LUISS

Master degree in Data Science and Management

Department of Management

Course of Machine Learning

The Role of Machine Learning in Automatic Fake News Detection

Supervisor Italiano Giuseppe

Candidate Bozzi Martina Matr. 754721

Co-Supervisor Sinaimeri Blerina

Academic Year 2022/2023

Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Abstract

This thesis aims at employing machine learning algorithms in order to develop a tool capable of detecting fake news online. Indeed, AI models can become a powerful ally in the fight against proliferation of fake news, especially in today's information-rich society, where social media plays a crucial role in informing (and misinforming) users. The key characteristic, and advantage of AI-models is that they are fast and efficient when analysing huge amount of text data, therefore becoming a valuable tool for checking the veracity of news articles present online, which would otherwise be a computationally expensive effort if performed manually by people. The Cambridge dictionary defines fake news as false stories that appear to be news, spread on the internet or other media, usually created to influence political views or as a joke [1].

The rise of fake news has significant implications for society, as it can influence people's decision-making process, such as their voting behavior, purchasing decisions, and even have repercussions on public health (as it happened with COVID-19 vaccines).

This thesis project stands out for its innovative approach in creating a tool from scratch to combat the proliferation of fake news. It combines natural language processing techniques, and machine learning algorithms to train models able to statistically determine the veracity of news articles. What sets this work apart from the existing literature is its focus on the Italian language, as the dataset was created by web-scraping the most popular italian web-pages of news, which are responsible for informing the majority of users.

Indeed, even tough the literature on automatic fake news detection is very broad, the latter has primarily been conducted on English text data and with datasets already present online. The contribution of this thesis is to deeply examine the literature that has been conducted, and to apply it to non-English text data, overcoming the specific issues related to it. The dataset used to conduct this work is balanced, and presents 7108 news articles labeled as fake or real. The distinction between these two

[1] Procter (2000)

categories relies on the News-Guardian report, an independent organization that monitors information sites worldwide.

The thesis is structured as follows: the first and second chapter focus on topic modelling, and text pre-processing. Indeed, since the dataset was obtained through web-scraping it also presented a lot of noise which needed to be removed in order for the algorithms to perform better.

The second part of the analysis is more technical as it shifts the attentions to text classification by employing three different models: Random Forest, Naive Bayes and K-nearest neighbors. Once those models have been trained, the confusion matrix and performance metrics of each of said algorithms is shown and used for model evaluation later on.

Another notable part of this thesis is the incorporation of the BERT (Bidirectional Encoder Representations from Transformers) model. Taking inspiration from the groundbreaking paper published by Google in 2020, the BERT model was employed to tackle the task of fake news detection. BERT, being a state-of-the-art transformer-based model, has demonstrated exceptional performance in various natural language processing tasks. The model used is BERT BASE the uncased version; which is less computationally expensive and performs better on small-medium size datasets compared to the BERT LARGE version.

The final part of the thesis explores the role of Chat GPT-3 (Generative Pre-trained Transformer 3) in the context of fake news detection. While previous sections focused on algorithmic approaches and machine learning models, this part shifts the focus towards the use of advanced language models from an ethical perspective.

Indeed, Chat GPT-3 is a powerful language model developed by OpenAI. Its unprecedented ability to generate human like text rises concerns regarding two key aspects: malicious manipulation, meaning the risk of people exploiting gpt3 to increase the proliferation of fake news present online; and bias which is generally a concern that needs to be addressed when creating any type of algorithm, but that is enhanced as Chat Gpt3 was trained on huge amounts of text data. Indeed,research shows that large datasets are more likely to favor and represent a niche of the actual population such as young users, English-speakers, and people from developed countries. Nevertheless, there are actions that could be undertaken to mitigate those aspects, such as enhancing digital literacy and increasing accountability and transparency of developers behind said algorithms.

Contents

Chapter 1

Introduction

1.1 The importance of automatic fake news detection

This thesis aims at employing machine learning algorithms in order to develop a tool capable of detecting fake-news present online. Indeed, in today information rich society, in which social media play a crucial role in informing (and dis-informing) the users; AI becomes a useful ally in the fight against fake-news proliferation. According to the Cambridge dictionary, fake-news can be defined as ”false stories that appear to be news, spread on the internet or using other media, usually created to influence political views or as a joke” [1].

Fake news can take many forms, including false news stories, political propaganda, and hoaxes.

The rise of fake news has significant implications for society, since it can influence people decision-making process. For example, fake news that are spread out during elections time can influence people's voting behavior, or false stories about a product or service can affect their purchasing decisions. Furthermore, the uncontrolled spread of fake news could also have significant repercussions on public health. Indeed, this was the case during COVID-19 when media and news web-sites led people into making uninformed decisions about vaccines and treatments, and had a great impact on people's confidence level in vaccines. Automatic fake news detection becomes therefore very important, as it provides a way to identify and mitigate the spread of fake news in real-time. Moreover, AI-developed tools could enhance accountability and transparency of news since these tools may be used to remove fake news from the internet and make the authors accountable.

[1] Cambridge Dictionaries Online, (2007)

1.2 Problem statement, and research question

The work presented in this project aims at creating a tool against fake-news proliferation by employing italian news published online in order to train AI models, which will determine the veracity of said news. Indeed, even though the literature on automatic fake news detection is rich; the innovation of this project stands in developing from scratch a tool able to work with italian text data, combining both natural language processing techniques, and machine learning algorithms (supervised and unsupervised). The dataset used for the analysis has been obtained through web-scraping; which is a technique leveraged to extract information present in HTML pages. In this way a labeled dataset is stored, in which the title, the content, and the source of said articles is present.
The distinction between fake and real news relies on the News-Guardian report, the independent organization that monitors information sites around the world, and which has tried to draw up these two difficult rankings (fake and real). Among the websites that have been declared as most unreliable in the italian scenario, we can find:

- Il primatonazionale.it, journal connected to the neo-fascist movement of Casa Pound;
- Scenari-economici.it, an information site with right-wing political positions, which publishes content in support of its nationalist and anti-immigration positions;
- Imolaoggi.it an information site with right-wing political positions, which publishes content in support of its nationalist and anti-immigration positions,
- Voxnews.info, a site that ”regularly publishes false information in support of its anti-immigration agenda. The site also published misinformation about the vaccine for COVID-19 and the presidential elections in the United States”.

On the opposite side, a list of ”most reliable websites” was drawn up , containing:

- Open.online, which deals with national and international news, designed to correctly inform millennials,
- Ilsole24ore.com, which mainly focuses on finance and economy,
- Ansa.it, the first multimedia information agency in Italy, and with a high ranking world-wide.

Taking this information into account, a specific dataset containing news content from these websites has been created, and used as basis in order to train classical machine learning algorithms such as KNN, Random forest and Naive Bayes. Subsequently, the same dataset will be employed in order to train BERT model, taking as reference the paper published by Google in 2019 ”BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”.

1.3 Literature Review and State of the Art

Despite the richness of literature present in the field of automatic fake news detection, a common agreed upon benchmark has not yet been reached.
In this section, the current state of the art, previous work and literature inherent to this thesis will be shown. Indeed, it is believed that one of the main limitations of this field of research, stands in the fact that most studies have been carried out relying on English text data. Among those, the most commonly used datasets are:

- FakeNewsNet: this is a public dataset which contains Tweets and articles collected from a selection of news websites. The dataset is labeled, and the labels indicate whether the content is real or fake,[2],
- The LIAR dataset: this dataset was created by the William and Mary Department of Computer Science and contains approximately 12,000 statements labeled as true, false, or misleading. The statements come from POLITIFACT.COM's API, and each statement is evaluated by a POLITIFACT.COM editor for its truthfulness. [3],
- COVID-19 dataset: containing data related to COVID-19 news articles and their labels indicating whether they are real or fake.

Some notable work in Italy has been carried out by researchers at universities like the University of Pisa and Milan Polytechnic which are active in detecting fake news. These studies aimed at developing different types of fake news detection systems in the Italian language, such as sentiment analyses, fact checking, analysis of propagation trends on social media and other techniques.
Nonetheless, fake news detection remains a field in which there is a lot of space for improvement in terms of accuracy and robustness of algorithms, furthermore the rapid evolution of language models makes it a constantly evolving field.

[2] Shu et al. (2020)
[3] Hendrixwilsonj (2021)

As said above, previous work in the field include also models trained in the italian language such as:

- Machine learning-based approaches: Support Vector Machines (SVM), Naive Bayes , and Convolutional Neural Networks (CNN) are some of the models used for this classification task. These studies revealed that certain features of datasets facilitate the task of fake news detection such as the title, the source, text description and language elements like references to several keywords,

- Linguistic features: other studies used AI tools to investigate linguistic features of text data such as sentiment analysis, named entity recognition, and text readability to detect fake news. Indeed, these features giving a deep insight on the tone and style of the artcile help in the distinction between fake and real,

- Multi-modal approaches: these line of research integrate text and visual representation (images) in news articles. For instance, coherence between image and text is investigated in order to determine if the information is trustworthy,

Nonetheless, it can be argued that research carried out in a non-English language enriches the existing literature by adding diversity in perspective, as it can offer unique insights when solving language-related problems.

1.4 Thesis Outline

The outline of this thesis is the following:

- Chapter 2 describes the methods, objective and the dataset employed for the analysis;
- Chapter 3 explains the ”classical” machine learning tools deployed to train the dataset in the achievement of fake news classification, and exposes the results achieved through the latter;
- Chapter 4 describes the theoretical approach that lies behind the BERT model developed by Google, and applies it to the dataset of this research;
- Chapter 5 analyses the potential risks related to GPT-3 from an ethical perspective in terms of manipulation and bias.

Lastly, in the conclusions the results achieved during the analysis, and the challenges encountered are exposed.

Chapter 2

Fake News Detection

2.1 Algorithmic approach to automatic fake news detection

From an algorithmic perspective, referring to fake news detection is essentially a classification problem. Indeed, the challenge is to classify the data present in the dataset, and make the algorithm decide whether it is reliable or not. In order to achieve this result, the data needs to be cleaned in a way that the algorithms can recognize and learn from patterns in it , and achieve classification goal.

The process of making algorithms understand the linguistic context of text data is obtained through Natural Language Pre-processing techniques, an evolving branch of AI which focuses on the interaction between computers and the human language.

Natural language pre-processing is a crucial step in the development of any natural language processing (NLP) tasks, or text analytics system. This involves a series of operations to be carried out in order to clean, normalize and prepare text data for further analysis. Pre-processing aims at transforming plain text into a form that is easy to comprehend, and which NLP algorithms are able to process.

One of the key aspects of pre-processing is text cleaning, which involves removing any irrelevant or distracting information from the text. This might include removing special characters, numbers, HTML tags, or punctuation marks that are not essential for the understanding of the text. Additionally, text cleaning may involve converting all text to lowercase, which helps to reduce the dimensionality of the data and simplify the analysis process.

Another important part of pre-processing is tokenization of the data, which involves dividing the document into smaller units, such as words or phrases. This is typically done using regular expressions or NLP libraries, such as NLTK or spaCy(as in this case that were both used). Tokenization is impor-

tant because it allows for the separation of individual words or phrases in the text (tokens), making it easier to identify patterns and relationships between words.

In addition to cleaning and tokenizing the text, pre-processing also includes stemming or lemmatization. Stemming is a process that consists of removing suffixes from words to obtain their root form. For example, the word “running” might be stemmed to “run”. Lemmatization is similar to stemming, but it takes into account the context of the words and the intended meaning. This helps to preserve the meaning of the words, even when they are reduced to their root form.

Pre-processing also involves removing stop words, which are common words that do not contribute to the meaning of the text, such as “and”, “the”, and “a”. Removing stop words helps to reduce the dimensionality of the data and simplify the analysis process.

Finally, pre-processing also allows the user to create numerical representations of the text data, such as word embeddings or bag-of-words representations. The latter are then given as input to NLP algorithms, which will pick up on the patterns in text data.

Lastly, after cleaning, normalizing and trasforming tezt data, pre-processing makes it possibile for NLP algoirthms to get a clear ”understanding” of the patterns in the data; therefore enabling a more accurate analysis. In the following chapter, pre-processing is performed on the dataset with the objective of both cleaning the data, and to extract relevant information from it.

2.2 Libraries

The current analysis has been carried out using Python and its related libraries. Specifically, three different virtual environments have been created: one for natural language pre-processing tasks, one for the implementation of classical machine learning algorithms, and one implementing and fine-tuning BERT.
In the following lines, a description of the libraries and their usage is given:

Pandas

Pandas is an open source Python package that is primarily used for data science, data analysis, and machine learning tasks. It is built on top of Numpy, a different package that offers support for multi-dimensional arrays [1]. With Pandas, users can easily and effectively perform operations like filtering, aggregation, transformation, and cleaning of data. Additionally, Pandas integrates well with other well-known data analysis libraries Additionally, Pandas integrates well with other popular data analysis libraries, such as NumPy and Matplotlib, making it an essential tool for data analysis and manipulation tasks in Python [2].

BeautifulSoup

Beuatiful Soup is a Python library for pulling data out of HTML and XML files.
It is designed to make it easier to extract data from websites, by providing convenient and efficient ways to search, navigate and modify the structure of HTML and XML content. Beautiful-Soup provides several methods for searching and navigating a document, including search by tag name, attribute value, or CSS class [3]. For the purpose of this analysis, this library was essential in order to scrape data from online news articles web-sites in order to create the dataset.

Requests

Requests allows you to send HTTP/1.1 requests in an easy yet efficient way, as there is no need to manually add query strings to your URLs.
Additionally, the library is particularly useful when working with APIs, as it makes it easy to send

[1] ActiveState, (2022)
[2] PyData , n.d.
[3] Beautiful Soup Documentation — Beautiful Soup 4.4.0 Documentation, n.d.

requests and receive responses in a structured format such as JSON. Additionally, it provides a variety of features for handling errors, cookies, sessions, and other aspects of web communication. [4]

Scikit-learn

Scikit learn is a free, open-source library in Python which contains a selection of algorithms for classification, regression, clustering, dimensionality reduction, and model selection, among others, and is known for its user-friendly API, ease of use and speed[5]

NLTK

NLTK is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces such as WordNet, along with several text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning [6]. In this analysis, the nltk library was employed in order to perform pre-processing tasks and to clean the data.

[4]Requests: HTTP for HumansTM — Requests 2.31.0 Documentation, n.d.
[5]Scikit-learn: Machine Learning in Python — Scikit-learn 1.2.2 Documentation, n.d.
[6]NLTK: Natural Language Toolkit, n.d.

WordCloud

WordCloud is a library in Python which use is to create Word-Clouds, that is to say visual representation of words' frequency in a text corpus [7].A set of words is given as input and the output image will show the words present in the corpus, with the size being the representation of the frequency of each word. This is useful because the generated word clouds can provide a quick and intuitive summary of the most frequently occurring words in a large text document. In the analysis it has been useful in order to highlight the main topics of the datasets under analysis.

Spacy

Spacy is a popular open-source library used for natural language processing (NLP) tasks. A set of tools useful to analyze and pre-process text data are present in this library, and specifically for this thesis the italian extension "it_core_news_sm" was used, which is a pre-trained statistical model for Italian language designed to perform a variety of NLP tasks, including tokenization, POS tagging, named entity recognition, and dependency parsing [8].

Transformers

Transformers provides APIs and tools to easily download and train state-of-the-art pretrained models. Using pretrained models can reduce your compute costs, carbon footprint, and save you the time and resources required to train a model from scratch [9].

Gensim

This library was used to buil the LDA model, since[10] the algorithms behind it automatically discover the semantic structure of documents by examining statistical co-occurrence patterns within a corpus of training documents. These algorithms are unsupervised, which means no human input is necessary – you only need a corpus of plain text documents [11].

[7] WordCloud for Python Documentation — Wordcloud 1.8.1 Documentation, n.d.
[8] Doc spaCy API Documentation, n.d.
[9] Transformers, n.d.
[10] https://radimrehurek.com/gensim/intro.html
[11] Gensim: Topic Modelling for Humans, n.d.

2.3 Dataset

The first part of the analysis focused on the creation of the dataset, using web-scraping techniques. Web scraping is a technique for extracting information from websites, consisting in making HTTP requests to a website's server, and downloading the HTML content of the web page. Given that the data was scraped by the internet , it presented a lot of impurities as html tags that subsequently needed parsing in order to to extract the information of interest.

This section will be focused on explaining the code used in order to retrieve the data. Since the structure of the code is the same for all the web-pages, just one of them will be included for clearness purposes:

```
1
2  url = "https://www.imolaoggi.it/category/polit/"
3  soup = BeautifulSoup(requests.get(url).content, "html.parser")
4
5  all_data = []
6  for title in tqdm(soup.select("h3.entry-title")):
7      t = title.get_text(strip=True)
8      u = title.a["href"]
9      s = BeautifulSoup(
10         requests.get( u).content, "html.parser"
11     )
12     text= s.find('div',{'class':'entry-content'}).text.strip()
13     text = s.select_one('[div="post-body entry-content float-container"]')
14     text = text(strip=True, separator="\n") if text else ""
15
16     all_data.append([t, text, u])
17
18
19 df = pd.DataFrame(all_data, columns=["Title", "content", "PageLink"])
```

The first line of the code specifies the URL of the web-page of interest. Through the library "Beautiful-Soup" a request of access to the web-page was made, and then an empty list is created in order to appended the parts of interest to include in the dataset.

Indeed, in this case the targeted parts of the article are the title (title.get_text),the link of the web-page (href tag), and the corpus (which is accessed through the href). Finally, the empty list is

converted into a DataFrame.
Afterwards, a new column called category is added, and based on the content of the article the variable takes as value:

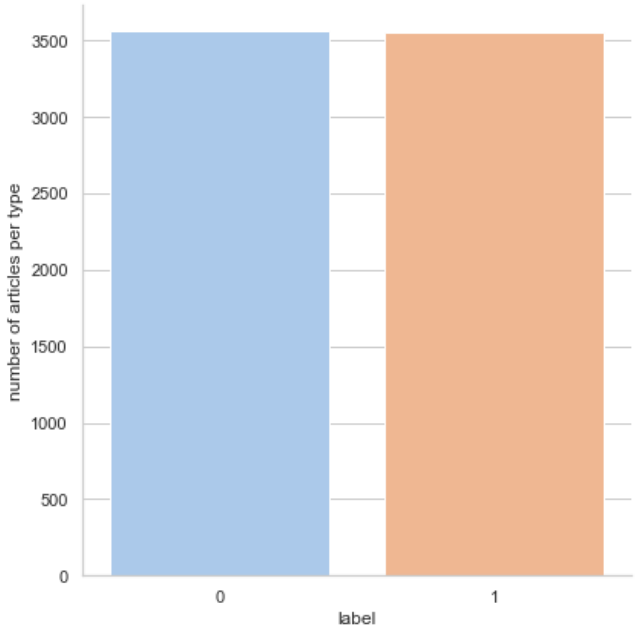
- Politics: online news related to politics and political events,
- World : online news related to the latest worldwide events,
- Economy: online news related to economical topics,
- News section: online news related to national news events,
- Science: online news related to science,
- Covid: online news related to covid and vaccines.

At the end of this proceeding, two datasets are obtained: one containing the reliable news , and one containing the fake news. Before concatenating the two dataframes, a variable called ”label” is added, which takes value 0 in case of fake news, and 1 in the presence of a real news. Finally the two datasets (fake and real) are concatenated together into one.
The final dataset contains 7108 observations, and 7 variables all categorical which are:

- Title: it refers to the tile of the article ,
- Content: it refers to the content of the article ,
- PageLink: it refers to the URL of the article
- Date: it refers to the date in which the article was published,
- Category: it refers to the category of the article mentioned above,
- Source: it refers to the name of the website by which the article was scraped,
- Label: it refers to the label of the article indicating whether it is a real of fake news.

The graph below shows the graphical distribution of the label variable present in the dataset. From the latter it can be observed that it is balanced dataset, presenting 3557 labeled as fake, and 3551 labeled as true.

Figure 2.1: Label distribution of dataset



2.4 Natural Language Pre-processing

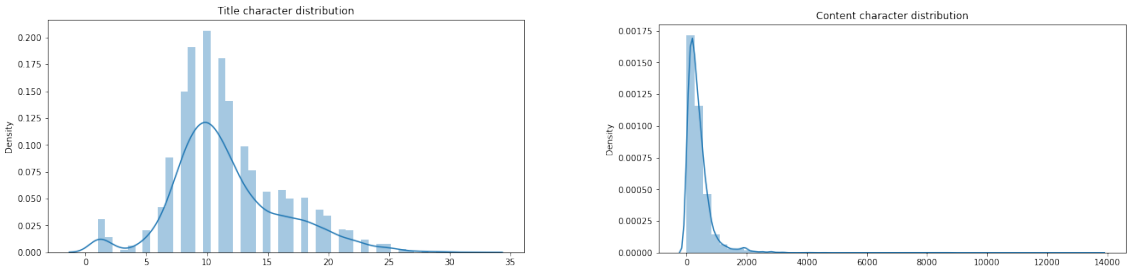
2.4.1 Data cleaning

Natural language pre-processing is a branch of machine learning that works with text-data. The goal is to enable computers with an understanding of languages that mimics the one of humans.
As said above, since the dataset used in this thesis is the result of web-scraping; it presented a lot of ”impurities” (links, emoticons, urls) which needed to be removed for a better performance of algorithms.
In order to do that, a pipeline of function was created. Text processing included the following steps:

- Convert the corpus of the text to lower case,
- Remove punctuations,
- Remove stop-words: from nltk library a series of italian stopwords was imported and then removed from the text. Stopwords are words such as preopositions and articles, that recurrently appear in text data, but need to be removed since they do not represent a pattern in the data,
- Remove emojis,
- Remove URLs,
- Remove htlm text,

Once the text was cleaned, a further inspection was performed concerning the length of text data. It emerged that the title has a mean of 11 characters, whereas the content has a mean of 397 characters. This information will be of use when fine-tuning the BERT model (since it has a maximum length of characters of 512) and was graphically represented in the following way:

Additionally, in order to get an insight of the topic of the dataset a Word-Cloud , also known as a tag cloud or text cloud, was created. The latter is a visual representation of the most frequently occurring words in a given text data. The words are displayed as individual elements, typically as differently-sized words, with the size of each word proportional to its frequency in the data. Their usage is about quickly finding out the most significant themes or topics present in the text data.
For the current analysis, it was believed that Word-cloud could be more of impact by isolating the original dataset by category and label, and subsequently comparing the results.

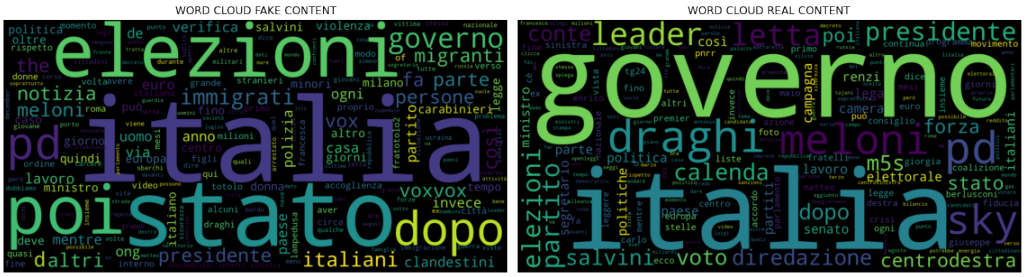


(a) title character distribution (b) content character distribution

Figure 2.2: Content and Title character distribution

In the picture below, the word clouds of political news for the real and fake datasets are represented:

Figure 2.3: Politics Fake and Real Word-cloud)



It is noticeable that while the two images present some words in common, such as ”italian”, and ”stato”; the most recurrent words in fake news articles of politics frequently include words like: meloni, immigrati, migranti; indicating that those are the main topics of those articles.

This difference becomes even more evident when analysing most recurring trigrams, which are sets of three consecutive tokens.

The graph below displays on the left the most recurring tri-grams of fake political news; and on the right the one of real news.

Figure 2.4: Politics Fake and Real Word-cloud with Tri-grams)

What emerges is the fact that once again in fake news data appear words and sentences that reflect hatred against communities and categories of people appear, such gas ”baby gang immigrati”, ”figli immigrati stuprano”, whereas on the other side real news use more neutral words when dealing with the same topics.

The last step of the pre-processing was stemming. Stemming is a technique used in Natural Language Processing to reduce words to their root form. The goal is to map related words to a common base form, so that words with similar meaning can be treated as the same token.

For example:

- ”running” can be stemmed to ”run”
- ”runner” can be stemmed to ”run”
- ”ran” can be stemmed to ”run”

In this way, different forms of the same word can be treated as the same term, reducing the dimensionality of the text data and improving the effectiveness of NLP tasks such as information retrieval or text classification.

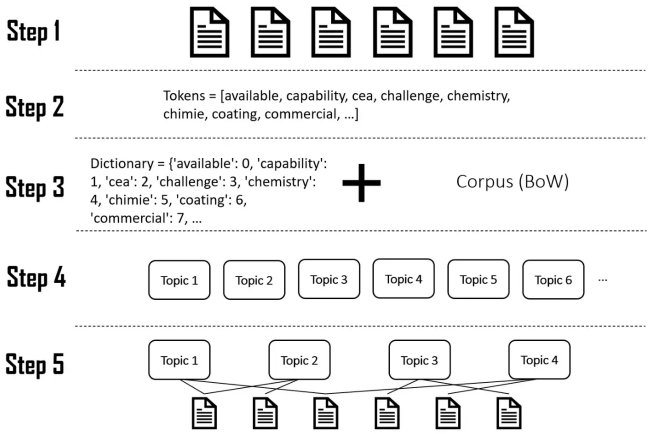
```
italian_stopwords = stopwords.words('italian')
stemmer = ItalianStemmer()
documents = list()
for k,v in enumerate(x):
    document = v.split()
    document = [stemmer.stem(word) for word in document]
    document = ' '.join(document)
    documents.append(document)
```

Since the text under analysis is in italian language, the italian extension of the Nltk python library for stemming was used.

2.4.2 Latent Dirichlet Allocation

In machine learning and natural language processing, a topic model is a type of statistical model for discovering the abstract "topics" that occur in a collection of documents [12]. It is based on the idea that each document in a corpus can be modeled as a mixture of topics, where each topic is characterized by a probability distribution over a set of words. The functioning of the algorithm can be explained with the picture below:

Figure 2.5: Picture from Towards Data Science[13]



As shown above, the algorithm presents a series of step [14]:

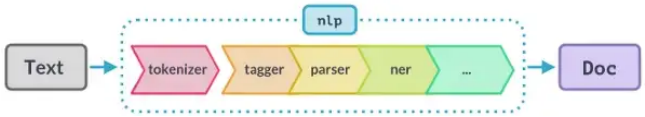
1. Collection of documents,
2. Cleaning and tokenization of data,
3. Creation of a dictionary based on the tokens present in the input documents,
4. Definition of a range of topics and reasoning of the optimal number of topics,
5. Find the distribution of topics in the document.

LDA has many applications, including text classification, information retrieval, and data visualization. It is also used for feature extraction and dimensionality reduction, where the learned topics can be used as features for further analysis. In general, LDA is an effective tool for uncovering the hidden topics present in large collections of text data and can provide valuable insights into the structure of the data. Since Spacy library offers multi-language models, and this

[12] Doll T., (2018)
[14] Ghanoum T., (2021,December,20)

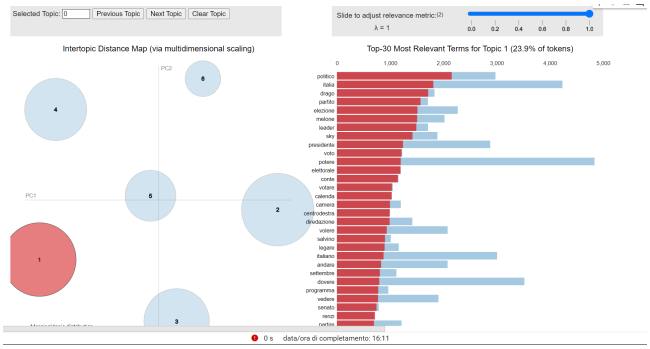
model will be trained on italian text data the first step is to import a pre-trained model called ”it_core_news_sm”. The Spacy pipeline works in the following way:

Figure 2.6: Picture from Towards Data Science[15]



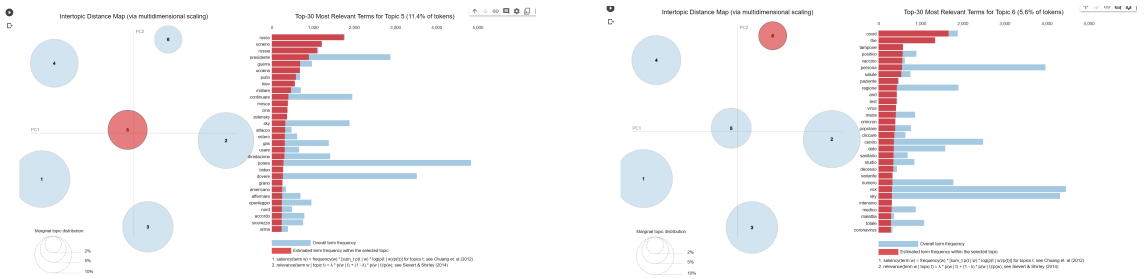
After tokenizing the data, a POS (part of speech) tagger is used. This is interesting since the tagger is a computational tool that labels each word in a text corpus with its corresponding part of speech; which combined with LDA can be used to extract meaningful information from text data.
Once the text is ready, it is essential to create a dictionary. The dictionary contains each token (character) of the corpus which is assigned to a unique id , and together with the corpus they will be the two inputs of the model. The corpus contains each token, and its frequency of appearance.
A crucial part in LDA is the pre-choice of the number of topics. In this case, since the dataset presents seven categories, a number of 6 topics will be assigned. Subsequently the lda model is fitted and its results plotted using pyLDAvis library.

Figure 2.7: LDA graphical representation using pyLDAvis library



The above chart represents the six topics as circles. Those circles are expected to not overlap (as in this case), implying a correct identification of the topics .
In detail, in the interactive graph when a circle is selected different words are displayed on the right, showing word frequency (blue) and estimated term frequency within the selected topic (red). Topics closer to each other are more related.

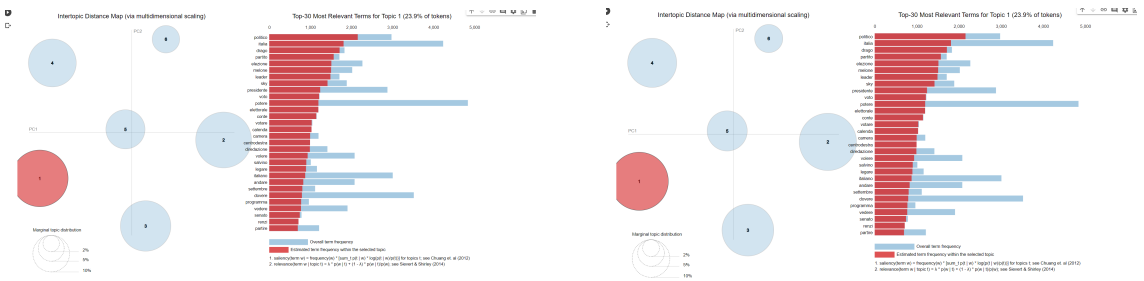
What emerges from the graph is the fact that topics 5 and 6 (which are believed to be the ones originally labeled as covid and world) are represented by a smaller circle, meaning that they present a smaller number of observations compared to other topics (as indeed it is). Topic 5 represents news about war events since the most recurring words are ”russo”, ”ucraino”, ”presidente”, ”guerra”, ”ucraina”, ”putin”. Topic 6 is believed to be about science, since its most frequent words are ”covid”, ”tampone”, ”vaccino”, ”positivo”, ”persona”, ”salute”.
Topic 1 and 4 represent political news and chronicle news, but it is believed that given the frequency



(a) Topic n.5 (b) Topic n.6

Figure 2.8: LDA model topic 5 and 6

of words in group 4 there can be found fake-news topics, as the most frequent words to appear are: ”seguire”, ”immigrato”, ”violenza”,”clandestino”, ”carabiniere”.

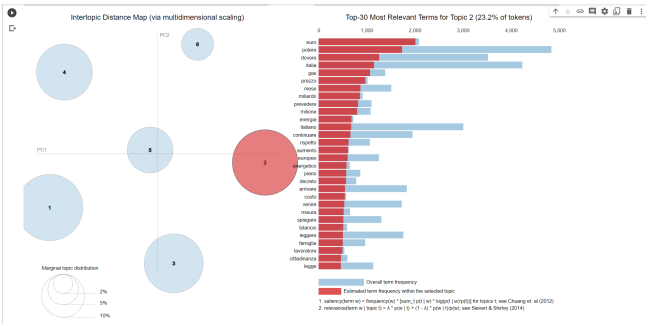


(a) Topic n.1 (b) Topic n.4

Figure 2.9: LDA model topic 1 and 4

Analysing group 2 and its frequency of words it is believed that this one is the topic related to economy, given words such as ”euro”,”potere”, ”gas”, ”prezzo”, ”energia”, ”prevedere”.

Figure 2.10: Topic n.2



Chapter 3

Classical machine learning algorithms

In this chapter three machine-learning algorithms will be trained in order to perform fake news classification. Later on, model evaluation will be performed by comparing the metrics of each algorithm. Subsequently, those same results will be compared also with the ones obtained by training the BERT model.

The first part of this analysis focused on creating three functions before fitting the models, which make the process less computationally expensive. Indeed, in this way it is only necessary to change the classifier, and the related results are then displayed in the same way for all models. Moreover, for each model the original dataset is split with a percentage of 80% for the train set, and 20% for the test set.

```
def train_model(model, x_test, y_train):
    model.fit(x_train, y_train)
    return model

def predict_model(model, x_test):
    y_pred = model.predict(x_test)
    return  y_pred

def evaluate_model(y_test, y_pred):
    print("----------- Confusion martix -----------------\n")
    print(confusion_matrix(y_test,y_pred))
    print("\n----------- Report ----------------- \n")
    print(classification_report(y_test,y_pred))
    print("\n----------- Accuracy -----------------\n")
```

```
15    print(accuracy_score(y_test, y_pred))
```

In detail:

1. train_model function takes as input three arguments: ”model”, which is the model to be trained, ”x_train” which is the dependent variable, and ”y_train” which is the target variable for the training data, in this case the label.

2. predict_model function takes in input two arguments: the ”model” which refers to the trained machine learning model, and ”x_test” which are the feature variables for the test data. The function uses the ”predict” method to make predictions on the test data and stores the predictions in the ”y_pred” variable.

3. evaluate_model function takes in two arguments: ”y_test” which is the target variables for the test data, and ”y_pred” which refers to the predictions made by the machine learning model. The function performs several evaluations on the model’s performance. It starts by printing the confusion matrix, which shows the number of correct and incorrect predictions for each class. Next, the function prints a classification report, which includes precision, recall, and f1-score for each class. Finally, the function prints the accuracy score, which is a scalar metric that gives an overall evaluation of the model’s performance.

3.1 Performance metrics in classification

The performance of each algorithm is assessed through a series of metrics that will be explained below. Since those metrics are used for all the algorithms deployed, the latter will be explained just once.

Therefore, it is important to remark that in classification alogirthm's erorrs can be categorized into:

- Type I Errors: False Positives, observations which are classified as positives but in reality they are truly negative;
- Type II Errors: False Negatives, individuals classified as negatives that are positive in reality [1]

Finally, in order to evaluate the performance of each model, a series of metrics is taken into account. Indeed, when evaluating the performance of a model which performs classification tasks it is common to based the evaluation on indicators such as accuracy, precision, recall, F1 score. In detail:

Accuracy

The accuracy metric is a scalar value that measures the relative number of accurate predictions over the total number of instances evaluated[2]. The formula for accuracy is:

$$Accuracy = (TruePositives + TrueNegatives)/TotalPredictions \tag{3.1}$$

Where True Positives are the number of instances correctly classified as positive, True Negatives are the number of instances correctly classified as negative, and Total Predictions is the total number of predictions made.

Precision

Precision is used to measure the positive patterns that are correctly predicted from the total predicted patterns in a positive class [3]. Precision is particularly important in cases where the cost of false positive predictions is high. The formula for precision is:

$$Precision = TruePositives/(TruePositives + FalsePositives) \tag{3.2}$$

Where False Positives are the number of instances classified as positive, but are actually negative.

[1] Chen (2021)
[2] Hossin and Sulaiman (2015)
[3] Ibidem

Recall

Recall is the ability of a model to identify relative cases within a dataset [4]. In detail, recall is used to measure the fraction of positive patterns that are correctly classified as positives [5].

$$Recall = TruePositives/(TruePositives + FalseNegatives) \tag{3.3}$$

F1-score The F1-score is an harmonic mean that takes into account both precision and recall, in the following formula:

$$F1Score = 2 * (Precision * Recall)/(Precision + Recall) \tag{3.4}$$

This evaluation metric in important since it gives equal wight to both measures, and is a specific example of the general F_β metric, where $_\beta$ can be adjusted to give more weight to either recall or precision [6].

[4] Koehrsen (2021)
[5] Hossin and Sulaiman (2015)
[6] Koehrsen (2021)

3.2 Random Forest

Random Forest is a supervised machine learning algorithm that could be employed both for regression and classification tasks. In this case, the latter is deployed for classification since the target variable of the dataset is binary and categorical, taking values of either 0 in case of true news; and 1 in the presence of fake news. Supervised learning means that the algorithm is trained on labeled data , and the goal is for the algorithm to predict the label of unseen data during testing.

Random Forest uses the ensemble method. Indeed, in statistics and machine learning "ensemble method" refers to the usage of multiple algorithm, that together outperform the results achievable by the single one. In this case Random Forest represents an ensemble of decision trees, that are merged together in order to produce more accurate predictions [7]. In general, it can be affirmed that in classification the performance of the algorithm increases when increasing the number of trees; but this method lacks generalization since other elements need to be taken into account such as the size of the dataset.

In the context of classification, each decision tree in the Random Forest algorithm is trained on a random subset of the data and a random subset of the features. The decision tree predicts the class label for a given data point by traversing the tree and arriving at a leaf node. At each internal node of the tree, the algorithm splits the data based on the feature that provides the maximum reduction in impurity. The impurity is typically measured using Gini impurity or entropy. Once the decision trees are trained, the Random Forest algorithm makes a prediction for a new data point by aggregating the predictions of all the trees. The prediction is typically made by taking the majority vote of the trees.

In mathematical terms, the prediction made by a Random Forest for a new data point "x" can be represented as:

$$\hat{y} = 1/n * (y_i) \tag{3.5}$$

In detail, the model has been fitted using Random Forest classifier by employing the functions described above in a pipeline that has the following structure:

```

from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators=1000, random_state=0)
classifier = train_model(classifier, x_train, y_train)
```

[7] Meltzer (2021)

```
y_pred = predict_model(classifier, x_test)

evaluate_model(y_test, y_pred)
```

The ”n_estimators” parameter in the code sets the number of decision trees in the Random Forest Classifier. The default number of parameters in Python is 100, but in this case a number of 1000 has been chosen. Indeed, being Random Forest an ensemble method, the larger the number of trees, the more diverse and robust the model becomes, each tree giving a different insight on the data. For example, setting ”n_estimators” to 1000 as in this case means that when making predictions, the algorithm will use all of the 1000 trees to make a prediction, and the final prediction will be the majority vote of all the trees. This allows the Random Forest Classifier to average out the biases and variances of the individual trees, resulting in a more stable and accurate prediction [8]. The ”random_state” parameter in the code controls the randomness of the algorithm. Setting the ”random_state” to a fixed value (in this case, 0) ensures that the same results will be obtained each time the code is run, provided that the input data and the code remain unchanged.

[8] Ellis (2022)

3.2.1 Results of Random Forest Classifier

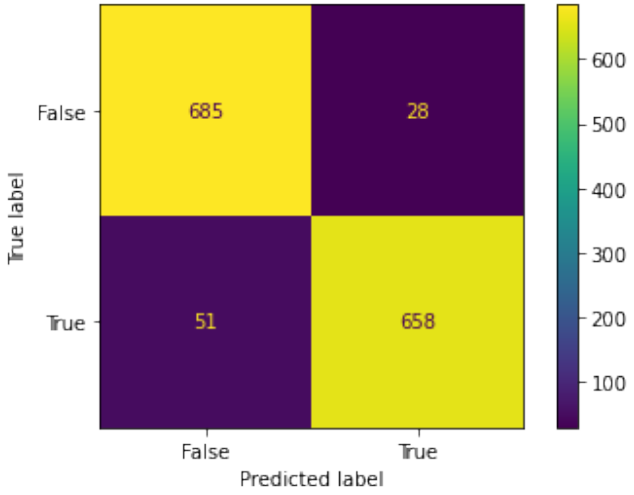
In this case the performance of the Random Forest algorithm can be represented in the following way:

Table 3.1: Evaluation metrics for Random Forest classifier

	Precision	Recall	F1-score	Support
0	0.93	0.96	0.95	713
1	0.96	0.93	0.94	709
Accuracy				0.94
Macro avg	0.94	0.94	0.94	1422
Weighted avg	0.94	0.94	0.94	1422

The table above includes five columns. The first column indicates the classes (0 and 1) for which the evaluation metrics are reported. The second, third, and fourth columns show the precision, recall, and F1-score for each class, respectively. The precision measures the proportion of true positives among all positive predictions, the recall measures the proportion of true positives among all actual positive instances, and the F1-score is a combined measure of precision and recall. The fifth column shows the number of instances in the test dataset belonging to each class. Finally, the table also reports the overall accuracy, which is the proportion of correct predictions over all instances in the test dataset. The table also provides macro and weighted averages of the precision, recall, and F1-score across both classes. The macro average is an unweighted mean of the evaluation metrics, whereas the weighted average takes into account the number of instances of each class in the test dataset.
Lastly, the confusion matrix is plotted:

Figure 3.1: Confusion Matrix for Random Forest Classifier



The figure above displays in the yellow cells the correct predictions which amount to 685 for the fake news correctly classified as false, and 658 for the true news correctly classified as true. The purple blocks represent the number of incorrectly classified news , respectively 51 predictions that were true and have been labeled as false, and 28 which were false are were classified as true.

3.3 Naive Bayes

Naive Bayes is a probabilistic machine learning algorithm based on the Bayes theorem. It's named "Naive" because it makes a strong assumption that all the features in the input data are independent of each other [9]. Overall, Naive Bayes algorithm are often employed for classification tasks since they have a good performance on textual analysis tasks such as automatic text detection, spam filtering , sentiment analysis.

There are three main variants of the Naive Bayes algorithm:

1. Gaussian Naive Bayes, which assumes that the features follow a normal distribution. It's mainly used for continuous data,

2. Multinomial Naive Bayes, which is used for discrete data such as text. It models the probabilities of observing the different outcomes (i.e., words) in the data,

3. Bernoulli Naive Bayes, which is similar to the multinomial Naive Bayes, but it's used for binary data where each feature can only take on two possible values (e.g., true or false)

Taking in consideration the characteristics of the dataset, the Multinomial Naive Bayes model has been chosen as classifier. This model is a good alternative to "heavy-AI" machine learning algorithms due to its performance, and simplification on text classification. The goal is to use this model to predict the class label of a new instance based on its features. The basic idea is to calculate the posterior probability of each class given the features of the new instance, using Bayes theorem [10]. In the case of multinomial Naive Bayes, the posterior probability is calculated as follows:

$$P(class|features) = \frac{P(features|class) * P(class)}{P(features)} \tag{3.6}$$

where

- $(class|features)$ is the posterior probability of class given the features of the new instance,
- $P(features|class)$ is the likelihood, which models the probability of observing the features given the class,
- $P(class)$ is the prior probability of the class,

[9] Huang and Li (2011)
[10] Ratz,(2022)

- $P(features)$ is the marginal likelihood, which is a normalization term to ensure that the posterior probabilities sum up to 1.[11]

In the case of the Multinomial Naive Bayes, the likelihood is modeled as the product of multinomial distributions for each feature, assuming that each feature is independent given the class:

$$P(features|class) = \prod_i P(feature_i|class) \tag{3.7}$$

where

- $\prod_i$ is the product over all features i,
- $P(feature_i|class)$ is the probability of observing the value of feature i given the class.[12]

With the use of maximum likelihood estimation, these probabilities can be calculated from the training data. Lastly, the class with the highest posterior probability is taken as reference to make the prediction for a new instance. The implementation of the model is as follows:

```
nb_classifier = MultinomialNB()
classifier = train_model(nb_classifier, x_train, y_train)
y_pred = predict_model(classifier, x_test)
```

[11] Loukas (2023)
[12] Ibidem

3.3.1 Results of Naive Bayes

The result of the algorithm is represented in the table below:

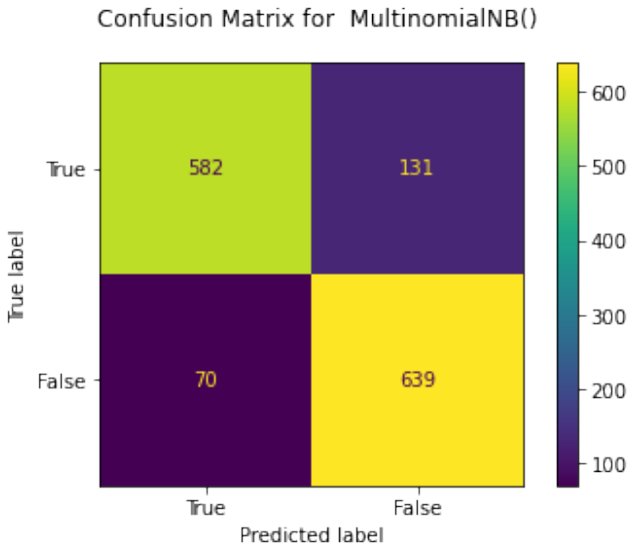
Table 3.2: Evaluation metrics for Naive Bayes classifier

	Precision	Recall	F1-score	Support
0	0.89	0.82	0.85	713
1	0.83	0.90	0.86	709
accuracy				0.86
macro avg	0.86	0.86	0.86	1422
weighted avg	0.86	0.86	0.86	1422

From the accuracy metrics, it is noticeable that the precision of the model is higher for observations belonging to class 0 (0.89) rather then class 1 (0.83). This changes when analysing the Recall, indicating its ability to identify a higher proportion of true positive samples belonging to class 1 compared to class 0. Finally, the F1 score being similar for both classes indicates that it performs equally well for both.

The confusion matrix of Naive Bayes Classifier has the following structure:

Figure 3.2: Confusion Matrix for Naive Bayes Classifier



The confusion matrix shows that the model predicted 582 instances as belonging to class 0 when the true label was also class 0. This is called true negatives (TN), which means that the model correctly identified 582 instances as negative for class 1. However, the model incorrectly predicted 131 instances as belonging to class 1 when the true label was actually class 0. This is called false

positives (FP), which means that the model incorrectly identified 131 instances as positive for class 1. Similarly, the model predicted 639 instances as belonging to class 1 when the true label was also class 1. This is called true positives (TP), which means that the model correctly identified 639 instances as positive for class 1. However, the model incorrectly predicted 70 instances as belonging to class 0 when the true label was actually class 1. This is called false negatives (FN), which means that the model failed to identify 70 instances as positive for class 1.

Overall, the confusion matrix shows that the model made 131 false positive predictions, which means that the model predicted some instances as positive for class 1 when they were actually negative for class 1. Similarly, the model made 70 false negative predictions, which means that the model predicted some instances as negative for class 1 when they were actually positive for class 1. On the other hand, the model made 582 true negative predictions and 639 true positive predictions.

3.4 K-nearest neighbors

K-Nearest Neighbors (KNN) is a supervised learning algorithm that can be used for both classification and regression tasks [13]. When given a new observation, the algorithm will predict its class based on the distance between other datapoints [14].

For classification tasks the formula of the algorithm is the following:

$$y = \arg\max_{j \in C_k} \sum_{i=1}^{K} w_i \mathbb{I}(y_i = j) \tag{3.8}$$

where

y is the predicted class, C_k is the set of possible classes, w_i is the weight assigned to the i-th neighbor, and $\mathbb{I}(y_i = j)$ is the indicator function that is equal to 1 if the i-th neighbor has class label j, and 0 otherwise. The weights w_i can be uniform (i.e., $w_i = 1$ for all neighbors) or distance-based (i.e., $w_i = \frac{1}{d(x_i, x_q)}$ where $d(x_i, x_q)$ is the distance between the i-th neighbor and the query point)[15].

Usually, the distance between the data points is calculated using either Manhattan, Euclidean and Hamming distance. The choice of the distance metric depends on the specific dataset and problem characteristics; and in this case the Manhattan distance has been employed. Indeed, the latter is the distance between real vectors using the sum of their absolute difference. In formula, the Manhattan distance can be written as:

$$d(x_i, x_j) = \sum_{k=1}^{n} |x_{ik} - x_{jk}| \tag{3.9}$$

and calculates distance between two points in the feature space as the sum of the absolute differences of their corresponding coordinates[16].

The Manhattan distance is less sensitive to outliers in the data, and is often used when the features are not continuous.

[13] Cristopher (2021)
[14] Gupta (2022)
[15] Lecture 2: K-nearest Neighbors / Curse of Dimensionality, n.d.
[16] Christopher (2021)

The code used to train the model is the following:

```
knn_classifier = KNeighborsClassifier(n_neighbors=2, metric="manhattan")
classifier = train_model(knn_classifier, x_train, y_train)
y_pred_k= predict_model(classifier, x_test)
```

where:

- n_neighbors=2 represents the number of nearest neighbors considered by the K-nearest neighbors (KNN) algorithm when making predictions. In this case,the KNN algorithm will consider the two closest neighbors to the test data point. The class label or target value of these two neighbors will then be used to make a prediction for the test data point. The majority class label among the neighbors will be assigned to the test data point as its predicted class,
- metric=”Manhattan”, as explained above is the distance metric used to calculate the distance between data-points.

3.4.1 Results of K-Nearest Neighbors Classifier

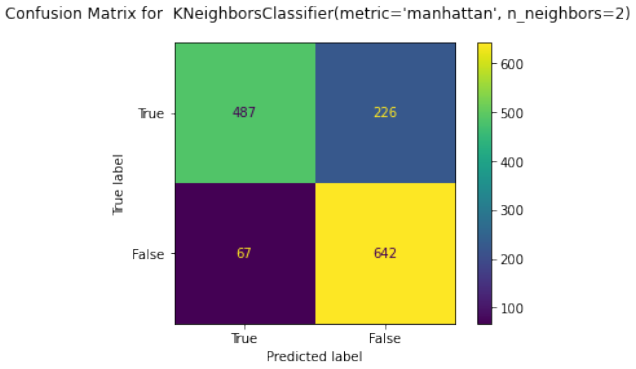
The evaluation metric of the algorithm is represented in the following table: The confusion matrix

Table 3.3: Evaluation metrics for KNN classifier

	Precision	Recall	F1-Score	Support
0	0.88	0.68	0.77	713
1	0.74	0.91	0.81	709
accuracy				0.79
macro avg	0.81	0.79	0.79	1422
weighted avg	0.81	.79	0.79	1422

indicates that the classifier correctly predicted 487 negative instances (class 0) and 642 positive instances (class 1). However, it incorrectly predicted 226 instances as positive (class 1) when they were actually negative (class 0), and 67 instances as negative (class 0) when they were actually positive (class 1). The precision of the classifier for class 0 is 0.88, indicating that when the algorithm predicted class 0, it was correct 88% of the time. The recall for class 0 is 0.68, indicating that the algorithm correctly identified 68% of the actual class 0 instances. The F1-score for class 0 is 0.77, which is the harmonic mean of precision and recall. Similarly, for class 1, the precision, recall, and F1-score are 0.74, 0.91, and 0.81, respectively. The accuracy of the classifier is 0.79, which indicates that it correctly predicted the class labels for 79% of the instances. The macro-averaged F1-score and weighted-averaged F1-score are both 0.79, indicating that the overall performance of the classifier is balanced across both classes. In general, an algorithm with high precision and recall values and a high F1-score is considered to be performing well. The accuracy of 79% suggests that the algorithm is performing reasonably well, but there is room for improvement, particularly in reducing the false positive and false negative rates.

Figure 3.3: Confusion Matrix for K-Nearest Neighbors Classifier



Chapter 4

BERT in Fake News Classification

4.1 BERT-Bidirectional Encoder Representations from Transformers

”BERT: Pre-training of Deep Bidirectional Transformers for language Understaning” is a paper published by researches of Google AI Language. This paper has sparked interest in the Machine Learning community due to its exceptional results in a wide range of Natural Language Processing (NLP) tasks, such as Question Answering (SQuAD v1.1) and Natural Language Inference (MNLI). The main innovation stands in the fact that BERT is not pre-trained using left-to-right right-to-left language models, but bidirectional training of Transformer, an attention model, for language modelling.
The researchers demonstrated that bidirectional training provides a deeper understanding of language context and flow compared to single-direction language models [1]. In the paper, they introduce a new technique called Masked LM (MLM) which enables bidirectional training in previously unattainable models.

4.1.1 Model Architecture

BERT is a multi-layer bidirectional Transformer encoder and is released in the tensor2tensor library [2]. The developers came up with two versions of the model: BERT base, and BERT large.
BERT Base base presents the same architecture as OPENAI GPT, and is made of 12 transformer encoder layers, 768 hidden units, and 110 million parameters; whereas BERT Large has 24 transformer

[1] (Devlin et al., 2019)
[2] (Devlin et al., 2019)

encoder layers, 1024 hidden units, and 340 million parameters, which makes it a much larger and more powerful model. However, it also requires more computation resources and training time. By creating two models with different sizes, the researchers aimed at providing a range of options that could balance the need for high performance with the practical constraints of computation resources and training time.

BERT makes use of Transformer, an attention mechanisms which is used to learn contextual relationships between words. The process is explained in the paper "Attention is al you need" which explains the function of Transformer (used in BERT). At each step the model is auto-regressive and makes use of encoders and decoders: the previously generated symbols are used as additional input when generating the next [3]. The encoder is composed of N=6 identical layers, each presenting other two sub-layers. The latter are a multi-head self-attention mechanism, and a fully connected feed-forward network. Each sub-layer has a residual connection and is followed by layer normalization. The output of each sub-layer is obtained by applying the sub-layer function to the input, adding it to the input using the residual connection, and then normalizing the result. All sub-layers, including the embedding layers, produce outputs of dimensionality 512, which enables the residual connections [4]. The decoder is also composed of N=6 identical layers, but in addition there is a third sub-layer which performs multi-head attention over the input of the encoded stack [5].

BERT uses a special type of self-attention mechanism called "masked self-attention" that allows it to handle inputs with missing tokens. During training, a small percentage of tokens in the input sequence are randomly masked out, and the model is trained to predict these masked tokens based on the surrounding context.

The masked self-attention mechanism in BERT works in a similar way to the multi-head self-attention mechanism in the encoder. It computes a weighted sum of the input embeddings, where the weights are based on the similarity between the query and key vectors for each token in the sequence. However, during masked self-attention, the model is only allowed to attend to tokens that are not masked out, which ensures that the model does not have access to information about the masked tokens.

[3](Vaswani et al., 2017)

[4]ibid

[5]Vaswani et al. (2017)

4.1.2 Pre-Training anf fine-tuning BERT

Pre-Training

The implementation of BERT model consists of two steps: pre-training and fine-tuning.
As said above, BERT uses a special kind of attention which is referred to as ”masked self-attention” (MLM). In order to train the bidirectional representation, 15% of the input is masked at random, and then only those masked tokens are predicted. In this phase, a mis-match between pre-triaining and fine-tuning is created due to the fact that the [MASK] tokens do not appear during fine-tuning. This effect is mitigated in order to prevent the model from simply memorizing the masked tokens and to encourage it to learn more general language representations, with the technique called ”masked token replacement” [6].
With this technique, the training data generator chooses 15% of the token positions at random for prediction. When the i-th token is chosen, it is then replaced with one of the following:

- The [MASK] token: 80% of the time, the i-th token is replaced with the special [MASK] token, which indicates that the original token is masked out,
- A random token: 10% of the time, the i-th token is replaced with a random token from the vocabulary, which encourages the model to learn more robust and general language representations
- The unchanged i-th token: 10% of the time, the i-th token is left unchanged to provide additional context to the model.

Subsequently, the model is trained to predict the original token based on the context provided by the remaining tokens. Specifically, the model is given the entire sequence of tokens as input, but the tokens that were selected for prediction are replaced with [MASK], a random token, or the unchanged token as described above. The model then generates a probability distribution over the entire vocabulary for each masked token, and the original token is predicted based on the highest-probability token in this distribution.
The second part of the pre-training consists of NSP (Next Sentence Prediction) used to make the model learn to understand the relationship between two consecutive sentence. In detail, pairs of sentences are created from a large corpus of text, and the model is trained to predict whether the second

[6](Devlin et al., 2019)

sentence is actually the next sentence that follows the first one or if it is a random sentence from the corpus. Specifically, when choosing the senteces A and B, 50% of the time B is the actual next sentence that follows A and 50% of the time it is a random sentence from the corpus [7].

Fine-Tuning BERT

Fine-tuning involves training the pre-trained model further on a smaller, task-specific dataset, which typically involves fine-tuning the parameters of the final layers of the model while keeping the pre-trained parameters fixed.
The fine-tuning process typically involves the following steps:

- Input representation: The input data is first pre-processed to match the format expected by the pre-trained BERT model. This involves tokenizing the text into word pieces (sub-words) and adding special tokens like [CLS] (for classification) and [SEP] (for separating sentences),
- Fine-tuning: The pre-trained BERT model is then fine-tuned on the downstream task by training the final layers of the model on the task-specific dataset. During this process, the pre-trained parameters of the model are typically kept fixed, while the parameters of the final layers are updated through back propagation. The goal is to optimize the model parameters for the specific task at hand, using techniques like gradient descent and stochastic optimization
- Evaluation: Once the model has been fine-tuned, it is evaluated on a separate validation or test set to measure its performance on the downstream task. This typically involves calculating metrics like accuracy, F1 score, or mean squared error, depending on the task,
- Iterative refinement: If the model's performance is not satisfactory, the fine-tuning process can be repeated with different hyper parameters or architectures, or by including additional training data. This process can be iterative until the desired level of performance is achieved

The fine-tuning process can be computationally expensive, especially for large datasets and complex tasks, but it is a powerful technique for leveraging the knowledge learned by pre-trained models like BERT to improve performance on a wide range of downstream tasks.

[7] (Devlin et al., 2019)

4.1.3 Implementation of BERT for automatic fake news detection task

This section focuses on explaining the implementation of the BERT model, as an additional tool in the detection of fake news, which is the aim of the thesis.

In the current analysis, the model used is BERT BASE the uncased version; which as said above is less computationally expensive and performs better on small-medium size datasets.

For this purpose, the dataset was split is three parts: 80% for the train set, 10% for validation set, and another 10% for test set. The validation set is important when training complex models such as BERT because it allows to perform operations like re-adjusting the learning rate, and batch size based on the model performance. The validation set can therefore be used to assess different combinations of these hyper-parameters and identify the optimal settings that lead to better model performance. Moreover, it is useful when monitoring the performance of the algorithm; since it can be used to continuously monitor the model's performance during training.

Specifically, the size of the three datasets is:

- train-set size: (4549, 2),
- validation-set size: (1137, 2)
- test-set size: (1422, 2)

Where 2 identifies the target, which is either true (0) or false (1).

After the data was split, a function with the main steps to prepare the data for the analysis was crated.

```
    def __getitem__(self, idx):
        if self.mode == 'test':
            statement, label = self.df.iloc[idx, :].values
            label_tensor = torch.tensor(label)
        else:
            statement, label = self.df.iloc[idx, :].values
            label_tensor = torch.tensor(label)

        word_pieces = ['[CLS]']
        statement = self.tokenizer.tokenize(statement)
        word_pieces += statement + ['[SEP]']
        len_st = len(word_pieces)

         tokens_b = self.tokenizer.tokenize(text_b)
         word_pieces += tokens_b + ["[SEP]"]
         len_b = len(word_pieces) - len_a

        ids = self.tokenizer.convert_tokens_to_ids(word_pieces)
        tokens_tensor = torch.tensor(ids)

        segments_tensor = torch.tensor([0] * len_st, dtype=torch.long)

        return (tokens_tensor, segments_tensor, label_tensor)

    def __len__(self):
        return self.len
```

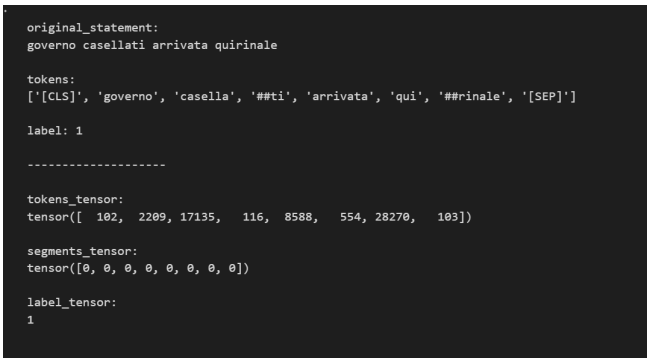
The "get_item" function is specified to enable indexing of the dataset, making it possible for individual samples to be accessed by index, and performs the following tasks:

- If the mode is 'test', it retrieves the statement and label values from the dataset at the specified index (idx) and converts the label to a PyTorch tensor (label_tensor),
- If the mode is 'train' or 'val', it performs the same steps as for 'test', retrieving the statement and label values and converting the label to a tensor,

- The statement is tokenized using the BERT tokenizer and the special tokens [CLS] and [SEP] are added to the token list. The token list is converted to token IDs using the tokenizer's convert_tokens_to_ids() method, and the resulting tensor is assigned to tokens_tensor,
- A segments tensor is created with all values set to 0, indicating that all tokens belong to the first sentence. The method returns a tuple containing the tokens_tensor, segments_tensor, and label_tensor

The image below represents an example of the output of the process described above. In the first line, there is the original statement retrieved from the dataset, and afterwards the same sentence after being processed by BERT tokenizer.The '##' symbol represents that the token is part of a larger word. The label indicates the label associated with the statement, which in this case is 1 (so fake), and tokens_tensor displays the tensor representation of the token IDs obtained after converting the tokenized statement using the tokenizer's convert_tokens_to_ids() function described above. The tensor contains the values [102, 2209, 17135, 116, 8588, 554, 28270, 103]. Moreover, segments_tensor represents the segments tensor, which is a tensor of zeros with the same length as the tokenized statement. In this case, all the values are 0, indicating that all tokens belong to the first sentence. Lastly. label_tensor shows the label converted to a PyTorch tensor. In this case, the label tensor has the value 1.

Figure 4.1: Snippet Output of tokenization



4.1.4 BERT Pre-training

The second part of the code, aims at preparing the input data for the BERT model by converting the samples into tensors, padding them, creating attention masks, and stacking the labels. This enables efficient batch processing and handling variable-length input sequences, which are important for training, and inference with BERT. The batch size is set at 16, meaning that during training, the model processes and updates its weights based on 16 training examples at a time. These 16 examples form a batch, and the model's parameters are updated based on the average gradient calculated over the batch. The choice of an appropriate batch size depends on various factors, including the available computational resources, the size of the dataset, and the complexity of the model. It often involves a trade-off between computational efficiency and the quality of the learned model.

The function created for achieving the described task is the following:

```
def create_mini_batch(samples):
    tokens_tensors = [s[0] for s in samples]
    segments_tensors = [s[1] for s in samples]


    if samples[0][2] is not None:
        label_ids = torch.stack([s[2] for s in samples])
    else:
        label_ids = None


    tokens_tensors = pad_sequence(tokens_tensors, batch_first=True)
    segments_tensors = pad_sequence(segments_tensors, batch_first=True)



    masks_tensors = torch.zeros(tokens_tensors.shape, dtype=torch.long)
    masks_tensors = masks_tensors.masked_fill(tokens_tensors != 0, 1)

    return tokens_tensors, segments_tensors, masks_tensors, label_ids
```

Tokens_tensors and segments_tensors are lists used to store the token tensors, and segment tensors for each sample in the input batch. The tokens tensor contains the tokenized input text; while the

segments tensor indicates which part of the input corresponds to the actual text and which part is padding.

The ”Label_ids” variable stores the label tensors for the samples in the input batch, after checking the each sample has a label. If the first sample has no label, label_ids is set to None.

Afterwards, padding begins: the tokens_tensors and segments_tensors are padded to have the same length within the batch. The pad_sequence() function is used with batch_first=True to pad the tensors with zeros. Padding involves adding special tokens (usually zeros) to the input sequences so that they match the length of the longest sequence in the batch.

In the attention mask part, the ”masks_tensors” are initialized as a tensor of zeros with the same shape as ”tokens_tensors”. The purpose of the attention mask is to indicate to BERT which positions in the input it should pay attention to. In this case, the positions of the actual tokens are set to 1, while the padded positions remain 0.

The function returns the processed tensors: tokens_tensors, segments_tensors, masks_tensors, and label_ids. These tensors will be used as input to the BERT model during training, validation, and testing.

For a better understanding of the process described, a snippet of the output is included.

Figure 4.2: Snippet Output token tensors

```
tokens_tensors.shape   = torch.Size([16, 21])
tensor([[  102,  2209, 17135,   116,  8588,   554, 28270,   103,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
             0],
        [  102,  1271,   594,  1339, 25282,  5524, 12459,   103,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
             0],
        [  102, 24240,  6389,  1057,  5764, 30940,  2347,  1986,  1291,  2407,
           103,     0,     0,     0,     0,     0,     0,     0,     0,     0,
             0],
        [  102, 25359, 30931,   127, 29697, 30876,   181,  5817,  1869,  1675,
         23436, 11481,   818,  1045, 15699, 30912, 21216,  1291,  2407,   103,
             0],
        [  102, 10438,   960,  5417,   710,   779, 30879, 24705, 30896,  5664,
          7331,  1472, 12459,   763,  6068, 24645,   181,  5817, 20352, 28064,
           103],
        [  102,  1871,   949,  3959, 10854,  1188,   763,  1522,  2744,  3823,
           103,     0,     0,     0,     0,     0,     0,     0,     0,     0,
```

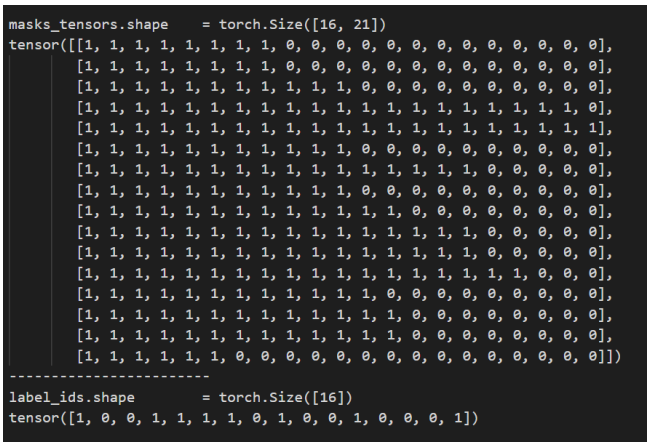
The tensor itself contains integer values that represent the tokens in the input sequences. Zeros in the tensor indicate the padding tokens that were added to make all sequences in the batch have the same length. Looking at the first row, the sequence starts with the [CLS] token (represented by 102), followed by several other tokens. The remaining entries in the row are zeros, which represent padding tokens. Similarly, the other rows in the tensor follow the same pattern. The number of padding tokens varies depending on the length of the corresponding sequence. All rows have been padded with zeros to match the length of the longest sequence in the batch.

Figure 4.3: Snippet Output of segment tensors

```
------------------------
segments_tensors.shape = torch.Size([16, 21])
tensor([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]])
------------------------
```

The tensor segments_tensors contains integer values that represent the segment IDs of the input tokens. In BERT, when processing a sequence, it is common to have two segments: Segment A and Segment B. However, in this case, since we are dealing with a single-sequence classification task, all the tokens belong to a single segment. The tensor segments_tensors in the example consists of all zeros. This indicates that all tokens in the input sequences belong to the same segment (Segment A).

Figure 4.4: Snippet Output of masked tensors



In the picture above, masks_tensors.shape is torch.Size([16, 21]), indicating that the masks_tensors tensor has a shape of 16 rows and 21 columns. The tensor masks_tensors is created to serve as an attention mask for the input tokens. It is used to indicate which tokens in the input sequences are actual tokens, and which ones are padding tokens. The attention mask helps the BERT model focus only on the relevant tokens and ignore the padding tokens during processing. In the example, the masks_tensors tensor consists of ones and zeros. The ones indicate the positions of the actual tokens in the input sequences, while the zeros indicate the positions of the padding tokens. By setting the attention mask to one for actual tokens and zero for padding tokens, the BERT model knows which tokens to attend to, and which tokens to disregard. For instance, in the first sequence [1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], the first eight positions have a value of one, indicating the presence of actual tokens, followed by thirteen zeros for padding. The masks_tensors tensor is an essential component in the BERT model's input because it helps ensure consistent sequence lengths within a batch and enables the model to attend to the meaningful tokens while ignoring the padding tokens.

Lastly, regarding the label_ids tensor, it has a shape of torch.Size([16]), indicating that it is a 1-dimensional tensor with 16 elements. It represents the labels for the corresponding input sequences in the batch. In the given example, each element in the label_ids tensor corresponds to the label of the respective sequence in the batch.

4.1.5 Model Construction

Subsequently to preparing the data, the model is loaded. As before, for a better understanding of the model architecture, a snippet of the code describing the configuration of the latter is included, and then explained.

Figure 4.5: Snippet Output of model configuration


```
BertConfig {
  "_name_or_path": "dbmdz/bert-base-italian-cased",
  "architectures": [
    "BertForMaskedLM"
  ],
  "attention_probs_dropout_prob": 0.1,
  "classifier_dropout": null,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 768,
  "initializer_range": 0.02,
  "intermediate_size": 3072,
  "layer_norm_eps": 1e-12,
  "max_position_embeddings": 512,
  "model_type": "bert",
  "num_attention_heads": 12,
  "num_hidden_layers": 12,
  "pad_token_id": 0,
  "position_embedding_type": "absolute",
  "transformers_version": "4.24.0",
  "type_vocab_size": 2,
  "use_cache": true,
  "vocab_size": 31102
}
```


The model configuration is the following:

- _name_or_path: specifies the name or path of the pre-trained model used as the base. In this case, it is ”dbmdz/bert-base-italian-cased”, which means that it makes a distinction between cased and uncased words, [8]
- architectures: Lists the architecture(s) used for the pre-trained model. In this case, it is ”BertForMaskedLM”, which indicates that the model was originally trained for masked language modeling,
- attention_probs_dropout_prob: specifies the dropout probability for the attention mechanism in the transformer layers. It is set to 0.1, meaning that during training, 10% of the attention weights are randomly set to zero to prevent over-fitting,

[8](Bert-base-cased · Hugging Face, n.d.)

- classifier_dropout: Specifies the dropout probability for the classifier layer. It is set to null, indicating that no additional dropout is applied specifically to the classifier layer,
- hidden_act: specifies the activation function used in the transformer layers' feed-forward networks. The "gelu" activation function is used, which is a smooth approximation of the rectified linear unit (ReLU) function.
- hidden_dropout_prob: Specifies the dropout probability for the hidden layers in the transformer. It is set to 0.1, meaning that during training, 10% of the hidden unit activation are randomly set to zero to prevent over-fitting.
- hidden_size: Specifies the size (dimensionality) of the hidden layers in the transformer. It is set to 768, indicating that the hidden layers have 768 units.
- initializer_range: Specifies the range for weight initialization. It is set to 0.02, meaning that the weights are initialized using a uniform distribution between -0.02 and 0.02,
- intermediate_size: Specifies the size of the intermediate (hidden) layer in the transformer's feed-forward networks. It is set to 3072.
- max_position_embeddings: Specifies the maximum length of input sequences. It is set to 512, meaning that input sequences longer than 512 tokens would need to be truncated or processed in chunks.
- num_attention_heads: Specifies the number of attention heads in the transformer. It is set to 12.
- vocab_size: Specifies the size of the vocabulary, i.e., the total number of unique tokens in the tokenizer's vocabulary. It is set to 31102.

4.1.6 Fine-Tuning

Finally, the last part of the code focuses on fine-tuning BERT and assessing its performance.

```
from sklearn.metrics import accuracy_score
from tqdm.notebook import tqdm

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print("device:", device)
model = model.to(device)

model.train()
optimizer = torch.optim.Adam(model.parameters(), lr=1e-5)
NUM_EPOCHS = 3

for epoch in range(NUM_EPOCHS):
    train_loss = 0.0
    train_acc = 0.0

    loop = tqdm(trainloader)
    for batch_idx, data in enumerate(loop):
        tokens_tensors, segments_tensors, masks_tensors, labels = [t.to(device)
    for t in data]



        optimizer.zero_grad()

        outputs = model(input_ids=tokens_tensors,
                        token_type_ids=segments_tensors,
                        attention_mask=masks_tensors,
                        labels=labels)

        loss = outputs[0]
        loss.backward()
        optimizer.step()

        logits = outputs[1]
        _, pred = torch.max(logits.data, 1)
```

```
34          train_acc = accuracy_score(pred.cpu().tolist() , labels.cpu().tolist())
35
36
37          train_loss += loss.item()
38
39
40
41          loop.set_description(f"Epoch [{epoch+1}/{NUM_EPOCHS}]")
42          loop.set_postfix(acc = train_acc, loss = train_loss)
```

The model is set to training mode: this is necessary to activate certain functionalities like dropout. Subsequently Adam optimizer is defined, for updating the model parameters during training. It takes the model.parameters() and a learning rate of 1e-5 as arguments.

The number of epochs is set to 3, and afterwards the code starts a loop over the specified number of epochs. The number of epochs refers to the number of times the model iterates over the train-set, so in this case three times. Within each epoch, it initializes the training loss and accuracy. The code iterates over the batches of data (data) using enumerate(loop).

At the beginning of the loop the trian_loss, and accuracy_loss are set to zero at the start of each epoch, in order to keep track of the loss value during the training phase. The loss is used to compute gradients using the loss.backward() function.

The optimizer.step() function is then called to update the model parameters based on the computed gradients. Calculate accuracy: The model predictions (logits) are obtained from the outputs, and the torch.max function is used to find the predicted class labels (pred). The accuracy_score function compares the predicted labels with the ground truth labels (labels) and calculates the accuracy. Note that .cpu() is used to move the tensors to the CPU for compatibility with the accuracy_score function.

In lines 23-26 the model object is called with the input tensors (tokens_tensors, segments_tensors, masks_tensors) and the labels (labels). This invocation of the model represents the forward pass. During the forward pass, the input tensors are passed through the model's layers, and the model computes the output.

Finally, in the last part of the code the training loss of the batch (loss.item()) is added to the running total (train_loss).

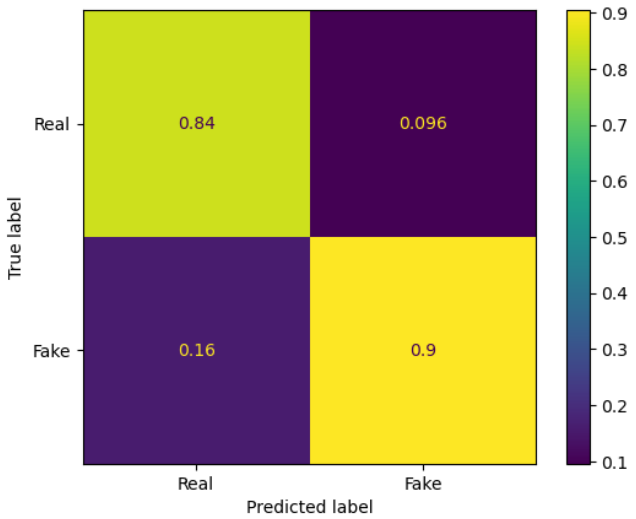
4.1.7 Results of BERT model

In this section, the results of the model previously constructed will be analyzed. As before, a snippet of the code will be included for a better understanding of the functions employed.

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

true=[]
predictions=[]
with torch.no_grad():
    model.eval()
    for data in testloader:
        if next(model.parameters()).is_cuda:
            data = [t.to(device) for t in data if t is not None]

        tokens_tensors, segments_tensors, masks_tensors = data[:3]
        test_outputs = model(input_ids=tokens_tensors,
                    token_type_ids=segments_tensors,
                    attention_mask=masks_tensors)

        logits = test_outputs[0]
        _, pred = torch.max(logits.data, 1)

        labels = data[3]
        true.extend(labels.cpu().tolist())
        predictions.extend(pred.cpu().tolist())


cm = confusion_matrix(true, predictions, labels=[1, 0], normalize='pred')
print(cm)

disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Real', 'Fake
    '])
disp.plot()

print('Acc: ', accuracy_score(predictions,true))
```

After importing the confusion matrix from sklearn, lines 6-21 evaluate the model by iterating over the test data. This is achieved setting the model to ”model.eval()”, and disabling the gradient computation ”torch.no_grad()” in order to save memory. The input tensors are then passed to the model, and the logits are obtained. The predicted labels are determined by selecting the class with the highest logit value. The true labels and predicted labels are then added to the respective lists. Afterwards, the confusion matrix is printed.

Figure 4.6: Confusion Matrix of BERT model



The confusion matrix above is a 2x2 matrix with the following structure [[0.84346701 0.09553159] [0.15653299 0.90446841]]. As for the previous models, the performance of the classification model is displayed by showing the proportion of true positive (TP), true negative (TN), false positive (FP), and false negative (FN) predictions. Specifically, from this values it can be said that:

- The top-left value of 0.84346701 represents the proportion of true positives (TP). It indicates that 84.35% have been classified as real by the model, and their label was indeed real;
- The top-right value of 0.09553159 represents the proportion of false negatives (FN). It indicates that 9.55% of the instances that truly belong to the positive class (’Real’) were incorrectly predicted as negative,
- The bottom-left value of 0.15653299 represents the proportion of false positives (FP). It indicates that 15.65% of the instances that truly belong to the negative class (’Fake’) were incorrectly predicted as positive,

- The bottom-right value of 0.90446841 represents the proportion of true negatives (TN). It indicates that 90.45% of the instances that truly belong to the negative class ('Fake') were correctly predicted as negative.

Moreover, a classification report which takes into account other evaluation metrics has been retrieved. Indeed, the precision, F1 score, recall, and support metrics of the model will be analyzed.

Table 4.1: Evaluation metrics for BERT MODEL

Class	Precision	Recall	F1-score	Support
0	0.90	0.83	0.87	708
1	0.84	0.91	0.88	714
accuracy			0.87	1422
macro avg	0.87	0.87	0.87	1422
weighted avg	0.87	0.87	0.87	1422

As mentioned above, the precision metric indicates the number of correct classification. In this case, for class 0, the precision is 0.90, which means that 90% of the instances predicted as 'Real' were actually 'Real'. For class 1, the precision is 0.84, indicating that 84% of the instances predicted as 'Fake' were indeed 'Fake'.

The "Recall" or sensitivity is the proportion of correctly predicted positive instances out of all the true positive instances. For class 0, the recall is 0.83, meaning that the model identified 83% of the 'Real' instances correctly. For class 1, the recall is 0.91, indicating that the model captured 91% of the 'Fake' instances.

The F1-score is also known as the harmonic mean. It takes into account precision and recall and outputs a single metric that balances them both. For class 0, the F1-score is 0.87, which combines the precision and recall values. For class 1, the F1-score is 0.88, indicating a balance between precision and recall.

The support stresses the number of observations in each class: in this case 708 instances of class 0 ('Real') and 714 instances of class 1 ('Fake').

The accuracy measures the overall performance of the model by calculating the proportion of correctly predicted instances out of all instances. The accuracy value is 0.87, indicating that the model predicted correctly for 87% of the instances.

The macro average calculates the average of precision, recall, and F1-score for both classes. In this case, it is 0.87 for all metrics, providing an overall average performance of the model.

The weighted average calculates the average of precision, recall, and F1-score, weighted by the support of each class. In this case, it is 0.87 for all metrics, indicating an average performance weighted by the number of instances in each class.

4.2 Model evaluation

Based on the results displayed above, a comparison between the BERT model and the other classical machine learning models can be made.

Based on the evaluation metrics, a rank of all the models can be made: **Random Forest** is the best performing model, with a precision of 0.93, recall: 0.96, f1-score: 0.95 ,accuracy: 0.94. This is followed by the **BERT model** which presents a precision of 0.90, recall of 0.83, f1-score of 0.87, accuracy of 0.87N. On the other hand, the worst performing model is **KNN model**, presenting the lowest precision, recall, and F1-score among the models. It achieves a precision of 0.88, recall of 0.68, and F1-score of 0.77. Its accuracy is 0.79, which is also lower than the other models.

Chapter 5

GPT3 and its role in misinformation

5.1 What is GPT3

In 2020, Open-AI developed GPT3, a neural language model that is capable of sophisticated natural language generation and completion of tasks like classification, question-answering, and summarization. GPT-3 emerged from a lineage of advancements in natural language processing (NLP) and deep learning. Building upon the successes of its predecessors, GPT-3 pushed the boundaries of what was previously thought possible in AI. It consists of 175 billion parameters [1], making it one of the most complex and advanced language models to date. The model uses deep learning algorithms to analyze and learn from vast amounts of text data, allowing it to generate coherent and contextually relevant text.

As said above, GPT-3 is trained employing the unsupervised method ”pre-training” followed by ”fine-tuning.” During pre-training, GPT-3 is exposed to a vast amount of text data and learns to predict the next word in a sentence or fill in missing words. This process helps the model develop a broad understanding of language, including grammar, syntax, and context. GPT-3 is trained to generate coherent and contextually appropriate responses based on this pre-training.

While GPT-3 has shown impressive capabilities in language generation and translation, it also poses a significant challenge in relation to the spread of fake news. As the model can generate human-like text that is difficult to distinguish from human-written content, it can be used to create fake news stories, articles, and social media posts.

[1] (Kaliyar et al., 2021)

5.2 Role of GPT3 in the spread of disinformation

The question of whether or not GPT3 is enhancing the spread of disinformation has been at the center of many studies. This risk is given by the astonishing capability of GPT to produce human-like text, and consequent inability of humans to recognize syntetic AI-generated text.

In the paper "The Radicalization Risks of GPT3 And Advanced Neural Language Models", the reaserchers analyse the impact of GPT3 in spreading extremists ideas. In the paper it is stressed out that the main impact of GPT3 in disinformation given by the fact that, in order to produce ideologically consistent fake-news, there is no longer the need to to feed the model with a large corpus of source material and hours of fine-tuning [2]. Indeed, GPT3 only requires as input a short text (ie a tweet), to immediately pick up on the pattern and intent of the author without any further training, and be prompted in order to produce a series of tasks, such as:

- mimicking the writing of the short-text that is given as input;
- reproducing multi-lingual biased and extremist text [3],
- influence the masses to spread misinformation which is not recognizable by humans as text written with AI.

One of the main concerns regarding GPT3 derives from its weaponizing in the production of human-like text. Indeed, misinformation risk is greatly amplified if AI-generated text is not controlled and is shared without critical evaluation, therefore amplyfing also the reach of those news. The lack of human recognition can contribute to the virality of misinformation, leading to potential societal consequence.

To mitigate these risks, it is crucial to promote media literacy, critical thinking, and fact-checking skills among individuals. Developing AI tools and techniques for detecting AI-generated text can also help in identifying and flagging potential instances of fake news. Additionally, responsible deployment of AI models, including appropriate disclaimers and transparent disclosure of AI-generated content, can help in raising awareness and fostering a more informed society.

[2] (McGuffie K. et al, 2020)
[3] ibid

5.3 Ethical issues and bias related to Open AI

The ethical issues related to GPT-3 is addressed by the authors of the 2020 Google paper as they state ”We focus on two primary issues: the potential for deliberate misuse of language models like GPT-3... and issues of bias, fairness, and representation within models like GPT-3” [4]. Those two issues are referred to as ”dark side of AI” [5], which comprehends all the negative aspects related to AI generated technology.

Regarding the first issue, that is to say manipulation, one of the main concerns in the misuse of GPT consist of malicious behaviour by actors who exploit it to produce credible, human-sounding text [6]. This could facilitate the spread of fake news, which could be used to target specific groups of individual and affect their decision-making process.This could result in people acting for ends that are not their own, and for reasons they have not chosen [7]. As chat gpt could potentially harm people's autonomy, the mitigation of this effect lies in the adoption of measures such as digital literacy, increase of transparency, and accountability for developers.

The second important concern related to open AI is bias. Indeed, when developing algorithms there is often the risk for ”inherited bias” which is the bias that algorithms inherit from the data they are trained on. This happens for instance if data are not representative of the population as a whole, leaving out certain groups of people. Bender et al. have demonstrated that usually large datasets do not equally represent online users but significantly over-represent younger users, people from developed countries, and English speakers [8].

[4] Brown et al., (2020)
[5] Mikalef et al. (2022)
[6] Chan (2022)
[7] Susser et al.,(2019)
[8] Bender et al.,(2022)

Conclusion

This research has highlighted the critical aspects and the challenges that lie behind the creation of an algorithm able to detect fake news present online.

As result, it emerged that it is indeed useful to use machine learning as a tool in the fight against fake news proliferation. The computational power of algorithms allows for the analysis of vast amounts of data in a relatively short amount of time. Without this capability, the task would require a significant amount of time and energy if done manually by individuals. On the other hand, it is important to stress out the fact that also algorithms have a margin of error that is not reducible to zero, suggesting that there still should be a human component in the analysis of text data, and context of articles.

Moreover, the importance of topic modelling algorithms and their ability to detect the subject of text data before reading the content o has been shown, and how text pre-processing is a crucial step in the creation of the latter.

In the more technical part of the thesis, it emerged how accurate classical models can be, as it was the case with Random Forest and Naive Bayes.

Those results could be improved using more sophisticated models like BERT, which on one hand requires more effort in the training part; and more time to obtain the results, but producing more accurate results.

Lastly, the research on the role of Chat-Gpt3 in the spread of fake news online has shown how its astonishing abilities to create human-like text could be a potential harm for the community of online users; but the improvement of the model privacy and safety measures has led to great reduction of this risk.

Bibliography

[1] Admin. (n.d.). admin2. Retrieved from https://www.professionereporter.eu/2021/12/ecco-i-10-siti-piu-affidabili-open-tpi-agi-internazionale-sole tirreno-nuova-sardegna/

[2] ActiveState. (2022, August 9). What is Pandas in Python? Everything you need to know - ActiveState. https://www.activestate.com/resources/quick-reads/what-is-pandas-in-python-everything-you-need-to-know/

[3] Beautiful Soup Documentation — Beautiful Soup 4.4.0 documentation. (n.d.). https://beautiful-soup-4.readthedocs.io/en/latest/

[4] Bender, E. M., Gebru, T., McMillan-Major, A., & Shmitchell, S.: On the dangers of stochastic parrots: can language models be too big? In: Paper Presented at the Conference on Fairness, Accountability, and Transparency (FAccT '21), March 3–10, 2021, Virtual Event, Canada. Association for Computing Machinery, New York, NY, USA, pp. 610–623. https://doi.org/10.1145/3442188.3445922(2021)

[5] Chan, A. (2022). GPT-3 and InstructGPT: technological dystopianism, utopianism, and "Contextual" perspectives in AI ethics and industry. AI And Ethics, 3(1), 53–64. https://doi.org/10.1007/s43681-022-00148-6

[6] Chen, L. (2021, December 14). Why do we have to talk about Type 1 error and Type 2 error? — Towards Data Science — Towards Data Science. Medium. https://towardsdatascience.com/programming-journal-4-why-do-we-have-to-talk-about-type-1-error-and-type-2-error-41b3ae68bb96

[7] Christopher, A. (2021, December 29). K-Nearest Neighbor - The Startup - Medium. Medium. https://medium.com/swlh/k-nearest-neighbor-ca2593d7a3c

[8] Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT) (Vol. 1, pp. 4171–4186).

[9] Doc · spaCy API Documentation. (n.d.). Doc. https://spacy.io/api/doc

[10] Doll, T. (2019, March 11). LDA Topic Modeling: An Explanation - Towards Data Science. Medium. Retrieved from https://towardsdatascience.com/lda-topic-modeling-an-explanation-e184c90aadcd

[11] Donald E. Knuth (1986) *The TEX Book*, Addison-Wesley Professional.

[12] Ellis, C. (2022, September 20). Number of trees in random forests. Crunching the Data. https://crunchingthedata.com/number-of-trees-in-random-forests/

[13] Gensim: topic modelling for humans. (n.d.). https://radimrehurek.com/gensim

[14] Ghanoum T., (2021,December,20), "Topic Modelling in Python with spaCy and Gensim" https://towardsdatascience.com/topic-modelling-in-python-with-spacy-and-gensim-dc8f7748bdbf

[15] Gupta, S. (2022, March 30). Understanding and using k-Nearest Neighbours aka kNN for classification of digits. Medium. https://towardsdatascience.com/understanding-and-using-k-nearest-neighbours-aka-knn-for-classification-of-digits-a55e00cc746f

[16] Hendrixwilsonj. (2021). LIAR Data analysis. Kaggle. https://www.kaggle.com/code/hendrixwilsonj/liar-data-analysis

[17] Hossin, M., & Sulaiman, M. R. (2015). A Review on Evaluation Metrics for Data Classification Evaluations. International Journal of Data Mining & Knowledge Management Process, 5(2), 01–11. https://doi.org/10.5121/ijdkp.2015.5201

[18] Huang, Y., & Li, L. (2011). Naive Bayes classification algorithm based on small sample set. https://doi.org/10.1109/ccis.2011.6045027

[19] Koehrsen, W. (2021, August 6). Beyond Accuracy: Precision and Recall - Towards Data Science. Medium. https://towardsdatascience.com/beyond-accuracy-precision-and-recall-3da06bea9f6c

[20] Kaliyar, R. K., Goswami, A., & Narang, P. (2021). FakeBERT: Fake news detection in social media with a BERT-based deep learning approach. Multimedia Tools and Applications, 80(8), 11765–11788. https://doi.org/10.1007/s11042-020-10183-2

[21] Lecture 2: k-nearest neighbors / Curse of Dimensionality. (n.d.). https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote02_kNN.html#:~:text=The%20k%2Dnearest%20neighbor%20classifier,%7Cp)1%2Fp

[22] Leslie Lamport (1994) *LaTeX: a document preparation system*, Addison Wesley, Massachusetts, 2nd ed.

[23] Winastwan, R. (2022b, January 4). Text Classification with BERT in PyTorch - Towards Data Science. Medium. https://towardsdatascience.com/text-classification-with-bert-in-pytorch-887965e5820f?gi=38b751b371c0

[24] Loukas, S., PhD. (2023, April 21). Text Classification Using Naive Bayes: Theory & A Working Example. Medium. https://towardsdatascience.com/text-classification-using-naive-bayes-theory-a-working-example-2ef4b7eb7d5a

[25] McGuffie K., Newhouse A., (2020), The Radicalization of GPT-3 and advanced Neural Language Models

[26] Meltzer, R. (2021, July 15). What is Random Forest? [Beginner's Guide + Examples]. CareerFoundry. https://careerfoundry.com/en/blog/data-analytics/what-is-random-forest/

[27] NLTK: Natural Language Toolkit. (n.d.). https://www.nltk.org/

[28] PyData —. (n.d.). https://pydata.org/

[29] Requests: HTTP for HumansTM — Requests 2.31.0 documentation. (n.d.). https://requests.readthedocs.io/en/latest/

[30] Ratz, A. V. (2022, April 11). Multinomial Naïve Bayes' For Documents Classification and Natural Language Processing (NLP). Medium. https://towardsdatascience.com/multinomial-nave-bayes-for-documents-classification-and-natural-language-processing-nlp-e08cc848ce6

[31] Scikit-learn: machine learning in Python — scikit-learn 1.2.2 documentation. (n.d.).https://scikit-learn.org/stable/

[32] Susser, D., Roessler, B., Nissenbaum, H.: Technology, autonomy, and manipulation. Internet Policy Rev. 8(2), 1–22 (2019). https://doi.org/10.14763/2019.2.1410

[33] Shu, K., Mahudeswaran, D., Wang, S., Lee, D., & Liu, H. (2020). FakeNewsNet: A Data Repository with News Content, Social Context, and Spatiotemporal Information for Studying Fake News on Social Media. Big Data, 8(3), 171–188. https://doi.org/10.1089/big.2020.0062

[34] Transformers. (n.d.). https://huggingface.co/docs/transformers/index

[35] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention is All you Need. In arXiv (Cornell University) (Vol. 30, pp. 5998–6008). Cornell University. Retrieved from https://arxiv.org/pdf/1706.03762v

[36] WordCloud for Python documentation — wordcloud 1.8.1 documentation. (n.d.). https://amueller.github.io/word_cloud/

List of Figures

List of Tables