indicates a highly positive sentiment and 0 a neutral sentiment.

The subjectivity is a float within the range $[0, 1]$ where 0 is very objective and 1 is very subjective.

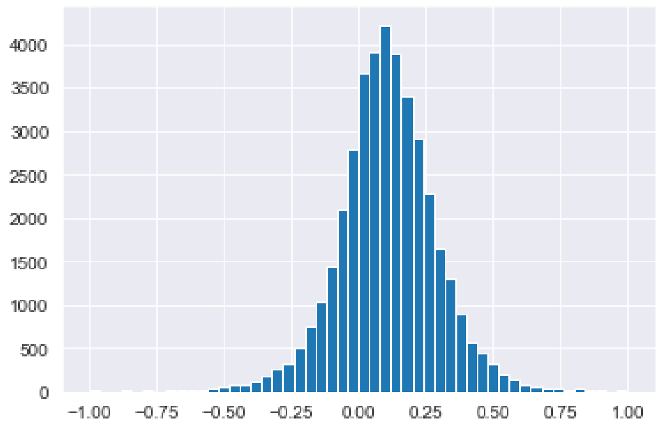
This is very useful for sentiment analysis.

[25]:
```
# pip install textblob
from textblob import TextBlob
```

```
data["sent_subjectivity"] = data["text_normalised"].apply(lambda x: TextBlob(x).
↪sentiment.subjectivity)
data["sent_polarity"] = data["text_normalised"].apply(lambda x: TextBlob(x).
↪sentiment.polarity)
```

```
[26]: data["sent_polarity"].hist(bins=50)
```

```
[26]: <AxesSubplot:>
```



Sentiment polarity seems to have a symmetric distribution around 0.15, whose frequency is the highest in this point and then, it decreases along the tails (similar to the shape of a normal distribution). This reflects that only some reviews have extreme sentiments, with positive sentiment as the most extreme one (0.75).

Vectorisation of text

GloVe, short for Global Vectors, is a word representation model that aims to capture distributed representations of words. It is an unsupervised learning algorithm designed to assign vector representations to words. The model maps words into a meaningful space, where the proximity between words reflects their semantic similarity.

To perform sentiment analysis on film reviews, we used the pre-trained word embeddings from the GloVe Twitter 25-dimensional model. This was achieved by importing the gensim.downloader module and using the load function from the api submodule.

The GloVe embeddings are powerful representations that capture the semantic meaning of words based on their contextual usage in a large corpus of Twitter text. Each word is transformed into a dense numerical vector of 25 dimensions, allowing us to analyse the sentiment expressed in the film reviews.

By loading the glove-twitter-25 model using api.load('glove-twitter-25'), we obtain access to the pre-trained word embeddings. These embeddings can then be used to represent the words in the film reviews as numerical vectors. This numerical representation enables us to apply various machine learning techniques to classify the sentiment of the reviews accurately.

The usage of pre-trained word embeddings saves us the time and computational resources required to train our own embeddings from scratch. It also benefits from the vast amount of data and context captured in the GloVe Twitter model, which makes it well-suited for sentiment analysis in the film review domain.

```
[27]: import gensim.downloader as api
      glove_emb = api.load('glove-twitter-25')
```

This get_average_vector() function allows us to compute the average vector representation for a given list of tokens. This way, we obtain a 25 dimensional vector representation for each review.

```
[28]: def get_average_vector(tokens):
          l = list()
          for i in tokens:
              try:
                  l.append(glove_emb.get_vector(i))
              except:
                  continue
          try:
              result = np.mean(l, axis=0)
          except:
              result = np.zeros(25)
          return result
```

```
[29]: data["embeddings"] = data["tokens_norm"].apply(lambda x: get_average_vector(x))
```

```
[30]: data.head()
```

```
[30]:                                                 review sentiment  \
      512    If I had known this movie was filmed in the ex...  negative
      2729   The cinema of the 60s was as much as time of r...  positive
      40138  Las Vegas is very funny and focuses on the sub...  positive
      6465   This musical has a deep meaning which is appre...  positive
      12190  Delightful Disney film with Angela Lansbury in...  positive

             word_count  char_len  \
      512           154       759
      2729          327      1737
      40138         117       614
      6465          214      1089
      12190         187       986

                                                   tokens_norm  \
      512    [known, movi, film, exasper, dogm, 95, style, ...
      2729   [cinema, 60, much, time, revolut, polit, music...
      40138  [la, vega, funni, focus, substanc, set, amaz, ...
      6465   [music, deep, mean, appreci, wise, see, wisdom...
      12190  [delight, disney, film, angela, lansburi, fine...
```

```
                                        text_normalised  sent_subjectivity  \
512    known movi film exasper dogm 95 style would ne...           0.241667
2729   cinema 60 much time revolut polit music filmma...           0.329634
40138  la vega funni focus substanc set amaz scene ou...           0.524286
6465   music deep mean appreci wise see wisdom contai...           0.563393
12190  delight disney film angela lansburi fine form ...           0.548512

       sent_polarity                                         embeddings
512         0.031250  [-0.23297037, 0.24585672, 0.090983965, -0.1447...
2729        0.030716  [-0.20859282, 0.171421, -0.007602942, -0.12960...
40138       0.161429  [-0.24537817, 0.21428183, 0.14903583, -0.08302...
6465        0.211445  [-0.14868726, 0.16624884, -0.1804222, 0.009106...
12190       0.189665  [-0.21085294, 0.12334834, -0.008020849, -0.081...
```

```
[31]: vector_data = pd.concat([data.embeddings.apply(pd.Series),
                     data[["sent_polarity","sent_subjectivity"]]], axis=1)
```

We have transformed text to a numerical representation with 27 features, which corresponds to word embeddings and sentiment polarity and subjectivity, for the 40000 sampled reviews.

```
[32]: vector_data.shape
```

```
[32]: (40000, 27)
```

```
[33]: vector_data.head()
```

```
[33]:             0         1         2         3         4         5         6  \
512   -0.232970  0.245857  0.090984 -0.144705  0.002647  0.000857  0.482371
2729  -0.208593  0.171421 -0.007603 -0.129600  0.110268 -0.023035  0.636843
40138 -0.245378  0.214282  0.149036 -0.083023  0.009157 -0.034927  0.397359
6465  -0.148687  0.166249 -0.180422  0.009106  0.051586  0.000221  0.664426
12190 -0.210853  0.123348 -0.008021 -0.081537 -0.124037 -0.165585  0.529820

             7         8         9  ...        17        18        19  \
512   -0.420550  0.034915 -0.096237  ... -0.147742  0.012249 -0.025490
2729  -0.443005  0.059925 -0.331034  ... -0.092161  0.067579 -0.143954
40138 -0.172392  0.002778 -0.344967  ... -0.335653 -0.066072  0.041860
6465  -0.308628  0.023470 -0.235530  ... -0.042695  0.336808 -0.323844
12190 -0.316194  0.119807 -0.311201  ... -0.039619 -0.059760  0.021153

            20        21        22        23        24  sent_polarity  \
512   -0.152912  0.348241  0.240295 -0.099608 -0.409727       0.031250
2729  -0.150108  0.188386 -0.174485 -0.064974 -0.320691       0.030716
40138  0.200282  0.115927  0.072215  0.059028 -0.308877       0.161429
6465  -0.166169  0.004295 -0.113732  0.017440 -0.232162       0.211445
12190 -0.168560  0.223524 -0.462363 -0.261190 -0.234969       0.189665
```

```
       sent_subjectivity
512             0.241667
2729            0.329634
40138           0.524286
6465            0.563393
12190           0.548512

[5 rows x 27 columns]
```

[34]:
```
vector_data = vector_data.fillna(0)
```

Now, we define a function to map the sentiments into numeric values.

[35]:
```
def map_sentiment(sentiment):
    if sentiment == "positive":
        return 0
    elif sentiment == "negative":
        return 1
    else:
        return None  # Handle any other cases if necessary

# Apply the function to create the numeric_sentiment column
data['numeric_sentiment'] = data['sentiment'].apply(lambda x: map_sentiment(x))
```

[36]:
```
X = sp.sparse.csc_matrix(vector_data)
y = data["numeric_sentiment"]
```

Machine Learning

Split the dataset into a training and test sets (70%-30% respectively).

[37]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=27,␣
 ↪test_size= 0.3)
print(X_train.shape)
print(X_test.shape)
```

```
(28000, 27)
(12000, 27)
```

We have obtained a training sample of 28000 film reviews and a test set of 12000 film reviews.

We define a function to perform cross validation with 10 folds across different models, which returns the final confusion matrix and the results for the given metrics.

[38]:
```
def model_evaluation(models, scores, X, y):
    global results
    global names
    results = {}
    names = []
    # For each model
    for name, model in models:

        KF = KFold(n_splits=10, shuffle=True, random_state=98)
        model_results = {}

        # Cross-validation
        for score in scores:
            cv_results = cross_val_score(model, X, y, cv=KF, scoring=score,␣
 ↪verbose=False)
            model_results[score] = cv_results

        y_pred = cross_val_predict(model, X, y, cv=KF)
        print(name)
        print(confusion_matrix(y, y_pred))
        print(classification_report(y, y_pred, digits=4))

        # Store results
        results[name] = model_results
        names.append(name)
```

[39]:
```
## Define the models
models = [
         ("Logistic", LogisticRegression()),
         ("GaussianNB",GaussianNB()),
         ("Tree", DecisionTreeClassifier()),
         ("XGBoost", XGBClassifier()),
         ("Gradient Boosting", GradientBoostingClassifier()),
         ("LDA", LinearDiscriminantAnalysis()),
         ("SVC", SVC())
]

scores = ["accuracy", "roc_auc"]

model_evaluation(models,  scores, X_train.toarray(), y_train)
```

```
Logistic
[[10468  3542]
 [ 3608 10382]]
              precision    recall  f1-score   support

           0     0.7437    0.7472    0.7454     14010
           1     0.7456    0.7421    0.7439     13990

    accuracy                         0.7446     28000
   macro avg     0.7446    0.7446    0.7446     28000
weighted avg     0.7446    0.7446    0.7446     28000
```

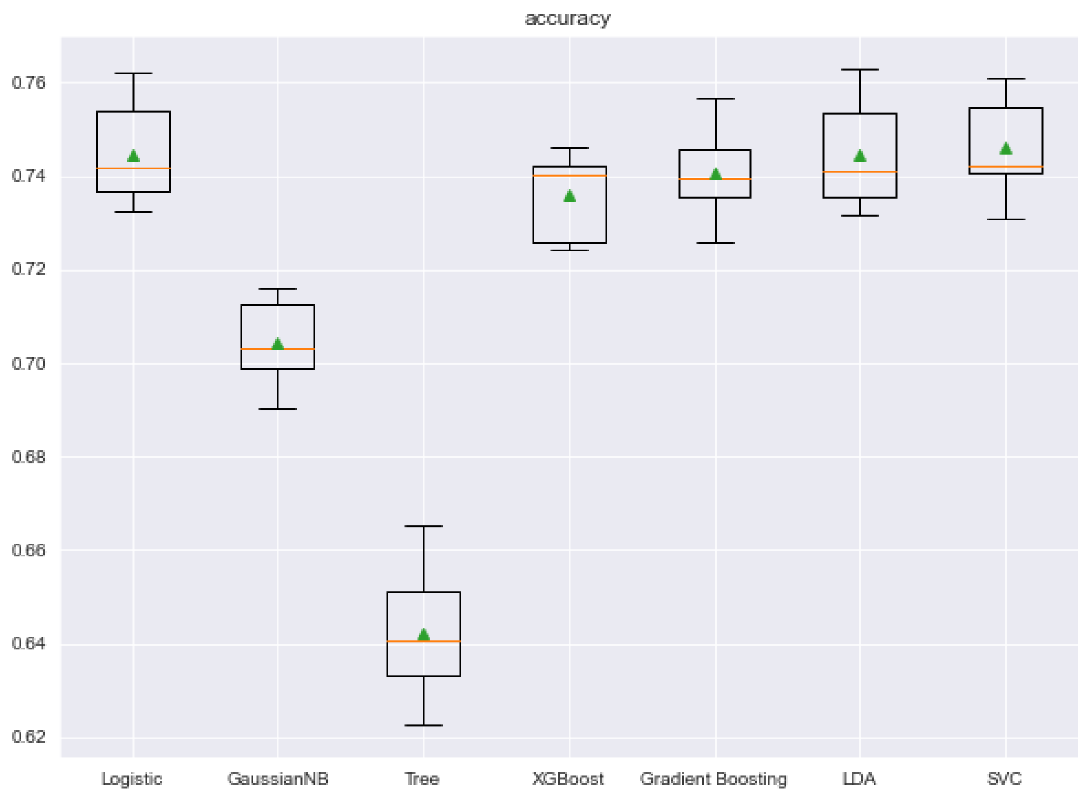
```
GaussianNB
[[ 9595  4415]
 [ 3862 10128]]
              precision    recall  f1-score   support

           0     0.7130    0.6849    0.6987     14010
           1     0.6964    0.7239    0.7099     13990

    accuracy                         0.7044     28000
   macro avg     0.7047    0.7044    0.7043     28000
weighted avg     0.7047    0.7044    0.7043     28000

Tree
[[8970 5040]
 [4978 9012]]
              precision    recall  f1-score   support

           0     0.6431    0.6403    0.6417     14010
           1     0.6413    0.6442    0.6428     13990

    accuracy                         0.6422     28000
   macro avg     0.6422    0.6422    0.6422     28000
weighted avg     0.6422    0.6422    0.6422     28000

XGBoost
[[10398  3612]
 [ 3782 10208]]
              precision    recall  f1-score   support

           0     0.7333    0.7422    0.7377     14010
           1     0.7386    0.7297    0.7341     13990

    accuracy                         0.7359     28000
   macro avg     0.7360    0.7359    0.7359     28000
weighted avg     0.7360    0.7359    0.7359     28000

Gradient Boosting
[[10556  3454]
 [ 3812 10178]]
              precision    recall  f1-score   support

           0     0.7347    0.7535    0.7440     14010
           1     0.7466    0.7275    0.7369     13990

    accuracy                         0.7405     28000
   macro avg     0.7407    0.7405    0.7405     28000
weighted avg     0.7407    0.7405    0.7405     28000
```

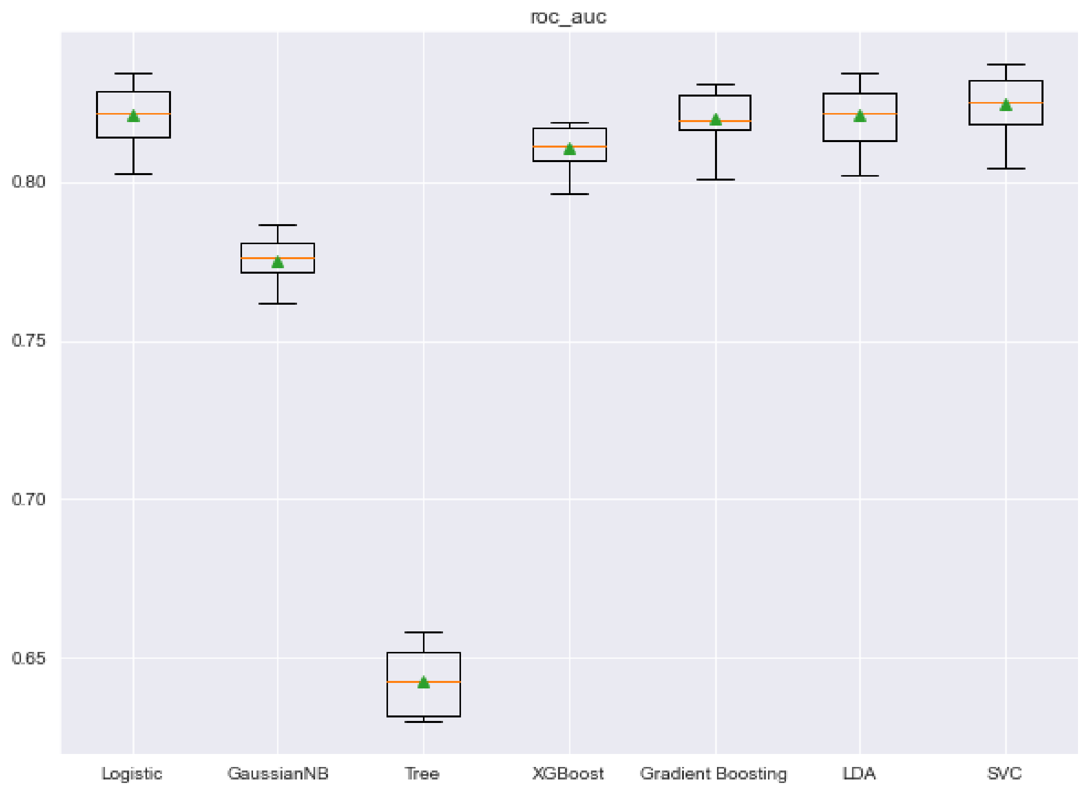
```
LDA
[[10498  3512]
 [ 3639 10351]]
              precision    recall  f1-score   support

           0     0.7426    0.7493    0.7459     14010
           1     0.7467    0.7399    0.7433     13990

    accuracy                         0.7446     28000
   macro avg     0.7446    0.7446    0.7446     28000
weighted avg     0.7446    0.7446    0.7446     28000

SVC
[[10532  3478]
 [ 3632 10358]]
              precision    recall  f1-score   support

           0     0.7436    0.7517    0.7476     14010
           1     0.7486    0.7404    0.7445     13990

    accuracy                         0.7461     28000
   macro avg     0.7461    0.7461    0.7461     28000
weighted avg     0.7461    0.7461    0.7461     28000
```

[40]:
```
def display_results(results, names, scores):
    # Create a DataFrame with mean and std for each score
    df_results = pd.DataFrame()
    df_results['Model'] = names
    for score in scores:
        df_results['Mean ' + score] = [result[score].mean() for result in␣
↪results.values()]
        df_results['Std ' + score] = [result[score].std() for result in results.
↪values()]

    display(df_results)

    # Plot boxplot with means
    fig, axes = plt.subplots(len(scores), figsize=(10, 8 * len(scores)))
    for i, score in enumerate(scores):
        axes[i].set_title(score)
        axes[i].boxplot([result[score] for result in results.values()],␣
↪labels=names, showmeans=True)

    plt.show()
```

```python
# Example usage
display_results(results, names, ['accuracy', 'roc_auc'])
```

```
              Model  Mean accuracy  Std accuracy  Mean roc_auc  Std roc_auc
0          Logistic       0.744643      0.009736      0.821323     0.009627
1        GaussianNB       0.704393      0.008310      0.775218     0.007537
2              Tree       0.642214      0.012242      0.642671     0.010289
3           XGBoost       0.735929      0.008710      0.810930     0.006704
4 Gradient Boosting       0.740500      0.009874      0.820305     0.008585
5               LDA       0.744607      0.010446      0.820955     0.009815
6               SVC       0.746071      0.009812      0.824763     0.009767
```


accuracy
0.76
0.74
0.72
0.70
0.68
0.66
0.64
0.62
Logistic
GaussianNB
Tree
XGBoost
Gradient Boosting
LDA
SVC


roc_auc
0.80
0.75
0.70
0.65
Logistic
GaussianNB
Tree
XGBoost
Gradient Boosting
LDA
SVC

For each model and from each fold, we obtain the accuracy and the roc auc, whose ditributions and means are visualised. In general, the majority of models (except GaussianNB and Tree) have good accuracy (around 75% on average) as well as good auc results (more than 0.8 over 1).

The best models are Support Vector Machines (Classifier), Logistic Regression and LDA, which have a similar performance, although it is true that SVM has achieved a slightly better performance on average. Therefore, the best sentiment review analyser is SVC().

Hyperparameter search

Once we have selected the best model, SVC, for our machine learning task, it becomes crucial to fine-tune its performance by optimising its hyperparameters. Performing a hyperparameter search involves systematically exploring different combinations of hyperparameters to find the optimal configuration that maximises the model's performance (regarding roc auc) on the given dataset. Since we have a balanced dataset, we could have used accuracy as metric instead.

Some SVC() hyperparameters we can tune are:

- C: it is known as the regularisation parameter, which determines the trade-off between achieving a low training error and having a simpler decision boundary. It controls the penalty for misclassifying training examples.
- gamma: defines the influence of each training example on the decision boundary. It controls the shape of the decision boundary and the flexibility of the model.
- kernel: determines the type of decision boundary that the SVM algorithm will learn. It specifies the function used to transform the input features into higher-dimensional space, where a linear decision boundary can be more effectively found.

[41]:
```
model = SVC()
grid = {'C':[1,10],'gamma':[1,0.1], 'kernel':['linear','rbf']}
cv = StratifiedKFold(n_splits=5)
grid_search = GridSearchCV(estimator=model, param_grid=grid, n_jobs=-1, cv=cv,␣
 ↪scoring='roc_auc',error_score=0)
```

[42]:
```
grid_result = grid_search.fit(X_train, y_train)
# summarise results
print("Best ROC AUC: %f with parameters %s" % (grid_result.best_score_,␣
 ↪grid_result.best_params_))
```

```
Best ROC AUC: 0.831763 with parameters {'C': 1, 'gamma': 1, 'kernel': 'rbf'}
```

The probability that the model will assign a larger probability to a random positive review than a random negative review is 0.831763, which is as a good value.

The kernel selected is the Gaussian Radial Basis Function (RBF). Mathematically:

$$k(x,y) = e^{-\frac{||x-y||^2}{2\sigma^2}}$$

```
[43]: # define models and parameters
      model = SVC(C = 1, gamma = 1, kernel = "rbf")
      model.fit(X_train,y_train)
```

```
[43]: SVC(C=1, gamma=1)
```

```
[44]: y_pred = model.predict(X_test)
      cm = confusion_matrix(y_test, y_pred)
      print(cm)
      print(classification_report(y_test,y_pred, digits = 4))
```

```
[[4649 1416]
 [1525 4410]]
              precision    recall  f1-score   support

           0     0.7530    0.7665    0.7597      6065
           1     0.7570    0.7430    0.7499      5935

    accuracy                         0.7549     12000
   macro avg     0.7550    0.7548    0.7548     12000
weighted avg     0.7550    0.7549    0.7549     12000
```

Interpretations

Precision + Out of all the reviews that the model predicted to be positive, “only” 75.3% actually were.

- Out of all the reviews that the model predicted to be negative, “only” 75.7% actually were.

Recall + Out of all the reviews that actually were positive, the model predicted this outcome correctly for 76.65% of those reviews.

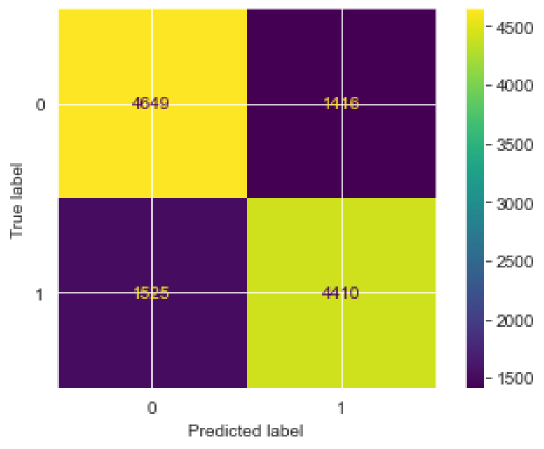
- Out of all the reviews that actually were negative, the model predicted this outcome correctly for 74.3% of those reviews.

Accuracy

- We can conclude that the model correctly predicted the outcome for approximately 75.49% of the reviews.

```
[45]: plt = ConfusionMatrixDisplay(cm)

      # Show the plot
      plt.plot()
```

```
[45]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1bbd4f9eac0>
```

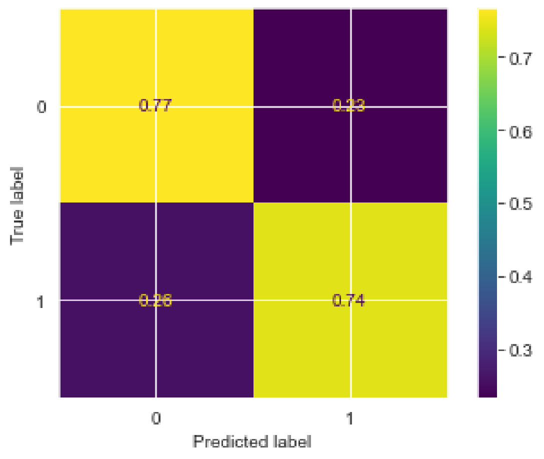


We can visualise that 4649 were correctly classified as true out of 6065 actual positive reviews and that 4410 were correctly classified as negative reviews out of 5935 actual negative reviews.

Another insight is that 4649 were actual positive reviews out of 6174 reviews classified as positive and that 4410 were actual negative reviews out of 5826 reviews classified as negative.

[46]:
```
cmn = confusion_matrix(y_test, y_pred, normalize = "true")
plt = ConfusionMatrixDisplay(cmn)
# if 'true', the confusion matrix is normalized over the true conditions (e.g.␣
↪rows);

# Show the plot
plt.plot()
```

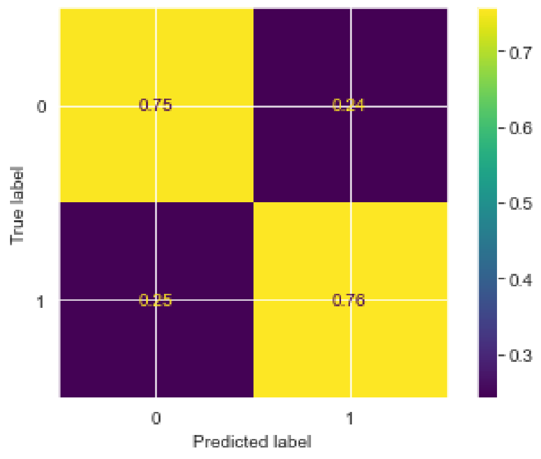
[46]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1bbd4f8ffd0>
```



This matrix shows the recall for each class since it has been normalised by rows (true).

```
[47]: cmn = confusion_matrix(y_test, y_pred, normalize = "pred")
      plt = ConfusionMatrixDisplay(cmn)
      # if 'pred', the confusion matrix is normalized over the predicted conditions (e.
       ↪g. columns);

      # Show the plot
      plt.plot()
```

```
[47]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1bbd4ecc520>
```



This matrix shows the precision for each class since it has been normalised by columns (predictions).

Conclusion

Based on the findings, we can conclude that the Support Vector Classifier (SVC) performed as the best model for our task with approximately 75.49% correct predictions on unseen data. Additionally, the ROC AUC value of 0.831763 demonstrates that the model exhibits good discriminatory power in distinguishing between the positive and negative reviews.

Moreover, our SVC model showed promising performance in terms of precision, recall and F1-score (around 75%). These results suggest that the SVC model is effective for our sentiment analysis task.

Chapter 7

Appendix

In this section, we will provide an overview of the Python code implemented for text preprocessing that gathers the concepts and procedures that have been explained along this work.

Firstly, we will dive into the morphological analysis explained in section (2.1), specifically in analysis stemming (section (2.1.1)) and lemmatisation (section (2.1.2)). In addition to this, we will look into the different modules WordNet have (section (2.1.2.1)) and how this information is presented. Then, different ways of tokenisation are shown together with their advantages and disadvantages (section (2.2.1)).

Brown corpus from NLTK will be preprocessed to check Zipf's Law hypothesis as it was mentioned in section (2.2.2). This justifies the removal of stopwords in the preprocess step.

Morphological Analysis

Stemming

Initialise the following stemming algorithms.

```
[1]: import nltk
     from nltk.stem import PorterStemmer, SnowballStemmer, LancasterStemmer
     # nltk.download("punkt")

     # Initialise Porter stemmer
     ps = PorterStemmer()
     # Initialise Snowball stemmer
     ss = SnowballStemmer("english")
     # Initialise Lancaster stemmer
     ls = LancasterStemmer()
```

```
[2]: # Example inflections to reduce
     examples = ["program","programming","programer","programs","programmed",␣
      ↪"people", "procedure", "successful",
                 "photographers", "photographs", "protographed", "unacceptable",␣
      ↪"fairly", "sportingly",
                "universal", "university", "data", "datum" ,"ultimately", "ultimatum"]
```

It is shown the following stemmisation of these words through Porter, Snowball and Lancaster Stemmer, which allow us to compare the differences between them.

```
[3]: # Perform stemming
     print("{0:15}{1:15}{2:15}{3:15}".format("--Word--","--PStem--", "--SStem--",␣
      ↪"--LStem"))
     for example in examples:
         print ("{0:15}{1:15}{2:15}{3:15}".format(example, ps.stem(example), ss.
      ↪stem(example), ls.stem(example)))
```

```
--Word--       --PStem--      --SStem--      --LStem
program        program        program        program
programming    program        program        program
programer      program        program        program
programs       program        program        program
programmed     program        program        program
```

```
people         peopl          peopl          peopl
procedure      procedur       procedur       proc
successful     success        success        success
photographers  photograph     photograph     photograph
photographs    photograph     photograph     photograph
protographed   protograph     protograph     protograph
unacceptable   unaccept       unaccept       unacceiv
fairly         fairli         fair           fair
sportingly     sportingli     sport          sport
universal      univers        univers        univers
university     univers        univers        univers
data           data           data           dat
datum          datum          datum          dat
ultimately     ultim          ultim          ultim
ultimatum      ultimatum      ultimatum      ultimat
```

Snowball Stemmer also is available in other languages such as Spanish. Here you can find an example.

[4]:
```
ss = SnowballStemmer("spanish")
examplesSpanish = ["niños", "trabajando", "trabajadores", "educación", "ciudad",␣
 ↪"persona"]
print("{0:15}{1:15}".format("--Word--", "--SStem--"))
for example in examplesSpanish:
    print ("{0:15}{1:15}".format(example, ss.stem(example)))
```

```
--Word--       --SStem--
niños          niñ
trabajando     trabaj
trabajadores   trabaj
educación      educ
ciudad         ciud
persona        person
```

Lemmatisation

Lemmatisation relies on the part-of-speech tag indicated and the result is an actual word.

[5]:
```
from nltk.stem import WordNetLemmatizer
# nltk.download("wordnet")
# nltk.download("omw-1.4")

# Initialise wordnet lemmatiser
wnl = WordNetLemmatizer()
```

[6]:
```
nouns = ["program", "programer", "people", "procedure", "photographers",␣
 ↪"photographs", "university", "data",
         "datum", "ultimatum"]
```

```
print("{0:15}{1:15}".format("--Word--","--Stem--"))
for example in nouns:
    print ("{0:15}{1:15}".format(example, wnl.lemmatize(example, pos="n")))
```

```
--Word--       --Stem--
program        program
programer      programer
people         people
procedure      procedure
photographers  photographer
photographs    photograph
university     university
data           data
datum          datum
ultimatum      ultimatum
```

[7]:
```
verbs = ["programming", "programs", "programmed", "photographed"]
print("{0:15}{1:15}".format("--Word--","--Stem--"))
for example in verbs:
    print ("{0:15}{1:15}".format(example, wnl.lemmatize(example, pos="v")))
```

```
--Word--       --Stem--
programming    program
programs       program
programmed     program
photographed   photograph
```

[8]:
```
adjectives = ["successfull", "unaceptable", "universal"]
print("{0:15}{1:15}".format("--Word--","--Stem--"))
for example in adjectives:
    print ("{0:15}{1:15}".format(example, wnl.lemmatize(example, pos="a")))
```

```
--Word--       --Stem--
successfull    successfull
unaceptable    unaceptable
universal      universal
```

[9]:
```
adverbs = ["fairly", "sportingly"]
print("{0:15}{1:15}".format("--Word--","--Stem--"))
for example in adverbs:
    print ("{0:15}{1:15}".format(example, wnl.lemmatize(example, pos="r")))
```

```
--Word--       --Stem--
fairly         fairly
sportingly     sportingly
```

WordNet

Bank synsets

Now, we retrieve the synsets (sets of synonymous words) associated with the word "bank" from WordNet. The output is a list of Synset objects from WordNet.

[10]:
```
from nltk.corpus import wordnet
syn = wordnet.synsets("bank")
print(syn)
```

```
[Synset('bank.n.01'), Synset('depository_financial_institution.n.01'),
Synset('bank.n.03'), Synset('bank.n.04'), Synset('bank.n.05'),
Synset('bank.n.06'), Synset('bank.n.07'), Synset('savings_bank.n.02'),
Synset('bank.n.09'), Synset('bank.n.10'), Synset('bank.v.01'),
Synset('bank.v.02'), Synset('bank.v.03'), Synset('bank.v.04'),
Synset('bank.v.05'), Synset('deposit.v.02'), Synset('bank.v.07'),
Synset('trust.v.01')]
```

We choose the object synset refering to a depository financial institution to better understand the concepts stored in it. First, we can know the definition of the words that from part of these synset (share semantic relation) and examples for illustration.

[11]:
```
print("depository_financial_institution.n.01 definition: " + syn[1].definition())
print(syn[1].examples())
```

```
depository_financial_institution.n.01 definition: a financial institution that
accepts deposits and channels the money into lending activities
['he cashed a check at the bank', 'that bank holds the mortgage on my home']
```

The lemmas represent the different forms of the words in this synset: depository financial institution, bank, banking concern, and banking company.

[12]:
```
print("depository_financial_institution.n.01 lemmas:")
print(syn[1].lemmas())
for lemma in syn[1].lemmas():
    print(lemma.name())
```

```
depository_financial_institution.n.01 lemmas:
[Lemma('depository_financial_institution.n.01.depository_financial_institution')
, Lemma('depository_financial_institution.n.01.bank'),
Lemma('depository_financial_institution.n.01.banking_concern'),
Lemma('depository_financial_institution.n.01.banking_company')]
depository_financial_institution
bank
banking_concern
banking_company
```

The hypernym of this synset (a term that includes the concept of depository financial institution) is financial institution.

```
[13]: print("depository_financial_institution.n.01 hypernyms:")
      print(syn[1].hypernyms())
```

```
depository_financial_institution.n.01 hypernyms:
[Synset('financial_institution.n.01')]
```

The terms that are encompassed (hyponyms) within a depository financial institution are: acquirer, agent bank, commercial bank, credit union, federal reserve bank, home loan bank, lead bank, member bank, merchant bank, state bank and thrift institution.

```
[14]: print("depository_financial_institution.n.01 hyponyms:")
      print(syn[1].hyponyms())
      for hyponym in syn[1].hyponyms():
          print(hyponym.name())
```

```
depository_financial_institution.n.01 hyponyms:
[Synset('acquirer.n.02'), Synset('agent_bank.n.02'),
Synset('commercial_bank.n.01'), Synset('credit_union.n.01'),
Synset('federal_reserve_bank.n.01'), Synset('home_loan_bank.n.01'),
Synset('lead_bank.n.01'), Synset('member_bank.n.01'),
Synset('merchant_bank.n.01'), Synset('state_bank.n.01'),
Synset('thrift_institution.n.01')]
acquirer.n.02
agent_bank.n.02
commercial_bank.n.01
credit_union.n.01
federal_reserve_bank.n.01
home_loan_bank.n.01
lead_bank.n.01
member_bank.n.01
merchant_bank.n.01
state_bank.n.01
thrift_institution.n.01
```

For a better understanding, we repeat the procedure with another synset of bank, which refers to the verb trust in this case.

```
[15]: print("trust.v.01 definition: " + syn[17].definition())
      print(syn[17].examples())
```

```
trust.v.01 definition: have confidence or faith in
['We can trust in God', 'Rely on your friends', 'bank on your good education',
"I swear by my grandmother's recipes"]
```

The lemmas represent the different forms of the words in this synset: trust, swear, rely and bank.

```
[16]: print("trust.v.01 lemmas:")
      print(syn[17].lemmas())
      for lemma in syn[17].lemmas():
          print(lemma.name())
```

```
trust.v.01 lemmas:
[Lemma('trust.v.01.trust'), Lemma('trust.v.01.swear'), Lemma('trust.v.01.rely'),
Lemma('trust.v.01.bank')]
trust
swear
rely
bank
```

The hypernym of this synset (a term that includes the verb trust) is the verb believe.

[17]:
```
print("trust.v.01 hypernyms:")
print(syn[17].hypernyms())
```

```
trust.v.01 hypernyms:
[Synset('believe.v.01')]
```

The terms that are encompassed (hyponyms) within trust are: count, credit and lean.

[18]:
```
print("trust.v.01 hyponyms:")
print(syn[17].hyponyms())
for hyponym in syn[17].hyponyms():
    print(hyponym.name())
```

```
trust.v.01 hyponyms:
[Synset('count.v.08'), Synset('credit.v.04'), Synset('lean.v.04')]
count.v.08
credit.v.04
lean.v.04
```

Siren synsets

[19]:
```
syn = wordnet.synsets("siren")
print(syn)
```

```
[Synset('siren.n.01'), Synset('enchantress.n.01'), Synset('siren.n.03'),
Synset('siren.n.04'), Synset('siren.n.05')]
```

[20]:
```
for i in range(0, len(syn)-1):
    print(syn[i], "definition: " + syn[i].definition())
    if (syn[i].examples() != []):
        print("Example:", syn[i].examples())
```

```
Synset('siren.n.01') definition: a sea nymph (part woman and part bird) supposed
to lure sailors to destruction on the rocks where the nymphs lived
Example: ["Odysseus ordered his crew to plug their ears so they would not hear
the Siren's fatal song"]
Synset('enchantress.n.01') definition: a woman who is considered to be
dangerously seductive
Synset('siren.n.03') definition: a warning signal that is a loud wailing sound
Synset('siren.n.04') definition: an acoustic device producing a loud often
wailing sound as a signal or warning
```

Path similarity

Since WordNet is a DAG, we can calculate the similarity between two synsets using the distace of the shortest path between them.

The value will be between 0 and 1, where 0 indicates no similarity and 1 indicates the highest similarity.

[21]:
```
dog = wordnet.synset('dog.n.01')
cat = wordnet.synset('cat.n.01')

similarity = dog.path_similarity(cat)
print(f"Path Similarity between 'dog.n.01' and 'cat.n.01': {similarity}")
```

```
Path Similarity between 'dog.n.01' and 'cat.n.01': 0.2
```

To obtain further information, we can dive into the hypernyms that these two synsets share.

[22]:
```
common_hypernyms = dog.common_hypernyms(cat)
print(f"Common Hypernyms between 'dog.n.01' and 'cat.n.01': {common_hypernyms}")
```

```
Common Hypernyms between 'dog.n.01' and 'cat.n.01': [Synset('mammal.n.01'),
Synset('animal.n.01'), Synset('living_thing.n.01'), Synset('chordate.n.01'),
Synset('object.n.01'), Synset('physical_entity.n.01'), Synset('placental.n.01'),
Synset('carnivore.n.01'), Synset('vertebrate.n.01'), Synset('whole.n.02'),
Synset('organism.n.01'), Synset('entity.n.01')]
```

[23]:
```
depository_financial_institution = wordnet.
 ↪synset('depository_financial_institution.n.01')
trust = wordnet.synset('trust.v.01')

similarity = trust.path_similarity(depository_financial_institution)
print(f"Path Similarity between 'depository_financial_institution.n.01' and␣
 ↪'trust.v.01': {similarity}")
```

```
Path Similarity between 'depository_financial_institution.n.01' and
'trust.v.01': 0.07142857142857142
```

Lexical Representations

Tokenisation

Regular Expressions

Word Tokenisation

Simple word tokenisation based on words boundaries.

[1]:
```
import re
textW = "There are different ways we can tokenise depending not only on the␣
 ↪language, but also on the technique."
```

[2]:
```
tokens = re.findall("[\w]+", textW)
print(tokens)
```

```
['There', 'are', 'different', 'ways', 'we', 'can', 'tokenise', 'depending',
'not', 'only', 'on', 'the', 'language', 'but', 'also', 'on', 'the', 'technique']
```

Tokenisation using regular expressions.

- + [] : A set of characters.
- \w : Returns a match where the string contains any word characters (characters from a to Z, digits from 0-9, and the underscore _ character).
- + : One or more occurrences.

Sentence tokenisation

Simple sentence tokenisationn based on punctuation marks, which does not distinguish other uses of punctuation marks such as abreviations and decimals.

[3]:
```
textS = "Dr. Henry drinks 2.5 litres of water per day. He also likes to drink␣
 ↪tea."
tokens_sent = re.compile('[.!?] ').split(textS) # Using compile method to␣
 ↪combine RegEx patterns
tokens_sent
```

[3]:
```
['Dr',
 'Henry drinks 2.5 litres of water per day',
 'He also likes to drink tea.']
```

NLTK

Word tokenisation

```
[4]: import nltk
     from nltk.tokenize import word_tokenize
     # nltk.download('punkt')
     tokens = word_tokenize(textW)
     print(tokens)
```

```
['There', 'are', 'different', 'ways', 'we', 'can', 'tokenise', 'depending',
'not', 'only', 'on', 'the', 'language', ',', 'but', 'also', 'on', 'the',
'technique', '.']
```

Since punctuation marks are considered as tokens, we can then add another preprocess step to remove them and only consider alphanumeric values.

```
[5]: textW.split()
```

```
[5]: ['There',
      'are',
      'different',
      'ways',
      'we',
      'can',
      'tokenise',
      'depending',
      'not',
      'only',
      'on',
      'the',
      'language,',
      'but',
      'also',
      'on',
      'the',
      'technique.']
```

Sentenece tokenisation

Sentence tokenisation based on NLTK module can make a distinction between the different uses of punctuation marks.

```
[6]: from nltk.tokenize import sent_tokenize

     text = """Characters like periods, exclamation point and newline char are used␣
      ↪to separate the sentences.
     But one drawback with split() method, that we can only use one separator at a␣
      ↪time!
```

```
So sentence tonenization wont be foolproof with split() method."""
sent_tokenize(text)
```

```
[6]: ['Characters like periods, exclamation point and newline char are used to
separate the sentences.',
 'But one drawback with split() method, that we can only use one separator at a
time!',
 'So sentence tonenization wont be foolproof with split() method.']
```

```
[7]: text = "Dr. Henry drinks 2.5 litres of water per day. He also likes to drink tea.
 ↪"
sent_tokenize(text)
```

```
[7]: ['Dr. Henry drinks 2.5 litres of water per day.',
 'He also likes to drink tea.']
```

Spacy

```
[8]: import spacy
```

First, we have to load a pre-trained language model specifically designed for English text processing. Then, a Doc object is created and then, we can use the attribute and methods to obtain further information.

```
[9]: # Load the small English model
nlp = spacy.load("en_core_web_sm")

# Create a Doc object
doc = nlp(textW)

# Iterate over the tokens in the Doc object
for token in doc:
    print(token.text)
```

```
There
are
different
ways
we
can
tokenise
depending
not
only
on
the
language
,
but
also
on
the
technique
.
```

Bag-of-Words

This section includes a simple example to understand bag-of-words from a collection of three documents, which are opinions regarding a TV show.

[10]:
```
import pandas as pd
import numpy as np
doc1 = 'Got Talent is an awesome tv show!'
doc2 = 'Got Talent is the best tv show! The best ever'
doc3 = 'Got Talent is so boring.'
```

[11]:
```
from sklearn.feature_extraction.text import CountVectorizer
vectoriser = CountVectorizer()
```

[12]:
```
X = vectoriser.fit_transform([doc1,doc2,doc3])
df_bow_sklearn = pd.DataFrame(X.toarray(),columns=vectoriser.
 ↪get_feature_names_out())
df_bow_sklearn.head()
```

[12]:
```
   an  awesome  best  boring  ever  got  is  show  so  talent  the  tv
0   1        1     0       0     0    1   1     1   0       1    0   1
1   0        0     2       0     1    1   1     1   0       1    2   1
2   0        0     0       1     0    1   1     0   1       1    0   0
```

For BoW implementation, CountVectorizer() is used to count the frequency of each word per document. We obtain a matrix where the element (i, j) refers to the number of times the word that represents column j appears in the i-th document.

As mentioned in Bag-of-Words section, it is useful to remove stopwors before constructing the bag-of-words representation. This is the reason why CountVectorizer allows to remove them easily.

[13]:
```
vectoriser = CountVectorizer(stop_words='english')
X = vectoriser.fit_transform([doc1,doc2,doc3])
df_bow_sklearn = pd.DataFrame(X.toarray(),columns=vectoriser.
 ↪get_feature_names_out())
df_bow_sklearn.head()
```

[13]:
```
   awesome  best  boring  got  talent  tv
0        1     0       0    1       1   1
1        0     2       0    1       1   1
```

```
2         0      0        1     1        1    0
```

In addition to this, CountVectorizer() can also consider n-grams to construct the bag-of-words and its representation. In this case, we are going to focus only on bigrams, but it allows for different ranges of n-grams.

[14]:
```
vectoriser = CountVectorizer(stop_words='english', ngram_range=(2,2))
X = vectoriser.fit_transform([doc1,doc2,doc3])
df_bow_sklearn = pd.DataFrame(X.toarray(),columns=vectoriser.
 ↪get_feature_names_out())
df_bow_sklearn.head()
```

[14]:
```
   awesome tv  best tv  got talent  talent awesome  talent best  \
0           1        0           1               1            0
1           0        1           1               0            1
2           0        0           1               0            0

   talent boring  tv best
0              0        0
1              0        1
2              1        0
```

TF-IDF

Now, we are going to implement TF-IDF to compare the results with the previous and illustrate this process with a simple example. For this purpose, we are going to use TfidVectorizer(), which has common arguments with CountVectorizer().

[15]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
vectoriserTF = TfidfVectorizer()
```

[16]:
```
X = vectoriserTF.fit_transform([doc1,doc2,doc3])
df_bow_sklearn = pd.DataFrame(X.toarray(),columns=vectoriserTF.
 ↪get_feature_names_out())
df_bow_sklearn.head()
```

[16]:
```
        an  awesome      best    boring      ever       got        is  \
0  0.48776  0.48776  0.000000  0.000000  0.000000  0.288079  0.288079
1  0.00000  0.00000  0.597527  0.000000  0.298763  0.176454  0.176454
2  0.00000  0.00000  0.000000  0.572929  0.000000  0.338381  0.338381

       show        so    talent       the        tv
0  0.370954  0.000000  0.288079  0.000000  0.370954
1  0.227217  0.000000  0.176454  0.597527  0.227217
2  0.000000  0.572929  0.338381  0.000000  0.000000
```

```
[17]: vectoriserTF = TfidfVectorizer(stop_words='english')
      X = vectoriserTF.fit_transform([doc1,doc2,doc3])
      df_bow_sklearn = pd.DataFrame(X.toarray(),columns=vectoriserTF.
       ↪get_feature_names_out())
      df_bow_sklearn.head()
```

```
[17]:    awesome      best    boring       got    talent        tv
      0  0.66284  0.000000  0.000000  0.391484  0.391484  0.504107
      1  0.00000  0.870714  0.000000  0.257129  0.257129  0.331100
      2  0.00000  0.000000  0.767495  0.453295  0.453295  0.000000
```

In this case, common words between different documents such as got and talent have the lowest not-null weight since they are not a key identifying feature to draw specific conclusions for that document.

```
[18]: vectoriserTF = TfidfVectorizer(stop_words='english', ngram_range=(2,2))
      X = vectoriserTF.fit_transform([doc1,doc2,doc3])
      df_bow_sklearn = pd.DataFrame(X.toarray(),columns=vectoriserTF.
       ↪get_feature_names_out())
      df_bow_sklearn.head()
```

```
[18]:    awesome tv   best tv  got talent  talent awesome  talent best  \
      0    0.652491  0.000000    0.385372        0.652491     0.000000
      1    0.000000  0.546454    0.322745        0.000000     0.546454
      2    0.000000  0.000000    0.508542        0.000000     0.000000

         talent boring   tv best
      0       0.000000  0.000000
      1       0.000000  0.546454
      2       0.861037  0.000000
```

In contrast to bag-of-words representation, which only considers the frecuency per document, TF-IDF reflects the importance a word has not only in a document but also within the whole corpus. This gives a more representative insight for each document.

Zipf's Law

We are going to analyse evidence of Zipf's Law in the Brown corpus.

```
[1]: import numpy as np
     import pandas as pd
     import matplotlib.pyplot as plt
     import seaborn as sns
     import string
     from math import log
     import numpy as np

     from nltk import FreqDist
     from nltk.corpus import brown
```

```
[2]: punctuations = list(string.punctuation)
     print(punctuations)
```

```
['!', '"', '#', '$', '%', '&', "'", '(', ')', '*', '+', ',', '-', '.', '/', ':',
';', '<', '=', '>', '?', '@', '[', '\\', ']', '^', '_', '`', '{', '|', '}', '~']
```

In this case, the preprocessing step involves tokenisation, lowercasing and punctuation removal.

```
[3]: tokens = [word.lower() for word in brown.words() if not all(char in punctuations␣
      ↪for char in word)]
     # in this case, words are splitted
     # lowercase and remove punctuation marks
     # We are only interested in words, not in punctuation marks
     print(tokens[:10])
```

```
['the', 'fulton', 'county', 'grand', 'jury', 'said', 'friday', 'an',
'investigation', 'of']
```

FreqDist tells us the frequency of each item in the text. In general, it could count any kind of observable event. It is a distribution because it tells us how the total number of word tokens in the text are distributed across the vocabulary items. The output is a dictionary whose values are sorted in decrease order.

```
[4]: word_freq = FreqDist(tokens)
     word_freq
```

```
[4]: FreqDist({'the': 69971, 'of': 36412, 'and': 28853, 'to': 26158, 'a': 23195,
     'in': 21337, 'that': 10594, 'is': 10109, 'was': 9815, 'he': 9548, ...})
```

```
[13]: print(f"The number of tokens in the Brown Corpus is {len(word_freq)}.")
```

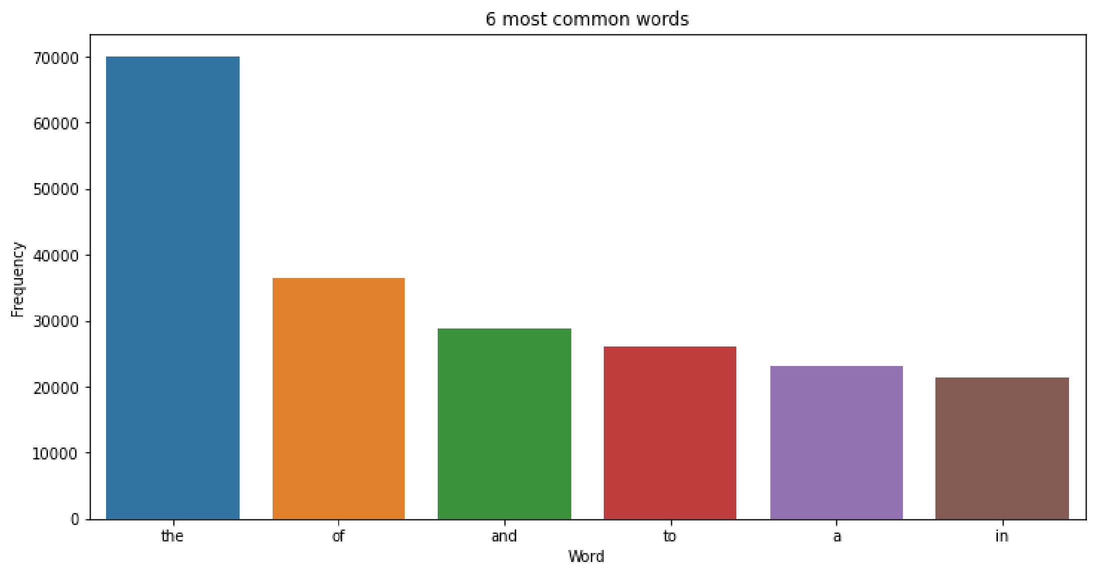
```
The number of tokens in the Brown Corpus is 49796.
```

We define a function that plot the distribution of the 10 most frequent words by default, although we can change this argument.

```
[14]: def ZipLaw(n=10):
          words_labels = [label[0] for label in word_freq.most_common(n)]
          words_freqs = [count[1] for count in word_freq.most_common(n)]
          plt.figure(figsize=(12,6))
          plt.title(str(n) + " most common words")
          plt.ylabel("Frequency")
          plt.xlabel("Word")
          plot = sns.barplot(x=words_labels,y=words_freqs)
          return plot
```

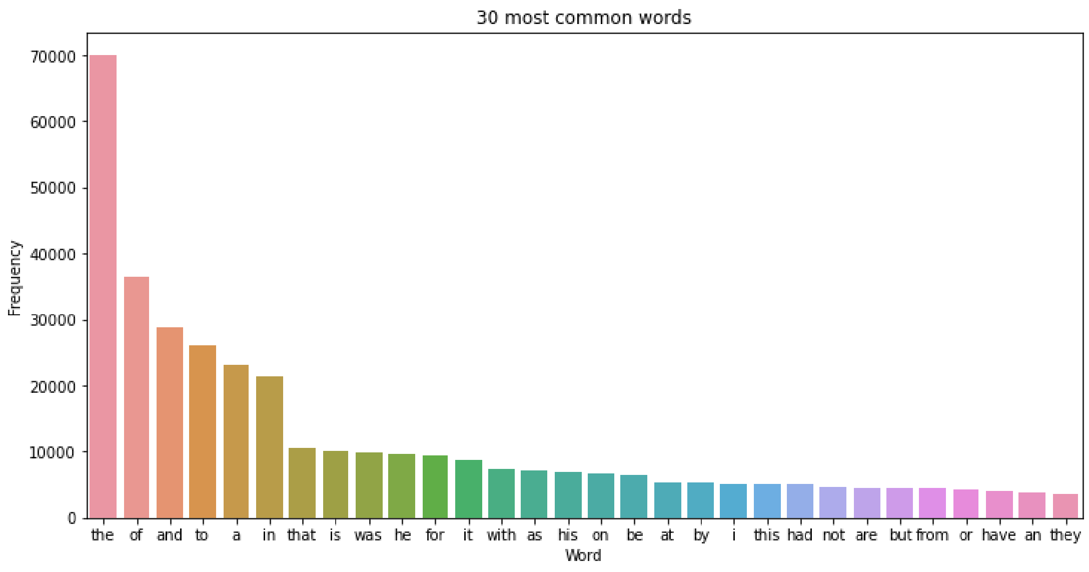
```
[15]: ZipLaw(6)
```

```
[15]: <AxesSubplot:title={'center':'6 most common words'}, xlabel='Word',
      ylabel='Frequency'>
```



```
[16]: ZipLaw(30)
      # plt.savefig("ZipLaw30.jpg")
```

```
[16]: <AxesSubplot:title={'center':'30 most common words'}, xlabel='Word',
      ylabel='Frequency'>
```



```
[17]: data = pd.DataFrame.from_dict(word_freq, orient="index", columns=["Frequency"])
      total_words = data["Frequency"].sum()
      total_words
      data = data.sort_values(by = "Frequency", ascending=False)
      data["% of total words"] = data["Frequency"]/total_words*100
      data["Rank"] = range(1, len(data)+1)
```

```
[18]: data.head(3)
```

```
[18]:      Frequency  % of total words  Rank
      the      69971          6.905131     1
      of       36412          3.593340     2
      and      28853          2.847376     3
```

In this dataframe, we can visualise the frequency and cumulative frequency together with their percentages and the rank of each token.

```
[19]: data["Cumulative frequency"]=np.cumsum(data["Frequency"])
      data["Cumulative %"]= data["Cumulative frequency"]/total_words*100
      data.head(135)
```

```
[19]:          Frequency  % of total words  Rank  Cumulative frequency  Cumulative %
      the          69971          6.905131     1                 69971      6.905131
      of           36412          3.593340     2                106383     10.498471
      and          28853          2.847376     3                135236     13.345847
```

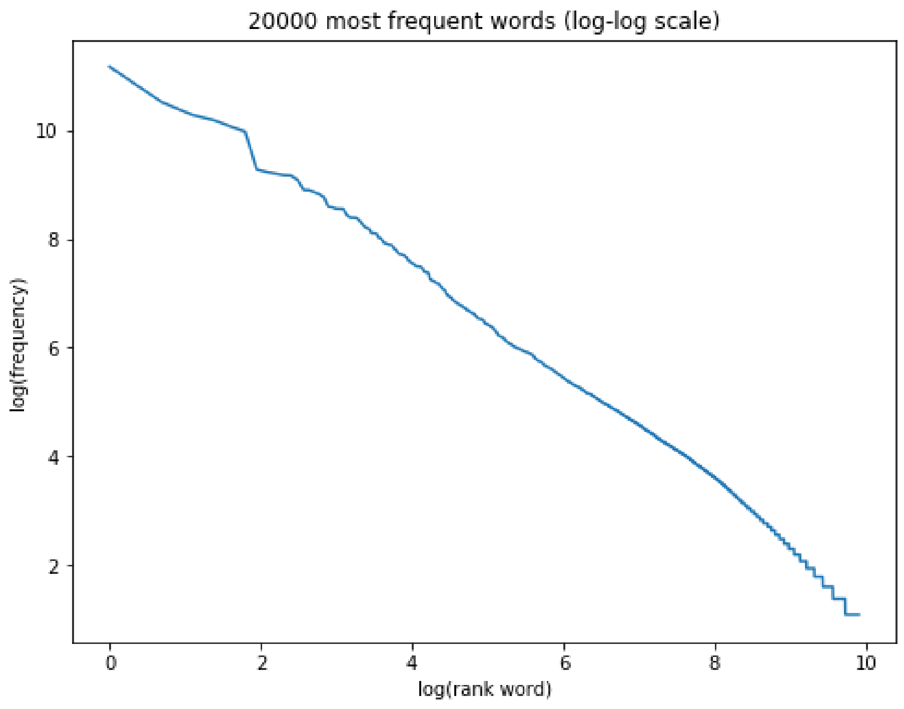
```
to           26158        2.581418    4           161394     15.927265
a            23195        2.289013    5           184589     18.216277
...            ...             ...  ...              ...           ...
day            687        0.067797  131           505471     49.882712
same           686        0.067698  132           506157     49.950410
another        684        0.067501  133           506841     50.017911
know           683        0.067402  134           507524     50.085314
while          680        0.067106  135           508204     50.152420

[135 rows x 5 columns]
```

Log-log plot

[20]:
```
def logZipLaw(n=5000):
    ranks_labels = list(map(np.log, np.arange(1,n+1)))
    words_freqs = [log(count[1]) for count in word_freq.most_common(n)]
    plt.figure(figsize=(8,6))
    plt.title(str(n) + " most frequent words (log-log scale)")
    plt.ylabel("log(frequency)")
    plt.xlabel("log(rank word)")
    plot = sns.lineplot(x=ranks_labels,y=words_freqs)
    return plot
```

[23]:
```
logZipLaw(20000)
# plt.savefig("LogZipLaw20000.jpg")
```

[23]:
```
<AxesSubplot:title={'center':'20000 most frequent words (log-log scale)'},
xlabel='log(rank word)', ylabel='log(frequency)'>
```


20000 most frequent words (log-log scale)
log(frequency)
log(rank word)
10
8
6
4
2
0
2
4
6
8
10

Bibliography

[1] Uday Kamath, John Liu, James Whitaker (2019), *Deep Learning for NLP and Speech Recognition.* Springer.

[2] Chomsky, Noam (1957), *Syntactic Structures.* The Hauge: Mounton & Co. Repreinted 1978, Peter Lang Publishing, 1957.

[3] Mathison Turin, Alan (1950), *Computing Machinery and Intelligence. Mind*, New Series, Vol. 59, No. 236, pp. 433-460

[4] W. John Hutchins, Leon Dostert & Paul Garvin (1955), *The Georgetown-I.B.M. experiment.* In: *In.* Jonh Wiley And Sons, pp. 124-135.

[5] S.R. Anderson (2012), *Languages: A Very Short Introduction.* OUP Oxford.

[6] Nizar Habash, Ramy Eskander & Abdelati Hawwri (2012), *A Morphological Analyzer for Egyptian Arabic.* In: *Proceedings of the Twelfth Meeting of the Special Interest Group on Computational Morphology and Phonology.* Association for Computational Linguistics, pp. 1-9.

[7] Beth Lovins, Julie (1968), *Development of a Stemming Algorithm* in *Mechanical Translation and Computational Linguistics.* Massachusetts Institute of Technology, Cambridge.

[8] F. Porter, Martin (1980), *An algorithm for suffix stripping.* URL: `https://tartarus.org/martin/PorterStemmer/`

[9] F. Porter, Martin (1980), *An algorithm for suffix stripping.* URL Algorithm: `https://tartarus.org/martin/PorterStemmer/def.txt`

[10] F. Porter, Martin, own Martin web page: `http://snowball.tartarus.org/` Retrieved in 2 September 2014.

[11] Datacamp, *Stemming and Lemmatization in Python.* URL: `https://www.datacamp.com/tutorial/stemming-lemmatization-python`

[12] Anjali Ganesh Jivani, *A Comparative Study of Stemming Algorithms.* Int. J. Comp. Tech. Appl., Vol 2 (6), 1930-1938. Retrieved in URL: `https://kenbenoit.net/assets/courses/tcd2014qta/readings/Jivani_ijcta2011020632.pdf`

[13] Andrea Gesmundo, Tanja Samardzic (2012), *Lemmatisation as a Tagging Task.* In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics.* Association for Computational Linguistics, pp. 368-372.

[14] Christiane Fellbaum (1998), WordNet. Official URL: `https://wordnet.princeton.edu/`

[15] Christiane Fellbaum, Editors: Roberto Poli, Michael Healy, Achilles Kameas (2010), *Theory and Applications of Ontology: Computer Applications.* Springer, pp. 231-243.

[16] Emiel van Miltenburg (2014), *WordNet Graphs.* Retrieved in: `http://www.cltl.nl/results/demos/wordnet-graphs/`

[17] Alina Ene, Kevin Wayne (2006), WordNet Programming Assignment. URL: `https://www.cs.princeton.edu/courses/archive/spring07/cos226/assignments/wordnet.html`

[18] Alina Ene, Kevin Wayne (2006), Programming Assignment 6: WordNet. URL: `https://www.cs.princeton.edu/courses/archive/spring17/cos226/assignments/wordnet.html`

[19] *Sample usage for WordNet.* URL: `https://www.nltk.org/howto/wordnet.html`

[20] Jonathan J. Webster & Chunyu Kit (1922), *Tokenisation as the initial phase in NLP.* Prof. of Coling-92.

[21] Julia Silge & David Robinson (2017), *Text Mining with R.* O'reilly. URL: `https://www.tidytextmining.com/`

[22] Zhixiang (Eddie) Xu, Minmin Chen, Kilian Q. Weinberger & Fei Sha (2013), *An alternative text representation to TF-IDF and Bag-of-Words.* URL: `https://arxiv.org/pdf/1301.6770.pdf`

[23] Martin Lotz (2020), *Mathematics of Machine Learning.* University of Warwick

[24] O. Bousquet, S. Boucheron & G. Lugosi (2004), *Introduction to Statistical Learning Theory.* In: O. Bousquet, U. von Luxburg, G. Rätsch (eds) Advanced Lectures on Machine Learning. ML 2003. Lecture Notes in Computer Science(), vol 3176. Springer, Berlin, Heidelberg. `https://doi.org/10.1007/978-3-540-28650-9_8`

[25] *What Is Overfitting?* MathWorks. Retrieved in: `https://ch.mathworks.com/discovery/overfitting.html`

[26] Gaël Varoquaux & Olivier Colliot. *Evaluating machine learning models and their diagnostic value. Olivier Colliot.* In *Machine Learning for Brain Disorders*, Springer, In press. HAL Id: ffhal-03682454v4f

[27] Vagelis Plevris Plevris, German Solorzano Solorzano, Nikolaos Bakas Bakas & Mohamed El Amine Ben Seghier Seghier (2022). *Investigation of performance metrics in regression analysis and machine learning-based prediction models.* Scipedia.

[28] Miriam Steurer, Robert J. Hill & Norbert Pfeifer (2020). *Metrics for Evaluating the Performance of Machine Learning Based Automated Valuation Models.* Department of Economics, University of Graz.

[29] Draper, N.R. & H. Smith (2014). *Applied regression analysis.* Applied Regression Analysis. 1-716

[30] Alice Zheng (2015). *Evaluating Machine Learning Models. A Beginner's Guide to Key Concepts and Pitfalls.* Chapter 3: Offline Evaluation Mechanisms: Hold-Out Validation, CrossValidation, and Bootstrapping, page 20. O'Reilly.

[31] Andrew Y. Ng & Michael I. Jordan. *On Discriminative vs. Generative classifiers: A comparison of logistic regression and naive Bayes.*

[32] *Sigmoid function.* Wikipedia. The Free Encyclopedia. URL: `https://en.wikipedia.org/wiki/Sigmoid_function`

[33] *Support Vector Machine.* Wikipedia. The Free Encyclopedia. URL: `https://en.wikipedia.org/wiki/Support-vector_machine`

[34] Andrew Ng. *CS229 Lecture notes Part V Support Vector Machines.* Stanford. URL: `https://www.cs.princeton.edu/courses/archive/spring16/cos495/slides/AndrewNg_SVM_note.pdf`

[35] S. Sumitra. *RKHS Methods in Machine Learning.* Department of Mathematics. Indian Institute of Space Science and Technology. Retrieved in: `https://www.iist.ac.in/sites/default/files/people/in12167/RKHS_0.pdf`

[36] Grace Zhang. *What is the kernel trick? Why is it important?* Medium. URL: `https://medium.com/@zxr.nju/what-is-the-kernel-trick-why-is-it-important-98a98db0961d`

[37] Dan Roth (2016). *Decision Trees.* CS 446 Machine Learning

[38] Juan M. Muñoz Pichardo & Rafael Pino Mejías. *Árboles de Clasificación y Regresión.* Statistics and Operational Research Department. University of Seville.

[39] Jean-Christophe Chouinard (2022). *Decision Trees in Machine Learning, with Examples (Python).* JC Chouinard. Retrieved in: `https://www.jcchouinard.com/decision-trees-in-machine-learning/`

[40] Stefanos Fafalios, Pavlos Charonyktakis, Ioannis Tsamardinos (2020). *Gradient Boosting Trees.*

[41] *Gradient Descent.* Wikipedia. The Free Encyclopedia. URL: `https://en.wikipedia.org/wiki/Gradient_descent`

[42] Stefanos Fafalios, Pavlos Charonyktakis & Ioannis Tsamardinos (2020). *Gradient Boosting Trees.* Gnosis Data Analysis PC. Retrieve in: https://www.gnosisda.gr/wp-content/uploads/2020/07/Gradient_Boosting_Implementation.pdf

[43] Stanley Chan (2020). *Lecture 5.2: Stochastic Gradient Descent.* ECE 595: Machine Learning I. School of Electrical and Computer Engineering Purdue University.

[44] Panos Achlioptas. *Stochastic Gradient Descent in Theory and Practice.* Stanford.

[45] Léon Bottou, Frank E Curtis, & Jorge Nocedal (2018). *Optimization methods for large-scale machine learning.* Siam Review, 60(2):223–311.

[46] Robert Mansel Gower, Nicolas Loizou, Xun Qian, Alibek Sailanbayev, Egor Shulgin, & Peter Richtárik (2019). *Sgd: General analysis and improved rates.*

[47] Akash Wagh (2022). *What Is Mean By Gradient Descent And Types Of Gradient Descent.* Medium.

[48] Rafael Pino Mejías. *Boosting.* Statistics and Operational Research Department. University of Seville.

[49] Schapire, R.E. *The strength of weak learnability.* Mach Learn 5, 197–227 (1990).

[50] Freddy Hernández (2023). *Modelos predictivos.*

[51] *XGBoost Documentation.* Retrieved in: https://xgboost.readthedocs.io/en/stable/

[52] *XGBoost - ML winning solutions (incomplete list)*(2016). GitHub. Retrieved in https://github.com/dmlc/xgboost/tree/master/demo#machine-learning-challenge-winning-solutions

[53] Constantino Malagón Luque (2003), *Clasificadores bayesianos. El algoritmo Naïve Bayes.* Universidad de Nebrija.

[54] Tharwat, Alaa (2015). *Linear Discriminant Analysis.*

[55] Olga Veksler. *Pattern Recognition.*

[56] Sebastian Raschka (2014). *Linear Discriminant Analysis – Bit by Bit.* https://sebastianraschka.com/Articles/2014_python_lda.html

[57] Sowmyaa Vajjala, Bodhisatttwa Majumder, Anuj Gupt, & Harshit Surana (2020). *Practical Natural Language Processing.* O'Reilly Media, Inc.

[58] Korde, Vandana (2012). *Text Classification and Classifiers: A Survey.* International Journal of Artificial Intelligence Applications. Vol 3. Pages 85-99.

[59] TAEHO JO (2019). *Text Mining: Concepts, Implementation and Big Data Challenge.* Springer.

[60] *Text Classification.* Machine Learning. Google Developers.

[61] T. KWARTLER. *Text Mining in Practice with R.* John Wiley & Sons, 2017.

[62] *Robert Plutchik.* Wikipedia. The Free Encyclopedia. URL: `https://en.wikipedia.org/wiki/Robert_Plutchik`

[63] DAVID M. BLEI, ANDREW Y. NG & MICHAEL I. JORDAN (2003). *Latent Dirichlet Allocation.* Journal of Machine Learning Research. Volume 3.

[64] SREELEKHA S. *Statistical Vs Rule Based Machine Translation; A Case Study on Indian Language Perspective.* Dept. of Computer Science & Engineering, Indian Institute of Technology Bombay, India

[65] OSBORNE, M. (2011). *Statistical Machine Translation.* In: Sammut, C., Webb, G.I. (eds) Encyclopedia of Machine Learning. Springer, Boston, MA. pp 912-915

[66] QUOC V. LE & MIKE SCHUSTER, Research Scientists, Google Brain Team (2016). *A Neural Network for Machine Translation, at Production Scale.* Google AI Blog. `https://ai.googleblog.com/2016/09/a-neural-network-for-machine.html`

[67] ZHIXING TANA, SHUO WANGA, ZONGHAN YANGA, GANG CHENA, XUANCHENG HUANGA, MAOSONG SUNA & YANG LIU (2020). *Neural Machine Translation: A Review of Methods, Resources, and Tools.* arXiv:2012.15515v1

[68] DANIEL JURAFSKY & JAMES H. MARTIN (2019). *Speech and Language Processing.* Chapter 25: Question Answering.

[69] FLORIAN HÖLN (2022). *Natural Language Question Answering Systems – Get Quick Answers To Concrete Questions.* SEEBURGER blog.

[70] LIN, J. (2007). *An exploration of the principles underlying redundancy-based factoid question answering.* ACM Transactions on Information Systems, 25(2).

[71] *Mean reciprocal rank.* Wikipedia. The free Encyclopedia. URL: `https://en.wikipedia.org/wiki/Mean_reciprocal_rank`

[72] W. S. EL-KASSAS, C. R. SALAMA, A. A. RAFEA, & H. K. MOHAMED (2021). *Automatic text summarization: A comprehensive survey.* Expert Systems With Applications. Vol. 165. Elsevier.

[73] RADA MIHALCEA AND PAUL TARAU. *TextRank: Bringing Order into Texts.* Department of Computer Science. University of North Texas.

[74] MAH, P.M.; SKALNA, I.; MUZAM, J. *Natural Language Processing and Artificial Intelligence for Enterprise Management in the Era of Industry 4.0.* Appl. Sci. 2022, 12, 9207.`https://doi.org/10.3390/app12189207`

[75] Silvia Quarteroni (2018). *Natural Language Processing for Industrial Applications.* Elca Informatique SA. Hauptbeitrag / Natural Language Processing for Industry. Springer.

[76] Bahja, Mohammed (05-2021). *Natural Language Processing Applications in Business.*

[77] Binggui Zhou, Guanghua Yang, Zheng Shi & Shaodan Ma (2022). *Natural Language Processing for Smart Healthcare.* IEEE Reviews in Biomedical Engineering. arXiv:2110.1580

[78] *Python.* Wikipedia. The Free Encyclopedia. URL. URL: `https://es.wikipedia.org/wiki/Python`

[79] Steven Bird, Edward Loper & Ewan Klein (2009), *Natural Language Processing with Python.* O'Reilly Media Inc. URL: `https://www.nltk.org/book/`

[80] Christopher D. Manning, Prabhakar Raghavan & Hinrinch Schütze (2008), *Introduction to Information Retrieval.* Cambridge University Press. URL: `https://nlp.stanford.edu/IR-book/html/htmledition/irbook.html`

[81] Jonh Paul Mueller (2018), *Embracing the Four Python Programming Styles.* URL: `https://newrelic.com/blog/nerd-life/python-programming-styles`

[82] Carl Eickson (2009), *Object Oriented Programming.* Atomic Object, LLC. Retrieved in URL: `https://uploads-ssl.webflow.com/628d552469d2a1b0c6b730a9/63486a5b1760683603644d82_ObjectOrientedProgramming.pdf`

[83] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel,P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot & E. Duchesnay (2011), *Scikit-learn: Machine Learning in Python.* Journal of Machine Learning Research Volumen 12, pp. 2825-2830. Retrieved in: `https://jmlr.csail.mit.edu/papers/volume12/pedregosa11a/pedregosa11a.pdf`

[84] Clément Bisaillon. *Fake and real news dataset.* Retrieved in: `https://www.kaggle.com/datasets/clmentbisaillon/fake-and-real-news-dataset?select=Fake.csv`

[85] Lakshmipathi N. *IMDB Dataset of 50K Movie Reviews.* Retrieved in: `https://www.kaggle.com/datasets/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews`