NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

RICARDO OLIVEIRA AFONSO

DEVELOPMENT OF A SMARTPHONE APPLICATION AND CHROME EXTENSION TO DETECT FAKE NEWS IN ENGLISH AND EUROPEAN PORTUGUESE

MASTER IN ELECTRICAL AND COMPUTER ENGINEERING

NOVA University Lisbon
September, 2023

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

DEPARTMENT OF ELECTRICAL
AND COMPUTER ENGINEERING

DEVELOPMENT OF A SMARTPHONE APPLICATION AND CHROME EXTENSION TO DETECT FAKE NEWS IN ENGLISH AND EUROPEAN PORTUGUESE

RICARDO OLIVEIRA AFONSO

Adviser: João Rosas
Full Professor, NOVA University Lisbon

Examination Committee

Chairs: Helena Fino
Associated Professor, NOVA University Lisbon
Ricardo Peres
Assistant Professor, NOVA University Lisbon

MASTER IN ELECTRICAL AND COMPUTER ENGINEERING

NOVA University Lisbon
September, 2023

Development of a Smartphone Application and Chrome Extension to Detect Fake News in English and European Portuguese

This document was created with the (pdf/Xe/Lua)LaTeX processor and the NOVAthesis template (v6.10.17) [1].

Acknowledgements

I would like to start by thanking Professor João Rosas for his availability and dedication towards not only this dissertation but also the many other projects and subjects he took part in during the past 5 years. The different relevant insights and suggestions provided in our back-to-back conversations during some phases of the project helped me persevere while also increasing my interest and curiosity in the field of Machine Learning.

I would also like to thank my family, especially my parents and sister, for supporting me throughout my entire life and for teaching me the values and principles that shape who I am today and who I strive to be one day.

I would also like to express my gratitude to Guilherme, João, Naves, Pedro, Rebeca and Tomás, who began as classmates and have since become cherished friends.

Finally, I want to thank everyone else who has made an impact during my journey, including my Professors, classmates, and friends who made this whole experience worthwhile and who I will always remember and appreciate.

”

“Greatness From Small Beginnings.”

— Sir Francis Drake

(English Navigator and Explorer)

Abstract

This project focused on fighting against the threat of fake news that has been increasing for the past few years. Many aspects play an important role when it comes to differentiating fake news from real news, making it challenging for humans to tell them apart. Fortunately, Artificial Intelligence methods can help in this troublesome process.

Although many projects have been developed regarding fake news detection in English, the same cannot be said about European Portuguese. As of now and based on the conducted search, only two projects have explored this important task in this language, leaving much more room for creativity and improvement. As a consequence, this project also explored different approaches that had not been considered before in the context of fake news detection in European Portuguese.

Many natural language processing and feature extraction techniques were used to gather relevant insights from data, along with different Machine Learning classifiers and Deep Learning models. The data for the English models was obtained from public datasets designed for fake news detection tasks, while the European Portuguese models had to resort to a dataset created from scratch by scraping data from fact-checking websites and real and fake news websites, given the lack of public datasets in this language.

The creation of the first public dataset for fake news detection in European Portuguese is a commendable step in addressing the challenge of this field, as it can pave the way for more research and innovation in this specific language. Furthermore, the new techniques allied with the scraped data also made it possible to achieve better results than previously developed work surrounding this important topic.

Therefore, this project will prove worthy not only for those who use the developed Chrome extension and smartphone application to analyse the content of websites in both English and European Portuguese languages, but also for any future researcher interested in this field and with the desire to contribute to this cause.

Keywords: Machine Learning, Deep Learning, Web Scraping, Natural Language Processing, Term Frequency-Inverse Document Frequency, Extra Gradient Boosting, Bidirectional Encoder Representations from Transformers, Flask, Docker, Cloud

Resumo

Este projeto teve um foco no combate contra o perigo das notícias falsas que tem vindo a aumentar nos últimos anos. São vários os aspetos a considerar quando se pretende diferenciar notícias verdadeiras de falsas, tornando este processo desafiante para o ser humano. Felizmente, os métodos de Inteligência Artificial podem ajudar neste processo problemático.

Apesar de já terem sido desenvolvidos vários projetos relativamente à identificação de notícias falsas em Inglês, o mesmo não se verifica em Português Europeu. Até ao momento e de acordo com a pesquisa realizada, apenas dois projetos exploraram esta importante tarefa neste idioma, havendo assim um maior espaço para melhoria e criatividade.

Foram utilizadas várias técnicas de processamento de linguagem natural e recolha de caraterísticas, juntamente com diferentes classificadores de *Machine Learning* e modelos de *Deep Learning*. Os dados para os modelos em Inglês foram obtidos através de *datasets* públicos especificamente criados para este tipo de tarefas, enquanto os modelos em Português Europeu tiveram de recorrer a um *dataset* criado de raíz, face à ausência de *datasets* públicos, com recurso a *web scrapers* que permitiram a extração de dados de *websites* de verificação de factos e *websites* de notícias verdadeiras e falsas.

A criação do primeiro dataset público para identificação de notícias falsas em Português Europeu representa um passo importante neste desafiante tópico, uma vez que pode abrir novos caminhos para mais investigação e inovação nesta língua específica. Além disso, as novas técnicas aliadas aos dados recolhidos permitiram obter melhores resultados do que os dos trabalhos previamente desenvolvidos.

Como tal, este projeto revelar-se-á extremamente útil não só para aqueles que utilizarem a extensão Chrome e aplicação *smartphone* desenvolvidas para analisar o conteúdo de websites em Inglês e Português Europeu mas também para qualquer investigador interessado nesta área e com o desejo de contribuir para esta causa.

Palavras-chave: Machine Learning, Deep Learning, Web Scraping, Natural Language Processing, Term Frequency-Inverse Document Frequency, Extra Gradient Boosting, Bidirectional Encoder Representations from Transformers, Flask, Docker, Cloud

Contents

List of Figures

List of Tables

Acronyms

AdaBoost	Adaptive Boosting
AI	Artificial Intelligence
ANN	Artificial Neural Network
ANOVA	Analysis of Variance
API	Application Programming Interface
AWS	Amazon Web Services
BERT	Bidirectional Encoder Representations from Transformers
Bi-LSTM	Bi-Directional Long Short-Term Memory
BoW	Bag-of-words
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CSS	Cascading Style Sheets
CSV	Comma Separated Values
CV	Count Vectorizer
DBSCAN	Density-based Clustering Algorithm
DistilBERT	Distilled BERT
DL	Deep Learning
DNN	Deep Neural Network
DOM	Document Object Model
EC2	Elastic Compute Cloud
GAN	Generative Adversarial Network
GBoost	Gradient Boosting
GPU	Graphical Processing Unit

HTML	HyperText Markup Language
IDE	Integrated Development Environment
iOS	iPhone Operating System
IP	Internet Protocol
JSON	Javascript Object Notation
KNN	K-Nearest Neighbours
LASSO	Least Absolute Shrinkage and Selection Operator
LFS	Linguistic Feature Sets
LSA	Latent Semantic Analysis
LSTM	Long Short-Term Memory
LSVM	Linear Support Vector Machine
MBERT	Multilingual BERT
ML	Machine Learning
MLP	Multilayer Perceptron
MNB	Multinomial Naïve Bayes
NB	Naïve Bayes
NER	Named-entity Recognition
NLP	Natural Language Processing
NLTK	Natural Language Toolkit
PCA	Principal Components Analysis
PHP	Hypertext Preprocessor
PLSA	Probabilistic Latent Semantic Analysis
POS	Part Of Speech
RAM	Random Access Memory
RNN	Recurrent Neural Network
RoBERTa	Robustly Optimized BERT Pretraining Approach
SFTP	Secure File Transfer Protocol
SQL	Structured Query Language)
SSH	Secure Shell
SVC	Support Vector Classifier
SVM	Support Vector Machine

TF-IDF	Term Frequency-Inverse Document Frequency
TFLite	TensorFlow Lite
TSV	Tabulation Separated Values
UI	User Interface
URL	Uniform Resource Locator
VSM	Vector Space Model
WE	Word Embeddings
WSGI	Web Server Gateway Interface
XGBoost	Extreme Gradient Boosting
XLM-R	Cross-lingual Language Model-RoBERTa
XLNET	Extreme Language Understanding Network
XML	Extensible Markup Language

1 Introduction

1.1 Motivation

The world we live in is constantly changing, with millions of events happening every time and everywhere. As a way to not only share information with the rest of the world but also to keep up with everything that is happening, we resort to media in the form of radio, television, press and internet, with the latter being one of the most prominent media found nowadays.

This obviously varies from country to country, but the amount of people using the internet to get access to news has been increasing significantly for the past few years, through websites, blogs, social media, and others. One only needs their own device and internet connection to easily gain access to the latest news, regardless of their location or time of day, which translates to a much quicker process when compared to watching TV or listening to the radio, especially when it comes to portable devices such as smartphones, tablets and computers that people can carry around whenever and wherever they go [2].

However, the ease of access to information characteristic of the internet and its almost worldwide availability come at the price of difficult monitoring, regulation and credibility check of not only the information shared but also the responsible entity, since anyone can create their own website or social media account for a wide variety of purposes, including more nefarious ones such as spreading false information of all kinds. This false information is often referred to as fake news.

The amount of fake news has been increasing for the past few years, and so has the need to develop solutions to fight against it. It is in this context that the desire to contribute to this cause and the interest in learning more about its complex processes arises.

1.2 Objectives

The objective of this project is to contribute to the fight against fake news by developing a smartphone application and a Chrome extension that will help their users identify and protect themselves against false information that they may find while searching for

information on the Internet.

The development of these applications is done by resorting to smartphone application development tools and a serverless backend. The mechanisms responsible for identifying and classifying fake news will be developed through Machine Learning (ML) and Deep Learning (DL) techniques.

Since ML mechanisms are not able to give correct answers every time, the application's performance will also benefit from a social network strategy in which each user can classify content as real or fake, which will be used to improve the models' performance.

The users' classifications will then be sent to a cloud-based server and then redirected to a local machine with better resources and capacities to perform the required steps behind the models' improvement.

As of today, a small number of projects have been developed to detect fake news in European Portuguese through ML and DL algorithms. Although Portugal is among the least affected countries, it is not immune to fake news and its dangers, with new cases of disinformation found every day.

The existence of so many websites and social media accounts devoted to spread disinformation leads to the development of methods to fight against it. As a consequence, this project will take both English and European Portuguese languages into consideration when analysing websites' content.

1.3 Document Structure

This document is organized into the following chapters: Introduction, State of Art, Project Development and Conclusion.

The motivation and objectives behind this dissertation are presented in the Introduction.

The State of Art explores a bit more the motivation behind the project by describing the concept of fake news and the dangers associated with it, along with the tools and processes involved in its detection. Many examples of the work that has been developed in this field are also explored, as well as some of the available frameworks, libraries and services.

The Project Development section not only describes the proposed techniques and processes, but also explores the challenges faced during the project, the respective solutions that made it possible to overcome said obstacles and the observed results of the different milestone achievements.

In Conclusion, the need to fight against fake news is reaffirmed and the different steps discussed throughout the project are summarized, along with the obtained results, faced challenges and future work.

2

State of art

2.1 Fake News and its Consequences

The concept of fake news has been around since the late 1890s and it was used to describe the false nature of certain reports in newspapers found during that era. However, the term became more popular in 2017 with the increasing amount of misinformation spread on social media and websites. Unlike trusted sources like journalists and media outlets, the internet has little to no regulation over shareable content and conduct principles, so one must filter both content and sources available. However, this is hard to do sometimes, as many publishers and individuals responsible for spreading fake news try to look trustworthy enough by impersonating well-known sources and organizations [3].

When used for more fun-related purposes, such as parodies or other forms of humour, fake news does not actually pose any sort of threat. The real threat emerges when fake news is used for more malicious purposes, such as manipulating one's principles, ideals, perceptions and, ultimately, behaviour, which has been found within political, social and economic affairs.

Such is the case of the 2016 presidential election in the United States of America, during which president Donald Trump resorted to false statements aimed at his political opponents, questioning the right of many Democrats to govern the country [4].

The on-going Covid-19 pandemic is another example, with many fake videos and photos regarding their origin and impacts being shared over social media, to the point where panic ends up spreading faster than the actual virus [5].

Many fake news articles revolving around the on-going war in Ukraine have also been spread online by both pro-Russian and pro-Ukrainian groups, which end up raising even more unnecessary conflicts and fear across the world [6].

Furthermore, fake news can also be used by hackers to convince people into clicking links that lead to fake websites as a way to gain access to their email accounts, bank accounts or even personal computers [7].

2.2 How to detect Fake News

Even though some organizations try to impersonate reputable sources as mentioned before, the vast majority of entities responsible for creating fake news often fail to stay undercover and end up being quite easy to unmask by analysing certain patterns found within the content of the news themselves. This is due to the fact that most fake news articles are generated by automated-bots and then spread by real people who are not aware of their false nature, with the remaining few being created by actual people who are hired for that specific role [8].

Despite the possibility of verifying whether a piece of news is real or fake, the sheer amount of information along with the different types of content makes it impossible for humans to review every news article and effectively separate what is fake from what is real. This has led many researchers and organizations to develop programs and tools capable of detecting fake news in websites and social media posts with automatic methods, mostly linked to ML and DL algorithms, which can predict the content's credibility by being trained with great amounts of fake news' and real news' data.

2.3 Fake News Detection Methods and Tools

Although many have their own, the vast majority of websites and tools characterize fake news based on the following categories:

- satire or parody, with more of a potential to fool rather than harm.
- false connection, found when headlines, visuals or captions have no relation with the actual content.
- misleading content, found when information is used in a misleading way to frame topics and individuals.
- false context, found when truthful content is used in a context different from reality.
- impostor content, found when false information is created by sources that impersonate trustful and well-known sources.
- manipulated content, found when content that was once real is manipulated to deceive, typically more related with photos.
- fabricated content, found when the content is entirely false as a way to deceive readers and harm those who are involved [9].

Some examples of these websites and tools include FactCheck.org and PolitiFact, in which journalists review American political news, Emergent.Info, in which rumours are submitted by users and reviewed by journalists, CaptainFact, which has a more user-based

verification approach towards videos, CrossCheck, which evaluates news according to their purpose and objectivity through Artificial Intelligence (AI), among many others [10].

For European Portuguese, a few websites have also been developed, such as Polígrafo from SAPO, Fact-Check from Observador and Prova dos Factos from PÚBLICO, in which journalists review the news' content and compare it with other sources. ContraFake is another project currently under development that aims to help in the detection of fake news through the use of AI as well [11], [12], [13], [14].

Many projects have been developed regarding the detection of fake news and, as a consequence, it is crucial to analyze some of the work that has been done in this field. Overall, the processes found within fake news detection can be divided into different phases, resorting to a wide variety of datasets, data processing techniques and classification models. Figure 2.1 illustrates a typical sequence of processes behind text classification:

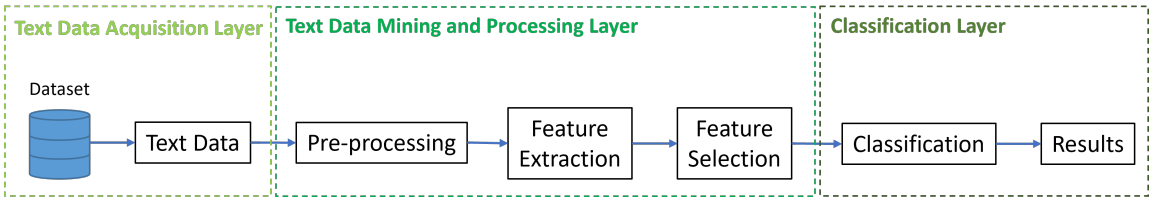


Figure 2.1: Text classification processes

It is of utmost importance to learn about each aspect and process related to text classification, starting with datasets.

2.3.1 Datasets

A dataset is often described as an aggregation of data organized in a wide variety of types and classifications, depending on their purpose. The most common types are displayed in figure 2.2.

A categorical dataset, also known as qualitative dataset, stores information related with the characteristics or qualities of people or objects.

A dichotomous dataset is a type of categorical dataset in which each row only as two categories. A dataset with news articles and statements that are labelled as either fake or real is an example of a dichotomous dataset.

A polytomous dataset, on the other hand, corresponds to a categorical dataset in which each entry can be represented by more than two categories. A dataset that stores multiple cake shapes desired by customers on a pastry is an example of a polytomous dataset.

A numerical dataset, also known as quantitative dataset, stores only numerical values. These include measurements such as the height and weight of patients in a certain clinic, the number of pages of all e-books on an online library, among others.

A bivariate dataset is a dataset with only two variables that can be related to one another. Figure 2.2 shows an example of a bivariate dataset that contains the average resting heart rate based on different ages.

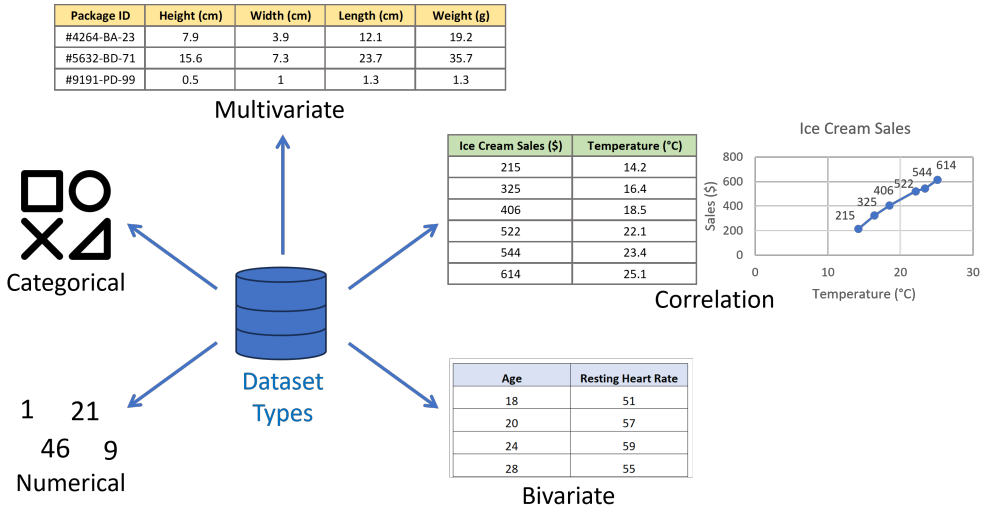


Figure 2.2: Most common dataset types

A multivariate dataset is a dataset with more than two variables that can be related to one another. Figure 2.2 shows an example of a multivariate dataset used in a warehouse to keep track of different packages and respective measurements that are represented as variables.

A correlation dataset is a bivariate or multivariate dataset in which the variables show some sort of dependency or relationship. Figure 2.2 shows an example of a correlation dataset in which the ice cream sales increase as the temperature increases, hence being designated as a positive correlation [15].

The data stored in datasets often falls under three main categories as well: structured, semi-structured and unstructured. Table 2.1 shows the main aspects of each category:

Table 2.1: Comparison between unstructured, semi-structured and structured data

	Unstructured data	Semi-structured data	Structured data
Representation	Text, image, video, audio and binary files	Key-value entries (JSON, CSV, HTML and XML)	Tabular (columns and rows)
Storage methods	File systems and cloud warehouses	NoSQL databases	Relational databases and local warehouses
Analysis techniques	NLP, image recognition, text, video and audio analysis	Query languages and data mining	SQL queries and data mining
Advantages	Variety of formats Easily and quickly collected High scalability Better insights with more data	Variety of formats High scalabity Versatile Schema More storable than unstructured	Less processing required Easier to manage Simplified usage and management
Disadvantages	Can be challenging to deal with Requires tools to be managed	Computers only interpret the data if it is first structured accordingly Higher costs that structured data	Pre-defined format limits its usability Not as scalable as others

Although this project will not explore many of the different storage methods and techniques mentioned above, they can help determine the use cases and limitations associated with each kind of data. Datasets play an important role in any AI task, and the

detection of fake news through ML and DL methods is no exception. After all, the data that is being processed will define the models' performance and behaviour, so the use of appropriate data translates to better future predictions and outcomes [16], [17], [18].

With these concepts in mind, many organizations and individuals have created several datasets after gathering unstructured data from different sources, such as news websites, blogs and social media, and transforming them into semi-structured data which researchers and developers then end up accessing during their projects' development.

The authors [19] in resorted to a social media fake news dataset from Kaggle to identify fake news from social media. The dataset included 7796 news articles, with information about each article regarding its title, body and either fake or real label.

Fake news in social media was also analysed in [20], more specifically on Facebook. The dataset was generated by collecting and analysing many profiles, as well as shared content. This was accomplished with the aid of a web crawler and Application Programming Interface (API), collecting more than 15000 news from a total of 5000 different profiles, including both fake and real news in different formats, such as text, images or videos.

The authors in [21] and [22] also collected several social media posts and news related with COVID-19 in English, which were classified as either real or fake. Non-English content was manually skipped, and a total of 10700 social media posts and news articles were gathered to create a dataset.

The spread of false information is not limited to social media. In fact, many datasets have been created by resorting to false statements and fake news articles spread throughout the web. In [23] an approach based on two models was developed to classify news as either fake or real, with a dataset provided by QICC made of 384 articles. Half of them were real and the other half were fake and most articles had information regarding their titles and content.

The authors in [24] resorted to the LIAR-PLUS Master, an extended version of the LIAR dataset which has evidence information extracted from the full-text verdict article published in Politifact by journalists.

The authors in [25] resorted to the Fake News Challenge (FNC-1) dataset, which is from a specific digital journalism project for rumour debunking found within the Emergent Dataset, with 1684 articles. As found in the previous datasets, this dataset included both headline and body of each article, but unlike the previous datasets, the news articles were not categorized as either fake or real. Instead, the journalists divided them into 4 different categories ("agree", "disagree", "discuss" and "unrelated") according to how related each article's title was to its corresponding body.

Two datasets from Kaggle with 23738 articles in total were considered in [26], along with the ISOT Fake News Dataset made of 21417 real and 23481 fake articles, in order to create a new dataset which contained fake and real news from many different domains, including politics, sports, technology and entertainment.

The authors in [27] analysed different methods used to detect fake news by studying a wide variety of papers and articles found within this field. Three main datasets were

taken into account: the LIAR dataset from Kaggle, the ISOT Fake News Dataset and a combined corpus dataset with fake and real news from different sources.

The WELFake dataset was created in [28] by merging popular datasets from different sources, including Kaggle, McIntire, Reuters, and BuzzFeed Political, with a total of 72,134 news articles (35,028 real and 37,106 fake news). The dataset was designed to prevent overfitting of classifiers and enable better ML training.

The authors in [5] resorted to a wide variety of fake and real COVID-19-related news articles that were manually gathered from different sources, such as Facebook, Twitter, The New York Times, Harvard Health Publishing, WHO, and more.

The Fake.Br Corpus was created in [29], which comprised a total of 7200 news articles in Brazilian Portuguese, half real, half fake, gathered from different online media sources and journals. The dataset included information about many aspects of each article, such as the authors, links, sources, grammatical errors, excessive punctuation marks, questionable layout, and more. The authors resorted to web crawling techniques to extract data from real news found in trustworthy sources and they also made use of reputable Brazilian fact-checkers to evaluate the credibility of each article. Each article was manually sorted afterwards into respective topics, including economy, science and technology, politics, and more.

A dataset made of 3764 news articles in European Portuguese was created in [30] after gathering data from different European Portuguese online news sources through web-scraping techniques, similar to the ones previously mentioned. The dataset included information about each article's website link, title, field category, source and classification tag (either fake or real). The author resorted to some online fact-checkers that have already been mentioned before, such as Polígrafo, Observador and Corona Verificado, to define the classification tag of each article. An article was considered real if it was classified by the fact-checkers as so, if there were not any signs of manipulation of its content and if its layout was equal or similar to the one typically found in real news. Otherwise, it would be defined as fake. Satirical articles were also defined as fake.

Another fake news dataset made of 708 news articles in European Portuguese was created in [31] after gathering data from different sources through web crawling and News Feeds APIs. The data included each article's author, title, content, Uniform Resource Locator (URL), category and more. Each article was manually obtained from News Feeds and Websites and, as found in some of the previous projects, the author also resorted to online fact-checkers, Polígrafo and Fact-Check Observador, to be more precise, when applying the different credibility levels, which were divided into two approaches: a multi-class approach, in which the credibility levels behind each article were categorized as "false", "partially false", "partially true" and "true", and a binary approach, which categorized as either "false" or "true".

Despite there being a wide variety of datasets focused on different fields, it is clear that, Kaggle's datasets such as the LIAR dataset along with the ISOT Fake News dataset are some of the most used datasets in fake news detection. However, there are also many

other popular datasets, such as BuzzFace with 2282 news articles related with the 2016 US Elections and whose credibility is categorized as "true", "non-factual content", "mixture of true and false" and "mostly false", and WSDM Cup, with 360,767 articles in which two articles were compared at a time in order to determine the relation between their titles and corresponding body [30].

Although datasets offer the ability to gain instant access to relevant data, some researchers opted to develop their own datasets, rather than using already existing ones, with the aid of Web Scrapers and Web Crawlers.

2.3.2 Web Scraping and Web Crawling

Web Scraping and web crawling are two processes that allow the automatic extraction of information from websites without human intervention. Figure 2.3 shows the main steps behind web scraping and web crawling:

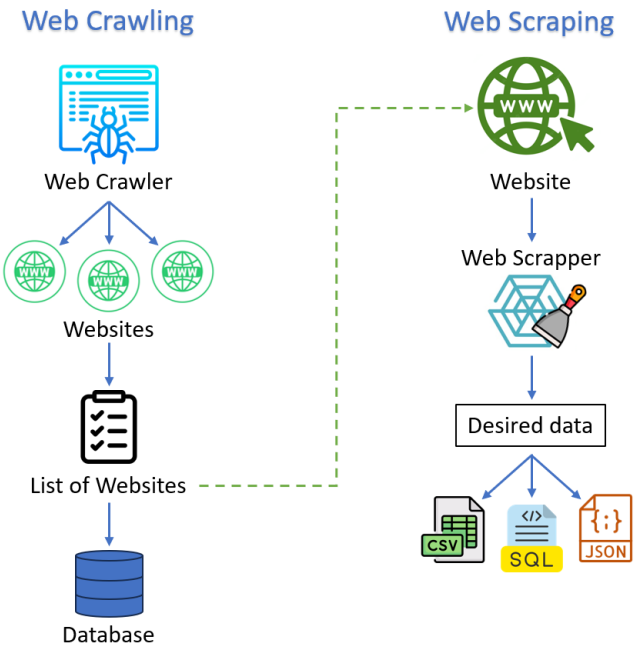


Figure 2.3: Main steps in web scraping and web crawling

As portrayed by Figure 2.3, web scraping is more centred around the extraction of data from one or more websites, whereas web crawling is more focused on discovering URLs or links across the internet. As a consequence, it is not possible to resort to web scraping unless one is familiar with the desired website URL, in which case an initial step involving the use of a web crawler can be beneficial, as a way of obtaining a list with the desired URLs that can then be used by a web scraper to extract the required data [32].

Besides URLs, the extraction of data from websites implies understanding how information is distributed in said websites as well, which are created through three main elements, namely HyperText Markup Language (HTML), Cascading Style Sheets (CSS)

and JavaScript, along with different frontend frameworks and libraries. CSS and JavaScript are responsible for the design and user interaction with a website, respectively, which is of small interest to Web Scraping in this context, while the same cannot be stated about HTML.

HTML describes the structure of web documents. The content on a page is structured by creating an architecture of text documentation in the form of elements identified by tags, which are used by browsers such as Chrome, Edge, Firefox and Safari to determine not only what content or information to display, but also how to display said content or information. The elements include titles, headings, subheadings, paragraphs, hyperlinks, tables, lists, basic text, images, interactable components such as buttons and text boxes, and much more [33].

All of these aspects underline the importance of HTML in Web Scraping, as it allows the identification of the elements found in news websites that contain the desired data to be extracted, which can then be saved in different document formats, such as Comma Separated Values (CSV) files, Structured Query Language) (SQL) files, Javascript Object Notation (JSON) files, among others [34], [35].

2.3.2.1 Web Scraping Libraries

Beautiful Soup is a Python library used to scrape information from web pages. Once a request is sent to a specific website, the respective page content can be accessed in order to iterate, search and modify the parse tree [36].

Selenium offers the ability to drive browsers as a user would, which is crucial to automate specific actions or movements required to scrape data. This includes closing pop-ups or tabs that can potentially prevent data extraction, as well as clicking on buttons or scrolling down to load more content on the webpage [37].

2.3.3 Text Data Mining and Processing

Despite there being many different datasets and news websites available, the vast majority share certain characteristics found within the data itself that are taken into account during content analysis, just as the journalists behind some of the fake news tools and websites do, as mentioned before. This translates to the need to process this same data carefully so that AI models are able to properly learn from it and predict the outcomes reasonably well, which can be achieved through text mining and Natural Language Processing (NLP) techniques.

Text mining is the process responsible for transforming unstructured text, such as social media, product reviews, videos and audio files, or semi-structured text, such as Extensible Markup Language (XML), JSON and HTML files, to a more structured format, as mentioned in table 2.1, in order to generate a better and cleaner dataset from which it is possible to underline patterns and derive future insights.

JSON techniques allow computers to read, process, analyze and interpret human language in both textual and verbal formats, making them extrMLy relevant for fake news detection through AI algorithms. Processing unstructured data with NLP techniques can become a very challenging task, hence the importance in using text mining alongside them.

Given the relevance of these processes, the following subsections will be devoted to the methods that take part in both text data mining and associated NLP techniques [38].

2.3.3.1 Pre-processing Phase and NLP Techniques

Typically, the first step in text mining involves a pre-processing stage, in which many NLP techniques can be used to filter and categorize data. The stopword filtering technique is one of them, which consists of the removal of stop words, common words that are not very relevant, as a way of allowing the program to focus on more important ones that help in the news content classification, such as the articles "the", "a" and "an" and certain prepositions such as "in" and "with". Table 2.2 shows some examples of stopword filtering:

Table 2.2: Stopword filtering examples

Text with stopwords	Text without stopwords
There is a tree near the river!	There tree near river!
This is a sample sentence, showing off the stop words filtration.	This sample sentence, showing stop words filtration.
The quick brown fox jumped over the lazy dog.	quick brown fox jumped lazy dog.

The tokenization of texts is another important technique given the nature of the required tasks. It involves splitting paragraphs and sentences into smaller units, often single words, and removing spaces and punctuation elements, such as dots, commas and quotations, for the algorithms to understand the data in a much better way. For example, tokenizing the sentence "There is a tree near the river!" would result in a list with the words "There", "is", "a", "tree", "near", "the" and "river" without the exclamation mark [19].

Another technique used in text pre-processing is stemming, which consists in reducing words into their base forms, by deleting prefixes and suffixes. Similar to stemming is lemmatization, in which inflectional endings from words are removed, as a way of obtaining the base or canonical form found in dictionaries. However, unlike in lemmatization, the removal of prefixes and suffixes in stemming can cause overstemming or understemming of certain words that may not be found in dictionaries.

Furthermore, lemmatization considers additional aspects during the analysis when compared to stemming, such as synonyms of words, which makes lemmatization a more complex and advanced technique than stemming [39]. Table 2.3 shows a few examples of stemming and lemmatization, including cases of overstemming (marked in red) and understemming (marked in yellow):

Table 2.3: Stemming and lemmatization examples

Normal Words	Stemmed Words	Lemmatized Words
Programming	Program	Program
Jumped	Jump	Jump
Universal	Univers	Universal
University	Univers	University
Alumnus	Alumnu	Alumnus
Alumni	Alumni	Alumnus

Part Of Speech (POS) is another text pre-processing technique that categorizes words as parts of speech, such as verbs, nouns and adjectives, with tags according to both their definition and corresponding context, making it possible to perform a semantic analysis on unstructured text [40], [24].

Another technique used is text sentiment analysis, which consists in detecting the polarity (positive or negative) and intensity of sentiments or feelings found in text. The use of this technique has been increasing for the past few years due to an increase in stylistic techniques found in fake news to affect readers' emotions [26].

Summarization is another method used to sum up the main ideas behind long pieces of text.

Segmentation is used to split the text into different sentences as a way to accomplish certain goals, such as differentiating the use of periods between abbreviations or fractional numbers and the ending of a sentence.

Change case is used to convert words to either lowercase or uppercase, as a way to set a string global format, with lowercase being the typically defined format.

Another technique used is spell correction, in which spelling mistakes found in text are corrected, although some authors do not resort to this technique due to the fact that fake news articles often have many more spelling mistakes when compared to real news.

Text normalization is responsible for converting certain pieces of text into their canonical form, often found in places with more informal writing such as social media applications, in which words are shortened or spelled differently.

Finally, text categorization is used to categorize synonyms and abbreviations with the intent of classifying texts into categories or topics [41], [38].

2.3.3.2 Feature Extraction Phase

The pre-processing step is followed by a feature extraction. This is an extrMLy important step since it allows the reduction of the number of resources and correspondent computation time needed by classification models to make much more accurate predictions, without the cost of losing vital information.

Furthermore, irrelevant or redundant features can increase a classifier's complexity and overfitting level, which should be avoided at all costs since overfitting happens when the model learns the particular training data so well that it performs poorly under different

data. Therefore, it is crucial to resort to this kind of processes, in order to remove these same features and, consequently, improve the performance of the models to the greatest extent with as many reliable predictions as possible [42], [43].

There are different types of features and corresponding text representation approaches found within feature extraction:

- Semantic features, which capture the meaning and interpretation of text, resulting in more significant data patterns.
- Lexical features, which vectorization techniques typically use for counting the occurrences of each unique word, including pronouns, verbs, adjectives, punctuation, word count, average word length, length of article and more.
- Sentence-level features, which are mostly used in text classification tasks. Some examples of these features include a Bag-of-Word approach, an n-gram approach and a part-of-speech-based approach.
- Psycholinguistic features, which are related with dictionary-based text mining techniques, such as word count and many others.

Term Frequency-Inverse Document Frequency (TF-IDF) is a vectorization model used to encode text by resorting to a Vector Space Model (VSM) format . It determines the importance of certain terms within a single document and within a whole collection of documents [44].

The term frequency (TF) of a term or word corresponds to the occurrences of said term in a document compared to the total number of words in said document, while the inverse document frequency (IDF) corresponds to the proportion of documents that contain the term and that comprise the dataset, as in equations 2.1 and 2.2:

$$TF = \frac{\text{Term's number of occurrences in the document}}{\text{Total number of terms in the document}} \tag{2.1}$$

$$IDF = log(\frac{\text{Number of documents in the corpus or dataset}}{\text{Number of documents which contain the term}}) \tag{2.2}$$

The TF-IDF of a term is obtained by multiplying both TF and IDF, as shown in equation 2.3:

$$\text{TF-IDF} = TF * IDF \tag{2.3}$$

TF-IDF is often used alongside a Bag-of-words (BoW) algorithm, also known as Count Vectorizer (CV), which is used to determine the frequency of each word found within a document. A list is generated for each document found in the corpus or dataset, with each key being the word and each value its corresponding number of occurrences on that

same document. BoW is similar to Word Embeddings (WE) like Word2Vec, which creates a vector per word.

It is important to note that the BoW model is an orderless document representation. In case it is desired to determine the probability of a certain word appearing after another, one can resort to the n-gram model, whose ability to store spatial information allows the calculation of not only the term frequency as described before but also the frequency of several terms found after each other.

A bi-gram corresponds to a sequence of 2 words (n=2), a tri-gram to a sequence of 3 words (n=3) and so on. The number of words to consider can be defined and changed as desired, which is crucial in bigger corpus or datasets, as a way to improve the classification models' understanding of the different term patterns and corresponding probabilities during its training [45], [46].

There are many other feature extraction techniques, such as Principal Components Analysis (PCA), in which a p number of principal components are chosen to represent the data, hence being considered a dimension reduction technique. This technique is simple and robust in approximating the covariance or correlation matrix, which are used to decode the linear relationship between two variables in a dataset, being one of the most powerful feature extraction techniques that can be used with variables that share a strong relationship [47].

Latent Semantic Analysis (LSA) is another feature extraction method that analyses the patterns shared by documents and the respective terms contained within them, through singular value decomposition. Although it is possible to find hidden patterns shared by said terms, it can be challenging to compare different documents as this technique is not able to detect polysemantic words [48].

A feature extraction technique capable of considering polysemy is Probabilistic Latent Semantic Analysis (PLSA), which is based on LSA and latent semantic features. In this model, one can either perform a factorization similar to the one found in LSA or resort to latent/hidden context variables that are assigned to observable variables, which determine the probability of words and documents using the convex combination of different features. Examples of latent/hidden variables include topics or concepts linked with documents or words, which represent the observable variables, respectively.

There are also Clustering-based methods that can be used in feature extraction, in which a certain number of clusters are formed or sorted. Although this method can be used to determine deceptive characteristics in fake news according to a normalized frequency of connections between units, it may perform poorly due to a possible non-existence of the required data in current fake news [27], [49].

2.3.3.3 Feature Selection Phase

Feature selection can be described as the process of selecting the subset of features to be used for training a classification model, while feature extraction is described as the process

of creating new features from already existing ones. Therefore, feature extraction should be performed first, as it allows the identification of many potentially useful features, and afterwards, feature selection can be performed to identify the best subset that contributes to the improvement of the model's performance [50].

When it comes to feature selection, there are three main types of methods to consider: Wrapper methods, Filter methods and Embedded methods.

Wrapper Methods

Wrapper methods iterate through different subsets of features, evaluating the importance of each feature for each iteration as a way to determine the most optimal model with the best combination. Figure 2.4 portrays said steps:

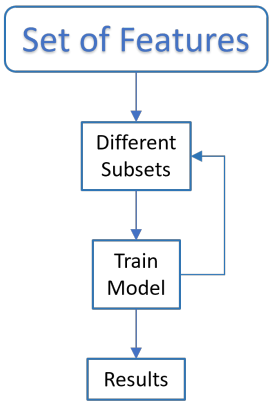


Figure 2.4: Wrapper methods' steps

Models tend to overfit when wrapping methods are used with small data points and the computation time required increases significantly when dealing with many features. The main feature selection wrapper methods include forward selection, backward selection and stepwise selection.

Forward selection consists in adding features one by one, starting from zero features. After performing the t-test or f-test, which are used to compare two related samples and to test the equality of two populations of samples, respectively, the first feature with the lowest p-value (probability) is added to the working model. Afterwards, another cycle is repeated, adding a second feature while also considering the previously selected one, and any features with insignificant p-values are excluded. This process is repeated until all features with relevant p-values are added [51].

Backward selection is similar to forward selection, but instead of starting from zero features, the model is run with all features from the dataset and the feature with the most insignificant p-value is excluded. This process is repeated until all features with irrelevant p-values are removed.

Stepwise selection consists in a hybrid method that involves both forward and backward selection. Starting from zero features, the first two features with the lowest significant

p-values are added. From this point, after another iteration and addition of the lowest significant feature, the features with an insignificant p-value are also removed in each cycle, until the final model is selected with only significant features.

On the one hand, wrapper methods are able to select a model with significant values, even when one is not familiar with the data or the relevance of the features. On the other hand, they do not cycle through every possible combination of features, so the resulting model may not correspond to the most optimal one, with worse predictions due to possible multicollinearity issues associated with inflated beta coefficients, i.e., two or more independent variables being highly correlated with each other.

Filter Methods

As opposed to wrapper methods in which models are tuned consecutively, filter methods resort to a ranking procedure based on a useful descriptive measure other than error, which is responsible for the selection of a subset. This measure is inherent to features and not dependent on a model, which translates to a very low computation time and overfitting, despite not being able to detect any interactions or correlations between features. Figure 2.5 shows the steps behind filter methods:

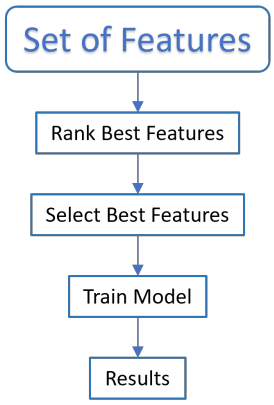


Figure 2.5: Filter methods' steps

In case one decides to add interactions to a model, an interaction term can also be used to quantify the relationship between two features, if the mentioned features depend on the value of one another. This can be beneficial for the model by reducing multicollinearity and, consequently, providing further insight into the data.

A few examples of filter methods include Pearson Correlation, Variance Thresholding and Analysis of Variance.

Pearson Correlation measures the linear correlation between two variables through a coefficient that ranges between -1 and 1. Two features are related if their correlation coefficient value is close to either 1 or -1. A heatmap containing all the correlations can be used to determine the features with the highest correlation, which should represent the majority of the relevant data.

Variance thresholding consists in determining the variance of a feature, which indicates its predictive power, and selecting all the features above a certain variance threshold, in order to keep most of the vital information found within the data.

Analysis of Variance (ANOVA) corresponds to a group of models that are used to identify variations in treatment (sample) means, which determines whether a feature is relevant to said models or not.

There are many other filter methods, such as Information Gain, which measures the contribution of a term to a classification task based on its presence or absence within a document, Chi-square Statistic, in which the independence between terms and classes is measured, and more [27].

Embedded Methods

Embedded methods can be described as the middle term between the two previous feature selection methods, since the selection and tuning of the subset of features are done during the model creation process. Figure 2.6 portrays said steps:

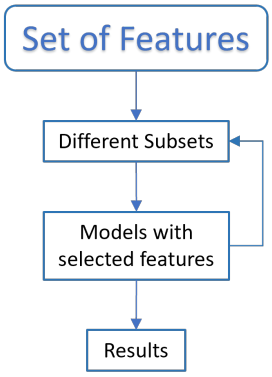


Figure 2.6: Embedded methods' steps

Least Absolute Shrinkage and Selection Operator (LASSO) Regression, also known as L1 Regularization, and Ridge regression, also known as L2 Regularization, are the two main embedded methods used in feature selection.

Ridge Regression resorts to a cost function with a lambda term that can be tuned to penalise features with large beta coefficients, as a way to reduce the impact of multi-collinearity by decreasing the correlation strength found within less relevant variables.

LASSO Regression is similar to ridge regression, with the most significant difference being the ability to remove features from a model by forcing their corresponding beta coefficient to zero, which also contributes to a reduction of the model's complexity.

A Decision Tree also generates models by resorting to feature selection, although it is considered a classification algorithm and, therefore, will be further discussed in the following subsection, along with many other classification algorithms [52], [49].

2.3.3.4 Text Data Mining and Pre-processing Libraries and Frameworks

Scikit-learn is an open-source python framework developed by Google with a wide variety of simple and efficient tools for data mining and analysis. It is one the most used frameworks in these fields, with a vast number of libraries including Numpy, Scipy, Matplotlib, Pandas, and more [53].

Natural Language Toolkit (NLTK) is a python framework focused on NLP and text analysis. It supports the Portuguese language and includes many text pre-processing techniques, such as tokenization, stemming, tagging, parsing and more, but it is slower and harder to use than the previous ones [54].

Spacy is an open-source python library with many pre-trained models that can be used for text pre-processing functions, including Tokenization, Lemmatization, Punctuation and Stopword Removal, Part of Speech Tagging, Entity Recognition, and more. It tends to be more efficient than NLTK and supports more than 69 languages, including Portuguese, with a pre-trained word vector that can also be used to extract information [55].

2.3.4 Classification Algorithms

Classification Algorithms are used by machines to learn and make predictions based on input data. The two types of learning most commonly used regarding the detection of fake news are Supervised Learning and Unsupervised Learning.

In Supervised Learning, algorithms are trained based on input and output pairs of data manually labelled by humans.

In Unsupervised Learning, on the other hand, machines derive predictions based only on input data, through the detection of possible patterns and relationships within it, and without any human intervention.

The terms regression and classification are used when defining the type of problems to solve. Regression is related to continuous quantity and infinite possible values, whereas classification refers to discrete class labels.

For example, when using AI for climate prediction, a regression approach would predict the temperature's value while a classification approach would determine whether it would be hot or cold based on temperature values. This means the detection of fake news involves a classification approach [56].

The following subsections will explore many different Machine Learning and Deep Learning algorithms that fall under these same types.

2.3.4.1 Machine Learning Algorithms

Machine Learning falls under the umbrella of Computer Science as a subset of Artificial Intelligence. The algorithms and techniques involved in ML aid computers when analysing problems and making decisions, in order to solve them.

Decision Trees is a supervised learning model used in classification and regression. The input data is divided recursively into subsets according to an attribute value test until all nodes in a subset share the same variable. Each node of the tree represents the possible outcomes and is able to evaluate both category and statistical data, as shown in figure 2.7:

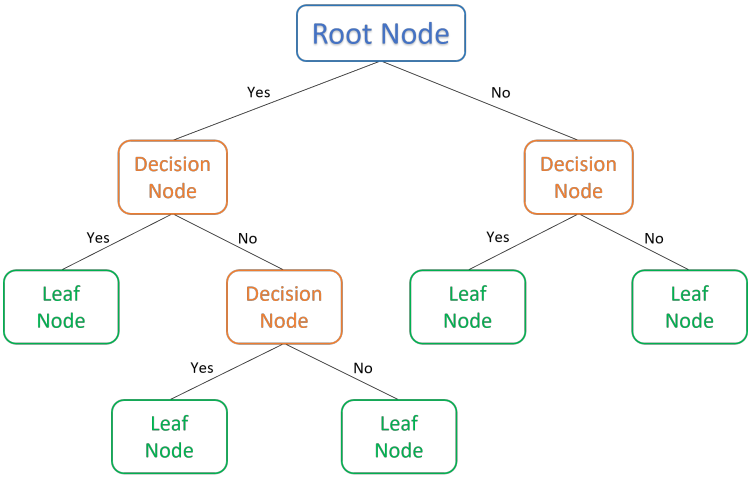


Figure 2.7: Decision Trees classifier

Random Forest is described as a collection of several decision trees, thus being a supervised learning model. The accuracy of this classifier increases with the number of trees and its predictions are not affected by overfitting or omitted values.

ExtrMLy Randomised Trees, also known as Extra Trees, is an algorithm similar to Random Forest, with the main difference being the use of a random subset of features for training, in order to estimate the importance of each of the model's characteristics and classify them accordingly.

Many algorithms such as Random Forest and Extra Trees resort to the combination of different weak classifiers to form a much stronger one, as a way to improve the resulting predictions and classifications. This is typically referred to as Ensemble Learning and it comprises three main distinct methods:

- Bagging, in which random data samples are initially used by a base learning algorithm in a primary model and then the output of all base models is combined, in order to achieve predictions with better accuracy and lower variance, as depicted in figure 2.8:

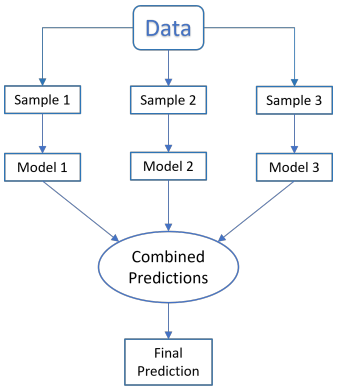


Figure 2.8: Bagging steps

- Boosting, in which the different base classifiers are sequentially arranged so that the preceding ones' mistakes are taken into account in the following ones, as opposed to the random data arrangement of Bagging and as shown in figure 2.9. This often results in a significant improvement in the predictions and performance of the classifier.

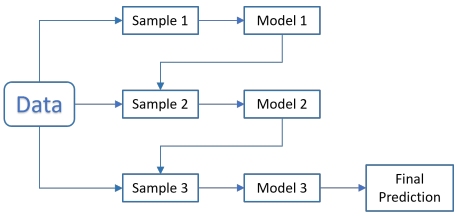


Figure 2.9: Boosting steps

- Stacking, in which various base learners are ensembled in parallel, as shown in figure 2.10, with the aim of combining their input in the best way possible to derive better predictions. Unlike bagging and boosting, stacking resorts to different learning algorithms and combines the base models using a final meta-model [57].

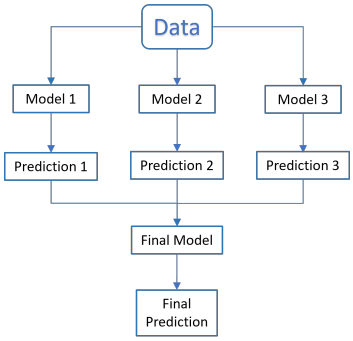


Figure 2.10: Stacking steps

Adaptive Boosting (AdaBoost) is an example of a boosting algorithm used in regression and classification problems. It is based on Decision Trees and, in order to improve the performance and predictions of a model, AdaBoost identifies the misclassified cases and penalises them by assigning more weightage to them, as a way to improve the performance of the following learning classifiers in each iteration and so that the final model is able to avoid said "mistakes".

Gradient Boosting (GBoost) is another boosting algorithm. Instead of penalising misclassified cases as found in AdaBoost, the different classifiers are gradually and sequentially trained in GBoost according to a loss function. GBoost typically performs better than AdaBoost, but it is more susceptible to overfitting and longer computational time.

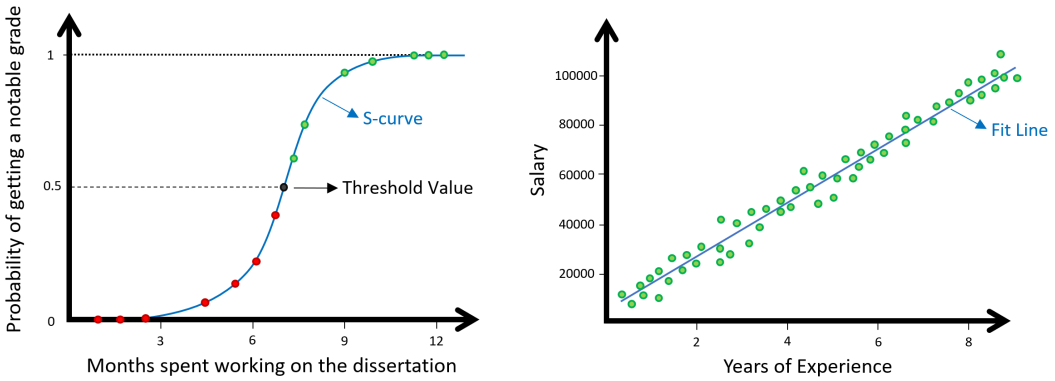
Extreme Gradient Boosting (XGBoost) represents an enhancement over the Gradient Boosting technique. The algorithm resorts to regularisation to reduce overfitting and parallel running to improve runtime speed, along with tree pruning [58].

K-Nearest Neighbours (KNN) is another supervised learning algorithm that is used to identify patterns and trends by considering k cases when analysing the whole dataset.

Logistic Regression is a supervised learning algorithm and is one of the most used algorithms for classification problems. The weighted sum of inputs is mapped between 0 and 1 through a Sigmoid curve (S-curve), which contains a threshold value that is used for predictions.

Linear Regression is also a supervised learning algorithm mainly used for regression problems, with the aim of finding connections between its predictions and the associated variables. These connections then lead to the best fit line that can accurately predict the output, which should only be in a continuous format like price, age, salary, and more [59].

Figure 2.11 depicts both of these two classifiers side by side. Despite being similar, Logistic Regression is more suited for fake news detection tasks than Linear Regression given their classification nature, as stated before.



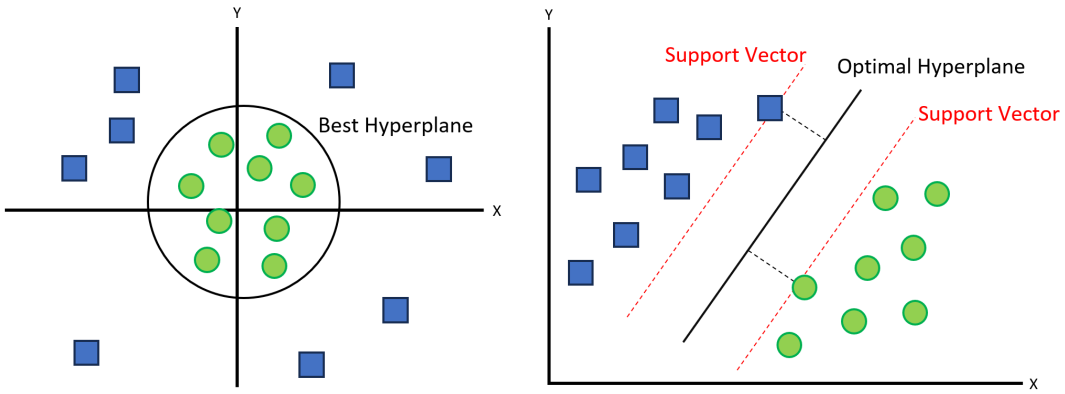
(a) Logistic Regression

(b) Linear Regression

Figure 2.11: Logistic and Linear Regression (adapted from [59])

Support Vector Machine (SVM) is another supervised learning algorithm that is used to solve linear classification and regression problems. The model is created according to a set of previously trained data and its creation is determined by the selection of the best hyperplane, which is a margin that separates the input data into their respective classes.

Linear Support Vector Machine (LSVM) is one of the most used algorithms for binary classification problems. The boundary that separates both classes is represented by a straight line, unlike SVM which can have different representations, as shown in Figure 2.12. Despite there being other algorithms with better memory and computational performance, it performs reasonably well even with large datasets [60].



(a) Support Vector Machine (b) Linear Support Vector Machine

Figure 2.12: Differences between SVM and LSVM (adapted from [60])

Naïve Bayes (NB) encompasses a family of classification algorithms that leverage Bayes' Theorem and is used when a decision of a certain class is influenced by many independent characteristics. The contribution of each characteristic is underlined not individually but rather as a group when several ones allow the algorithm to detect the corresponding class. It works well with large datasets and many variables and it can also be used in incremental learning, which will be discussed later on.

Multinomial Naïve Bayes (MNB) is one of the many specific instances of Naive Bayes that resorts to multinomial distribution and it can be used to identify an article's topic, such as politics, entertainment, sports, and others.

2.3.4.2 Deep Learning Algorithms

Deep Learning (DL) is a subset of Machine Learning that attempts to simulate the behaviour of the human brain through the use of neural networks with multiple layers that allow a progressive extraction of higher-level characteristics and features from input data [61], [62].

An Artificial Neural Network (ANN) has many node layers, also known as neurons, comprised of three main layers: an input layer, a hidden layer and an output layer. These

layers are connected to one another with associated weight and threshold levels. Figure 2.13 shows the architecture of an ANN:

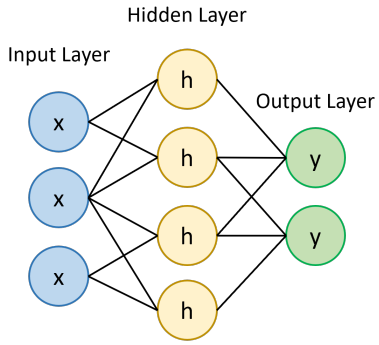


Figure 2.13: Artificial Neural Network's architecture

Hidden layers apply different weights to inputs and redirects as outputs. A node is activated when its output combined with the input layer exceeds a predefined threshold value, which then translates to the data transmission to the subsequent layer of the network. A Deep Neural Network (DNN) is an ANN with more than one hidden layer, with different weights applied to the inputs. [63].

A Convolutional Neural Network (CNN) is a class of DNNs that resorts to convolution, in which a matrix multiplication is used to generate outputs used in future training. This neural network is commonly used in image processing, although it can also be used in text classification with the aid of word vectors. Figure 2.14 shows the architecture of a CNN:

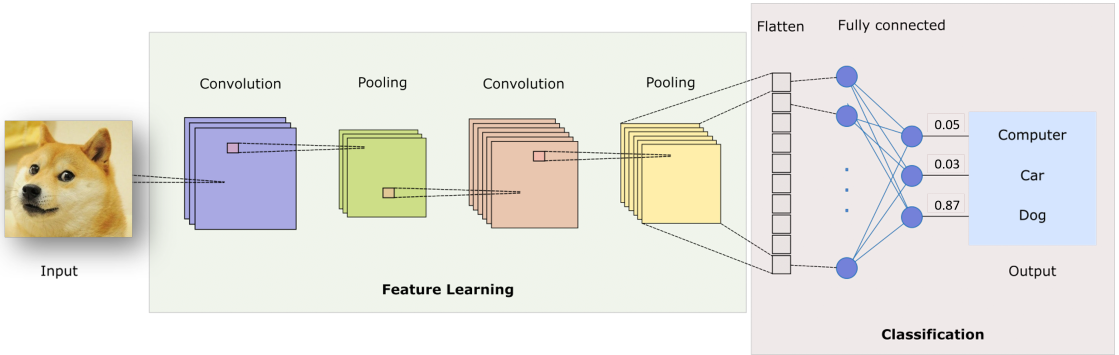


Figure 2.14: Convolutional Neural Network's architecture (adapted from [64])

A CNN has a Convolutional Layer in which a map with in-depth convoluted features is created, a Pooling Layer in which the output of the network is represented by the average mean of nearby outputs, which helps decrease both computation requirements and weights, and a Fully Connected Layer, which defines the outputs are represented, with neurons connected with preceding and succeeding ones [65], [64].

A Recurrent Neural Network (RNN) resorts to sequential data processing, as shown in figure 2.15:

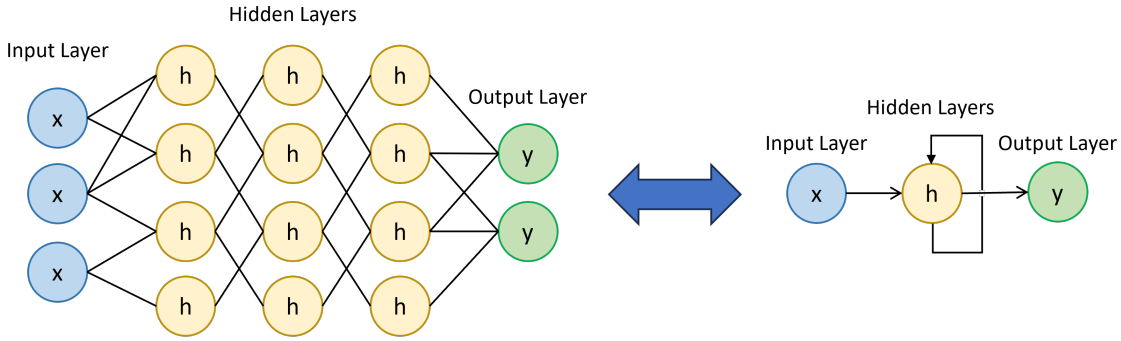
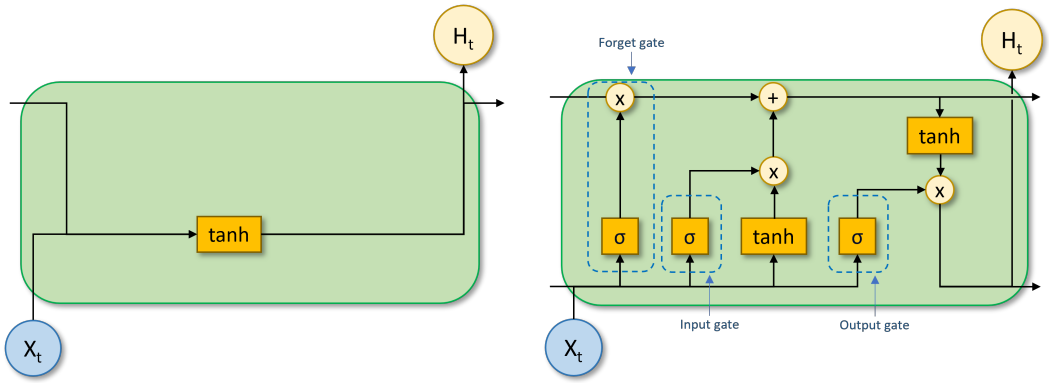


Figure 2.15: Recurrent Neural Network's architecture

While DNNs perceive inputs and outputs as two independent elements, RNNs consider information from preceding inputs to determine the current input and output of the sequence, while also sharing parameters across each layer of the network. These aspects make it possible for RNNs to capture relevant contextual information by finding patterns associated with training data dependence. This neural network is used for many purposes, including NLP, sentiment analysis, voice recognition and more.

An increase of layers using activation functions makes it harder to train due to the decreasing gradient value, which corresponds to a loss of past information. Long Short-Term Memory (LSTM) is a type of RNN capable of avoiding this issue by memorising prior information and, as a consequence, interpreting data in different ways [66].

As portrayed in figure 2.16, when compared to an RNN unit which only has a hyperbolic tangent function to regulate the output of the neural network, an LSTM unit has and additional layer/gate called "forget gate" with a sigmoid function that determines which long-term information should be kept.



(a) RNN unit (b) LSTM unit

Figure 2.16: Differences between RNN and LSTM (adapted from [67])

Moreover, the input layer also resorts to a sigmoid function to select important input

data, along with the output layer in which the output is built with both long-term and recent short-term information. This is one of the most used neural networks in NLP given its ability to grasp the context of words while also being able to filter relevant information, in order to grasp their meaning and derive accurate predictions [67].

Bi-Directional Long Short-Term Memory (Bi-LSTM) is similar to LSTM, but its input flow is bi-directional, which means both future and past information can be preserved. This is accomplished by using two separate LSTMs that register information from all input and output nodes [27], [31].

2.3.4.3 Machine and Deep Learning Libraries

TensorFlow is an open-source library developed by Google with many tools to process and load data and build Machine Learning models. The recommended language to use is python, although TensorFlow also supports C++ and JavaScript. TensorFlow 2 is the improved version of TensorFlow and is better than its predecessor, given the fact that it was developed taking into account many drawbacks of TensorFlow [68], [69].

Keras is TensorFlow 2's high-level API, an accessible and efficient interface for addressing machine learning challenges, with an emphasis on Deep Neural Networks. [70].

Transformers provides APIs and tools that grant access to state-of-the-art pre-trained deep learning models that can be trained and fine-tuned to accomplish a wide variety of tasks from many fields, including natural language processing and text classification [71].

Sklearn's libraries also offer many different machine learning algorithms, alongside data mining and analysis tools.

2.3.5 Performance Results and Evaluation

The performance of each classification model can be evaluated by resorting to different validation procedures, such as K-Fold Cross-Validation and other variants, followed by the analysis of their corresponding Confusion Matrices.

2.3.5.1 K-Fold Cross-Validation and Variants

K-Fold Cross-Validation is described as a resampling procedure used to evaluate classification models. The data sample is divided into k groups in order to iteratively generate training and test subsets of data that are used afterwards to estimate the performance of the model with future data that is bound to be different from the one used during its training. Besides allowing the reduction of overfitting, this method is simple to use and often estimates future predictions reliably.

After shuffling the dataset randomly and splitting it into k groups, each group goes through the following steps:

1. Select the group as either a hold-out or test data set.

2. Select the remaining groups to form the training data set.

3. Train a model using the training set and evaluate its performance on the test set.

4. Keep the evaluation score and discard the model.

Finally, the skill of the model is summarized and estimated based on the evaluation scores. In order to obtain a reliable estimation, it is crucial to apply a reasonable value to the parameter k. Otherwise, a poor estimation is very likely to happen, either with a high variance, which changes based on the data used, or high bias, such as very optimistic.

Applying a value of 5 or 10 to k tends to generally generate an estimation with low bias and reasonable variance. The higher the value, the smaller the difference between the training set's size and the subsets, which translates to a less significant bias nature.

There are also many other variants of Cross-Validation. Train/Test Split is one of them, in which a single split of the dataset into a training and test subset is done, thus applying a value of 2 to k.

Leave-One-Out Cross-Validation is another variant, in which the value of k is set to the total size of the dataset. Despite its longer computation time, this method considers every single observation in the hold-out dataset.

Another variant is Stratified Cross-Validation, in which the dataset splitting into folds is performed according to certain criteria, such as ensuring that each fold has the same proportion of observations as the class outcome value, for example.

Nested Cross-Validation or Double Cross-Validation is also another variant. Each fold of cross-validation may employ k-fold cross-validation, in order to conduct hyperparameter tuning during model evaluation [72].

When it comes to Deep Learning, one can resort to Cross-Validation as long as the dataset does not have a considerable size, otherwise, the process tends to be very slow. A typical solution to this problem is selecting a random subset of training data as a holdout set for the model's validation [73].

It is important to note that it is possible to obtain several different results for the same algorithm, same data and same machine. This often happens due to many reasons, including the evaluation procedure characteristic of the previously mentioned Cross-Validation methods and the stochastic nature of many learning algorithms, which are considered beneficial since they allow the generation and selection of different models with diverse behaviour and predictions [74].

2.3.5.2 Confusion Matrix

A confusion matrix can be applied to each algorithm after a Cross-Validation has been performed to evaluate their predictions even further. This is accomplished by dividing the possible outcomes into 4 categories:

- True Positive (TP), which is when the prediction turns out positive and the actual value is indeed positive (1 to 1).
- True Negative (TN), which is when the prediction turns out negative and the actual value is indeed negative (0 to 0).
- False Positive (FP), which is when the prediction turns out positive, but the actual value is negative (1 to 0).
- False Negative (FN), which is when the prediction turns out negative, but the actual value is positive (0 to 1).

The format of the confusion matrix is shown in figure 2.17:

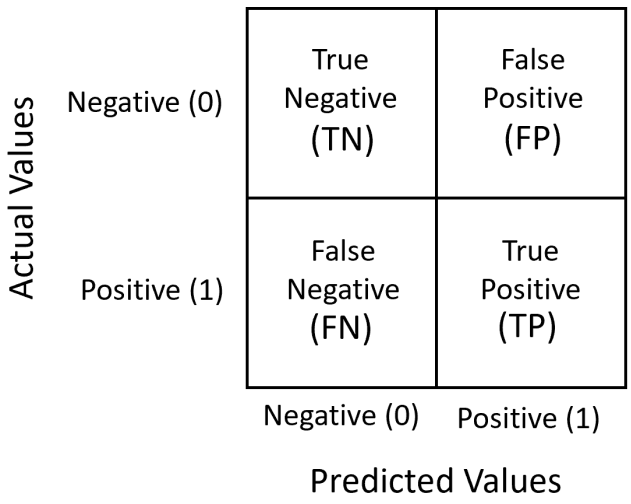


Figure 2.17: Confusion Matrix format (adapted from [75])

It is then possible to determine the Recall, Precision, Accuracy and F-measure of each model.

Recall indicates how many positive classes were predicted correctly. In this context, it measures the number of fake news articles that were correctly classified as fake out of all the actual fake news articles, as shown in equation 2.4. A higher recall means fewer instances of actual fake news articles being misclassified as real news.

$$Recall = \frac{TP}{TP + FN} \tag{2.4}$$

Precision indicates, from all the classes predicted as positive, how many are actually positive. In this context, it measures the number of correctly predicted fake news articles out of all the articles predicted as fake, as shown in equation 2.5. A higher precision means fewer instances of real news articles being incorrectly labelled as fake.

$$Precision = \frac{TP}{TP + FP} \tag{2.5}$$

Accuracy indicates, how many classes were predicted correctly, as shown in equation 2.6. Typically, a higher accuracy translates to better results, but this is not always the case. In fact, in the context of fake news detection, a model with high accuracy can predict an article as real when it is actually fake (false positive) or if an article was predicted as fake when it is real (false negative), hence the importance of the previous parameters.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.6}$$

F-measure, also known as F-score or F1-score, helps compare different models by measuring both Precision and Recall at the same time, as shown in equation 2.7. In the context of fake news, it is really important to consider a model's F-measure as it helps compare models with different Recall and Accuracy values. A higher F-measure translates to better results, so one should typically consider the classifiers with the highest f-measure values [75].

$$F - measure = \frac{2 * Recall * Precision}{Recall + Precision} \tag{2.7}$$

2.4 Web and Cloud Services

Similarly to what was described in the Web Scraping and Crawling section, the creation of a Browser Extension through typical web development elements such as HTML, CSS and JavaScript allows users to analyze the content of websites. This is accomplished by storing the developed classification models in a cloud instance, to which users can communicate through GET and POST requests that allow the sharing of not only the desired news article's data required for classification but also the respective models' predictions [20], [76].

2.4.1 Web Frameworks

Flask is an open-source microframework for web development in Python with RESTful web services that allow client requests such as POST, GET, PUT, and DELETE via HTML in many formats, including JSON, Python, Hypertext Preprocessor (PHP) or plain text [77], [78].

Django is another open-source framework developed in Python. It is a well-known framework and very easy and quick to use, but it does not support RESTful services.

CherryPy is another web-oriented framework developed in Python. It is object-oriented, hence not having such a straightforward approach when compared to previous ones [31].

2.4.2 Cloud Platforms

The most popular Cloud Platforms nowadays are Amazon Web Services (AWS), Microsoft Azure and Google Cloud Platform. Each platform has its own pros and cons regarding the number of tools and services offered, the cost, scalability, reliability, security and more.

Although any of these three platforms can be used for this project, it is important to mention that AWS is the most used Cloud Platform, holding 33% of the market share of the world's leading cloud infrastructure service providers as of the second quarter of 2023, followed by Microsoft Azure and Google Cloud Platform. This is due to AWS being established before the remaining cloud platforms, which translates to an overall better experience for a wide variety of fields [79], [80].

2.5 Smartphone Development Environments

Android Studio is the free and official Integrated Development Environment (IDE) for Android app development. It offers the ability to use a wide variety of packages and dependencies for different purposes, including access to the device's sensors, location, internet and more. Its languages include Java and Kotlin, with the latter being the most modern one with more recent features and concise syntax, hence the continuous transition from Java to Kotlin by many developers [81], [82].

Even though Android Studio also offers the ability to develop cross-platform code and modules to make Android applications work on iPhone Operating System (iOS), XCode is the free and official IDE for all Apple software, including iOS. Unlike Android which is used by many companies, iOS devices belong to Apple only, which translates to much fewer dependencies and package conflicts, as well as a quicker development process. However, only iOS applications can be developed with XCode [83], [84], [85].

2.6 Related Work

This section will explore different projects, papers and articles that have been developed regarding the detection of fake news with AI, with a focus on the processes and techniques used, along with their corresponding results and possible encountered challenges. The datasets used by the authors and researchers will not be deeply described since they have

already been mentioned before in the "Datasets" section. Different tables will then be presented summarizing the characteristics, models and techniques used in the approaches described below.

In [19] a social media fake news dataset was used to identify fake news from social media.

Firstly, the dataset went through a pre-processing stage in order to convert the unstructured data into structured data and, ultimately, generate a clean dataset to be used. To achieve this goal, a stopword filtering technique was used, as a way to allow the program to focus on more important ones that help in the news' content classification. Many words were removed, such as the articles "the", "a" and "an" and certain prepositions such as "in" and "with". Furthermore, non-English words were removed and the HTML tags of each text's structure were also removed with the `remove_tags` python function.

Next, four feature extraction methods were considered:

- TF-IDF, which proved to be the best feature extraction technique, with an average accuracy of 91.23%.
- CV, which determined the number of repetitions of a term in a document by using a two-dimensional matrix that stored each term's occurrences.
- N-Gram level vector which aided in the identification of sequences of words, thus improving the selection of the right features and corresponding numerical values [46].
- Character level vector, which contributed to understanding how convolutional neural networks performed with letters, words and whole sentences.

The tokenization of texts was also very important given the nature of the required tasks, splitting paragraphs and sentences into smaller units for the machine to understand in a better way.

After the feature extraction stage, Machine Learning and Deep Learning models were used. The Machine Learning classifiers included Random Forest, KNN, LSVM and Logistic Regression.

Boosting classifiers that also fall under the Machine Learning category were also used, namely AdaBoost, which proved very useful by getting an accuracy of 100% with the TF-IDF feature extraction method, and XGBoost, which was important in correcting prediction errors [86].

The Deep Learning models were the following:

- Artificial Neural Network.
- Recurrent Neural Network with Long Short-term Memory (RNN+LSTM), two different classifiers that were used together. As stated before, the sequence of data in

RNN is based on the current input state and the previous input state, which can be of utmost importance to determine future data. However, this is only possible by using small data sequences at a time and by forgetting previous ones, which is why LSTM was used, as they offer the ability to memorize short-term memory much better and to distinguish between relevant and irrelevant information much better [87].

- Convolutional Neural Network with Long Short-term Memory (CNN+LSTM), in which data features were extracted without prior knowledge and human intervention, which reduced the complexity and overfitting level. This approach had 100% accuracy as well.

On the one hand, this project showed different possible approaches regarding fake news detection and underlined the great performance of the Term frequency-inverse document method for feature extraction, which scored 91.23% accuracy, and the AdaBoost and CNN+LSTM models, which scored 100% accuracy. On the other hand, only one single dataset was used, which means the data complexity could have been higher, perhaps to the point where the accuracy of the models would be affected. Furthermore, there are also fake news articles made of only fake pictures or videos, to which the authors suggest the usage of datasets with this kind of data along with CNNs to classify content made of both textual and non-textual properties.

The authors in [23] developed an approach based on two models to classify news as either fake or real, with a dataset provided by QICC.

The first model corresponded to a feature-based approach in which the title and body of each article are separated to better extract similarity, using cosine similarity to compare each article's heading with the top five google search results for the same heading, along with N-grams, char-grams and tokenization. Lexicon features such as assertive verbs, factive verbs, report verbs and subjectivity were also considered, in order to determine the frequency of certain words found in each article. All these features were passed on to several machine learning classification models such as Support Vector Machine, Naïve Bayes, Random Forest and XGBoost.

The second model corresponded to a deep learning approach with a profound understanding of the English language, as well as its semantics, which helped identify stylistic differences between real news and fake news. As a consequence, top Natural Language Processing models were chosen, namely:

- Bidirectional Encoder Representations from Transformers (BERT), which helps computers understand the meaning of ambiguous language in text by resorting to surrounding text as a way to grasp its context, as we humans do when reading polysemous words, which are words with different meanings. This model is used by Google to optimize the interpretation of user search queries [88].

- Extreme Language Understanding Network (XLNET), which allows learning bidirectional contexts and overcomes certain limitations of BERT, such as its characteristic pre-train-finetune discrepancy [89], [90].

- Robustly Optimized BERT Pretraining Approach (RoBERTa), which has an improved mask language modelling when compared to BERT and, as a consequence, better downstream task performance [91].

According to the authors, these models are pre-trained language models, so there was no pre-processing step needed. Each model can process each word with their own tokenizer, although the authors indicated the need to fine-tune the classification layer of each model based on the task at hand of detecting fake news.

In terms of results, the deep learning models achieved the best performance, with the XLNET classifier achieving 98% accuracy, and the best feature extraction technique was the cosine similarity feature method, which underlines the importance of identifying the news' source and comparing it with well-known sources. The authors indicated that only one epoch was needed for the models to properly learn the data, meaning one complete pass through the training data was sufficient [92]. Finally, they also suggest using fact-checking websites along with the Google searches performed in the cosine similarity feature method as a way to gather data from more different sources and, consequently, increase the model's performance.

These machine learning classifiers were also considered in [24] to detect fake news.

After converting the dataset's Tabulation Separated Values (TSV) format to CSV, the dataset is pre-processed through what the authors call "rough noise removal". Similarly to what has been described in the pre-processing steps of previous approaches, this step consists of the removal of ids, dots, commas and quotations, as well as stemming and deleting suffixes with the aid of NLP NLTK libraries and SAFAR v2 library. This was followed by tokenization as well with POS.

The feature extraction process that followed also resorted to TF-IDF n-gram features as already mentioned before, with the TF-IDF Vectorizer function of python Sklearn, taking into account lexical features such as word count, average word length, article's length, number of adjectives and more.

The dataset was divided into 70% for training the classifiers and 30% to test them, as well as 75% and 25% respectively, an aspect that was not mentioned in previous papers and that can affect the models' accuracy, with both situations in the recommended range of training/testing relation percentage [93].

The machine learning classification models used include XGBoost, Random Forest, Naive Bayes, K-Nearest Neighbours, Decision Trees and Support Vector Machine. As found in previous papers, the best models were XGBoost, with an accuracy of 75%, and SVM and Random Forest, with 73% accuracy.

The authors in [25] used different types of neural networks to detect fake news.

As expected, the project begins with the pre-processing phase of the dataset by resorting to many techniques that have already been mentioned before, such as stop word removal with NLTK python library, punctuation removal and stemming. After the pre-processing step, two feature extraction methods were used in order to convert the raw text into numerical features: BoW and TF-IDF.

The resulting dataset was then split with a 67/33 % ratio for the models' training and testing, respectively. One thing the authors also mention is the division of the training data into validation sets (80/20), which was used to evaluate the models in order to perform model selection, with a 3-Fold Cross-Validation setup [94].

In terms of classification models, three different variations of neural networks were used along with the two feature extraction methods explained before:

- TF-IDF with DNN.
- BoW with DNN.
- Pre-trained word embedding with Neural Networks.

The best-performing model was the TF-IDF Vectors with Dense Neural Network model, which achieved an accuracy of 94.31%, but some misclassification rate of the stance "disagree" as "agree" was found. The authors also mention the Hyperparameter Tuning Performance phase that helped tune the neural network's hyperparameters, mainly:

- Final layer activation function, which varies on the project's goal, with Softmax being the best [95].
- Batch size, the number of samples (single rows of data) to work through, which was set to 64 [96].
- Dropout rate, used to prevent overfitting and set to 0.1.
- Epochs, the number of times an algorithm cycles through the entire data set, with 50 epochs achieving the best results.
- L2 penalty, used to prevent overfitting and set to 0.0001.

For future work, the authors mention the possibility of analysing new datasets from other sources, such as Twitter and Facebook.

A more specific approach was considered in [26], with only Machine Learning Ensemble Methods to detect fake news.

The authors make a very notable point when it comes to the type of solution obtained based on the dataset involved. As a consequence, it is suggested an approach with the goal of detecting fake news from different fields rather than a single one, including politics, sports, technology, entertainment, among others.

The pre-processing step is very similar to previous ones, as well as the feature extraction phase, although the latter involved the extraction of 93 different features from text, including the percentage of words that portrayed positive or negative feelings, stop words, informal language, punctuation, adjectives, verbs, prepositions and more. The resulting dataset was divided into 70% for training the classifiers and 30%, as seen in previous approaches.

In terms of classification models, the Multilayer Perceptron (MLP) was used, an artificial neural network that was fine-tuned to use ReLU as the activation function with the Adam Solver and 3 hidden layers, along some other models that have already been previously mentioned, including Logistic Regression, Random Forest, KNN, SVM, XGBoost and AdaBoost. The classifier's performance varied within the 3 datasets, but the models with the best performance in the final dataset with all data were Random Forest with an F1-score of 91% and Multilayer Perceptron, decision trees and XGBoost with 90%.

The authors in [27] analysed the different methods used to detect fake news by studying a wide variety of papers and articles found within this area. Despite the authors not putting any of the techniques into practice, the work developed is extrMLy important since it explores many of the possible text processing approaches, which were similar to previous ones, and classifiers involved in the detection of fake news, as well as the performance expected of each classification model based on the respective dataset.

Three datasets were taken into account:

- Combined corpus dataset with fake and real news, in which Bi-LSTM achieved an F1-score of 95%, followed by Naive Bayes, CNN and LSTM with 93%.
- LIAR dataset from Kaggle, in which Naive Bayes achieved an F1-score of 59%, followed by Bi-LSTM and CNN with 58%.
- Fake news dataset comprised of data gathered from sources such as Kaggle and ISOT, in which Generative Adversarial Network (GAN) achieved an F1-score of 87%, followed by Naive Bayes and CNN with 86%.

On the one hand, the authors stated the performance of deep learning methods was far superior to machine learning ones, apart from Naïve Bayes which had similar scores to DL models. On the other hand, the authors commented on deep learning models being more prone to overfitting when applied to datasets with great amounts of data, such as the corpus dataset that was used.

Nevertheless, Bi-LSTM was the model with the best overall scores, with an average 95% accuracy and F1-score.

A dataset was created in [20] with the aid of a web crawler and the Facebook API, which had information about each user, including their profile name, age, number of friends, profile picture, number of groups and events participated, number of liked pages, news interest, number of following users, number of links shared in posts, number of hashtags, updated location, and more. In terms of news content, the dataset included different aspects, such as the source, headline, body, type (text, images or videos), writing style (use of chapters, quotes and external links), date of post and more.

After pre-processing the stored information, the authors resorted to different ML and DL models, including KNN, SVM, Logistic Regression, Decision Trees, Naïve Bayes and LSTM. To evaluate their performance, the authors resorted to a 10-fold Cross-Validation technique with different feature combinations. The results showed the combination of user's content and news content achieved the best accuracy and the number of messages with hashtags, URL, and pictures with multiple captions contained more fake news than other posts.

The best algorithm was LSTM with 99.42% accuracy, followed by KNN and SVM, with 99.3%. The authors then proceeded to the implementation of a chrome extension to allow users to detect fake news by selecting pieces of text and applying the different classification models previously mentioned.

For future work, the use of other deep learning algorithms such as Bi-LSTM is considered, along with boosting classifiers and more datasets.

To identify fake news in [28], the authors resorted to pre-processing steps such as stopword removal and stemming. Duplicated values were removed, but null ones were still considered to avoid removing whole entries with possible relevant information.

The pre-processing step was followed by a Feature Engineering step, which aimed to create a feature set that would summarize the original dataset to speed up the models' training phase and respective learning accuracy. A total of 87 linguistic features were used, including the number of special characters, determinants, verbs, adjectives, adverbs, capital letters, words, syllables, text polarity and more.

In terms of feature extraction, Count Vectorizer and TF-IDF were considered, along with Word Embeddings over Linguistic Feature Sets (LFS). Different experiments determined that CV with LFS achieved the best accuracy, hence being used with various ML models.

Different ML models were used, including SVM, NB, KNN, Decision Trees, Bagging, and AdaBoost. The train and test subsets were split in different ratios, but 70/30 ratio achieved the best results. SVM achieved an F1-score of 96.56%, followed by AdaBoost with 95.02% and Bagging with 95% on the WELFake dataset, which was higher than state-of-the-art models such as BERT and CNN with F1-scores of 93.75% and 92.36%, respectively.

For future work, the authors intend to resort to knowledge graphs and user credibility to verify the model's output.

The authors in [5] resorted to different pre-processing techniques to clean it, including the removal of URLs and punctuation marks and tokenization.

A Named-entity Recognition (NER) approach was considered for feature extraction, which is able to classify unstructured text and extract relevant features categorized into linguistic and sentiment aspects. Some of these features included the news source, number of stopwords, number of "@", number of numeric values and lowercase characters, text language, character count, average word length, positive, neutral, negative and compound sentiment values, number of punctuation marks and tags, sentence length, and more.

In terms of classifiers, many different ML algorithms were used, namely Random Forest, AdaBoost, Decision Trees, KNN, Support Vector Classifier (SVC), which is an SVM used for linear-based approaches, and Bagging. The dataset was split into training and testing subsets with a 70/30% ratio and two different experiments were conducted, before and after Feature Extraction, in Google Colab, an open-source cloud-based Jupyter Notebook environment with Graphical Processing Unit (GPU) access [97].

The results showed the importance of feature extraction, with better performance in the second experiment due to the removal of frequent words that had no benefit to the classification phase. SVC achieved the highest F1-measure, scoring 88.76% and Random Forest achieved the highest accuracy and precision scores, with 88.5% and 87.77%, respectively.

For future work, the author underlines the importance of taking into account audio clips, images and video clips in the dataset, as well as Deep Learning approaches and more NLP processing techniques.

The authors in [21] resorted to a few pre-processing techniques, including link, non-alphanumeric characters and stopword removal, along with TF-IDF vectorizer for feature extraction.

In terms of ML classifiers, Logistic Regression, SVM with linear kernel, Decision Trees and Gradient Boosting were used for the binary classification task. In the end, SVM was the best classifier in both validation and test subsets, with an F1-score of 93.46% and 93.32%, respectively, followed by Logistic Regression with an F1-score of 92.75% and 91.96%, respectively.

For future work, the authors state that DL models would be worth exploring as well, along with a larger dataset and more details regarding the posts and articles, including the reasoning behind their respective labels.

After creating the "Fake.Br corpus" dataset in [29], the authors considered three different feature extraction approaches: a BoW approach, a Word2Vect approach and a FastText approach. Many classifiers were considered as well, including, Decision Trees, Random Forest, Logistic Regression, SVM and Bagging. Different pre-processing techniques were also used, including stop-words removal and lemmatization.

The different experiments showed the best feature extraction approach was BoW and the best classifier was Logistic Regression, with an F1-score of 95%.

After creating the "FakePT" dataset in [30], many pre-processing techniques were applied afterwards to clean the dataset, including punctuation removal, tokenization, stop-words removal, change case to lowercase, POS and n-grams.

For feature extraction, the author resorted to a Word Embedding approach and a TF-IDF with BoW approach to transform unstructured text data into many relevant features. One Hot Encoding was used to represent news categories, in which a sparse vector is used for each article to determine their corresponding category.

Many ML and DL models were used, including Random Forest, Extra Trees, MNB, K-Nearest Neighbours, LSVM, SVM, Gradient Boosting, CNN and LSTM. The data was divided into a training and testing dataset according to a 70 to 80%/30 to 20% ratio, respectively. Each classifier's performance was evaluated with two Cross-Validation techniques: K-Fold Cross-Validation and Stratified K-Fold. Stratified K-Fold consists in shuffling the dataset just once before splitting the dataset into k folds, followed by a fold split with the same ratio of observations in each class.

For feature selection, many different tests were conducted by combining the title, summary, category and source of each article with the feature extraction approaches previously mentioned. The results showed that the average accuracy of the dimension "Title" was worse than Summary.

According to the author, this difference in performance was due to the fact that, on average, an article's summary had almost four times more words than its title, and this extra information is crucial for classifiers. As a consequence, the best set of pre-processing techniques was "Punctuation Removal; Lemmatization; Stopwords removal; N-grams TF-IDF; Lowercase" using the "Summary" dimension, Bag of Words with TF-IDF approach, a balanced dataset, and MNB as the classifier.

Once the best set of pre-processing techniques was determined, two different approaches were considered as a way to evaluate the performance of the previous techniques and classifiers: one with a balanced dataset, with half fake, half real news, and another with an unbalanced dataset, 77% fake and 23% real, more specifically. The K-Fold Cross-Validation technique was used in the balanced dataset, while the Stratified K-Fold was used in the unbalanced one.

For the balanced dataset, the best F1-score was 0.74, using the combination of "Summary + Source", Bag of Words with TF-IDF as the text representation technique, and Extra Trees as the classifier. For the unbalanced dataset, the best F1-score was 0.73, using the combination of "Title + Summary" and "Category" as the dimensions, Bag of Words with TF-IDF and One Hot Encoding for feature extraction and MNB as the classifier.

Finally, the author comments on the possibility of identifying news topics to improve predictions, as well as using real news instead of news from fact-checkers, which tend to be false most of the time and, as a consequence, may lead to overfitting.

The authoress in [31] resorted to many different pre-processing techniques, including punctuation removal, lowercase, tokenization with uni-grams and bi-grams, stop-words removal and stemming. These were applied to "title", "content" and "description" variables, while the variables "category" and "source" were associated with Label Encoder and One-Hot Encoding approaches.

Label Encoder was used to assign a number to each category and One-Hot Encoding was used to convert all sources to a single matrix with binary values. TF-IDF and CV were used for feature extraction and, afterwards, several experiments with different combinations of variables were conducted regarding both multi-class and binary approaches.

In the multi-class approach, three classifiers were considered, namely MNB, Random Forest and Gradient Boosting, with the data split into training and testing subsets with an 80/20% ratio, respectively. TF-IDF allowed the generation of a less complex model with improved performance when compared to CV, but the results had low accuracy, mainly due to the dataset's subjective criteria, which had a negative impact on models.

In the binary approach, the classes were manually adapted: news classified as "partially true" or "true" were joined in a single "true" class, while "partially false" and "false" ones were joined in a single "false" class, in order to avoid the problems previously mentioned. After several experiments with different combinations of variables, the title + description combination stood out, along with source and category. The data was split into training and testing subsets with a 90/10% ratio, while also resorting to Uni-grams and Bi-grams.

Three different models were developed afterwards. The first one focused on a text analysis of the titles and descriptions of news articles, with TF-IDF and MNB classifier, which scored 95% accuracy. The second one focused on categories and sources of news articles, along with their titles and descriptions, and Random Forest achieved 87% accuracy. The third model was created by joining the other two models, in which SVM achieved an accuracy of 92%. In this third model, news articles ended up being classified as "True", "Doubtful" or "False" based on the probability of "True" and "False" classes.

For future work, the author indicates the use of a more balanced dataset may increase the robustness of the classification models, given the fact there were much more news articles classified as "true" rather than "false" in the dataset, which may have affected the classifiers' performance and predictions. Furthermore, the implementation of a web crawler could also be considered as a way to automate news articles extraction.

2.6.1 Methods and Techniques Summary

The following tables present a summary of the main techniques and methods regarding both text processing and classification phases that have been previously described, starting off with the main text processing techniques in table 2.4:

Table 2.4: Main text processing techniques

	Stop-words	Tokenization	Lemmatization	Stemming	Punctuation	POS	Sentiment Analysis	N-grams	NER	One-Hot Encoding
Abdulrahman and Baykara	x	x						x		
Antoun et al.		x					x	x		
Khanan et al.		x		x	x	x		x		
Thota et al.	x			x	x					
Ahmad et al.	x				x		x			
Mishra et al.	x		x					x		
Verma et al.	x			x						
Khan et al.	x				x		x		x	
Patwa et al.	x									
Silva et al.	x		x							
Rodrigues	x		x		x	x		x	x	x
Teixeira	x	x		x	x			x		x

Certain techniques stand out from others, such as stopword, punctuation removal, n-grams and tokenization. The existent works dealing with European Portuguese news classification lack any insights regarding the impact of sentiment analysis, thus raising a possible approach to determine whether fake news and real news differ from each other in that regard.

Table 2.5 shows the main feature extraction techniques explored in the State of Art:

Table 2.5: Main feature extraction techniques

	TF-IDF	BoW	Word Embedding
Abdulrahman and Baykara	x	x	
Antoun et al.			x
Khanan et al.	x		
Thota et al.	x	x	x
Mishra et al.	x		
Verma et al.	x	x	x
Khan et al.			
Patwa et al.	x		
Silva et al.		x	x
Rodrigues	x	x	x
Teixeira	x	x	

TF-IDF was the main technique, followed by BoW and Word Embeddings.

Table 2.6 shows the main ML classifiers considered in the literature:

Table 2.6: Main ML classifiers

	Decision Trees	Random Forest	Extra Trees	KNN	SVM and Variants	Naïve Bayes and Variants	Logistic Regression
Abdulrahman and Baykara		x		x	x		x
Antoun et al.		x			x	x	
Khanan et al.	x	x		x	x	x	
Ahmad et al.		x		x	x		x
Mishra et al.	x				x	x	x
Sahoo and Gupta	x			x	x	x	x
Verma et al.	x			x	x	x	
Khan et al.	x	x		x	x		
Patwa et al.	x				x		x
Silva et al.	x	x			x		x
Rodrigues		x	x	x	x	x	x
Teixeira	x	x		x	x	x	x

Many projects share a wide variety of ML classifiers, with SVM and variants, Logistic Regression, KNN, Decision Trees and Random Forest being the most common ones.

Table 2.7 shows the main ML ensemble classifiers found in the projects of the State of Art section:

Table 2.7: Main ML ensemble classifiers

	Bagging	Adaboost	XGBoost	Gradient Boosting
Abdulrahman and Baykara		x	x	
Antoun et al.			x	
Khanan et al.			x	
Ahmad et al.		x	x	
Mishra et al.		x		
Verma et al.	x	x		
Khan et al.	x	x		
Patwa et al.				x
Silva et al.	x			
Rodrigues				x
Teixeira			x	x

AdaBoost and XGBoost were the most used ML ensemble classifiers, followed by Bagging and Gradient Boosting.

Table 2.8 shows the main DL models explored in the State of Art:

Table 2.8: Main DL models

	ANN	DNN	CNN	RNN	LSTM	Bi-LSTM	MLP	GAN	BERT	RoBERTa	XLNET
Abdulrahman and Baykara			x	x	x						
Antoun et al.									x	x	x
Thota et al.	x	x									
Ahmad et al.							x				
Mishra et al.			x		x	x		x	x	x	
Sahoo and Gupta					x						
Verma et al.			x								
Rodrigues			x		x				x		
Teixeira					x						

The most common DL models include LSTM, CNN and BERT. Table 2.9 shows the best classifiers and models of the previous projects:

Table 2.9: Best classifiers and models of each project

Abdulrahman and Baykara	Adaboost and CNN + LSTM (100% Acc)
Antoun et al.	XLNet (98% F1-score)
Khanan et al.	XGBoost (75% Acc)
Thota et al.	DNN (94.31% Acc)
Ahmad et al.	Random Forest (91% F1-score)
Mishra et al.	Bi-LSTM (95% F1-score)
Sahoo and Gupta	LSTM (99.42% Acc)
Verma et al.	SVM (96.56% F1-score)
Khan et al.	SVC (88.76% F1-score)
Patwa et al.	SVM (93.46% F1-score)
Silva et al.	Logistic Regression (95% F1-score)
Rodrigues	MNB (73% F1-score)
Teixeira	MNB (95% Acc)

Overall, SVM and variants and LSTM were the ones with the best results.

3

Project Development

3.1 Proposed Approaches

Taking into account all the previous techniques and processes, it is possible to develop different approaches regarding the detection of fake news in both English and European Portuguese languages. Unlike in the English language in which a great number of projects have already been developed, only a few methods have been put into practice when detecting fake news in European Portuguese.

As a consequence, the room for creativity and improvement is much larger in the latter, especially considering certain methods that have not been used yet. These include the sentiment analysis of news content during the pre-processing phase, along with different feature selection methods that may prove useful after feature extraction, in order to not only better select important features, but also remove irrelevant ones that may affect the training of the models and resulting predictions. Furthermore, there are many other ML and DL models that have not been used yet.

Since no dataset is publicly available with both real and fake news in European Portuguese, the need to create one from scratch by resorting to Web Scraping arises, as found in some of the projects discussed in the previous chapter. As for the English language, many different datasets can be used.

With these aspects in mind, the approach regarding the detection of fake news in English can be divided into the following steps:

1. Choose the dataset(s).
2. Apply different pre-processing techniques to obtain a cleaner version of the dataset.
3. Use different feature extraction and selection methods to identify and filter relevant characteristics within the data.
4. Train different ML and DL models and evaluate their performance with Cross-Validation techniques and corresponding Confusion Matrices.
5. Identify the model with the best performance for future predictions.

Figure 3.1 represents the approach previously mentioned:

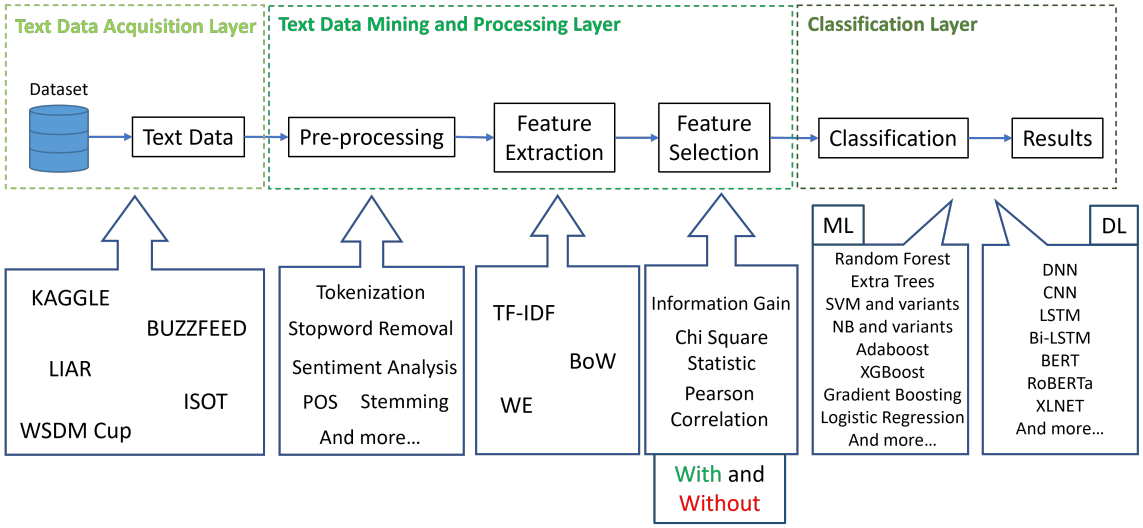


Figure 3.1: Fake news identification in English

The approach regarding the detection of fake news in European Portuguese is very similar to the previous one, apart from the use of datasets in the first step, which is replaced by the implementation and use of Web Scrapers to create a dataset by gathering real and fake news articles from trustworthy and untrustworthy websites, respectively. Figure 3.2 represents this approach:

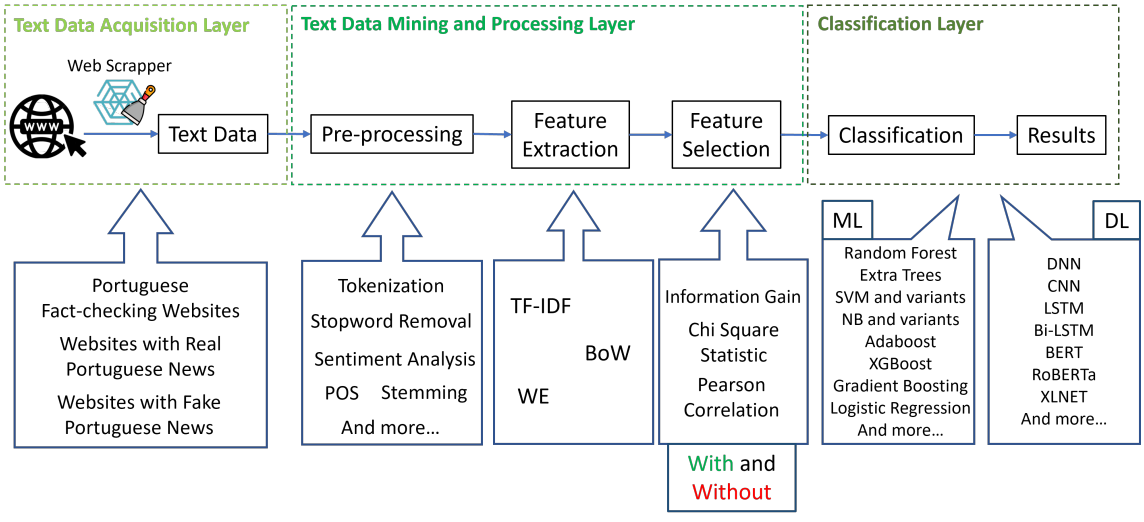


Figure 3.2: Fake news identification in European Portuguese

The resulting dataset and classification models will be available to users via cloud and web-based services. This would allow both analysis and submission of content classified as either real or fake, in order to improve future predictions. Figure 3.3 illustrates this process:

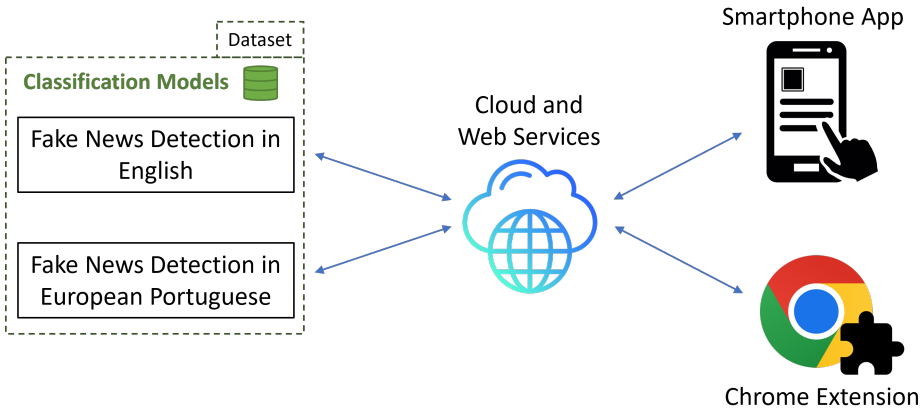


Figure 3.3: Cloud and web services with Chrome extension and smartphone app

Figure 3.4 shows the final pipeline with all the previous steps that were considered and developed:

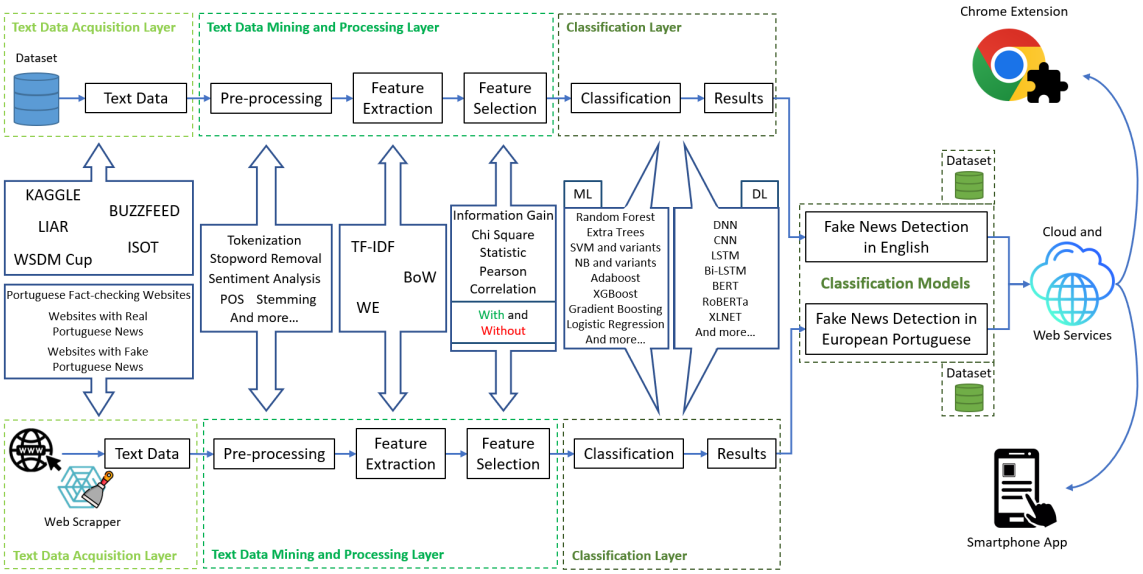


Figure 3.4: Final pipeline of the project

3.2 English Approach

This section will explore the different steps behind the English approach, starting off with the Dataset.

3.2.1 Dataset

The first step in the English Approach is related with the dataset creation. The final English dataset comprises three different datasets:

1. The "WELFAKE dataset", which, as already described before, contains 72,134 news articles, with 35028 real and 37106 fake news extracted from Kaggle, McIntire, Reuters and BuzzFeed Political [98].

2. The "Misinformation & Fake News text dataset" created by Steven from Kaggle, which contains 79000 articles of misinformation, fake news and propaganda. The real articles were extracted from reputable sources such as Reuters and the New York Times while the fake articles were extracted from right-wing extremist websites such as Redflag Newsdesk and Breitbart, from the public dataset described by Ahmed et al. in their article and from the EUvsDisinfo project, which fact checks disinformation cases originating from pro-Kremlin media shared across Europe [99], [100].

3. The COVID-19 Fake News dataset created by Patwa et all. that was previously mentioned, with a total of 10479 statements from several social media posts, comments and news. The dataset was split into different training, testing and validation subsets for people to train and test their models and for the contest's organizers to measure their performance, so these three subsets were merged to create the third and final dataset of this project [101].

Combining these datasets into a single one makes it possible to gather a lot of data that is fit for the task at hand. A wide variety of sources were also considered, ranging from well-known fake news websites and real news websites to fact-checking websites and social media. Different topics were also taken into account, including Politics, the COVID-19 Pandemic and the on-going war between Russia and Ukraine, thus improving the models' predictions on future data related to said topics.

Even though these datasets offer the ability to get access to relevant data for the tasks at hand in a very straightforward way, the text data that they provide still requires some cleaning before the pre-processing phase. This includes removing rows with null or duplicated elements to avoid feeding the models with irrelevant data, as well as filtering only text data rows written in English, which was accomplished by resorting to the library Langid to classify each text's language and removing non-English entries [102].

3.2.2 Exploratory Data Analysis

To achieve a more objective analysis and interpretation of future results, it is convenient to start with Exploratory Data Analysis. The plot in figure 3.5 shows the resulting balance of real and fake news after cleaning the three datasets and merging them into a single one:

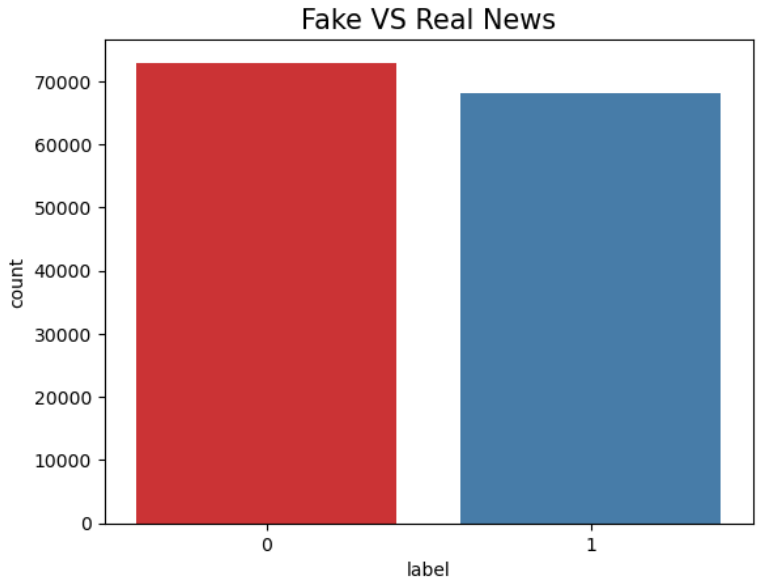


Figure 3.5: Balance of fake (red) and real (blue) news in English

The English dataset has over 140 000 articles, with a total of 72925 fake and 68167 real news and statements. The word cloud in figure 3.6 was created with the WordCloud package and it shows the 500 most frequent words in the entire dataset, including "trump", "president", "said", "would", "people" and more:

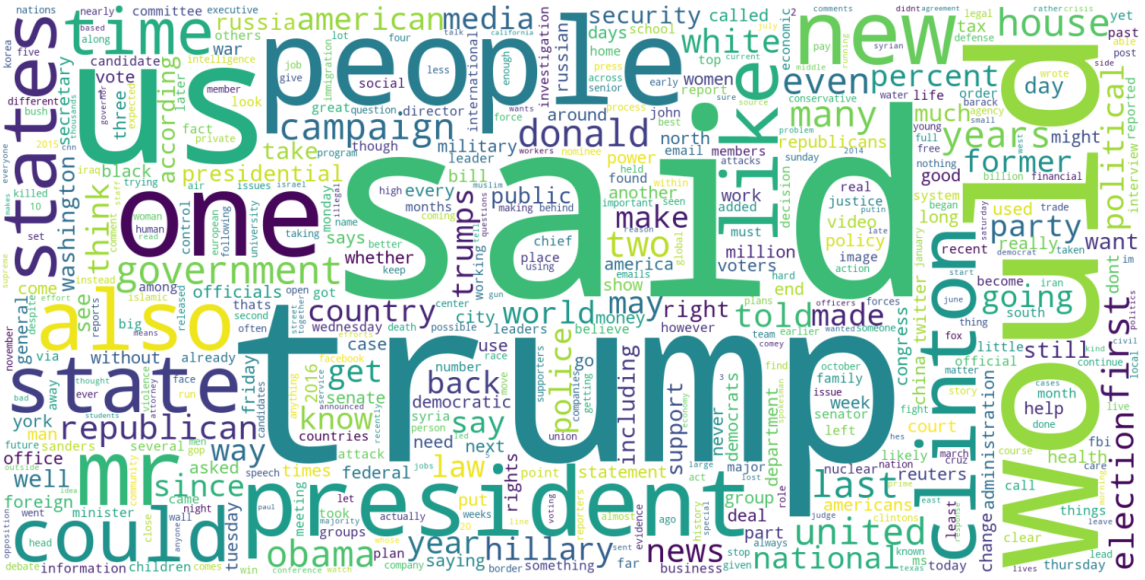


Figure 3.6: The 500 most common words in the English dataset

The word cloud in figure 3.7 shows the 500 most frequent words in fake news, including "trump", "president", "said", "would", "people" and more:

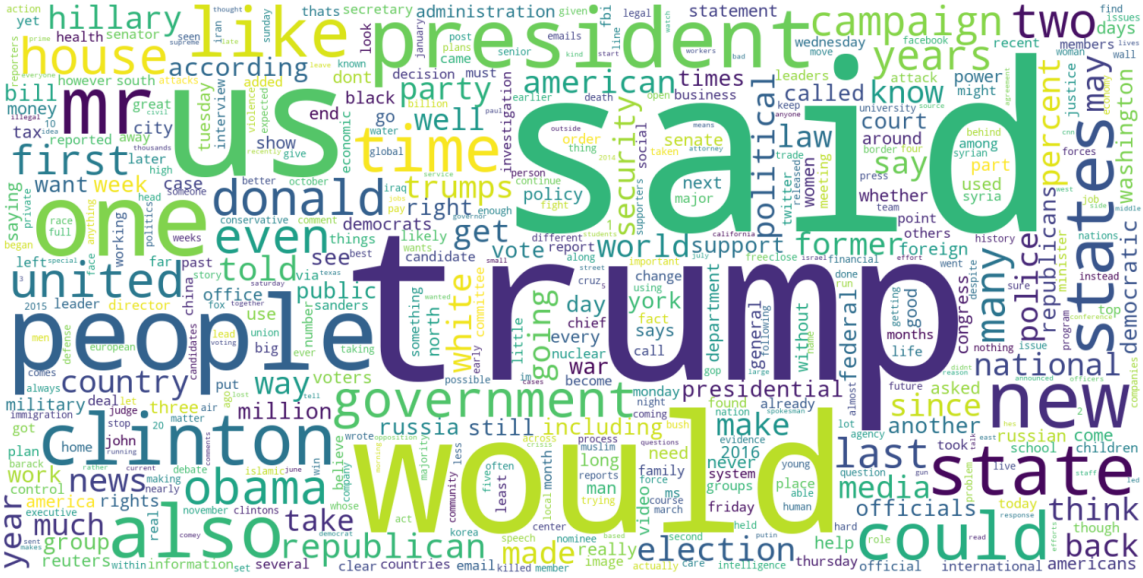


Figure 3.7: The 500 most common words in English fake news

The word cloud in figure 3.8 shows the 500 most frequent words in real news, including "trump", "president", "said", "would", "people" and more:

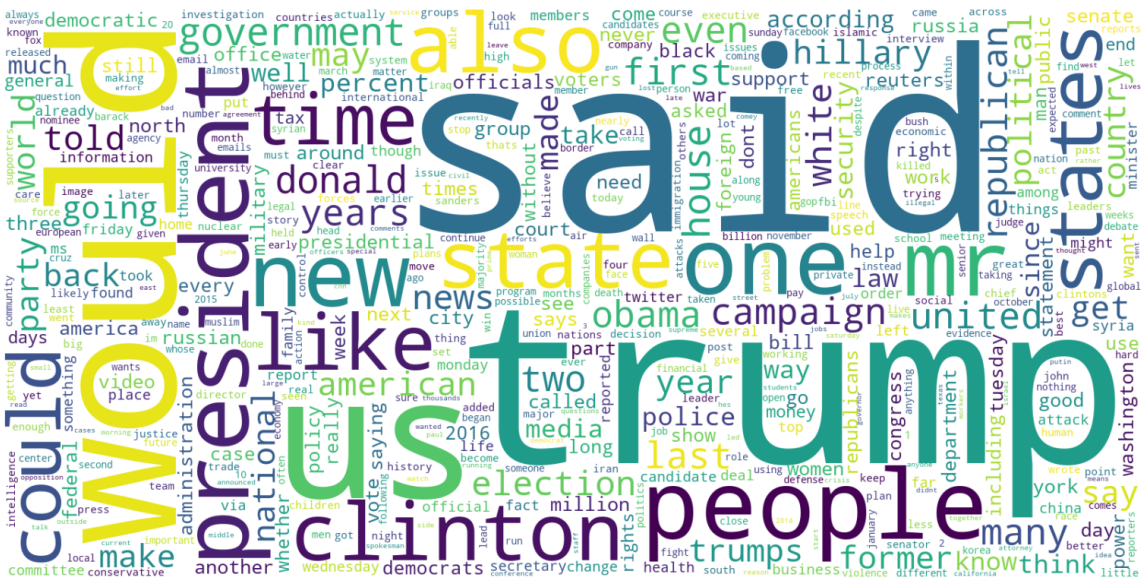


Figure 3.8: The 500 most common words in English real news

The word cloud in figure 3.9 shows the 300 most frequent bi-grams in fake news, including "Donald Trump", "United States", "white house", "New York" and more:

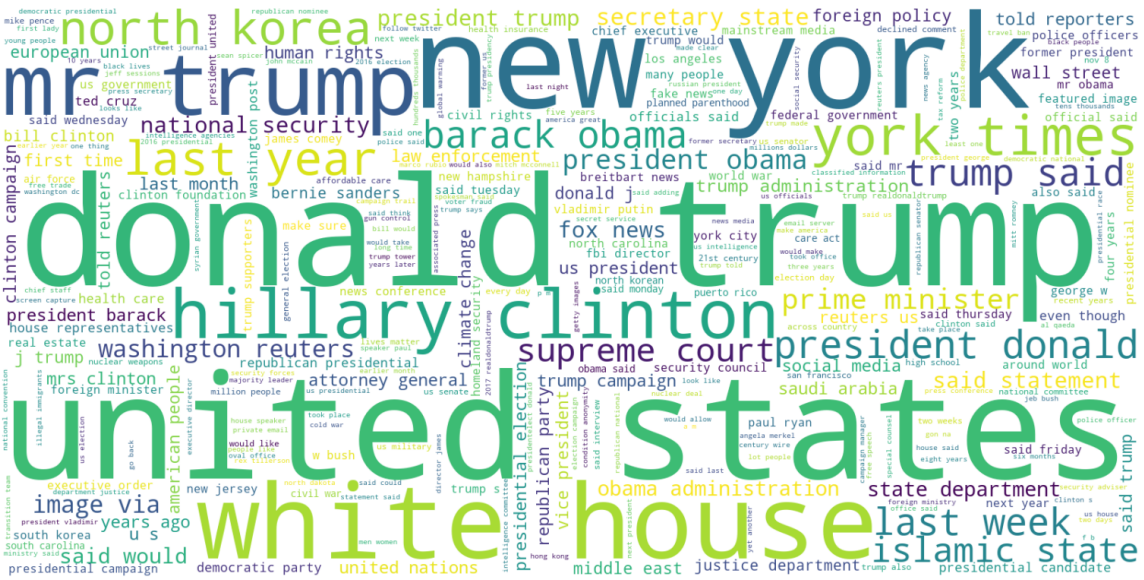


Figure 3.9: The 300 most common bi-grams in English fake news

The word cloud in figure 3.10 shows the 300 most frequent bi-grams in real news, including "Donald Trump", "United States", "white house", "New York" and more:

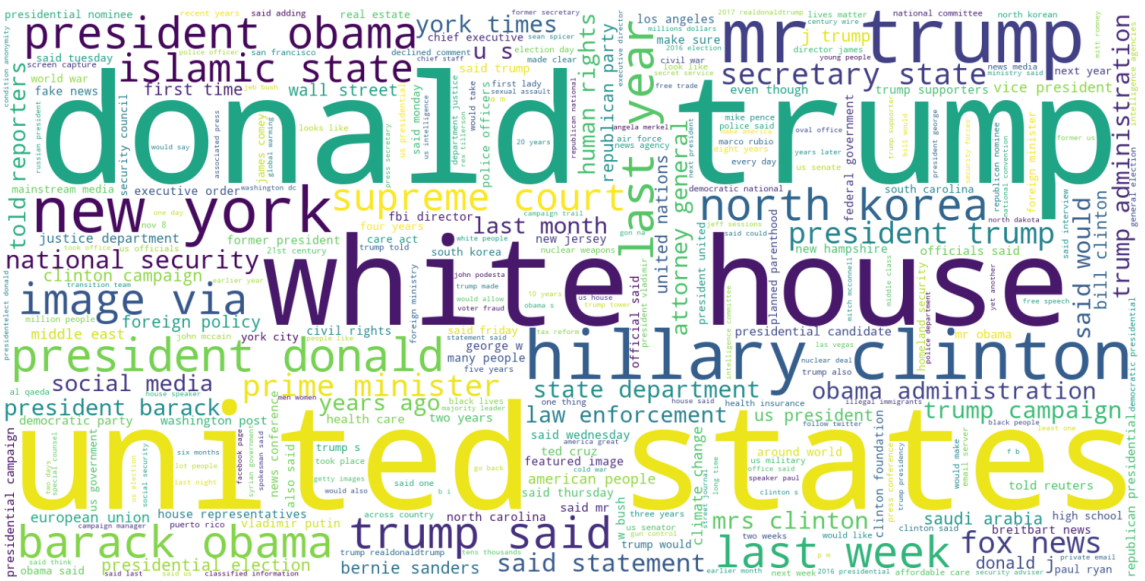


Figure 3.10: The 300 most common bi-grams in English real news

Taking into account the previous results, it is clear that Politics is the main topic discussed in the vast majority of the dataset, but the rows with articles and statements about the COVID-19 pandemic and the on-going war between Russia and Ukraine represent more recent and relevant events in the context of fake news and misinformation, thus playing an important role in this project, nevertheless.

Moreover, both fake and real news share many common words, even bi-grams, which could prove challenging in both training and prediction phases as it can impact the models' capacity to set the line separating real and fake news. On the other hand, this similarity is also related to the text data chosen for this task, naturally, but it portrays how difficult it can get to tell the difference between real and fake statements, especially with little knowledge about the respective field or topic at hand.

Although the number of fake and real entries is not the same, considering that approximately 51.7% of the dataset is labelled as fake and 48.3% as real, the dataset is still considered to be balanced. This concern with the dataset's balance should typically be taken into account since a balanced dataset is often able to avoid creating biased models that tend to make better predictions for the majority class. In the context of fake news detection, it is intended to create a model that is able to not only successfully identify fake news, but also avoid constant misclassifications, hence the importance of a balanced dataset.

Since Sentiment Analysis will also be considered in this approach, it is relevant to explore the sentiment distribution of the dataset, which is presented in figure 3.11:

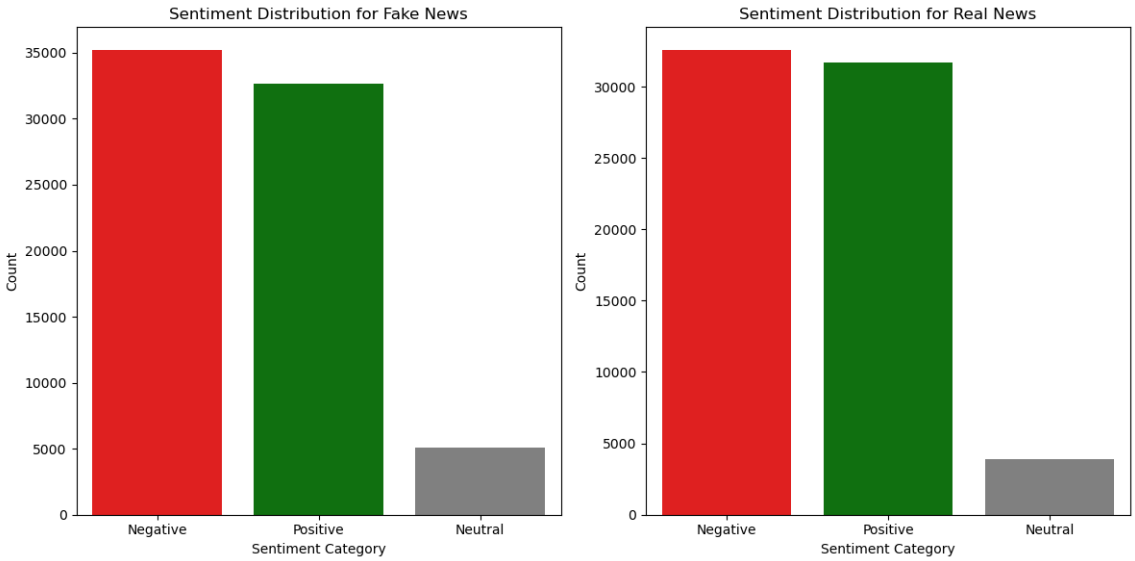


Figure 3.11: Sentiment distribution in the English dataset

Even though there are not any clear differences between fake and real news, the sentiment analysis may still prove useful in the training phase.

With a cleaned and balanced dataset ready to be used, different experiments were conducted to create and improve ML and DL models. Although a wide variety of techniques were combined in different ways, only two main experiments will be discussed given the more significant impact they had on the English approach.

3.2.3 Experiment number 1

This experiment was the first one being conducted and it comprised the best techniques and models identified in the projects and papers explored in the State of Art.

3.2.3.1 Data Pre-processing

The pre-processing phase consisted of the main text processing techniques used in the papers and projects explored in the State of Art: stopword and punctuation removal, changing text to lowercase, stemming and lemmatization. Sentiment Analysis and Part-of-Speech tagging were also used in order to gather possible linguistic and stylistic characteristics that could be used as features in the feature extraction and selection phase.

3.2.3.2 Feature Extraction and Selection

For the Feature Extraction phase, TF-IDF, BoW and Word Embeddings (Word2Vec) were considered, along with Chi-square Statistic and Pearson Correlation for the Feature Selection phase.

3.2.3.3 Machine Learning Models and Results

Many different ML models were trained with the previous techniques. Tables 3.1, 3.2, 3.3 and 3.4 show the results with TF-IDF for feature extraction with different maximum features and the n-gram hyperparameter set to uni-grams, bi-grams and tri-grams, along with the respective time it took to train and test them in the hh:mm:ss.SSS format:

Table 3.1: ML models with TF-IDF and 1000 maximum features

Model	Accuracy	Precision	Recall	F1-Score	Training Time	Evaluation Time
Decision Trees	0.56	0.56	0.56	0.56	00:06:37.820	00:00:00.162
Random Forest	0.51	0.51	0.51	0.51	00:24:46.151	00:00:01.250
Extra Trees	0.4	0.4	0.4	0.4	00:50:16.969	00:00:01.765
XG Boost	0.7	0.7	0.7	0.7	00:01:05.132	00:00:00.024
Adaboost	0.69	0.69	0.69	0.69	00:02:23.067	00:00:01.297
Gradient Boosting	0.71	0.71	0.71	0.71	00:11:30.585	00:00:00.058
Bagging	0.58	0.58	0.57	0.57	00:50:09.311	00:00:00.448
Logistic Regression	0.67	0.67	0.67	0.67	00:00:03.130	00:00:00.001
SVC	0.64	0.64	0.64	0.64	04:17:05.486	00:36:28.438
Linear SVC	0.67	0.67	0.67	0.67	00:00:04.047	00:00:00.007
Multinomial NB	0.6	0.61	0.59	0.58	00:00:00.051	00:00:00.020
Bernoulli NB	0.57	0.57	0.57	0.57	00:00:00.211	00:00:00.080
Gaussian NB	0.57	0.57	0.57	0.57	00:00:01.982	00:00:00.665
KNN (k = 20)	0.47	0.44	0.46	0.42	00:00:00.557	01:25:56.042

Based on the results, only a few models were used in the remaining tests with more features, mainly XGBoost, AdaBoost and Gradient Boosting.

Table 3.2: ML models with TF-IDF and 50000 maximum features

Model	Accuracy	Precision	Recall	F1-Score	Training Time	Evaluation Time
XG Boost	0.75	0.75	0.75	0.75	00:03:07.388	00:00:00.067
Adaboost	0.71	0.72	0.71	0.71	00:07:09.588	00:00:04.301
Gradient Boosting	0.73	0.73	0.73	0.73	00:33:05.698	00:00:00.123
Logistic Regression	0.63	0.63	0.63	0.63	00:00:12.664	00:00:00.030
Linear SVC	0.6	0.6	0.6	0.6	00:00:07.063	00:00:00.015
Multinomial NB	0.53	0.53	0.53	0.52	00:00:00.199	00:00:00.050
Bernoulli NB	0.52	0.52	0.52	0.52	00:00:00.621	00:00:00.151

Table 3.3: ML models with TF-IDF and 100000 maximum features

Model	Accuracy	Precision	Recall	F1-Score	Training Time	Evaluation Time
XG Boost	0.75	0.75	0.75	0.75	00:03:38.198	00:00:00.105
Adaboost	0.71	0.72	0.71	0.71	00:07:58.575	00:00:06.435
Gradient Boosting	0.73	0.73	0.73	0.73	00:36:59.846	00:00:00.178
Logistic Regression	0.62	0.62	0.62	0.62	00:00:16.529	00:00:00.043
Linear SVC	0.57	0.57	0.57	0.57	00:00:09.811	00:00:00.049
Multinomial NB	0.49	0.49	0.49	0.48	00:00:00.342	00:00:00.075
Bernoulli NB	0.51	0.51	0.51	0.5	00:00:00.843	00:00:00.226

Table 3.4: ML models with TF-IDF and 200000 maximum features

Model	Accuracy	Precision	Recall	F1-Score	Training Time	Evaluation Time
XG Boost	0.75	0.75	0.75	0.75	00:03:54.401	00:00:00.124
Adaboost	0.71	0.72	0.71	0.71	00:08:41.488	00:00:05.978
Gradient Boosting	0.73	0.73	0.73	0.73	00:41:43.807	00:00:00.211
Logistic Regression	0.6	0.6	0.6	0.6	00:00:30.423	00:00:00.048
Linear SVC	0.55	0.55	0.55	0.55	00:00:10.359	00:00:00.044
Multinomial NB	0.45	0.44	0.44	0.43	00:00:00.362	00:00:00.086
Bernoulli NB	0.48	0.47	0.47	0.47	00:00:01.012	00:00:00.381

Surprisingly, the remaining feature extraction techniques had similar to worse results, even after trying out different combinations of hyperparameters and combining different approaches, including TF-IDF vectors with Word2Vec and TF-IDF vectors with BoW. The feature selection methods were not able to improve the results either, with a noticeable decrease in the performance of the models when applied to their input data.

Overall, the macro average F1-score slightly above 0.70 achieved by XGBoost, AdaBoost and Gradient Boosting is considered to be good, but this was very unexpected for some of the approaches and models which achieved much better results, as described in the State of Art, such as XGBoost, AdaBoost, Random Forest and Logistic Regression.

Since the dataset that was used has relevant news articles and statements in the context of fake news detection, the reason behind such results does not lie in the data, but rather in the possibility of models struggling with capturing the context and patterns behind the text data merged from the three datasets. This could have happened due to either poorly applied pre-processing techniques or feature extraction methods.

3.2.3.4 Deep Learning Models and Results

Table 3.5 shows the results obtained for a wide variety of deep learning models with TF-IDF and 1000 features:

Table 3.5: DL models with TF-IDF and 1000 maximum features

Model	Accuracy	Precision	Recall	F1-Score	Training Time	Evaluation Time
DNN	0.68	0.68	0.68	0.68	00:01:40	00:00:02
CNN	0.52	0.51	0.5	0.38	00:03:25	00:00:04
LSTM	0.53	0.53	0.51	0.44	00:02:35	00:00:06
Bi-LSTM	0.53	0.52	0.52	0.48	00:10:29	00:00:16
BERT	0.59	0.67	0.6	0.55	01:43:24	00:03:42
RoBERTa	0.64	0.64	0.64	0.64	02:01:12	00:04:01
XLNET	0.64	0.66	0.63	0.62	03:10:37	00:05:53

DL models proved harder to train when compared to ML ones, mainly because of the size of the dataset and the respective higher GPU memory required to train the models with much more TF-IDF features. Without enough features to represent the dataset, not even Bi-LSTM or state-of-the-art models like BERT and RoBERTa were able to grasp the context of sentences and capture better linguistic patterns within them.

One thing was certain: a new approach had to be considered to achieve a better outcome. After an exhaustive experimenting phase with different combinations of pre-processing and feature extraction techniques, a very surprising result was obtained.

3.2.4 Experiment number 2

Although it is being referenced as "Experiment number 2", this experiment was actually one of the very last ones attempted in the English approach. The idea behind it was based on the results of several other previous approaches without any substantial increase in the performance of the models.

Before this experiment, all text data went through the previous pre-processing and feature extraction methods. The importance of said steps and consequent increase in models' performance were claimed and described in so many different projects of the State of Art that it blinded any initial thoughts on how both ML and DL models would behave when submitted to text data almost void of any pre-processing or feature extraction techniques.

As a consequence, a new set of tests was done, this time feeding the models with text data slightly pre-processed and with as little feature extraction as possible. Not much was to be expected, after all, if the models were not able to perform much better with methods that are known for helping them interpret data more easily, why would the original text data increase their performance?

3.2.4.1 Machine Learning Models and Results

Tables 3.6, 3.7 and 3.8 show the performance of the ML models after using TF-IDF to vectorize the text data with different maximum features:

Table 3.6: ML models with TF-IDF and 1000 maximum features

Model	Accuracy	Precision	Recall	F1-Score	Training Time	Evaluation Time
Decision Trees	0.56	0.56	0.56	0.56	00:07:11.852	00:00:00.183
Random Forest	0.51	0.51	0.51	0.51	00:24:44.453	00:00:01.275
Extra Trees	0.39	0.39	0.39	0.39	00:52:27.235	00:00:01.854
XG Boost	0.69	0.69	0.69	0.69	00:01:07.437	00:00:00.034
Adaboost	0.69	0.7	0.7	0.69	00:02:29.815	00:00:01.352
Gradient Boosting	0.71	0.71	0.71	0.71	00:11:31.849	00:00:00.068
Bagging	0.57	0.57	0.56	0.56	00:50:20.248	00:00:00.483
Logistic Regression	0.67	0.67	0.67	0.67	00:00:04.710	00:00:00.002
SVC	0.63	0.63	0.64	0.63	04:11:42.588	00:37:03.235
Linear SVC	0.67	0.67	0.67	0.67	00:00:05.387	00:00:00.016
Multinomial NB	0.58	0.6	0.57	0.55	00:00:00.103	00:00:00.022
Bernoulli NB	0.55	0.55	0.55	0.55	00:00:00.275	00:00:00.074
Gaussian NB	0.56	0.56	0.56	0.56	00:00:02.062	00:00:00.853
KNN (k = 20)	0.48	0.46	0.47	0.43	00:00:00.678	01:22:11.402

Based on the results, only a few models were used once again in the remaining tests with more features, mainly XGBoost, AdaBoost and Gradient Boosting.

Table 3.7: ML models with TF-IDF and 100000 maximum features

Model	Accuracy	Precision	Recall	F1-Score	Training Time	Evaluation Time
XG Boost	0.75	0.75	0.75	0.75	00:03:36.331	00:00:00.105
Adaboost	0.72	0.73	0.72	0.71	00:07:54.803	00:00:05.174
Gradient Boosting	0.73	0.74	0.73	0.73	00:36:16.264	00:00:00.215
Logistic Regression	0.62	0.62	0.62	0.62	00:00:47.348	00:00:00.058
Linear SVC	0.57	0.57	0.57	0.57	00:00:10.293	00:00:00.046
Multinomial NB	0.5	0.5	0.5	0.5	00:00:00.314	00:00:00.088
Bernoulli NB	0.52	0.51	0.51	0.51	00:00:00.932	00:00:00.230

Table 3.8: ML models with TF-IDF and 200000 maximum features

Model	Accuracy	Precision	Recall	F1-Score	Training Time	Evaluation Time
XG Boost	0.75	0.75	0.75	0.75	00:04:08.346	00:00:00.167
Adaboost	0.72	0.72	0.72	0.71	00:09:03.727	00:00:05.792
Gradient Boosting	0.73	0.74	0.74	0.73	00:40:03.938	00:00:00.157
Logistic Regression	0.6	0.6	0.6	0.6	00:00:46.737	00:00:00.032
Linear SVC	0.55	0.55	0.55	0.55	00:00:09.261	00:00:00.032
Multinomial NB	0.46	0.45	0.45	0.45	00:00:00.279	00:00:00.073
Bernoulli NB	0.5	0.49	0.49	0.49	00:00:00.882	00:00:00.250

As expected, the ML models were not able to achieve better results when less pre-processing and feature extraction techniques were applied.

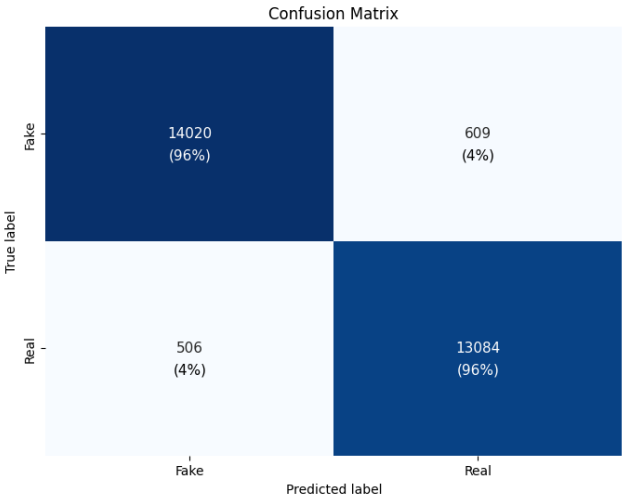
3.2.4.2 Deep Learning Models and Results

To simplify the pre-processing and feature extraction techniques applied to DL models, Keras models only resorted to the Keras' Tokenizer to tokenize text data, while Transformers models resorted to their own pre-trained tokenizers. The performance of each model is presented in table 3.9:

Table 3.9: DL models with only tokenized text data

Model	Accuracy	Precision	Recall	F1-Score	Training Time	Evaluation Time
DNN	0.55	0.59	0.54	0.48	00:00:43	00:00:02
CNN	0.58	0.69	0.57	0.49	00:00:56	00:00:02
LSTM	0.63	0.7	0.62	0.58	00:11:48	00:00:19
Bi-LSTM	0.66	0.68	0.65	0.64	00:33:30	00:00:40
BERT	0.89	0.89	0.89	0.89	02:03:03	00:03:36
RoBERTa	0.77	0.77	0.77	0.77	03:09:27	00:03:33
XLNET	0.83	0.83	0.83	0.83	03:24:56	00:05:52

Out of all the models used, one clearly stands out from the rest: the BERT model, which was able to achieve a macro average F1-score of 0.89. This was a very surprising result given that the text data did not undergo a very detailed pre-processing phase. Training the model with its layers unfrozen in Google Colab made it possible to increase the performance even more, up to 0.96 macro average F1-score. The respective confusion matrix and model summary are presented in figure 3.12:



(a) Confusion Matrix

Layer (type)	Output Shape	Param #
input_ids (InputLayer)	[(None, 128)]	0
tf_bert_model (TFBertModel)	TFBaseModelOutputWithPoolingAndCrossAttentions(last_hidden_state=(None, 128, 768), pooler_output=(None, 768), past_key_values=None, hidden_states=None, attentions=None, cross_attentions=None)	109482240
flatten (Flatten)	(None, 98304)	0
dense (Dense)	(None, 128)	12583040
dropout_37 (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 1)	129

Total params: 122,065,409
Trainable params: 122,065,409
Non-trainable params: 0

(b) Model summary

Figure 3.12: Confusion matrix and summary of the best model in the English approach

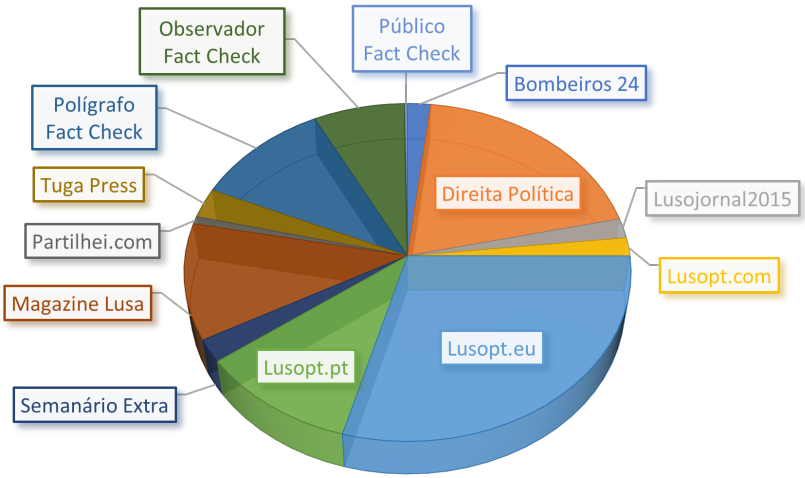
In the end, BERT still achieved remarkable results, proving once again its ability and potential in text classification tasks as a state-of-the-art model.

3.3 European Portuguese Approach

This section will explore the different steps behind the European Portuguese approach, starting off with the Dataset created through Web Scraping.

3.3.1 Web Scraping and Dataset Creation

The many different websites that were considered when creating the dataset for fake news detection in European Portuguese are presented in the charts of figures 3.13 and 3.14, along with the number of articles and statements scraped:

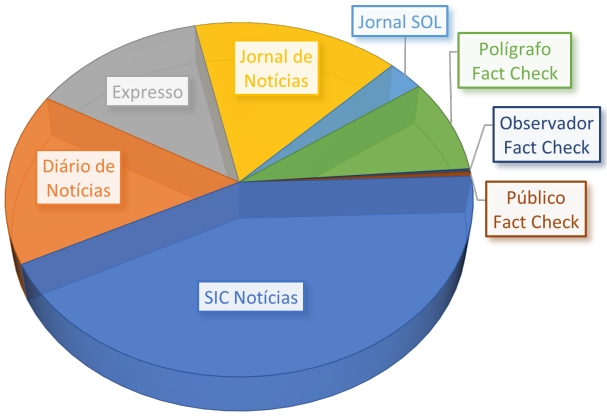


(a) Fake news distribution chart

Fake News	
Website	Count
Bombeiros 24	602
Direita Política	6425
Lusojornal2015	623
Lusopt.com	574
Lusopt.eu	9620
Lusopt.pt	3486
Semanário Extra	710
Magazine Lusa	3652
Partilhei.com	238
Tuga Press	927
Polígrafo Fact Check	3559
Observador Fact Check	2385
Público Fact Check	80

(b) Fake news websites

Figure 3.13: Portuguese fake news distribution chart and table



(a) Real news distribution chart

Real News	
Website	Count
SIC Notícias	14718
Diário de Notícias	5369
Expresso	4537
Jornal de Notícias	5369
Jornal SOL	950
Polígrafo Fact Check	2925
Observador Fact Check	90
Público Fact Check	143

(b) Real news websites

Figure 3.14: Portuguese real news distribution chart and table

The Beautiful Soup package allowed the extraction of news data from each website by sending several requests to specify the HTML tags with the desired information. Selenium was also used given that some websites required user interaction to load and display content. This included clicking buttons, scrolling up and down and closing tabs.

Regarding the resulting dataset, 4 columns were created to store the text, label, source and URL of each web-scraped article. As a way to extract real news, the web scraping process only took into account trustworthy websites widely known and used by many people, while websites that had been identified for spreading misinformation were used to extract fake news. Certain fake news websites are not accessible anymore, but it was still possible to gather some data by resorting to "Arquivo.pt" which had stored some data from past versions of these websites [103], [104].

The trustworthy websites also included fact-checks from reputable sources such as Polígrafo, Público and Observador. Unlike in [30] which resorted to the fact-checks overview in the main webpage to gather data, the data from fact-checks in this project was extracted from the actual fact-checks instead of their overviews. It was observed that the vast majority of fact-checks have an introductory section with 4 paragraphs and one image dedicated exclusively to the context of the fact-check, which often also includes the original statement that is fact-checked, as well as the source behind it.

These introductory sections obviously have much more information than their respective overviews in the main webpage, which can help improve the training phase and consequent predictions of the ML and DL models. The remaining paragraphs of the article explained the facts and reasoning behind the final rating of the fact-check, which were not relevant to the task at hand, thus being ignored. However, it would make sense to use the remaining paragraphs if it was explained why the fake news articles and posts from fake news websites were labelled as false.

Figure 3.15 shows which fact-checks' ratings were converted to "Real" and "Fake" when stored in the dataset for binary classification:

Polígrafo Rating	Label
Verdadeiro	Real
Verdadeiro, mas...	Real
Descontextualizado	Fake
Impreciso	Fake
Manipulado	Fake
Falso	Fake
Pimenta na Língua	Fake

(a) Poligrafo's conversion

Observador Rating	Label
Certo	Real
Praticamente certo	Real
Esticado	Fake
Inconclusivo	Fake
Enganador	Fake
Errado	Fake

(b) Observador's conversion

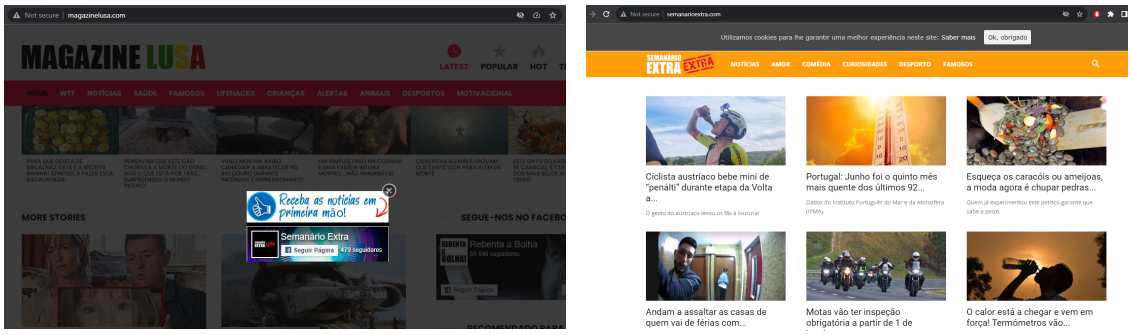
Público Rating	Label
Verdadeiro	Real
Parcialmente verdadeiro	Real
Parcialmente falso	Fake
Falso	Fake
Inconclusivo	Fake

(c) Público's conversion

Figure 3.15: Fact-checking websites' categories conversion

Some of the fake news websites have a very similar website structure to each other and they do not really seem to bother to hide it. In fact, there are even fake news websites

that also promote news from other websites known for spreading misinformation, which confirms the close bonds shared between them! Figure 3.16 shows an example of this self-promotion, where the fake news website "Magazine Lusa" promotes the Facebook page of another fake news website called "Semanário Extra":



(a) Magazine Lusa promoting Semanário Extra (b) Semanário Extra

Figure 3.16: Portuguese fake news websites promoting each other

Another aspect that fake news websites have in common is the constant spam of certain advertisements that are not found on credible news websites. Many tabs and pop-ups suddenly open when users left-click or scroll up and down and most of these are used to advertise gambling websites, although there are also advertisements with pornographic content as well. "Direita Política" was one of the websites that advertised the latter type of content and, curiously, stopped being accessible a few weeks later after being scraped.

Regarding the topics and fields discussed by the extracted articles, these were often divided into different categories in each website such as politics, science, sports, fame, justice, society and more. There were also both real and fake news articles about COVID-19 and the on-going war between Russia and Ukraine. Some fake news websites also had articles tagged as jokes or satire, which were ignored during the web scraping step.

It is also important to mention that, although the fake news websites had, obviously, fake news, many of these websites had news from credible sources as well, including ones that were used to extract real news data. This was probably done in an attempt to fool readers into thinking that each of these websites were credible sources, which made their misinformative nature not so obvious and easily spotted. To avoid possible incorrect labels, all news from misinformation websites with credible sources were ignored, while those with an untrustworthy source were labelled as fake.

On the other hand, there were also certain news articles with trustworthy sources mentioned within the text/body itself. As a consequence, articles without any source at the end were also labelled as fake and had their "source" column set to None, in order to avoid confusing certain words with reputable sources' names. For example, "Público" as in both public and the news agency or "Sábado" as in both Saturday and the news agency.

The dataset and web scrapers are available here. After checking and removing repeated entries, the dataset was cleaned and ready for Exploratory Data Analysis.

3.3.2 Exploratory Data Analysis

Figure 3.17 shows the balance of real news and fake news in the dataset:

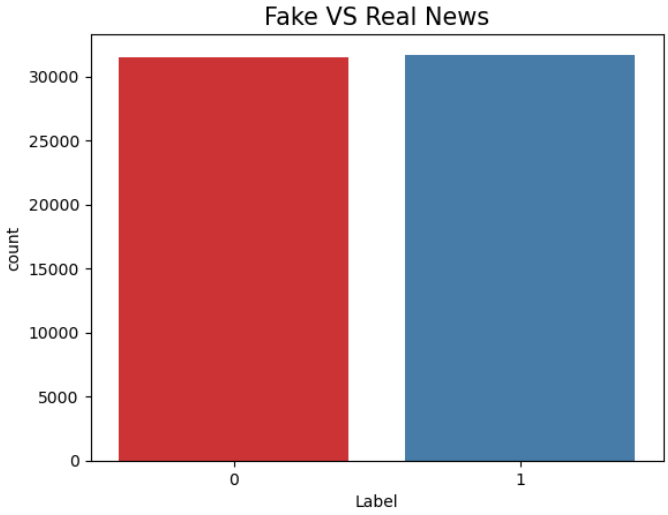


Figure 3.17: Balance of fake (red) and real (blue) news in European Portuguese

The Portuguese dataset has over 62000 articles, with a total of 31520 fake and 31716 real news and statements, hence being balanced. The word cloud in figure 3.18 shows the 500 most frequent words in the entire dataset, including "anos", "euros", "sobre", "ainda", "Portugal" and more:

Figure 3.18: The 500 most common words in the Portuguese dataset

The word cloud in figure 3.19 shows the 500 most frequent words in fake news, including "anos", "euros", "sobre", "ainda", "Portugal" and more:

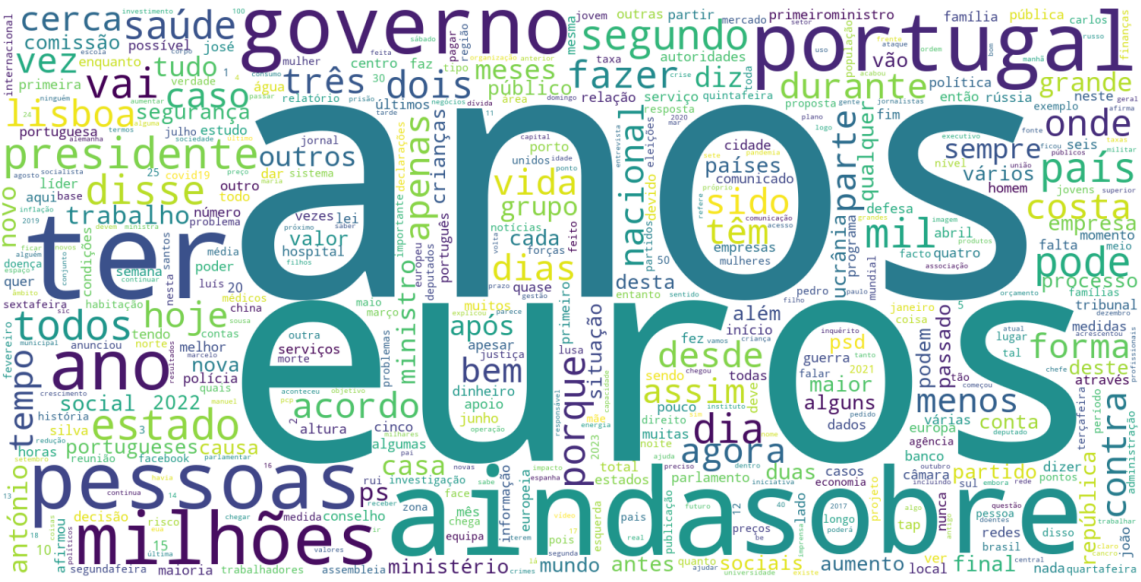


Figure 3.19: The 500 most common words in Portuguese fake news

The word cloud in figure 3.20 shows the 500 most frequent words in real news, including "anos", "euros", "sobre", "ainda", "Portugal" and more:

Figure 3.20: The 500 most common words in Portuguese real news

The word cloud in figure 3.21 shows the 300 most common bi-grams in fake news, including "milhões euros", "redes sociais", "António Costa" and more:

Figure 3.21: The 300 most common bi-grams in Portuguese fake news

The word cloud in figure 3.22 shows the 300 most common bi-grams in fake news, including "milhões euros", "António Costa", "Estados Unidos" and more:

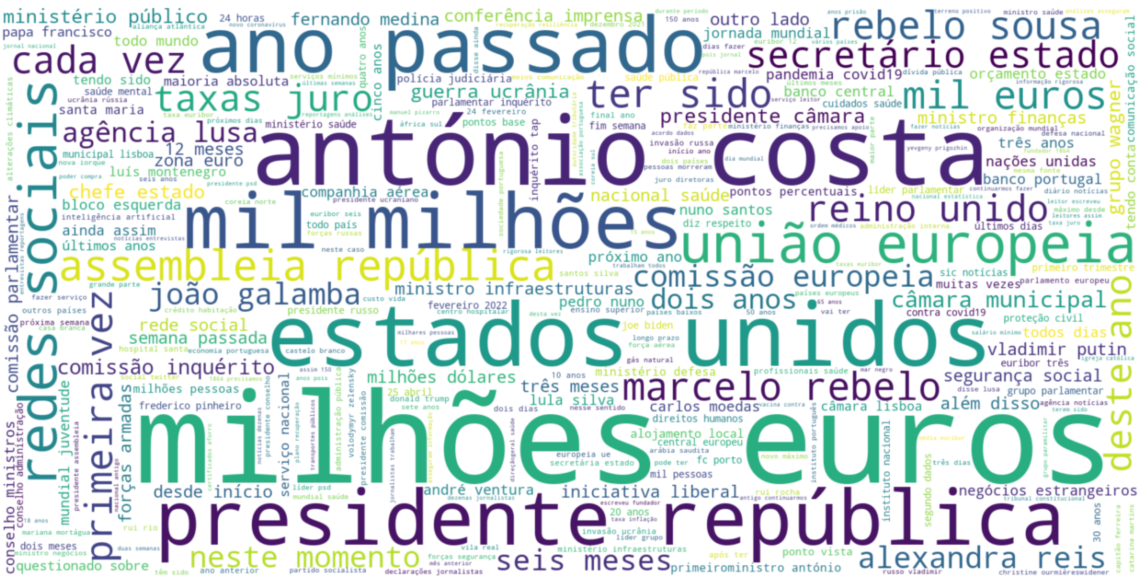


Figure 3.22: The 300 most common bi-grams in Portuguese real news

Unlike in the English approach, this time the most common words in fake news articles were a bit different from the ones in real news articles. There are also bi-grams with more occurrences in fake news than real news and vice-versa, such as "José Sócrates", "Cristiano Ronaldo", "redes sociais" which translates to "social networks" and more. Even though it makes sense for fake news articles to share a different set of words from real news articles, this aspect could also be related to the data that was available to scrape.

As opposed to fake news websites, the real news websites that were used probably had a lot of articles removed or hidden because the oldest date of publication of most real news articles was in 2021, around the time when the COVID-19 pandemic started to raise a lot of awareness. When it comes to fake news, the dates of the articles range from 2014 to 2023, which could raise difficulties for ML and DL models during training given the time gap between real and fake news and the events that took place during those years.

Since Sentiment Analysis will be considered once again in this approach, it is relevant to explore the sentiment distribution of the dataset, which is presented in figure 3.23:

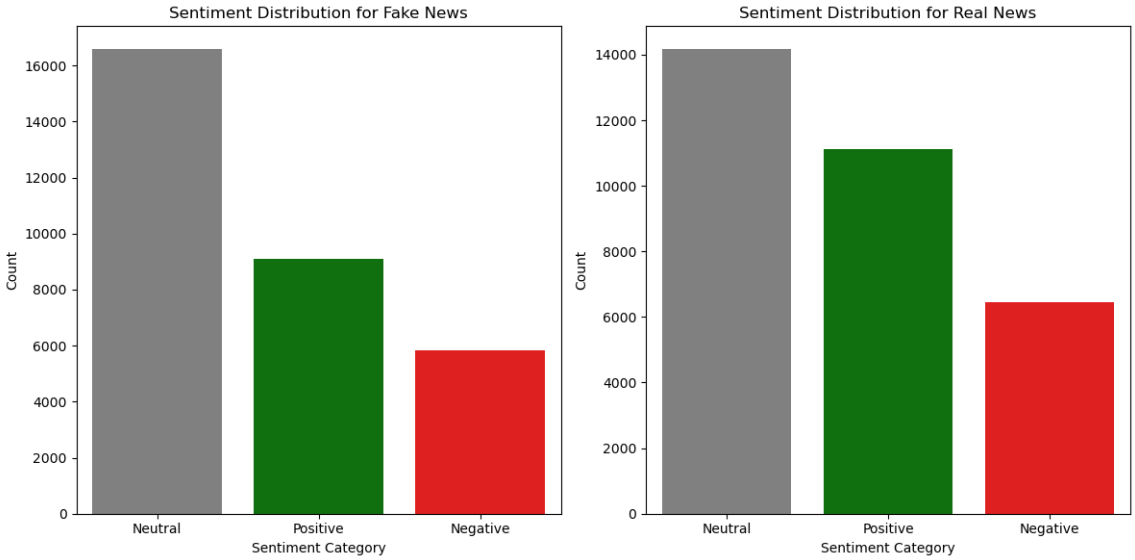


Figure 3.23: Sentiment distribution in the Portuguese dataset

The previous plots show clear differences regarding the sentiment distribution between fake and real news. There are more real news articles and statements that portray a stronger positive sentiment than fake news, which seem to have a more neutral stance. These differences could prove useful as additional features during the models' training phase.

Once the data was cleaned and analysed, four main experiments were conducted. Taking into account what had happened in the English Approach, the European Portuguese approach started with simple techniques and gradually increased in complexity.

3.3.3 Experiment number 1

3.3.3.1 Data Pre-processing

In the first experiment, no pre-processing techniques were applied, which means the text data was used directly as input for ML models, whereas DL models received a tokenized version of said text data.

3.3.3.2 Feature Extraction and Selection

Once again, TF-IDF, BoW and Word Embeddings (Word2Vec) were considered for Feature Extraction, along with Chi-square Statistic, Pearson Correlation and Recursive Feature Extraction for the Feature Selection phase.

3.3.3.3 Machine Learning Models and Results

Tables 3.10, 3.11 and 3.12 show the ML models that were trained and with the feature extraction techniques mentioned previously:

Table 3.10: ML models with original text and TF-IDF

Model	Accuracy	Precision	Recall	F1-Score	Training Time	Evaluation Time
Decision Trees	0.84	0.84	0.84	0.84	00:50:21.325	00:00:00.752
Random Forest	0.89	0.89	0.89	0.89	02:03:38.129	00:00:20.254
Extra Trees	0.9	0.9	0.9	0.9	06:00:17.518	00:00:17.818
XG Boost	0.91	0.91	0.91	0.91	00:02:13.073	00:00:01.864
Adaboost	0.83	0.83	0.83	0.83	08:22:49.235	00:00:08.427
Gradient Boosting	0.84	0.85	0.84	0.84	16:57:32.781	00:00:00.727
Bagging	0.88	0.88	0.88	0.88	05:04:02.616	00:00:10.241
Logistic Regression	0.91	0.91	0.91	0.91	00:02:35.933	00:00:00.091
SVC	0.92	0.92	0.92	0.92	03:03:08.392	00:14:16.519
Linear SVC	0.93	0.93	0.93	0.93	00:00:10.476	00:00:00.085
Multinomial NB	0.88	0.89	0.88	0.88	00:00:01.188	00:00:00.234
Bernoulli NB	0.86	0.88	0.87	0.86	00:00:01.475	00:00:00.741
KNN (k = 20)	0.8	0.8	0.8	0.8	00:00:00.546	07:10:45.434

Table 3.11: ML models with original text, TF-IDF and WE

Model	Accuracy	Precision	Recall	F1-Score	Training Time	Evaluation Time
Decision Trees	0.84	0.84	0.84	0.84	00:00:07.462	00:00:00.024
Random Forest	0.89	0.89	0.89	0.89	00:00:51.884	00:00:00.251
Extra Trees	0.9	0.9	0.9	0.9	00:00:11.165	00:00:00.316
XG Boost	0.91	0.91	0.91	0.91	00:00:11.603	00:00:00.043
Adaboost	0.83	0.83	0.83	0.83	00:00:37.446	00:00:00.263
Gradient Boosting	0.84	0.85	0.84	0.84	00:03:07.552	00:00:00.050
Bagging	0.83	0.83	0.83	0.83	00:00:57.153	00:00:00.145
Logistic Regression	0.79	0.79	0.79	0.79	00:00:00.919	00:00:00.020
SVC	0.83	0.83	0.83	0.83	00:04:29.145	00:00:35.546
Linear SVC	0.79	0.79	0.79	0.79	00:00:28.112	00:00:00.022
Bernoulli NB	0.7	0.7	0.7	0.7	00:00:00.147	00:00:00.039
KNN (k = 20)	0.82	0.82	0.82	0.82	00:00:00.061	00:00:01.574

Table 3.12: ML models with original text, TF-IDF and BoW

Model	Accuracy	Precision	Recall	F1-Score	Training Time	Evaluation Time
Decision Trees	0.84	0.84	0.84	0.84	00:02:12.670	00:00:00.099
Random Forest	0.89	0.89	0.89	0.89	00:05:11.324	00:00:00.889
Extra Trees	0.9	0.9	0.9	0.9	00:08:29.214	00:00:01.087
XG Boost	0.91	0.91	0.91	0.91	00:00:26.560	00:00:00.047
Adaboost	0.83	0.83	0.83	0.83	00:04:56.991	00:00:00.928
Gradient Boosting	0.84	0.85	0.84	0.84	00:12:45.976	00:00:00.027
Bagging	0.88	0.88	0.88	0.88	00:18:37.571	00:00:00.355
Logistic Regression	0.92	0.92	0.92	0.92	00:00:10.564	00:00:00.014
SVC	0.93	0.93	0.93	0.93	00:47:48.239	00:03:57.599
Linear SVC	0.92	0.92	0.92	0.92	00:00:06.441	00:00:00.003
Multinomial NB	0.87	0.87	0.87	0.87	00:00:00.039	00:00:00.024
Bernoulli NB	0.84	0.85	0.84	0.84	00:00:00.127	00:00:00.039
KNN (k = 20)	0.73	0.73	0.73	0.73	00:00:00.043	00:25:20.145

The differences explored between fake and real news in the "Exploratory Data Analysis" subsection turned out to be a good sign for text classification since ML models were able to achieve outstanding results with just the original text. As observed in the English approach, none of the feature selection techniques seemed to improve the performance of the models once again.

3.3.3.4 Deep Learning Models and Results

Table 3.13 shows the DL models that were trained, as well as their respective results:

Table 3.13: DL models with original text

Model	Accuracy	Precision	Recall	F1-Score	Training Time	Evaluation Time
DNN	0.89	0.89	0.89	0.89	00:01:21	00:00:01
CNN	0.92	0.92	0.92	0.92	00:02:20	00:00:13
LSTM	0.92	0.92	0.92	0.92	00:07:12	00:00:22
Bi-LSTM	0.92	0.92	0.92	0.92	00:17:48	00:00:40
BERT	0.86	0.86	0.86	0.86	00:50:12	00:01:38
RoBERTa	0.5	0.5	0.5	0.5	00:36:16	00:04:01
XLNET	0.86	0.86	0.86	0.86	04:25:53	00:04:10

The DL models achieved great results as well. The models that had performed worse in the English approach performed much better this time, including DNN, CNN, LSTM and Bi-LSTM. The state-of-the-art models BERT, RoBERTa and XLNET did not perform as well as the others, especially RoBERTa, which may have struggled when capturing the context and patterns of the text.

The BERT model was also trained with its layers unfrozen but the macro average F1-score did not increase beyond 0.92 either.

3.3.4 Experiment number 2

3.3.4.1 Machine Learning Models and Results

Tables 3.14, 3.15 and 3.16 show the results of the ML models with pre-processed text:

Table 3.14: ML models with pre-processed text and TF-IDF

Model	Accuracy	Precision	Recall	F1-Score	Training Time	Evaluation Time
Decision Trees	0.86	0.86	0.86	0.86	00:44:28.095	00:00:00.431
Random Forest	0.89	0.89	0.89	0.89	01:56:19.525	00:00:15.647
Extra Trees	0.9	0.9	0.9	0.9	05:00:16.169	00:00:13.818
XG Boost	0.9	0.9	0.9	0.9	00:02:09.034	00:00:02.519
Adaboost	0.83	0.83	0.83	0.83	07:52:30.936	00:00:07.866
Gradient Boosting	0.84	0.85	0.84	0.84	16:40:19.789	00:00:00.257
Bagging	0.89	0.89	0.89	0.89	04:47:50.509	00:00:04.896
Logistic Regression	0.91	0.91	0.91	0.91	00:01:38.100	00:00:00.064
SVC	0.92	0.92	0.92	0.92	02:07:10.335	00:09:47.267
Linear SVC	0.93	0.93	0.93	0.93	00:00:07.085	00:00:00.042
Multinomial NB	0.89	0.9	0.89	0.89	00:00:00.875	00:00:00.167
Bernoulli NB	0.85	0.87	0.85	0.84	00:00:01.223	00:00:00.702
KNN (k = 20)	0.83	0.83	0.83	0.83	00:00:00.280	03:20:40.272

Table 3.15: ML models with pre-processed text, TF-IDF and WE

Model	Accuracy	Precision	Recall	F1-Score	Training Time	Evaluation Time
Decision Trees	0.82	0.82	0.82	0.82	00:00:08.618	00:00:00.018
Random Forest	0.89	0.89	0.89	0.89	00:00:54.311	00:00:00.223
Extra Trees	0.89	0.89	0.89	0.89	00:00:09.325	00:00:00.280
XG Boost	0.89	0.89	0.89	0.89	00:00:10.105	00:00:00.039
Adaboost	0.81	0.81	0.81	0.81	00:00:35.591	00:00:00.316
Gradient Boosting	0.85	0.85	0.85	0.85	00:03:11.035	00:00:00.056
Bagging	0.86	0.86	0.86	0.86	00:01:01.202	00:00:00.155
Logistic Regression	0.81	0.81	0.81	0.81	00:00:00.702	00:00:00.021
SVC	0.88	0.88	0.88	0.88	00:03:59.346	00:00:26.795
Linear SVC	0.81	0.81	0.81	0.81	00:00:22.397	00:00:00.020
Bernoulli NB	0.77	0.77	0.77	0.77	00:00:00.175	00:00:00.047
KNN (k = 20)	0.86	0.86	0.86	0.86	00:00:00.069	00:00:01.333

Table 3.16: ML models with pre-processed text, TF-IDF and BoW

Model	Accuracy	Precision	Recall	F1-Score	Training Time	Evaluation Time
Decision Trees	0.84	0.84	0.84	0.84	00:02:19.392	00:00:00.075
Random Forest	0.9	0.9	0.9	0.9	00:05:02.901	00:00:00.805
Extra Trees	0.9	0.91	0.9	0.9	00:08:37.436	00:00:00.937
XG Boost	0.9	0.91	0.9	0.9	00:00:22.998	00:00:00.037
Adaboost	0.83	0.83	0.83	0.83	00:04:11.358	00:00:00.646
Gradient Boosting	0.84	0.84	0.84	0.84	00:10:45.364	00:00:00.027
Bagging	0.88	0.88	0.88	0.88	00:16:53.116	00:00:00.254
Logistic Regression	0.92	0.92	0.92	0.92	00:00:04.964	00:00:00.007
SVC	0.93	0.93	0.93	0.93	00:42:49.811	00:03:16.770
Linear SVC	0.91	0.91	0.91	0.91	00:00:05.028	00:00:00.005
Multinomial NB	0.87	0.87	0.87	0.87	00:00:00.041	00:00:00.010
Bernoulli NB	0.85	0.85	0.85	0.85	00:00:00.105	00:00:00.035
KNN (k = 20)	0.76	0.76	0.76	0.76	00:00:00.034	00:15:28.868

The stopword removal and lemmatization techniques behind the pre-processing phase resulted in different outcomes for ML models, as some had slightly increased macro average F1-score results while others had their performance slightly decreased when compared to the first experiment.

3.3.4.2 Deep Learning Models and Results

Table 3.17 shows the performance of the DL models:

Table 3.17: DL models with pre-processed text

Model	Accuracy	Precision	Recall	F1-Score	Training Time	Evaluation Time
DNN	0.92	0.92	0.92	0.92	00:01:01	00:00:01
CNN	0.92	0.92	0.92	0.92	00:02:12	00:00:12
LSTM	0.92	0.92	0.92	0.92	00:08:06	00:00:18
Bi-LSTM	0.92	0.92	0.92	0.92	00:15:36	00:00:44
BERT	0.5	0.5	0.5	0.5	00:27:39	00:01:34
RoBERTa	0.5	0.5	0.5	0.5	00:36:16	00:04:01
XLNET	0.87	0.87	0.87	0.87	04:16:30	00:02:17

The pre-processed text had a general negative impact, as they did not improve the overall results of the DL models when compared to the previous experiments.

3.3.5 Experiment number 3

In this experiment, POS tagging and Sentiment Analysis were once again added to gather the number of words, nouns, verbs, adverbs and adjectives in each text and the average sentiment portrayed (negative, neutral or positive). The source of each text was also considered.

3.3.5.1 Machine Learning Models and Results

Tables 3.18, 3.19 and 3.20 show the results of the ML models:

Table 3.18: ML models with pre-processed text, news source, POS tagging, Sentiment Analysis and TF-IDF

Model	Accuracy	Precision	Recall	F1-Score	Training Time	Evaluation Time
Decision Trees	0.94	0.94	0.94	0.94	00:25:26.930	00:00:00.472
Random Forest	0.92	0.92	0.92	0.92	01:34:41.779	00:00:13.617
Extra Trees	0.92	0.92	0.92	0.92	03:34:42.819	00:00:14.334
XG Boost	0.95	0.96	0.95	0.95	00:02:14.068	00:00:05.058
Adaboost	0.94	0.94	0.94	0.94	08:15:34.912	00:00:08.373
Gradient Boosting	0.95	0.95	0.95	0.95	17:22:59.443	00:00:00.554
Bagging	0.95	0.95	0.95	0.95	02:52:59.832	00:00:05.972
Logistic Regression	0.93	0.93	0.93	0.93	00:02:55.621	00:00:00.143
SVC	0.94	0.94	0.94	0.94	02:23:57.998	00:10:40.638
Linear SVC	0.95	0.95	0.95	0.95	00:00:08.245	00:00:00.052
Multinomial NB	0.91	0.91	0.91	0.91	00:00:00.929	00:00:00.194
Bernoulli NB	0.87	0.89	0.87	0.87	00:00:01.219	00:00:00.674
KNN (k = 20)	0.85	0.85	0.85	0.85	00:00:01.115	03:54:53.794

Table 3.19: ML models with pre-processed text, news source, POS tagging, Sentiment Analysis, TF-IDF and WE

Model	Accuracy	Precision	Recall	F1-Score	Training Time	Evaluation Time
Decision Trees	0.83	0.83	0.83	0.83	00:00:07.097	00:00:00.033
Random Forest	0.89	0.89	0.89	0.89	00:00:52.830	00:00:00.193
Extra Trees	0.89	0.89	0.89	0.89	00:00:09.255	00:00:00.260
XG Boost	0.89	0.89	0.89	0.89	00:00:10.901	00:00:00.043
Adaboost	0.81	0.81	0.81	0.81	00:00:34.666	00:00:00.241
Gradient Boosting	0.85	0.85	0.85	0.85	00:02:54.488	00:00:00.046
Bagging	0.87	0.87	0.87	0.87	00:00:58.596	00:00:00.139
Logistic Regression	0.82	0.82	0.82	0.82	00:00:00.629	00:00:00.018
SVC	0.88	0.88	0.88	0.88	00:02:54.201	00:00:23.063
Linear SVC	0.81	0.81	0.81	0.81	00:00:19.720	00:00:00.016
Bernoulli NB	0.77	0.77	0.77	0.77	00:00:00.137	00:00:00.044
KNN (k = 20)	0.87	0.87	0.87	0.87	00:00:00.060	00:00:01.104

Table 3.20: ML models with pre-processed text, news source, POS tagging, Sentiment Analysis, TF-IDF and BoW

Model	Accuracy	Precision	Recall	F1-Score	Training Time	Evaluation Time
Decision Trees	0.94	0.94	0.94	0.94	00:01:16.085	00:00:00.078
Random Forest	0.93	0.93	0.93	0.93	00:04:18.991	00:00:00.771
Extra Trees	0.93	0.93	0.92	0.93	00:07:56.180	00:00:00.901
XG Boost	0.95	0.95	0.95	0.95	00:00:21.661	00:00:00.027
Adaboost	0.94	0.94	0.94	0.94	00:04:15.216	00:00:00.642
Gradient Boosting	0.95	0.95	0.95	0.95	00:10:57.032	00:00:00.026
Bagging	0.95	0.95	0.95	0.95	00:08:54.955	00:00:00.289
Logistic Regression	0.95	0.95	0.95	0.95	00:00:04.502	00:00:00.005
SVC	0.95	0.95	0.95	0.95	00:43:10.131	00:03:00.345
Linear SVC	0.94	0.94	0.94	0.94	00:00:03.692	00:00:00.005
Multinomial NB	0.89	0.89	0.89	0.89	00:00:00.044	00:00:00.011
Bernoulli NB	0.87	0.87	0.87	0.87	00:00:00.108	00:00:00.037
KNN (k = 20)	0.82	0.82	0.82	0.82	00:00:00.030	00:17:03.724

The characteristics extracted from POS tagging and Sentiment Analysis improved the performance of the models even further, alongside the source of each text. This proves the impact that sentiment analysis can have in fake news detection when dealing with data that has different sentiment scores, which is the case taking into account the sentiment distribution previously explored.

Moreover, POS tagging also contributed to better results given the fact that many fake news websites often have much shorter news than the ones found in real news websites. The words and n-grams and respective occurrences obtained and presented in the previous word clouds are also behind the improvement of the models since there are clear differences between fake and real news.

Finally, the source of each news also helped improve the results since it might be more accessible for models to establish similar patterns shared between the news from fake news websites, while also differentiating them from real ones.

3.3.5.2 Deep Learning Models and Results

Table 3.21 shows the performance of the DL models:

Table 3.21: DL models with pre-processed text, news source, POS tagging and Sentiment Analysis

Model	Accuracy	Precision	Recall	F1-Score	Training Time	Evaluation Time
DNN	0.73	0.73	0.73	0.73	00:00:40	00:00:01
CNN	0.78	0.78	0.78	0.78	00:07:12	00:00:12
LSTM	0.69	0.7	0.69	0.69	00:03:48	00:00:02
Bi-LSTM	0.7	0.7	0.7	0.7	00:12:22	00:00:04
BERT	0.87	0.87	0.87	0.87	03:24:24	00:01:45
RoBERTa	0.87	0.87	0.87	0.87	03:22:59	00:01:45
XLNET	0.87	0.87	0.87	0.87	03:51:46	00:02:20

Although some of the DL models had their performance slightly increased, they were not able to take advantage of all the additional features, at least not as much as the ML models did, which could be explained by a lack of better-suited layers and respective hyperparameters, along with overfitting. Training the state-of-the-art models with their layers unfrozen did not improve their performance either.

3.3.6 Experiment number 4

In an attempt to increase the best models' results even more, a fourth and last experiment was conducted. Given that ML models had better results than DL models, a grid-search approach with 5-fold cross-validation was considered with different hyperparameters, in order to tune the best ML models of the third experiment.

Furthermore, the pre-trained models of the previous experiments were pre-trained with English data, so it would be interesting to check the performance of pre-trained models on Portuguese data with both text data and features as input.

Tables 3.22 and 3.23 show the results of the grid-searches, while tables 3.24 and 3.25 show the results obtained with the model BERTimbau pre-trained in Brazilian Portuguese, along with Multilingual BERT (MBERT) and Cross-lingual Language Model-RoBERTa (XLM-R) pre-trained in multiple languages:

Table 3.22: Grid-search of the best ML models with TF-IDF and BoW

Model	Accuracy	Precision	Recall	F1-Score	Training Time	K-Fold Cross-Validation F1-Scores (k=5)
Decision Trees	0.95	0.95	0.95	0.947	23:27:19	[0.945835, 0.948491, 0.947017, 0.948191, 0.946915]
XG Boost	0.96	0.96	0.96	0.956	06:15:54	[0.956112, 0.955121, 0.957399, 0.955510, 0.956107]
Adaboost	0.95	0.95	0.95	0.947	04:06:39	[0.947415, 0.946028, 0.948010, 0.947407, 0.946611]
Logistic Regression	0.95	0.95	0.95	0.946	01:26:46	[0.948310, 0.944257, 0.944950, 0.947118, 0.943065]
Linear SVC	0.95	0.95	0.95	0.946	00:18:43	[0.948112, 0.946727, 0.945839, 0.946227, 0.944745]

Table 3.23: Grid-search of the best ML models with TF-IDF

Model	Accuracy	Precision	Recall	F1-Score	Training Time	K-Fold Cross-Validation F1-Scores (k=5)
Decision Trees	0.95	0.95	0.95	0.948	15:27:39	[0.951467, 0.950676, 0.948603, 0.945729, 0.945828]
XG Boost	0.96	0.96	0.96	0.957	39:39:18	[0.957099, 0.957689, 0.955917, 0.956502, 0.955908]
Logistic Regression	0.95	0.95	0.95	0.952	08:40:41	[0.953844, 0.953150, 0.951966, 0.950676, 0.951664]
Linear SVC	0.95	0.95	0.95	0.947	07:52:29	[0.949199, 0.948109, 0.947913, 0.942372, 0.946326]

Grid-searches proved useful as most ML models had their performance slightly improved. The results were very close, but the TF-IDF approach achieved the best ones.

Table 3.24: Performance of different pre-trained models on Portuguese data (text only)

Model	Accuracy	Precision	Recall	F1-Score	Training Time	Evaluation Time
BERTimbau	0.95	0.95	0.95	0.947	01:58:38	00:01:51
BERTm	0.88	0.88	0.88	0.88	00:28:38	00:01:39
XLM-R	0.9	0.9	0.9	0.9	01:34:48	00:01:47

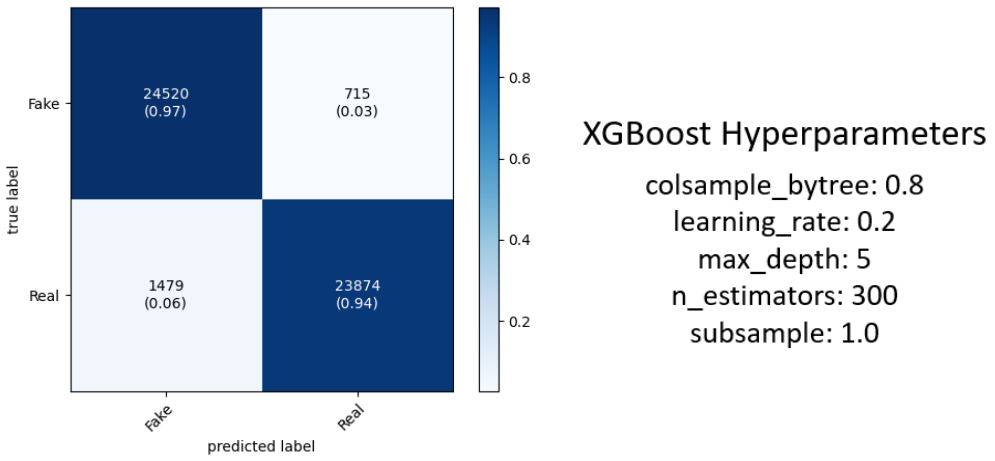
Table 3.25: Performance of different pre-trained models on Portuguese data (features)

Model	Accuracy	Precision	Recall	F1-Score	Training Time	Evaluation Time
BERTimbau	0.95	0.95	0.95	0.943	01:15:23	00:01:56
BERTm	0.94	0.94	0.94	0.94	03:07:14	00:01:59
XLM-R	0.94	0.94	0.94	0.94	02:22:38	00:01:57

The pre-trained models on Portuguese data showed better results than the ones in previous experiments. Unfreezing the pre-trained models' layers with only text as input also increased the results up to 0.947 F1-score, but ML models still took the lead.

Despite the results being very close, XGBoost with TF-IDF had the best performance with an F1-score of 0.957, followed by XGBoost with TF-IDF and BoW with 0.956 and Logistic Regression with TF-IDF and BoW with 0.952.

Figure 3.24 shows the confusion matrix and hyperparameters of the best classifier:



(a) Confusion matrix of XGBoost with TF-IDF

(b) XGBoost Model hyperparameters

Figure 3.24: Confusion Matrix and hyperparameters of XGBoost with TF-IDF

3.4 Applications Development and Deployment

With two models able to detect fake news in English and European Portuguese, it was relevant to establish a method for their practical implementation. As a result, different processes were considered, which are portrayed in figure 3.25:

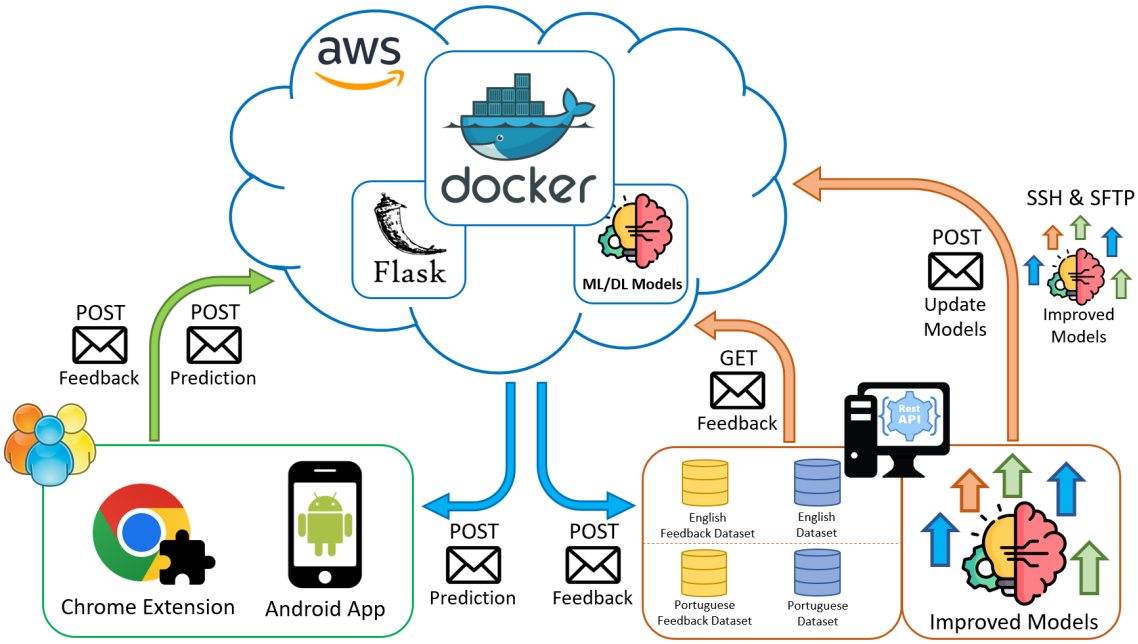


Figure 3.25: Processes behind the application development and deployment phases

This section will explore the steps involved in said processes, starting off with the Flask application.

3.4.1 Flask Application

A Flask application was developed in order to receive requests from users and send the desired data respectively. Once the models are loaded, the Flask app awaits for either a prediction or feedback request. Both POST and GET requests are sent and received in the form of JSON messages since they are easy to identify and parse, which is important when data is constantly being exchanged from one side to another and vice-versa.

The prediction request sent by the user also includes either the English or Portuguese language to identify the respective model to use for the prediction. The Flask app then sends back another message to the user containing either "fake" or "real".

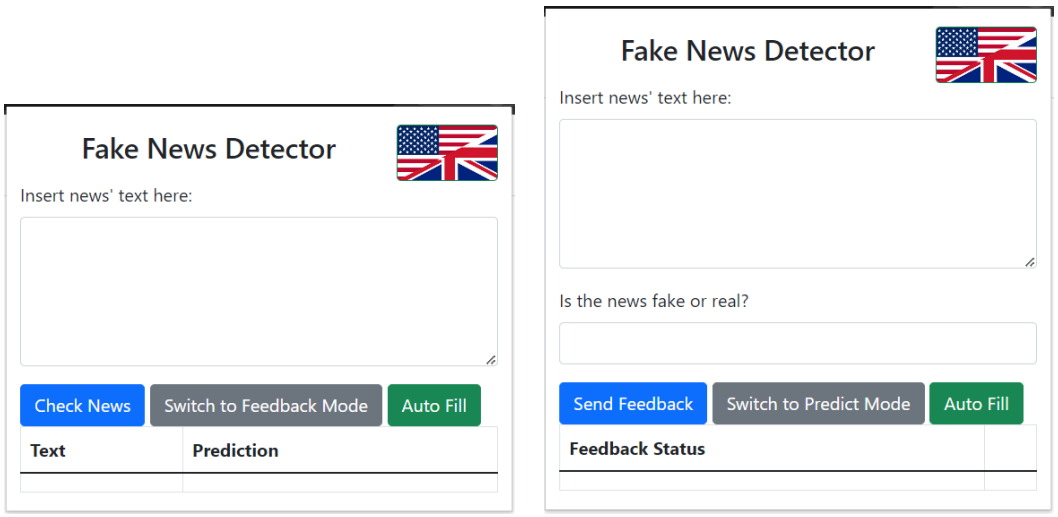
The feedback request is sent by the user to report either fake or real news, hence containing the text and respective label. Once the message is received, the Flask app stores its contents in a list for future use, sending afterwards another message to notify the user that the feedback was received successfully.

In case any request is not sent or received successfully, the Flask app sends a message notifying users of the possible cause behind such errors.

The development of a Flask app able to receive data and send models' predictions was followed by the development of a Chrome extension.

3.4.2 Chrome Extension

The Chrome extension is divided into prediction mode and feedback mode. Figure 3.26 shows the User Interface (UI) of both of these modes:



(a) Predict mode of the Chrome extension (b) Feedback mode of the Chrome extension

Figure 3.26: Predict and feedback modes of the Chrome extension

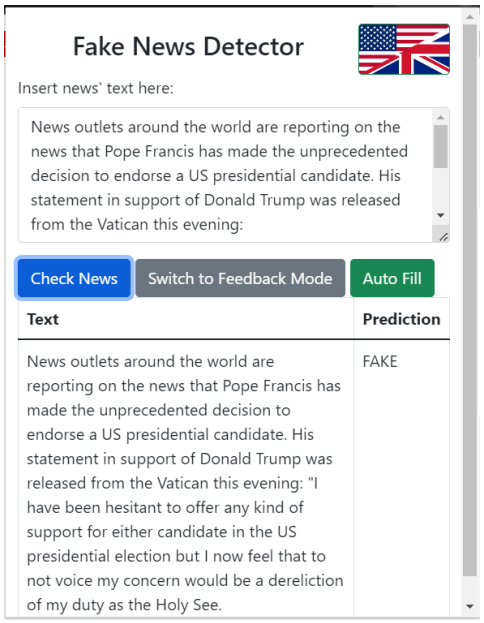
To request a prediction of a given text, users can type or copy and paste the desired text in the input field and then click on the *Check News* button to send the JSON message to the Flask app. The returned prediction is assigned to the table underneath the buttons, which shows the text and respective prediction.

To change between modes, users can click on the *Switch to Feedback Mode* and *Switch to Predict Mode* buttons. Changing between modes resets all input fields and tables as well.

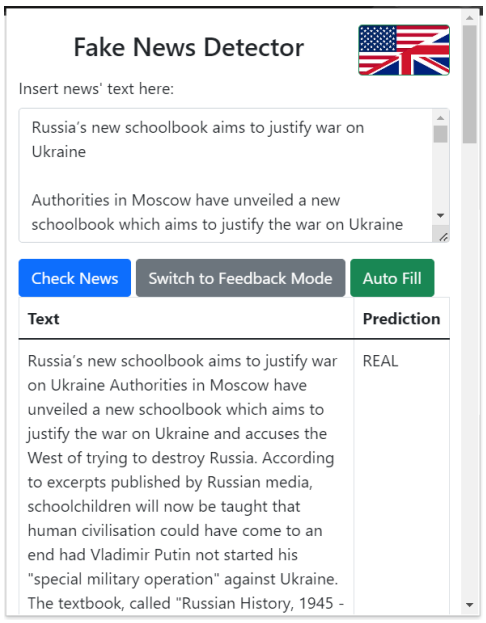
To report fake and real news, users can type or copy and paste the desired text and its respective label ("real" or "fake") in the appropriate input fields. Once the *Send Feedback* button is clicked, the feedback is sent and a status message is received stating whether the Flask app received the JSON message or not, as mentioned before.

Users can also select which model should be used to predict or to improve with their feedback by clicking on the flag on the top right corner of the extension, which changes between the English and Portuguese models.

Figures 3.27 and 3.28 show different examples of both fake and real news being correctly predicted by both models:

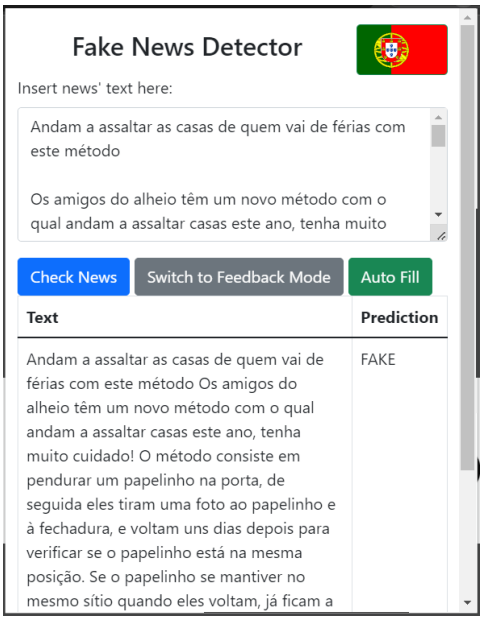


(a) Fake prediction on English news

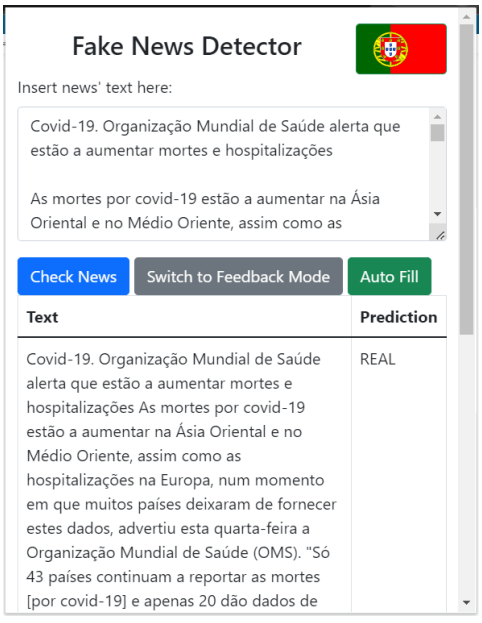


(b) Real prediction on English news

Figure 3.27: Fake and real predictions of English news on the Chrome extension



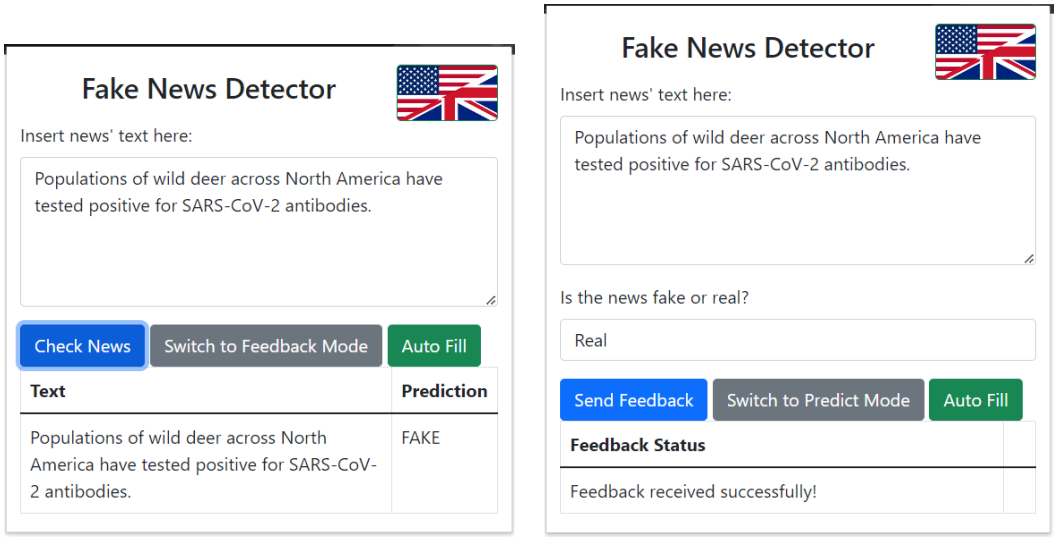
(a) Fake prediction on Portuguese news



(b) Real prediction on Portuguese news

Figure 3.28: Fake and real predictions of Portuguese news on the Chrome extension

Figure 3.29 shows an example of reporting a statement as real after the model incorrectly predicted it as fake:



(a) Incorrect prediction on the Chrome Extension (b) Feedback to improve the model

Figure 3.29: Real statement sent as feedback after being incorrectly predicted

Each mode also has the *Auto Fill* button which, as the name suggests, tries to fill the input field with the desired news title and text of the current tab in which the extension is open. This was added as a way to improve the users' experience by removing the hassle of having to constantly copy and paste the relevant text for predictions and feedback.

The auto fill process resorts to a background script, which extracts the Document Object Model (DOM) of the current webpage that contains the desired HTML tags, and port messaging, which makes it possible to exchange JSON messages with HTML data between the background script and the main script [105], [106].

Many tests with different fake and real news websites were conducted and most of them use the "h1" tag for the title and "p" tags for the different paragraphs found within the body of the news article. Since many elements can have paragraphs, the parent elements of all "p" elements are considered and the parent element with most paragraphs is set as the most common parent with the news' body element.

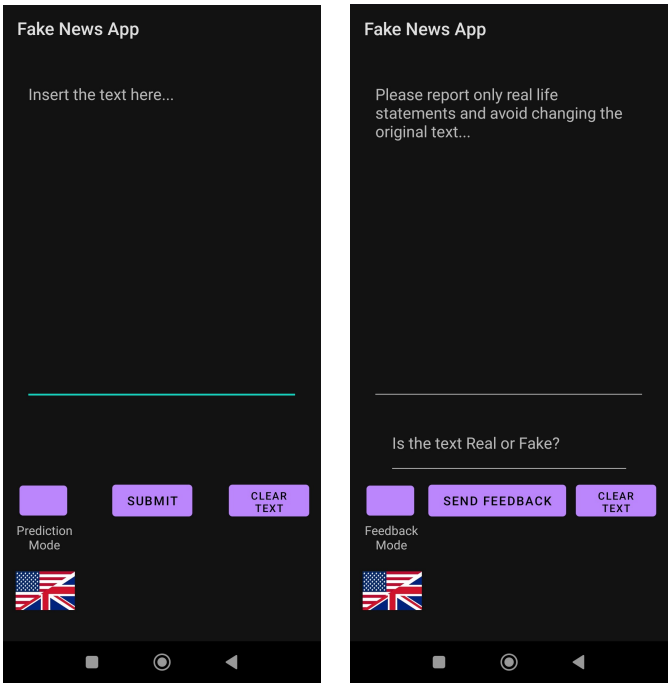
Some websites also had their cookies policies and terms of use in paragraph format, which raised a concern as they ended up being set as the news' body incorrectly due to their numerous paragraphs. Since these are typically found at the end of the webpage, only the first 8 parent elements are considered to filter the desired paragraphs.

Even though this process is complex enough to identify the title and body paragraphs of a news article, this will obviously not work for all websites due to how different their HTML structure can be. As a consequence, users have to manually copy and paste the text when the program is not able to automatically fill it in.

Once the Chrome extension was ready to be used, an Android application was also developed.

3.4.3 Android Application

An Android Application was developed using Android Studio. The app is naturally divided into prediction and feedback modes as well, which are presented in figure 3.30:



(a) Predict mode (b) Feedback mode

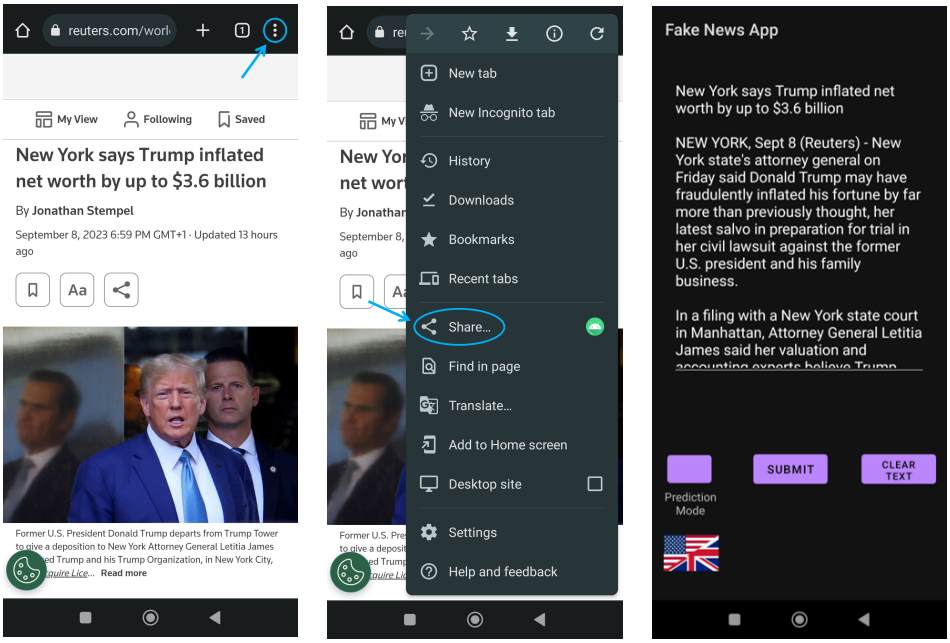
Figure 3.30: Predict and feedback modes of the Android app

The Android app is very similar to the Chrome extension in terms of UI, with buttons to send JSON messages and change between modes and models, as well as input fields to insert the text and label. To improve the user's experience, a new button was added to instantly clear the text of the input fields because it takes longer to clear them than on the Chrome Extension. The data is also being exchanged between the Android app and the Flask app with JSON messages and POST and GET requests with the Retrofit package [107].

The key difference between the Chrome extension and the Android app has to do with the auto fill function and how the HTML data is extracted: unlike the Chrome extension which has direct access to the HTML DOM of a webpage, the Android app resorts to the JSoup package which is able to obtain the page document from a given URL and gather the desired HTML elements and respective data [108].

To access the webpage's URL, the app also resorts to Intents that allow users to share data between apps, including the webpage currently opened in their browser to the Android app, which is then able to extract the URL required by JSoup. The app then opens with the news' text loaded in the respective input field, ready to either ask for a prediction or send as feedback. The content of a website is being selected and extracted similarly to what was described previously with the Chrome extension [109].

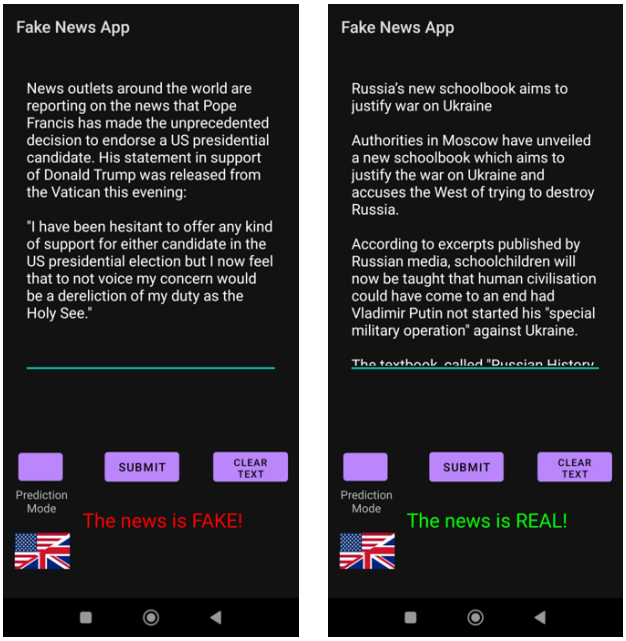
Figure 3.31 shows the steps behind the auto fill process of the Android App:



(a) Open browser settings (b) Share the URL (c) The data is loaded

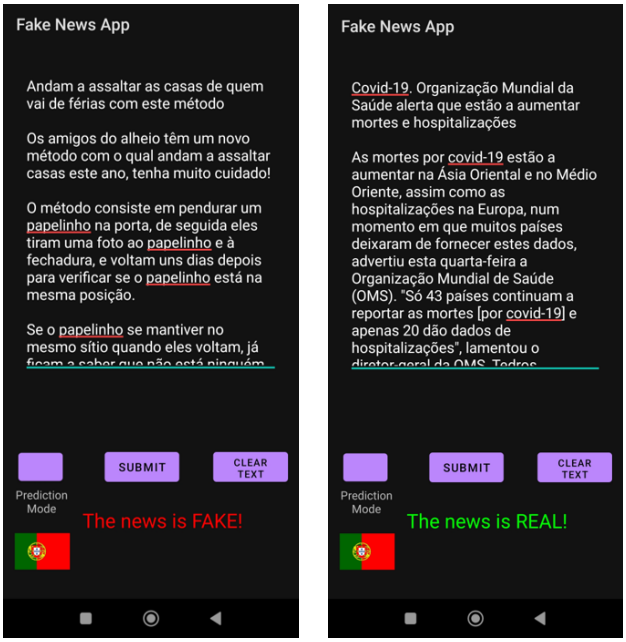
Figure 3.31: Steps behind the auto fill functionality of the Android app

Figures 3.32 and 3.33 show different examples of both fake and real news being correctly predicted by both models:



(a) Fake prediction (b) Real prediction

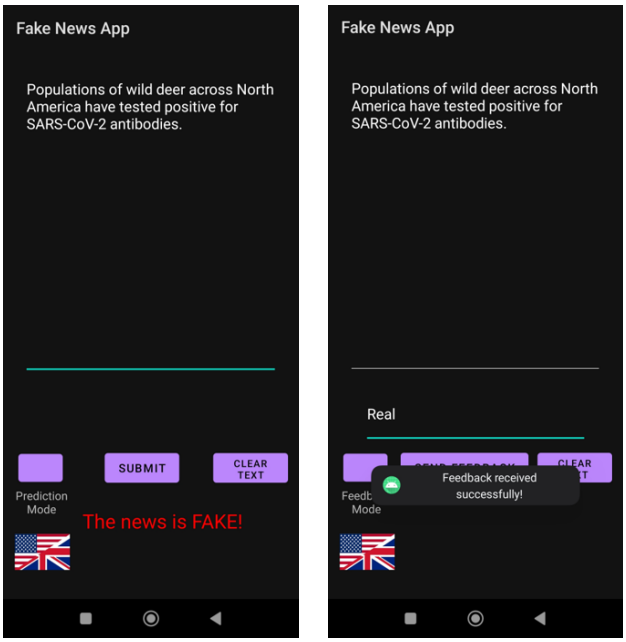
Figure 3.32: Fake and real predictions of English news on the Android app



(a) Fake prediction (b) Real prediction

Figure 3.33: Fake and real predictions of Portuguese news on the Android App

Figure 3.34 shows an example of reporting a statement as real after the model incorrectly predicted it as fake:



(a) Incorrect prediction (b) Feedback improvement

Figure 3.34: Real statement sent as feedback after being incorrectly predicted

With both Chrome extension and Android app working, it was time to deploy the Flask app on the cloud.

3.4.4 Application Containerization and Cloud Deployment

All POST and GET requests had been tested via a local computer and local Wi-fi connection, so deploying the Flask app to the cloud would be a step closer to a real service or production system, hence definitely worth considering. AWS was chosen as the cloud-based service given its popularity, as mentioned in the State of Art section. AWS also offers a free trial for new accounts, with an Elastic Compute Cloud (EC2) instance that has access to a machine with 1 GB of Random Access Memory (RAM), 30 GB of space volume and a Central Processing Unit (CPU) with 2 cores [110].

To ensure that the Flask App is able to run on any machine, one can resort to Virtual Machines and Containerization, which is where Docker comes in. A Docker container ensures that all the packages and dependencies required by the Flask app are met to run as intended. Since a Linux Virtual Machine with the Amazon Linux 2 AMI was chosen to run the container, the Flask app was adapted to use Gunicorn as the Web Server Gateway Interface (WSGI). This improves the system's scalability by setting the number of workers to the number of CPU cores, in order to handle multiple concurrent requests [111].

Once Docker was installed in the EC2 instance, the Flask app and the ML and DL models were copied over as well. Once the container with the Flask app is run, the Chrome Extension and the Android app can communicate with the cloud by requesting predictions and reporting fake or real news. Furthermore, a mount was also bound to the folder inside the container where the Flask app is running, so that the models can be updated in real-time after their improved versions are sent to replace the old ones.

However, given the limited resources provided in the free trial, the Flask app was not able to run the models that were created. The BERT model just on its own required at least 1.3 GB of RAM, which already exceeded the maximum amount provided for free, let alone both models while while receiving requests at the same time.

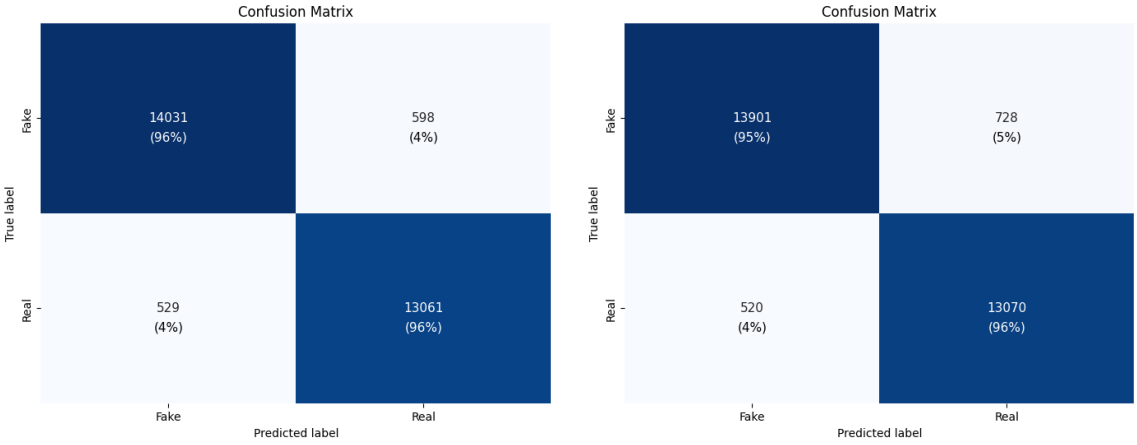
To avoid any additional costs, the models had to be optimized. Regarding the BERT model, a very similar version of it was trained called Distilled BERT (DistilBERT) model. Its performance comes very close to BERT with fewer resources required. In fact, the macro average F1-score of DistilBERT was still 0.96 with a smaller size required on disk of 910 MB when compared to the BERT model's size of 1.37 GB.

Despite the use of lighter models, the Flask app still crashed on the cloud instance, so the model was optimized even further with the help of the TFLiteConverter Package. This package is typically used when developing ML and DL models that are supposed to run locally on smart mobile devices, especially for purposes where the devices' sensors are constantly used for predictions and more [112].

Although they are not supposed to be running on mobile devices, this approach proved extrMLy crucial to allow the models to be used by the Flask app on the cloud instance, as they significantly reduce the resources required without sacrificing the performance that much. The size of the model reduced from 910 MB to 75.6 MB, which is impressive considering that the macro average F1-score was still 0.96!

This same problem was also evidenced regarding the Portuguese model. Although TF-IDF is known for being computationally cheap alongside ML models, it still surpassed the 1 GB of RAM limitation, which led to the training of another DistilBERT model in Google Colab and consequent conversion to a TensorFlow Lite (TFLite) file. In the end, the Portuguese model that would be used on the cloud instance achieved a macro average F1-score of 0.92, which is still very good, despite being lower than the previously mentioned XGBoost classifier.

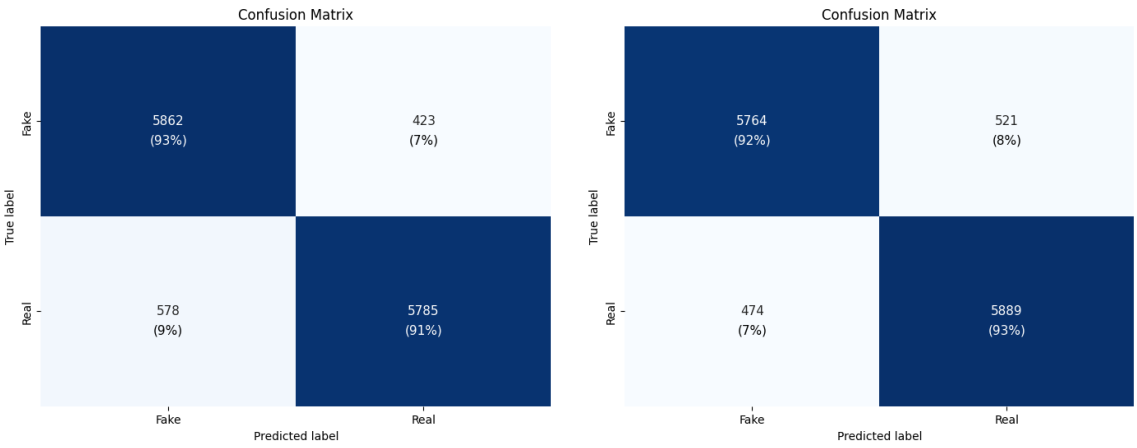
Figures 3.35 and 3.36 show the confusion matrix of the different DistilBERT and TFLite models:



(a) Confusion Matrix of DistilBERT

(b) Confusion Matrix of TFLite model

Figure 3.35: Confusion matrices of DistilBERT and respective TFLite English models



(a) Confusion Matrix of DistilBERT

(b) Confusion Matrix of TFLite model

Figure 3.36: Confusion matrices of DistilBERT and respective TFLite Portuguese models

There was, however, still an issue to tackle: the feedback data gathered from users was supposed to be used to improve the existing models by fine-tuning the DistilBERT models

and converting them to their respective TFLite versions, but the cloud instance was not able to handle these processes given its limited resources. As a result, a new approach had to be considered, which is where the RESTful script plays a crucial role.

3.4.5 RESTful Script for User Feedback and Model Improvement

The main problem of improving the DL models with user feedback revolved around the need of a GPU and more RAM. Instead of paying for another instance with more resources, a script was created and run on a local computer with a dedicated GPU.

By resorting to POST and GET requests, this program works as an API that fetches the feedback sent by users to the Flask app on the cloud instance. Once the Flask app sends the feedback data over to the local computer, the lists are cleared so that new future feedback can be stored.

There are, however, a few aspects to take into account during this final phase. The first one has to do with how the feedback data is filtered, after all, users can send whatever they want, which means some news articles or statements can end up being incorrectly reported, whether by mistake or on purpose.

On the other hand, using repeated or very similar entries as input can also result in bias since the model would be trained on a very specific set of data instead of a more diversified one, which typically leads to a worse ability to generalize and interpret unseen data, as already described before.

These details should be considered since fine-tuning models with irrelevant data proves detrimental to their performance, to the point where said models can stop being viable to perform their role for the task at hand. To overcome this possible outcome, a similarity-based approach was developed, in which the different texts reported by users are aggregated into different groups based on how similar they are. Repeated entries from the same user are also removed, so that only unique entries are considered and consequently assimilated.

Different approaches were considered during this step, including Sentiment Analysis to find common sentiment patterns and clustering algorithms such as Density-based Clustering Algorithm (DBSCAN) and Agglomerative Clustering, which divide the texts into different clusters based on lower and higher density regions and thresholds. However, many of the performed tests showed that these methods were struggling when finding distinct clusters, perhaps due to data points not being well-separated enough in the feature space [113], [114].

In the end, an approach involving a DistilBERT model and Cosine Similarity from sklearn achieved the best similarity results. This combination proved very useful because the DistilBERT model is able to capture more context and provide better sentence embeddings, which alongside Cosine Similarity allow the program to measure the agreement among users for a particular text or statement [115].

Figure 3.37 shows an example of using text embeddings and cosine similarity to measure how close certain sentences are to each other:

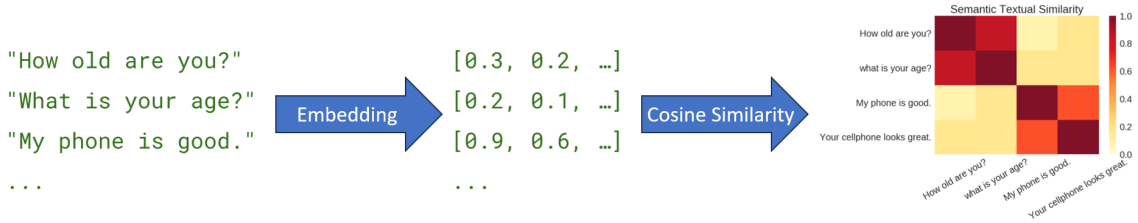
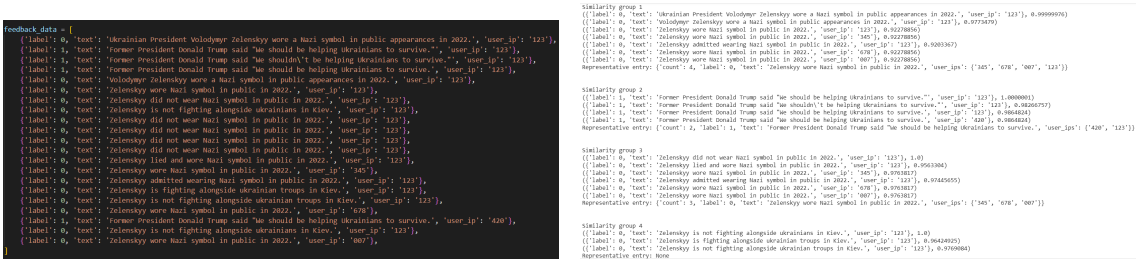


Figure 3.37: Text comparison using embeddings and cosine similarity (adapted from [116])

With different similarity groups, a representative text is then selected for each group, in order to generalize the feedback and extract the most meaningful and relevant statements, while also decreasing the model bias as stated before. The representative text is mainly selected based on how many users reported it and how similar it is to the remaining texts within the respective similarity group.

Figure 3.38 shows an example of similarity aggregation using the developed approach:



(a) Feedback example for similarity aggregation

(b) Resulting similarity groups

Figure 3.38: Example of feedback similarity aggregation

The similarity groups in figure 3.38 include the different users that reported each entry and a float value that corresponds to the cosine similarity score that was set based on the remaining entries of the respective group. Although the program identified the same representative entry for two groups, only the first one will be considered to reduce bias.

Even though this method allows the API to filter the feedback data, it also has its disadvantages, especially when it comes to similar texts that contradict each other or similar texts that have different labels. Developing a negation-handling approach based on Sentiment Analysis or common words with negative connotations would avoid these problems, but that could also result in misclassifications where the negation is more complex or subtle.

For example, the sentence "I do not disagree with him," has a double negation that makes it challenging to determine the actual sentiment. Another example is, for instance, "Despite some flaws, the movie is not bad", which has different negative words such as "flaws" and "bad", but its overall sentiment is positive.

Given the complexity behind this issue and the challenge it carries, the Law of Large Numbers was considered, which, in the context of fake news detection, states that, between two contradicting texts, the number of users that reported the correct one will in the end be higher than its counterpart [117].

These approaches make it possible to filter the feedback, which is then saved in the respective feedback dataset stored in the local computer, which is made of four columns, namely the text, label, the number of times said text and label were reported and the Internet Protocol (IP) addresses of the users that reported them.

Once filtered, the feedback data can finally be used to improve the existing models. This phase also raises another question since many different approaches can be taken into account. Training the models with the feedback data is probably the first one that comes to mind, but this method can significantly reduce the performance of the model in case the feedback data is very different from the data used during the training phase.

In fact, this same problem was observed several times when fitting the models. This goes against the very purpose of improving the models with user feedback, as the topics of news and statements discussed at the time are bound to change in the future, hence not being the most suitable approach.

Active Learning is another method used to label and select the most informative samples and fine-tune the models. Although this method would also help identify the data that would be likely to have the most impact when improving both models, this technique often requires human intervention to filter the relevant data.

Figure 3.39 shows an example in which human intervention is fundamental to differentiate confusing pictures of cupcakes and puppies through active learning:

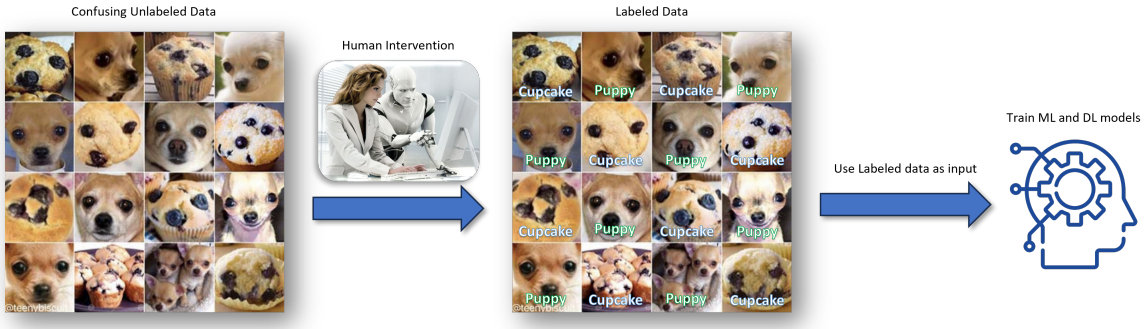


Figure 3.39: Labelling confusing photos with active learning (adapted from [118])

Given that it was intended to automate this process as much as possible, this technique was not used [118].

The models ended up being fine-tuned by resorting to Transfer Learning, which is often used in Deep Learning models such as Neural Networks or Transformers' pre-trained models.

Figure 3.40 shows the general concept behind the model's improvement with transfer learning:

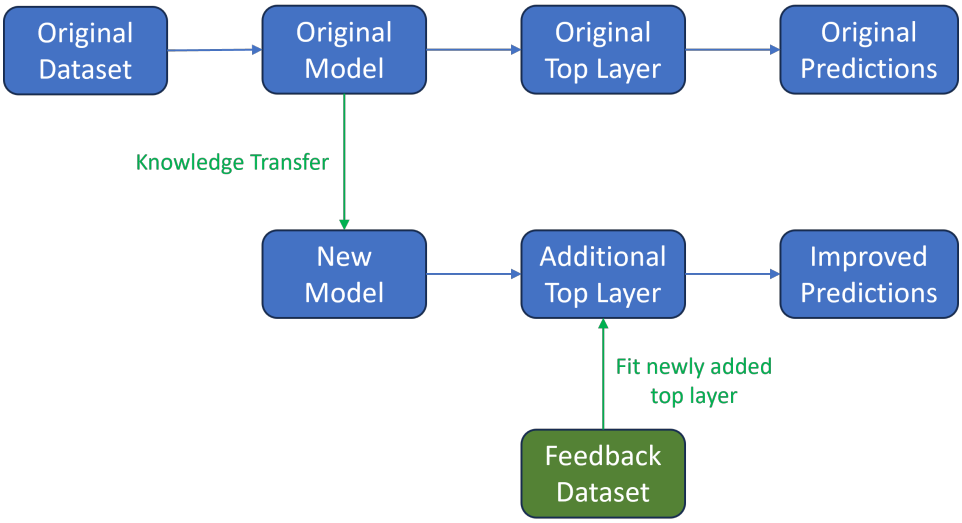


Figure 3.40: Use of transfer learning to improve the models with feedback data

Instead of fitting the whole model with the feedback data as described before, the pre-trained layers are frozen and a few new layers are added on top, which will be trained on the new user-generated data. This means the models are able to retain the knowledge and patterns acquired during the initial training phase with the data from the original dataset while fine-tuning their performance on new data [119].

Moreover, improving the models through transfer learning ensures that the received feedback is able to always teach the model about new topics or unfolding events while also improving already existing ones within the scope of its knowledge.

The fine-tuning process is followed by the models' conversion to their respective TFLite versions, which are then sent over to the cloud instance by resorting to the Paramiko package, which offers the ability to establish a connection between the script and the cloud instance with Secure Shell (SSH) commands and send the TFLite versions of the fine-tuned models through Secure File Transfer Protocol (SFTP) commands [120], [121], [122].

Once the cloud instance receives the models, the folder where the Flask app is running inside the container's volume is also updated because of the mount bound early on. The API then sends a POST request to the Flask app, so that the old TFLite models are replaced by the new improved ones, which are able to make better predictions based on the relevant feedback sent by users.

The datasets, the cloud-based web application source code, the Flask and Android applications, and the ML and DL models are available in this GitHub repository [123].

3.5 Results Discussion and Comparison

This section will analyse and reflect on the results obtained in the previous sections. Following the sequential flow of the project, the unexpected behaviour of the models in the English approach can be explained by a possible loss of context and patterns shared by the different text data entries once many pre-processing and feature extraction techniques were applied.

The sentiment distribution of fake and real news was very similar, so Sentiment Analysis did not improve the results either. This would also explain why BERT performed much better in the end, when only the text data was used as input, as the model was able to interpret the relationships and differences found within the data of fake and real news.

Although the data used in this approach is relevant to the task at hand, the combination of the three datasets may have significantly increased the complexity of the patterns, to the point where most models struggled during the training phase, despite performing well in each isolated dataset on their own with over 0.90 F1-score. Taking this into account, it does not make much sense to compare the BERT model of the English approach with the remaining projects, especially due to how many different datasets and projects exist.

The Portuguese approach, on the other hand, had its data extraction phase much more limited without the existence of public datasets. This also applies to the other European Portuguese projects explored in the State of Art that also resorted to similar sources, which translates to a much more coherent comparison. The main aspects of this comparison are presented in table 3.26:

Table 3.26: Portuguese approach compared with the previously explored projects

	Rodrigues [30]	Teixeira [31]	Afonso (Proposed Approach)
Sources	Fact-checking websites	Fake and real news websites	Fact-checking websites Fake and real news websites
Dataset Size	3764 (872 real and 2889 fake)	718 (543 real and 175 fake)	63236 (31716 real and 31520 fake)
Best Input Combination	Summary and source of each fact-check	Title and description of each news article	Title, source and first four paragraphs of each fact-check Title, source and description of each news article
Best Pre-processing Techniques	Punctuation Removal Tokenization Stopword Removal Lowercase	Stemming Tokenization Stopword Removal Lowercase	Lemmatization Stopword Removal Sentiment Analysis POS tagging
Best Feature Extraction Techniques	Bag of Words with TF-IDF and One Hot	TF-IDF (Uni and Bi-grams)	TF-IDF (Uni, Bi and Tri-grams)
Best model	Extra Trees (0.74 F1-score)	MNB (0.95 Accuracy)	XGBoost (0.957 F1-score)

The proposed Portuguese approach can be summarized as a mix of the other approaches with extra features and characteristics. Fact-checking websites and fake and real news websites were considered when extracting data and the resulting dataset is much bigger than the rest. The extra context of each fact-check described in the first four paragraphs provided more information when compared to each fact-check's summary.

Besides extracting much more data than the previous European Portuguese projects by resorting to web scraping, a much wider variety of sources was also considered. These aspects play an important role when it comes to acquiring a more representative sample of fake and real news articles found in real life, which consequently translated to the

development of the first publicly available fake news dataset in European Portuguese.

Moreover, the vast majority of fake news websites promoted other fake news websites as well, which were often used as the source of many articles, hence the importance of each news source. Sentiment Analysis and POS tagging also helped grasp relevant features, alongside feature extraction techniques such as TF-IDF and BoW.

Unlike the English Approach in which the majority of models struggled generalising the data, the conducted experiments with the previous techniques in the European Portuguese approach showed impressive results with both ML and DL models, thus confirming the importance and relevance of the different processes that were chosen.

XGBoost achieved the best F1-score, which was also higher than the previously mentioned projects, thus proving the impact of boosting algorithms in ML tasks, more specifically in the context of fake news detection.

This achievement is even more remarkable given the fact that much more data was considered when training the models than the previous projects, which can be challenging due to the increase of data patterns, thus leading to the development of much more robust and complex models designed to detect fake news in this language.

The Chrome extension and Android application developed alongside the Flask application that was run on Docker and deployed on the AWS EC2 cloud instance allowed the creation of a system that can be easily scalable and in which the developed models could be put to practice, proving their usefulness and possible impact when protecting their users from the dangers of fake news.

Furthermore, the development of a RESTful script capable of improving the models by filtering user feedback is a notable step, as it helps future-proof the ML and DL models and, ultimately, the system as a whole.

4

Conclusion

4.1 Developed Project Summary

The use of Artificial Intelligence is mandatory in fake news detection, hence the importance behind the creation of ML and DL models through text data mining and processing techniques. Two datasets were created, an English one from merging existing datasets and another in European Portuguese through web scraping. Based on conducted searches, this is the first dataset with real and fake news in European Portuguese publicly available.

Many different methods were considered to create ML and DL models, including pre-processing techniques and feature extraction and selection methods. Furthermore, certain techniques such as Sentiment Analysis and POS tagging were also considered, in order to obtain additional features found within fake and real news.

Finally, the developed models were optimized and deployed to the cloud, alongside a Flask app which receives prediction requests from a Chrome Extension and Android Application. Users can also send feedback which is processed in a script on a local computer to fine-tune the models through transfer learning and then send them over to the cloud instance to overwrite the old ones.

The contributions of this project were published in an article under the same title as this dissertation and submitted to the IEEE Latin America Transactions journal. The reviewers approved the article, and it is awaiting publication in the next issue.

4.2 Most Relevant Results and Faced Challenges

A wide variety of models and methods were considered, with BERT and respective tokenizer achieving the best F1-score of 0.96 in the English approach and XGBoost with pre-processing techniques, Sentiment Analysis, POS tagging and TF-IDF achieving the best F1-score of 0.957 in the European Portuguese approach.

A few obstacles were faced during the project's development. Although the web scraping phase went well for most European Portuguese websites, a few had their structure changed throughout the years, which translated to constant changes on the web scrapers.

Moreover, the creation of the English model proved quite challenging as well since the different techniques and processes explored in the State of Art with great results did not live up to expectations, unlike the European Portuguese approach in which said techniques allowed the models to perform very well.

The Chrome Extension also had its share of troubles, especially regarding the background script implementation to extract HTML data from the active tab and the consequent communication established between said script and the main script, in which the prediction and feedback requests were sent over to the Flask app on the cloud.

The final steps of the project revolving around the cloud deployment and the local script to improve the models packed quite a few challenges as well. These included the conflicts of different dependencies specified in the Dockerfile that were required to create the docker image and run the respective container with the Flask app, the models' optimizations that were mandatory given the cloud instance's limited resources, the similarity comparison approach developed in the RESTful script.

4.3 Future Work

Regarding the developed local script with RESTful methods, an Active Learning approach could also be developed, in which users rate each other's feedback, thus improving the filtering phase and the resulting models. Although the purpose of the feedback dataset is to receive the feedback when fetched by the RESTful script, they could also be merged with the datasets created with web scraping, in order to create and train future models from scratch instead of only using the feedback data during the fine-tuning step.

Furthermore, saving the data to a database rather than multiple files may be better in the long-term regarding the system's scalability, with more news articles being added by users through the web application as time passes.

The world of Deep Fakes has also been drastically increasing and evolving for the past few years, so developing a model capable of detecting them is something worth considering, although this could be challenging for European Portuguese given the kind of required data that is publicly available, which as of now is almost slim to none, unlike in English which already has many different datasets for such tasks.

Even though different models and techniques were considered, many others could also be used, along with more data from diversified topics and more ways to predict news besides the typical binary classification with "fake" or "real". These include using Twitter and Facebook APIs to extract social media posts or taking advantage of AI chatbots like ChatGPT from OpenAI or Bard from Google to extract even more data.

One is led to conclude that, at the end of the day, fake news websites are only concerned about the revenue received from advertisements, without any care in the world about the type of content that is shared or advertised, along with the implications they bring to readers and, ultimately, the whole world. Such indifference must not fly under the radars of governments and entities with enough power to put a stop to them.

Bibliography

[1] J. M. Lourenço. *The NOVAthesis LaTeX Template User's Manual*. NOVA University Lisbon. 2021. URL: https://github.com/joaomlourenco/novathesis/raw/master/template.pdf (cit. on p. ii).

[2] *Social media as a news source*. en. Page Version ID: 1131348711. 2023-01. URL: https://en.wikipedia.org/w/index.php?title=Social_media_as_a_news_source&oldid=1131348711 (visited on 2023-01-04) (cit. on p. 1).

[3] J. McGarrigle. *Explained: What is Fake news? | Social Media and Filter Bubbles*. en-GB. 2018-06. URL: https://www.webwise.ie/teachers/what-is-fake-news/ (visited on 2023-01-04) (cit. on p. 3).

[4] S. Maheshwari. "10 Times Trump Spread Fake News". en-US. In: *The New York Times* (2017-01). ISSN: 0362-4331. URL: https://www.nytimes.com/interactive/2017/business/media/trump-fake-news.html (visited on 2023-01-15) (cit. on p. 3).

[5] S. Khan et al. "Detecting COVID-19-Related Fake News Using Feature Extraction". en. In: *Frontiers in Public Health* 9 (2022-01), p. 788074. ISSN: 2296-2565. DOI: 10.3389/fpubh.2021.788074. URL: https://www.frontiersin.org/articles/10.3389/fpubh.2021.788074/full (visited on 2023-02-07) (cit. on pp. 3, 8, 36).

[6] *Five of the most viral misinformation posts since Ukraine war began*. en. 2022-08. URL: https://www.euronews.com/my-europe/2022/08/24/ukraine-war-five-of-the-most-viral-misinformation-posts-and-false-claims-since-the-conflic (visited on 2023-09-23) (cit. on p. 3).

[7] Heather. *What Are The Dangers of Fake News? | The Risk of Fake News*. en-US. 2021-11. URL: https://www.peoplesbanknet.com/the-dangers-of-fake-news/ (visited on 2023-01-04) (cit. on p. 3).

[8] *How is fake news spread? Bots, people like you, trolls, and microtargeting*. URL: https://www.cits.ucsb.edu/fake-news/spread (cit. on p. 4).

[9] *Fake news*. en. Page Version ID: 1130893324. 2023-01. URL: https://en.wikipedia.org/w/index.php?title=Fake_news&oldid=1130893324 (visited on 2023-01-04) (cit. on p. 4).

[10] *Tools That Fight Disinformation Online*. en. URL: https://www.rand.org/research/projects/truth-decay/fighting-disinformation/search.html (visited on 2023-01-15) (cit. on p. 5).

[11] SAPO. *Polígrafo*. pt. URL: https://poligrafo.sapo.pt/fact-checks (visited on 2023-01-16) (cit. on p. 5).

[12] Observador. *Fact Check – notícias, opinião, rádio, fotos e podcasts*. pt-PT. URL: https://observador.pt/seccao/observador/fact-check/ (visited on 2023-01-16) (cit. on p. 5).

[13] G. Brockell. *Prova dos Factos*. pt. URL: https://www.publico.pt/prova-dos-factos (visited on 2023-01-16) (cit. on p. 5).

[14] *O que é o projeto «Combate às fake news»/«Contra Fake»*. Section: Sem categoria. 2020-08. URL: https://combatefakenews.lusa.pt/o-projeto-combate-as-fake-news-contra-fake/ (visited on 2023-01-16) (cit. on p. 5).

[15] *What Is a Data Set? (With Definition, Types and Examples) | Indeed.com*. en. URL: https://www.indeed.com/career-advice/career-development/what-is-data-set (visited on 2023-09-15) (cit. on p. 6).

[16] a. Tondak. *Structured Vs Unstructured Data*. en-US. 2022-11. URL: https://k21academy.com/microsoft-azure/dp-900/structured-data-vs-unstructured-data-vs-semi-structured-data/ (visited on 2023-09-15) (cit. on p. 7).

[17] C. Woolard. *What's the difference between structured, semi-structured, and unstructured data?* en-US. 2023-06. URL: https://automationhero.ai/blog/whats-the-difference-between-structured-semi-structured-and-unstructured-data/ (visited on 2023-09-15) (cit. on p. 7).

[18] *Semi-Structured Data - javatpoint*. en. URL: https://www.javatpoint.com/semi-structured-data (visited on 2023-09-15) (cit. on p. 7).

[19] A. Abdulrahman and M. Baykara. "Fake News Detection Using Machine Learning and Deep Learning Algorithms". en. In: *2020 International Conference on Advanced Science and Engineering (ICOASE)*. Duhok, Iraq: IEEE, 2020-12, pp. 18–23. ISBN: 978-1-66541-579-8. DOI: 10.1109/ICOASE51841.2020.9436605. URL: https://ieeexplore.ieee.org/document/9436605/ (visited on 2023-01-06) (cit. on pp. 7, 11, 30).

[20] S. R. Sahoo and B. Gupta. "Multiple features based approach for automatic fake news detection on social networks using deep learning". en. In: *Applied Soft Computing* 100 (2021-03), p. 106983. ISSN: 15684946. DOI: 10.1016/j.asoc.2020.106983. URL: https://linkinghub.elsevier.com/retrieve/pii/S1568494620309224 (visited on 2023-02-04) (cit. on pp. 7, 28, 35).

[21] P. Patwa et al. "Fighting an Infodemic: COVID-19 Fake News Dataset". In: *Combating Online Hostile Posts in Regional Languages during Emergency Situation*. Springer International Publishing, 2021, pp. 21–29. DOI: 10.1007/978-3-030-73696-5_3. URL: https://doi.org/10.1007%2F978-3-030-73696-5_3 (cit. on pp. 7, 36).

[22] P. Patwa et al. "Overview of CONSTRAINT 2021 Shared Tasks: Detecting English COVID-19 Fake News and Hindi Hostile Posts". In: *Proceedings of the First Workshop on Combating Online Hostile Posts in Regional Languages during Emergency Situation (CONSTRAINT)*. Springer, 2021 (cit. on p. 7).

[23] W. Antoun et al. "State of the Art Models for Fake News Detection Tasks". en. In: *2020 IEEE International Conference on Informatics, IoT, and Enabling Technologies (ICIoT)*. Doha, Qatar: IEEE, 2020-02, pp. 519–524. ISBN: 978-1-72814-821-2. DOI: 10.1109/ICIoT48696.2020.9089487. URL: https://ieeexplore.ieee.org/document/9089487/ (visited on 2023-01-07) (cit. on pp. 7, 31).

[24] Z. Khanam et al. "Fake News Detection Using Machine Learning Approaches". en. In: *IOP Conference Series: Materials Science and Engineering* 1099.1 (2021-03), p. 012040. ISSN: 1757-8981, 1757-899X. DOI: 10.1088/1757-899X/1099/1/012040. URL: https://iopscience.iop.org/article/10.1088/1757-899X/1099/1/012040 (visited on 2023-01-08) (cit. on pp. 7, 12, 32).

[25] A. Thota et al. "Fake News Detection: A Deep Learning Approach". en. In: 1.3 (2018) (cit. on pp. 7, 33).

[26] I. Ahmad et al. "Fake News Detection Using Machine Learning Ensemble Methods". en. In: *Complexity* 2020 (2020-10). Ed. by M. I. Uddin, pp. 1–11. ISSN: 1099-0526, 1076-2787. DOI: 10.1155/2020/8885861. URL: https://www.hindawi.com/journals/complexity/2020/8885861/ (visited on 2023-01-09) (cit. on pp. 7, 12, 34).

[27] S. Mishra, P. Shukla, and R. Agarwal. "Analyzing Machine Learning Enabled Fake News Detection Techniques for Diversified Datasets". en. In: *Wireless Communications and Mobile Computing* 2022 (2022-03). Ed. by M. F. Hashmi, pp. 1–18. ISSN: 1530-8677, 1530-8669. DOI: 10.1155/2022/1575365. URL: https://www.hindawi.com/journals/wcmc/2022/1575365/ (visited on 2023-01-16) (cit. on pp. 7, 14, 17, 25, 34).

[28] P. K. Verma et al. "WELFake: Word Embedding Over Linguistic Features for Fake News Detection". In: *IEEE Transactions on Computational Social Systems* 8.4 (2021), pp. 881–893. DOI: 10.1109/TCSS.2021.3068519 (cit. on pp. 8, 35).

[29] R. M. Silva et al. "Towards automatically filtering fake news in Portuguese". In: *Expert Systems with Applications* 146 (2020), p. 113199. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2020.113199. URL: https://www.sciencedirect.com/science/article/pii/S0957417420300257 (cit. on pp. 8, 36).

[30] J. F. C. Rodrigues. "Fake News Classification in European Portuguese Language". en. In: (2020). URL: http://hdl.handle.net/10071/22194 (cit. on pp. 8, 9, 37, 57, 83).

[31] M. R. P. Teixeira. "Índice de Credibilidade de Conteúdos Noticiosos em Língua Portuguesa para Uso em Ambiente Escolar". pt. In: (). URL: http://hdl.handle.net/10400.22/18330 (cit. on pp. 8, 25, 29, 38, 83).

[32] *Web Scraping Vs Web Crawling | Zyte*. en-GB. Section: Learn. 2021-01. URL: https://www.zyte.com/learn/difference-between-web-scraping-and-web-crawling/ (visited on 2023-02-01) (cit. on p. 9).

[33] M. Wong. *Basic Frontend Knowledge*. en. 2020-08. URL: https://levelup.gitconnected.com/basic-frontend-knowledge-737702051bd8 (visited on 2023-02-02) (cit. on p. 10).

[34] *What is Web Scraping and How to Use It?* en-us. Section: GBlog. 2020-06. URL: https://www.geeksforgeeks.org/what-is-web-scraping-and-how-to-use-it/ (visited on 2023-02-02) (cit. on p. 10).

[35] *Web Crawling vs Web Scraping: What Is the Difference?* en. 2022-12. URL: https://soax.com/blog/web-crawling-vs-web-scraping-what-is-the-difference (visited on 2023-09-17) (cit. on p. 10).

[36] L. Richardson. *beautifulsoup4: Screen-scraping library*. URL: https://www.crummy.com/software/BeautifulSoup/bs4/ (visited on 2023-08-21) (cit. on p. 10).

[37] *WebDriver*. en. URL: https://www.selenium.dev/documentation/webdriver/ (visited on 2023-08-21) (cit. on p. 10).

[38] *What is Text Mining? | IBM*. en-us. URL: https://www.ibm.com/topics/text-mining (visited on 2023-01-18) (cit. on pp. 11, 12).

[39] V. Balakrishnan and L.-Y. Ethel. "Stemming and Lemmatization: A Comparison of Retrieval Performances". en. In: *Lecture Notes on Software Engineering* 2.3 (2014), pp. 262–267. ISSN: 23013559. DOI: 10.7763/LNSE.2014.V2.134. URL: http://www.lnse.org/show-34-165-1.html (visited on 2023-01-18) (cit. on p. 11).

[40] K. Pykes. *Part Of Speech Tagging for Beginners*. en. 2020-11. URL: https://towardsdatascience.com/part-of-speech-tagging-for-beginners-3a0754b2ebba (visited on 2023-01-07) (cit. on p. 12).

[41] *A Guide to Text Preprocessing Techniques for NLP - Blog*. URL: https://exchange.scale.com/public/blogs/preprocessing-techniques-in-nlp-a-guide (visited on 2023-01-31) (cit. on p. 12).

[42] *The Role of Feature Extraction in Machine Learning*. en-US. URL: https://www.snowflake.com/guides/feature-extraction-machine-learning (visited on 2023-01-06) (cit. on p. 13).

[43] *Overfitting*. en. Page Version ID: 1115154474. 2022-10. URL: https://en.wikipedia.org/w/index.php?title=Overfitting&oldid=1115154474 (visited on 2023-01-12) (cit. on p. 13).

[44] *TF-IDF — Term Frequency-Inverse Document Frequency – LearnDataSci*. URL: https://www.learndatasci.com/glossary/tf-idf-term-frequency-inverse-document-frequency/ (visited on 2023-01-06) (cit. on p. 13).

[45] *Bag-of-words model*. en. Page Version ID: 1136283063. 2023-01. URL: https://en.wikipedia.org/w/index.php?title=Bag-of-words_model&oldid=1136283063 (visited on 2023-02-03) (cit. on p. 14).

[46] S. Srinidhi. *Understanding Word N-grams and N-gram Probability in Natural Language Processing*. en. 2020-01. URL: https://towardsdatascience.com/understanding-word-n-grams-and-n-gram-probability-in-natural-language-processing-9d9eef0fa058 (visited on 2023-01-06) (cit. on pp. 14, 30).

[47] *Correlation Vs Covariance in Data Science*. en. URL: https://www.projectpro.io/article/correlation-vs-covariance/489 (visited on 2023-02-03) (cit. on p. 14).

[48] MarketMuse. *What is Latent Semantic Analysis (LSA) - Latent Semantic Analysis (LSA) Definition from MarketMuse Blog*. en-US. URL: https://blog.marketmuse.com/glossary/latent-semantic-analysis-definition/ (visited on 2023-02-03) (cit. on p. 14).

[49] F. P. Shah and V. Patel. "A review on feature selection and feature extraction for text classification". en. In: *2016 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)*. Chennai, India: IEEE, 2016-03, pp. 2264–2268. ISBN: 978-1-4673-9338-6. DOI: 10.1109/WiSPNET.2016.7566545. URL: http://ieeexplore.ieee.org/document/7566545/ (visited on 2023-02-02) (cit. on pp. 14, 17).

[50] M. Oleszak. *Feature Selection Methods and How to Choose Them*. en-US. 2022-09. URL: https://neptune.ai/blog/feature-selection-methods (visited on 2023-02-02) (cit. on p. 15).

[51] *Difference Between T-test and F-test (with Comparison Chart)*. en-US. 2017-01. URL: https://keydifferences.com/difference-between-t-test-and-f-test.html (visited on 2023-02-04) (cit. on p. 15).

[52] M. McCombe. *Intro to Feature Selection Methods for Data Science*. en. 2019-06. URL: https://towardsdatascience.com/intro-to-feature-selection-methods-for-data-science-4cae2178a00a (visited on 2023-02-03) (cit. on p. 17).

[53] *scikit-learn: machine learning in Python — scikit-learn 1.2.1 documentation*. URL: https://scikit-learn.org/stable/ (visited on 2023-02-05) (cit. on p. 18).

[54] *NLTK :: Natural Language Toolkit*. URL: https://www.nltk.org/ (visited on 2023-02-05) (cit. on p. 18).

[55] *Language Processing Pipelines · spaCy Usage Documentation*. en. URL: https://spacy.io/usage/processing-pipelines (visited on 2023-02-05) (cit. on p. 18).

[56] *Regression vs. Classification in Machine Learning: What's the Difference?* en. Section: Data Science. 2021-10. URL: https://www.springboard.com/blog/data-science/regression-vs-classification/ (visited on 2023-02-06) (cit. on p. 18).

[57] *Stacking in Machine Learning - Javatpoint*. URL: https://www.javatpoint.com/stacking-in-machine-learning (visited on 2023-02-06) (cit. on p. 20).

[58] S. Wu. *How to choose between different Boosting Algorithms*. en. 2021-06. URL: https://towardsdatascience.com/how-to-select-between-boosting-algorithm-e8d1b15924f7 (visited on 2023-02-06) (cit. on p. 21).

[59] *Linear Regression vs Logistic Regression - Javatpoint*. en. URL: https://www.javatpoint.com/linear-regression-vs-logistic-regression-in-machine-learning (visited on 2023-09-19) (cit. on p. 21).

[60] *Support Vector Machine (SVM) Algorithm - Javatpoint*. en. URL: https://www.javatpoint.com/machine-learning-support-vector-machine-algorithm (visited on 2023-09-19) (cit. on p. 22).

[61] *Deep learning*. en. Page Version ID: 1137673005. 2023-02. URL: https://en.wikipedia.org/w/index.php?title=Deep_learning&oldid=1137673005 (visited on 2023-02-06) (cit. on p. 22).

[62] *What is Deep Learning? | IBM*. en-us. URL: https://www.ibm.com/topics/deep-learning (visited on 2023-02-06) (cit. on p. 22).

[63] *What are Neural Networks? | IBM*. en-us. URL: https://www.ibm.com/topics/neural-networks (visited on 2023-02-06) (cit. on p. 23).

[64] *Convolutional Neural Networks with PyTorch | Domino Data Lab*. URL: https://domino.ai/blog/gpu-accelerated-convolutional-neural-networks-with-pytorch (visited on 2023-09-19) (cit. on p. 23).

[65] M. Mishra. *Convolutional Neural Networks, Explained*. en. 2020-09. URL: https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939 (visited on 2023-02-06) (cit. on p. 23).

[66] C.-F. Wang. *The Vanishing Gradient Problem*. en. 2019-01. URL: https://towardsdatascience.com/the-vanishing-gradient-problem-69bf08b15484 (visited on 2023-02-07) (cit. on p. 24).

[67] J. Dancker. *A Brief Introduction to Recurrent Neural Networks*. en. 2022-12. URL: https://towardsdatascience.com/a-brief-introduction-to-recurrent-neural-networks-638f64a61ff4 (visited on 2023-09-20) (cit. on pp. 24, 25).

[68] *TensorFlow*. en. URL: https://www.tensorflow.org/ (visited on 2023-02-05) (cit. on p. 25).

[69] *Tensorflow 1.0 vs. Tensorflow 2.0: What's the Difference?* en. Section: Data Science. 2021-10. URL: https://www.springboard.com/blog/data-science/tensorflow-1-0-vs-tensorflow-2-0/ (visited on 2023-02-05) (cit. on p. 25).

[70] *Keras: Deep Learning for humans*. original-date: 2015-03-28T00:35:42Z. 2023-02. URL: https://github.com/keras-team/keras (visited on 2023-02-05) (cit. on p. 25).

[71] *Hugging Face Transformers*. URL: https://huggingface.co/docs/transformers/index (visited on 2023-08-21) (cit. on p. 25).

[72] J. Brownlee. *A Gentle Introduction to k-fold Cross-Validation*. en-US. 2018-05. URL: https://machinelearningmastery.com/k-fold-cross-validation/ (visited on 2023-02-04) (cit. on p. 26).

[73] V. Lyashenko. *Cross-Validation in Machine Learning: How to Do It Right*. en-US. 2022-07. URL: https://neptune.ai/blog/cross-validation-in-machine-learning-how-to-do-it-right (visited on 2023-02-04) (cit. on p. 26).

[74] J. Brownlee. *Why Do I Get Different Results Each Time in Machine Learning?* en-US. 2020-08. URL: https://machinelearningmastery.com/different-results-each-time-in-machine-learning/ (visited on 2023-02-04) (cit. on p. 26).

[75] S. Narkhede. *Understanding Confusion Matrix*. en. 2021-06. URL: https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62 (visited on 2023-01-08) (cit. on pp. 27, 28).

[76] H. K. Gupta. *How to make a make a Google Chrome Extension*. en. 2020-05. URL: https://medium.com/tech-iiitg/how-to-make-a-make-a-google-chrome-extension-4867d3bffec (visited on 2023-02-05) (cit. on p. 28).

[77] *Flask (web framework)*. en. Page Version ID: 1120483147. 2022-11. URL: https://en.wikipedia.org/w/index.php?title=Flask_(web_framework)&oldid=1120483147 (visited on 2023-02-05) (cit. on p. 28).

[78] *What is a REST API?* en. URL: https://www.redhat.com/en/topics/api/what-is-a-rest-api (visited on 2023-02-05) (cit. on p. 28).

[79] *What's the Difference Between AWS vs. Azure vs. Google Cloud?* en. 2023-06. URL: https://www.coursera.org/articles/aws-vs-azure-vs-google-cloud (visited on 2023-08-22) (cit. on p. 29).

[80] A. Bartwal. *AWS vs Azure vs GCP: Which is Best Cloud Platform*. en-US. 2023-06. URL: https://k21academy.com/amazon-web-services/aws-solutions-architect/aws-vs-azure-vs-gcp/ (visited on 2023-08-22) (cit. on p. 29).

[81] *Meet Android Studio*. en. URL: https://developer.android.com/studio/intro (visited on 2023-08-22) (cit. on p. 29).

[82] *Kotlin VS Java – What's the Difference?* en. 2023-03. URL: https://www.freecodecamp.org/news/kotlin-vs-java-whats-the-difference/ (visited on 2023-08-22) (cit. on p. 29).

[83] *Make your Android application work on iOS – tutorial | Kotlin*. en-US. URL: https://kotlinlang.org/docs/multiplatform-mobile-integrate-in-existing-app.html (visited on 2023-09-26) (cit. on p. 29).

[84] A. Inc. *Xcode 15*. en. URL: https://developer.apple.com/xcode/ (visited on 2023-08-22) (cit. on p. 29).

[85] *What Is Xcode and How to Use It?* en. URL: https://www.netguru.com/blog/what-is-xcode-and-how-to-use-it (visited on 2023-08-22) (cit. on p. 29).

[86] J. Brownlee. *Extreme Gradient Boosting (XGBoost) Ensemble in Python*. en-US. 2020-11. URL: https://machinelearningmastery.com/extreme-gradient-boosting-ensemble-in-python/ (visited on 2023-01-06) (cit. on p. 30).

[87] V. Choubey. *Undestanding Recurrent Neural Network (RNN) and Long Short Term Memory(LSTM)*. en. 2020-07. URL: https://medium.com/analytics-vidhya/undestanding-recurrent-neural-network-rnn-and-long-short-term-memory-lstm-30bc1221e80d (visited on 2023-01-06) (cit. on p. 31).

[88] *What is BERT (Language Model) and How Does It Work?* en. URL: https://www.techtarget.com/searchenterpriseai/definition/BERT-language-model (visited on 2023-01-07) (cit. on p. 31).

[89] P. Kandru. *Guide to XLNet for Language Understanding*. en-US. 2021-03. URL: https://analyticsindiamag.com/guide-to-xlnet-for-language-understanding/ (visited on 2023-01-07) (cit. on p. 32).

[90] Z. Yang et al. *XLNet: Generalized Autoregressive Pretraining for Language Understanding*. en. arXiv:1906.08237 [cs]. 2020-01. URL: http://arxiv.org/abs/1906.08237 (visited on 2023-01-07) (cit. on p. 32).

[91] *RoBERTa: An optimized method for pretraining self-supervised NLP systems*. en. URL: https://ai.facebook.com/blog/roberta-an-optimized-method-for-pretraining-self-supervised-nlp-systems/ (visited on 2023-01-07) (cit. on p. 32).

[92] *Epoch*. 2019-05. URL: https://deepai.org/machine-learning-glossary-and-terms/epoch (visited on 2023-01-08) (cit. on p. 32).

[93] A. Gholamy, V. Kreinovich, and O. Kosheleva. "Why 70/30 or 80/20 Relation Between Training and Testing Sets: A Pedagogical Explanation". In: *Departmental Technical Reports (CS)* (2018-02). URL: https://scholarworks.utep.edu/cs_techrep/1209 (cit. on p. 32).

[94] G. Myrianthous. *Training vs Testing vs Validation Sets*. en. 2021-08. URL: https://towardsdatascience.com/training-vs-testing-vs-validation-sets-a44bed52a0e1 (visited on 2023-01-08) (cit. on p. 33).

[95] S. Ronaghan. *Deep Learning: Which Loss and Activation Functions should I use?* en. 2019-08. URL: https://towardsdatascience.com/deep-learning-which-loss-and-activation-functions-should-i-use-ac02f1c56aa8 (visited on 2023-01-08) (cit. on p. 33).

[96] J. Brownlee. *Difference Between a Batch and an Epoch in a Neural Network*. en-US. 2022-08. URL: https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/ (visited on 2023-01-08) (cit. on p. 33).

[97] *Google Colab*. en. URL: https://colab.google/ (visited on 2023-09-27) (cit. on p. 36).

[98] *Fake News Classification*. en. URL: https://www.kaggle.com/datasets/saurabhshahane/fake-news-classification (visited on 2023-08-17) (cit. on p. 46).

[99] *Misinformation & Fake News text dataset 79k*. en. URL: https://www.kaggle.com/datasets/stevenpeutz/misinformation-fake-news-text-dataset-79k (visited on 2023-08-17) (cit. on p. 46).

[100] H. Ahmed, I. Traore, and S. Saad. "Detection of Online Fake News Using N-Gram Analysis and Machine Learning Techniques". In: 2017-10, pp. 127–138. ISBN: 978-3-319-69154-1. DOI: 10.1007/978-3-319-69155-8_9 (cit. on p. 46).

[101] *CodaLab - Competition*. URL: https://competitions.codalab.org/competitions/26655#learn_the_details (visited on 2023-08-17) (cit. on p. 46).

[102] *Detect an Unknown Language using Python*. en-us. Section: Python. 2020-01. URL: https://www.geeksforgeeks.org/detect-an-unknown-language-using-python/ (visited on 2023-09-20) (cit. on p. 46).

[103] *Arquivo.pt - pesquise páginas do passado!* pt. URL: https://arquivo.pt/ (visited on 2023-08-23) (cit. on p. 57).

[104] G. M. Group. *Fake news: sites portugueses com mais de dois milhões de seguidores*. pt-PT. 2018-11. URL: https://apav.pt/apav_v3/index.php/pt/1866-diario-de-noticias-fake-news-sites-portugueses-com-mais-de-dois-milhoes-de-seguidores (visited on 2023-08-24) (cit. on p. 57).

[105] *chrome.scripting*. en. URL: https://developer.chrome.com/docs/extensions/reference/scripting/ (visited on 2023-09-08) (cit. on p. 73).

[106] *Chrome Extensions Message passing*. en. 2012-09. URL: https://developer.chrome.com/docs/extensions/mv3/messaging/ (visited on 2023-09-08) (cit. on p. 73).

[107] Piotr. *Retrofit With Kotlin- The Ultimate Guide*. en-US. 2022-03. URL: https://codersee.com/retrofit-with-kotlin-the-ultimate-guide/ (visited on 2023-09-08) (cit. on p. 74).

[108] pradeep. *Using Jsoup With Kotlin To Scrape Wiki Pages*. en. 2019-03. URL: https://thetechstack.net/using-jsoup-with-kotlin-to-scrape-wiki-pages/ (visited on 2023-09-08) (cit. on p. 74).

[109] *Send simple data to other apps*. en. URL: https://developer.android.com/training/sharing/send (visited on 2023-09-08) (cit. on p. 74).

[110] T. Jain. *Simple way to deploy machine learning models to cloud*. en. 2019-04. URL: https://towardsdatascience.com/simple-way-to-deploy-machine-learning-models-to-cloud-fd58b771fdcf (visited on 2023-08-27) (cit. on p. 77).

[111] S. Ilarslan. *What is Gunicorn?* en. 2022-05. URL: https://medium.com/@serdarilarslan/what-is-gunicorn-5e674fff131b (visited on 2023-08-27) (cit. on p. 77).

[112] *TensorFlow Lite | ML for Mobile and Edge Devices*. en. URL: https://www.tensorflow.org/lite (visited on 2023-09-07) (cit. on p. 77).

[113] *sklearn.cluster.DBSCAN*. en. URL: https://scikit-learn/stable/modules/generated/sklearn.cluster.DBSCAN.html (visited on 2023-09-07) (cit. on p. 79).

[114] *sklearn.cluster.AgglomerativeClustering*. en. URL: https://scikit-learn/stable/modules/generated/sklearn.cluster.AgglomerativeClustering.html (visited on 2023-09-08) (cit. on p. 79).

[115] *sklearn.metrics.pairwise.cosine_similarity*. en. URL: https://scikit-learn/stable/modules/generated/sklearn.metrics.pairwise.cosine_similarity.html (visited on 2023-09-07) (cit. on p. 79).

[116] *NLP Town*. URL: http://nlp.town/blog/sentence-similarity/ (visited on 2023-09-22) (cit. on p. 80).

[117] *Law of large numbers*. en. Page Version ID: 1173350484. 2023-09. URL: https://en.wikipedia.org/w/index.php?title=Law_of_large_numbers&oldid=1173350484 (visited on 2023-09-07) (cit. on p. 81).

[118] A. C. John. *Active Learning: Strategies, Tools, and Real-World Use Cases*. en-US. 2022-08. URL: https://neptune.ai/blog/active-learning-strategies-tools-use-cases (visited on 2023-09-07) (cit. on p. 81).

[119] D. Mwiti. *Transfer Learning Guide: A Practical Tutorial With Examples for Images and Text in Keras*. en-US. 2022-07. URL: https://neptune.ai/blog/transfer-learning-guide-examples-for-images-and-text-in-keras (visited on 2023-09-07) (cit. on p. 82).

[120] *Welcome to Paramiko! — Paramiko documentation*. URL: https://www.paramiko.org/ (visited on 2023-09-07) (cit. on p. 82).

[121] *What is SSH (Secure Shell) and How Does it Work? Definition from TechTarget*. en. URL: https://www.techtarget.com/searchsecurity/definition/Secure-Shell (visited on 2023-09-07) (cit. on p. 82).

[122] *What is Secure File Transfer Protocol (SFTP)? A Definition from TechTarget.com*. en. URL: https://www.techtarget.com/searchcontentmanagement/definition/Secure-File-Transfer-Protocol-SSH-File-Transfer-Protocol (visited on 2023-09-07) (cit. on p. 82).

[123] R. O. Afonso. *ro-afonso/fake-news-pt-eu*. 2023. URL: https://github.com/ro-afonso/fake-news-pt-eu (cit. on p. 82).