Politecnico di Torino

POLITECNICO DI TORINO

Master's Degree in Data Science and Engineering

Academic Year 2021-2022

Detecting fake news using Natural Language Processing

Supervisors

Prof. Elena BARALIS

Prof. Raphaël TRONCY

Candidate

Giulio ALFARANO

Abstract

The enormous dissemination of data and information enabled by the Internet has brought great freedom of expression and information to almost the entire world population. However, at the same time the spread of information also brings with it the spread of false information, known in the field as "Fake news" or "misinformation". This phenomenon has led the scientific community to become interested in and develop solutions that can curb the problem: a fundamental building block is Natural Language Processing, an algorithmic technique developed through Machine Learning and artificial intelligence in general. In this work we address two challenges carried out during 2021, FEVEROUS and MediaEval 2021, that concern the identification of fake news through natural language processing and the most widely used models for text classification. The first challenge dealt with the truthfulness of various claims within an ad hoc provided database by retrieving evidence from the English Wikipedia corpus; on the other hand, the second challenge concerned the identification of misinformation in a dataset composed of real tweets about the recent Covid-19 pandemic and related conspiracy theories. Among the tested models, the most explored techniques are the classification algorithms following a pre-processing and encoding process of the text and, secondly, the Transformers, considered the current state-of-art for the main Natural Language Processing tasks, such as the one analysed, but also used in other fields. The promising results show that this type of solution can be an excellent tool to help the community face this problem from a social and technical point of view.

Acknowledgements

I would like to express my gratitude to all the people who helped me during the processing of this thesis and in general during my university years.
A big thank you goes to professor Elena Baralis and professor Raphael Troncy, who allowed me to work on such an interesting topic in the stimulating environment of EURECOM and guided me throughout my thesis.
Thanks to all the EURECOM staff who welcomed me as one of their own. A special thank you goes to professor Paolo Papotti, Mohammed Saeed and Youri Peskine who stimulated me and helped me in the practical activities behind this work.
I thank my mom Cinzia, my dad Luigi and my brothers Giorgio and Giacomo for their unconditional love and support that bridges the physical distance that separates us. Thanks to Irene, for always being by my side for the past 4 years and more to come with infinite love, and for protecting me from all the fears of adulthood.
Thanks to Matteo, without which I would not have made it through my university years with such success and for being a central hub of friendship. Thanks to Marina, for being the best accomplice I could ask for in this shared path in Turin and for being an unwitting personal guide.
Thanks to Ilaria, Stefano, Chiara, Erika, Pierluigi, Riccardo and Desirèe for their long-standing friendship that I hope will last forever. Lastly, thank you to all my friends in Turin for making these years the best of my life.

Table of Contents

List of Tables

List of Figures

Acronyms

AI
Artificial Intelligence

NLP
Natural Language Processing

NLI
Natural Language Inference

NER
Named Entity Recognition

ML
Machine Learning

DL
Deep Learning

NLTK
Natural Language Toolkit

BoW
Bag of Words

TF-IDF
Term Frequency-Inverse Document Frequency

PCA
Principal Component Analysis

ERM
: Empirical Risk Minimization

TP
: True Positive

FP
: False Positive

TN
: True Negative

FN
: False Negative

ROC
: Receiver Operating Characteristic

AUC
: Area Under the Curve

MCC
: Matthews Correlation Coefficient

AdaBoost
: Adaptive Boosting

SVM
: Support Vector Machine

NN
: (Artificial) Neural Network

ReLU
: Rectified Linear Unit

MLP
: Multi-Layer Perceptron

CNN

Convolutional Neural Network

RNN

Recurrent Neural Network

LSTM

Long-Short Term Memory

BERT

Bidirectional Encoder Representations from Transformers

MLM

Masked Language Model

NSP

Next Sentence Prediction

RoBERTa

Robustly optimized BERT approach

FC

Fact-Checker

API

Application Programming Interface

FACT

Framework for Analysis and Capture of Twitter Graphs

GDELT

Global Dataset of Events, Location, and Tone

FANG

Factual News Graph

IR

Information Retrieval

"To square the circle of the world in your own interpretations is the onset of shortsightedness in an eye that thinks it has perfect vision."
Roberto Saviano, *ZeroZeroZero*

Chapter 1

Introduction

Recent years have seen a surge in the use of the internet and social networks in particular. This situation increases in parallel a large flow of information circulating on the web. The case of fake news is not a contemporary problem, but it developed simultaneously with the spread of the media. However, with the advent of the Internet, the phenomenon has become increasingly present and constant, also because of the *echo-chamber* effect, where information, ideas or beliefs are reinforced by communication and repetition within a defined group of people. This is a relevant situation especially when events of global interest occur: the most striking and recent case is the Covid-19 pandemic. Fake news includes not only false textual news, but also news or phrases of various kinds that have been manipulated, de-contextualised or created with the aim of doing harm, thus threatening both the collective knowledge of a given event and the actions that follow it.

This type of phenomena has stimulated the scientific community in the field of data science, which developed in the past statistical and Machine Learning techniques firstly to make natural language understandable from computer, and consequently to perform specific tasks on textual data, such as *machine translation*, *question-answering*, *topic modeling*, etc. From statistical tools, configurations have evolved to the latest technologies that most often include models based on complex neural networks.

In the last years the research community experimented with the application of new architectures and technologies to the field of fake news detection, studying if these could be usable tools to tackle the problem. Numerous researches and surveys have been carried out in order to initially study the characteristics of misinformation in depth, and subsequently to be able to identify, block and prevent its spread. This area of research has also been encouraged by the numerous organisations that have been set up in recent years with the aim of curbing the problem of misinformation. Despite this, current technologies do not allow us to have accurate solutions, both

because of a limitation in terms of computational power and because of people's distrust of automated systems. However, this does not exclude the possibility of implementing tools that can help humans in this task.
In this work we approach two different challenges held in 2021, both focused on fake claims and disinformation detection in two different fields and with different configurations. Before diving into the two task, in chapter 2 we present a general overview of the most widely used Machine Learning and Deep Learning techniques, from the most basic natural language textual encoding up to the state-of-art architectures based on deep neural networks. Following, in chapter 3 we introduce the social context in which the tasks are designed and placed, together with motivations about the importance of developing solutions, some definitions about misinformation and related concepts, and a brief description of current state-of-art in fake news field.
The core sections of this work are chapter 4 and chapter 5, which present two challenges. The first one is a task held by the FEVER workshop concerning the identification of claim veracity from a dataset of annotated sentences. These claims have been generated by some volunteer annotators starting from a set of Wikipedia elements. The web encyclopedia is central for this work: indeed, the authors provided a corpus created from the English Wikipedia, through which the algorithm has to extract structured and unstructured evidences in order to predict the factuality of the input claims. Our solution was based on a combination of transformer-based and neural re-ranker models that allowed us to finish in fifth place, above the baseline score. Instead, the second challenge was held by MediaEval organization, which concern a more real-world problem, specifically the misinformation about Covid-19 pandemic. In fact, the provided dataset consists in a set of tweet text scraped from the famous social network, which focus on the emergency pandemic situation and on the related conspiracy theories that circulate online. The authors decided to extract 9 different conspiracies through keywords and hashtags, and the main task of the challenge is to identify if a tweet is discussing a theory and which of the conspiracies it is about. We managed to reach the first position through a state-of-art transformer-based solutions combined with an ensembling model, winning the contest.
Finally, in chapter 6 we summarize the results we obtained in order to contextualize them.

Chapter 2

Preliminaries on Machine and Deep Learning

Due to the great amount of data from the Internet, especially textual information, the usage of algorithms in this sense have been broad explored through Machine Learning and, from a different point of view, Deep Learning. This approach to textual data processing gave birth to *Natural Language Processing* and it enabled the resolution of important tasks and challenges, with the aim of achieving *Natural Language Understanding*, considered by the community as the "holy grail" of this field. In this chapter we provide a summary of the principal ML and DL algorithms, up to the latest and most widely used architectures with the best performance.

2.1 Natural Language Processing

Natural Language Processing is a subject related to linguistics, computer science and therefore artificial intelligence, that deals with the analysis of human-computer interaction through natural language, which can be in the form of sound or text. This type of technology is concerned with analysing and extracting information from natural language sources and often making predictions about the extracted data. The wide field of NLP covers a diversified set of tasks and problems which aim to different objectives; the most popular are:

- **Text classification**: it is the task that deals with processing unstructured data (such as texts) and making predictions and analyses on the characteristics extracted from them. This task is widely use in AI scientific community with broad applications other than fake news detection, such as sentiment analysis, topic labeling, spam detection, and intent detection.
- **Natural Language Inference** (NLI): it is the task of determining whether

an *hypothesis* query is an *entailment*, a *contradiction*, or *neutral* with respect to a given *premise* query. An example is shown in Figure 2.1.

- **Named Entity Recognition** (NER): it is the task which aims to identify and classify named entities (real-world object denoted with a name) mentioned in unstructured data; this can be done through a *grammar-based* or a *statistical-based* strategy.

- **Text pre-processing**: it is explained in subsection 2.1.1

Before diving into the numerous ML techniques that are used for classification, it is useful to understand the characteristics of unstructured data in exam and, in particular, how it can be processed in order to make it algorithmically usable by ML and DL techniques.

$\mathbf{P}^a$	A senior is waiting at the window of a restaurant that serves sandwiches.	Relationship
$\mathbf{H}^b$	A person waits to be served his food.	Entailment
	A man is looking to order a grilled cheese sandwich.	Neutral
	A man is waiting in line for the bus.	Contradiction
$^a\mathbf{P}$, Premise. $^b\mathbf{H}$, Hypothesis.		

Figure 2.1: NLI example

2.1.1 Pre-processing

Unstructured natural language, being such, is by nature ambiguous and sloppy; especially on the Internet, one can find such text modifications that make it difficult to process by current technologies: padding, unconventional characters, typos, wrong capitalisation, abbreviations, etc. Furthermore, it is good to remember that the algorithms do not operate on words or letters but on numbers - in particular in the case of Machine Learning on numerical vectors - therefore it is necessary to find a suitable technique for the conversion of sentences and words into numerical feature vectors that can integrate both their semantic content and their meaning within the context of the sentence. This must also integrate a model that can measure the similarity between terms.
For what concern natural language difficulties, we have numerous cases in which

the similarity of words in sense or form can cause prediction problems. Because of this, it might be useful to use a *dictionary* to distinguish the semantic meaning of various words. In this regard, it is important to identify how we categorise the words when they appear in textual data:

- **Lemma** is the canonical, dictionary or citation form of a word
- **Wordform** is the "inflected" word as it appears in the text
- **Word sense** is the discrete representation of an aspect of a word's meaning

Using these definitions we can define how to organise a possible dictionary: through *semasiological approach*[1] entries are arranged by lexemes or words and each of them describe their sense; on the other hand the *onomasiological approach*[2] organises the entries by senses or meanings. While constructing this dictionary, one must take into account the problems often present in written plain text, among them:

- **Homonymy**: it is a relation between words that have the same form but unrelated meanings, examples are *homographs* that have the same written form and *homophones* that have the same pronunciation. It is the main cause of NLP problems, especially in information retrieval, machine translation and text-to-speech algorithms.
- **Polysemy**: it is a systematic relation between words that share the same form and have *related* meanings or senses e.g. the word "bank" can mean both the building and the organisation depending on the context. This problem is often solved through Name Entity Recognition.
- **Synonymy**: it is a relation between words that share the same meaning but with different form in some or all contexts. Two lexemes can be defined as synonyms if one can substitute the other in some or all contexts.
- **Antonymy**: it is a relation between words that do not share meaning nor form, but their senses are strictly related since they represents opposites with respect to one feature of meaning.
- **Hyponymy** and **Hypernymy**: one word is defined as *hyponymy* of another if its sense is more specific, denoting a subclass of the other word. Conversely, *hypernymy* defines a superclass of the other word, having a wider sense.

[1] https://en.wikipedia.org/wiki/Semasiology

[2] https://en.wikipedia.org/wiki/Onomasiology

All these features make the creation of a dictionary difficult, which is why it is necessary to apply methods of **text normalization** the text in order to simplify it and remove unnecessary elements. The normalisation process consists of several steps, which are not always applied in their entirety to the whole text, but it depends mainly on data characteristics. The main steps are structured as follows:

- **Punctuation removal**: this is usually the first step to apply, it removes all the punctuation in order to simplify the text. The task may also include emoticon/emoji and HTML tags removal.
- **Stop word removal**: this step removes the words that are not useful for encoding and text understanding because they act as padding and/or they have low entropy.
- **Capitalization**: this concerns the substitution of all the capital letters with lower case ones. It is not always used because it often depends on the theme of the text being analysed, but it is a general rule to apply it for long texts.
- **Tokenization**: it is a process that divide the plain text/sentence in atomic elements that are semantically useful for processing, called *tokens*; punctuation and white spaces may or may not be included in the final tokens.
- **Lemmatization**: it is a process that apply a morphological analysis of the word, removing the inflectional endings of the word and returning only its *lemma*, so the canonical form.

After applying these processes, it is necessary to adopt a model for organising words so that they can be codified. In the field of NLP, there are two macro-approaches, which in turn contain more specific ones: the *thesaurus-based approach* and the *distributional approach*.

2.1.2 Thesaurus-based approach

A *thesaurus* is a set of terms used to classify documents and data in order to fed them to a model; more specifically, in this scenario, it is a hierarchically organized lexical database which contains words related among them. The most used is the *WordNet*[3], which is a database consisting of an on-line thesaurus that is enriched by some dictionary aspects. Wordnet is structured in a semasiological way, so each entry is a term whose different senses and meanings are defined. It has both instances and classes: instances define a proper, individual name that identifies a single entity, while classes are words that indicate a super-class, which may

[3]http://wordnetweb.princeton.edu/perl/webwn

contain several sub-classes and instances depending on the term. The hierarchical structure of the thesaurus makes it possible to establish and calculate a measure of similarity or, more precisely, relatedness, since similarity implies that the terms are near-synonymous, whereas relatedness implies that the words are related in any way. So, it defines that two concepts are similar if they are near each other in the thesaurus hierarchy: shorter is the path separating the terms/senses, more related they are. Figure 2.2 shows the logic of the path computing in the hierarchy. In formulas:

$$pathlen(c_1, c_2) = 1+\text{number of edges in the shortest path between senses nodes } c_1 \text{ and } c_2$$

$$simpath(c_1, c_2) = \frac{1}{pathlen(c_1, c_2)}$$

$$wordsim(w_1, w_2) = \max_{c_1 \in senses(w_1), c_2 \in senses(w_2)} sim(c_1, c_2)$$

WordNet has been also used as a corpus in Python language by the NLTK library[4], as well as in the Java language. Despite its structure, this approach has weaknesses, for example this paradigm assumes that each edge of the tree has a weight of 1 and for some sense nodes this may be misleading; furthermore, there is no thesaurus for every existing language and they must be hand-built. The next step to improve word encoding is the distributional approach.

2.1.3 Distributional approach

Distributional models are embedded vector-space models of meaning, since every word is embedded through a numerical vector in a multi-feature space. This means that words are represented with vectors and if they are related in the context, they are placed "nearby" in the feature space. Because of this, the approach tends to have lower precision but higher recall. The first and simpler type of embedding is the *Bag of Words* model: having a certain number of documents or sentences, each word is coded by the frequencies with which they occur in each document. The BoW model is also referred to as term-document matrix if the terms in the entries are not only single words but also combination of them as n-grams. Through this model, the calculation of similarity is more intuitive, since it is closely linked to the context in which the words are inserted: two words are similar in meaning if their context vectors are similar. Mathematically, this measure is computed through the

[4]https://www.nltk.org/howto/wordnet.html

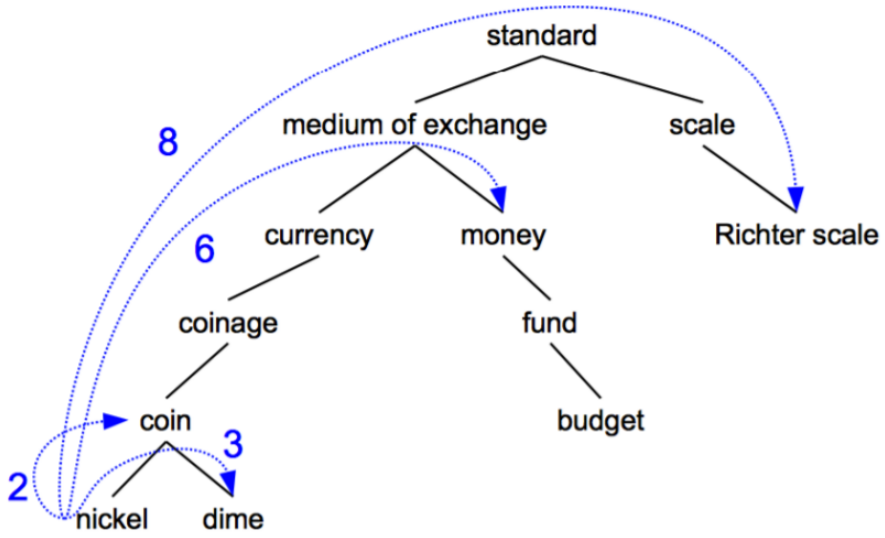

Figure 2.2: Logic of thesaurus' hierarchy

cosine similarity between vectors; in this way, the more two vectors have similar direction and versor, the more the two words are related:

$$cosine(\vec{v} \cdot \vec{w}) = \frac{\sum_{i=1}^{N} v_i w_i}{\sqrt{\sum_{i=1}^{N} v_i^2} \sqrt{\sum_{i=1}^{N} w_i^2}} \tag{2.1}$$

Although the frequency of words is useful for representation, if no stop word removal is applied, the most frequently occurring terms and lemmas are the ones that acts as paddings, so they are useless for the context, e.g. "the", "and", "among", etc. To overcome this problem, various types of function are often applied, in order to re-weight the vectors.

TF-IDF

TF-IDF stands for *Term Frequency-Inverse Document Frequency* and it is a re-weighting technique for term-frequency matrix in order to re-balance the importance of a single term in every document. Due to its configuration, it often results in very sparse vectors, but despite this [1] showed how this technique leads to decent accuracy and results in text classification field; for this reason it is often used as baseline system together with basic machine learning models. Its value for a term is proportional to the word frequency, but it is normalized by the number of documents in the database: in such way, the importance of a single term is

adjusted by its frequency in all the documents. Having the term t, the document d, and the total number of documents D, the expression is:

$$\text{tfidf}(t, d, D) = \text{tf}(t, d) \cdot \text{idf}(t, D) \tag{2.2}$$

The *term frequency* defines the frequency of a single word or n-gram that depends on the raw counts of a term t in a document d:

$$\text{tf}(t, d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}}$$

While the *identity frequency matrix* define the "importance" of the word across all the documents, i.e. how much the term appears in all the corpus, through the logarithmic scaled inverse fraction of number of documents that contain the word and the total number of documents:

$$\text{idf}(t, D) = \log \frac{|D|}{|\{d \in D : t \in d\}|}$$

So, words such as "the", "alone", etc. has a normalized count among all the documents, since they will tend to have low *idf* because of their high frequency in all the documents.
It often happens that a certain problem needs the normalized version of both the elements that compose TF-IDF: for *tf*, it is often logarithmic scaled or a double normalization is scaled, on the other hand *idf* is computed as a *probabilistic inverse document frequency*.

$$\text{tf}(t, d) = \log(1 + f_{t,d}) \qquad \text{idf}(t, D) = \log \frac{|D| - n_t}{n_t}$$

Where n_t is the number of documents where the term t is present. Regardless of the type of normalisation, the distance between words or n-grams can be calculated using the cosine formula 2.1 as in the previous case.
Other solutions for encoding words and phrases consist of processing them using more or less complex models in order to obtain *embeddings*: this is the case of **Word2Vec**[5]. This type of embedding is characterized by way shorter vector, but denser as well, so generalizing better the word features. Due also to its small size, this kind of vector is easier to process for machine learning models, since they will have less weights subject to tuning. The approach consists in fed a two layer neural network that is fed with corpus of text and returns a vector space where every word is represented. This technique has the advantage of being able

[5]Code: https://code.google.com/archive/p/word2vec/

to represent the linguistic context of individual words, thereby also being able to better distinguish synonyms and other similar semantic entities. Despite these advantages, it has drawbacks too: since it relies on local information, it cannot handle out-of-vocabulary words and it needs a larger corpus to be train to in order to avoid sub-optimal results. Moreover, parameters cannot be shared among language, so the model must be trained from scratch for each language.

2.2 Introduction to Machine Learning

In this section we briefly describe the concept of Machine Learning, while tackling specifically the problem of classification in a general way, how it is handled and evaluated. **Machine learning** is a sub-field of AI which has the aim to describe a problem in a logical form, so with uncertainty, and to apply general deduction procedures to solve it: in other words, it is a process that gives the ability to *learn* to a computer model. Having a ML model, the elements that it needs are the *data from experience* E, the *task to solve* T and the *performance measure* P. So, an *agent* learns when its performance in solving T, measurably P, improves with experience E. Moreover, experience E is described by *data points* or *instances*, which in turn are characterised by sets of qualitative or quantitative fields, called *features*.
Depending on how E, P and T are defined, we can have different approaches of learning, the four main types are presented here:

- **Supervised learning**: it is a learning process where the input data is characterized by its features and the labels (which can be categorical or a continuous real value); the model learns the relationship between the data points and the related label, applying this mapping process to new unseen data instances.

- **Unsupervised learning**: it is a learning process is a process in which the model is fed with data without related labels; so, in this case, the learning agent has to identify structures, such as patterns and clusters, within the data space, based on their features.

- **Semi-supervised learning**: it is a learning process that falls between supervised and unsupervised learning, since it uses a combination of labelled and unlabelled data in order to return better performances in specific tasks.

- **Reinforcement learning**: it is a learning process in which the agent is placed in an environment where each action can correspond to a cumulative reward and a current state of the surrounding conditions; its aim is to maximize the function that defines the reward.

Depending on the type of learning, we mention specific tasks that identify the most popular problems within the ML field:

- **Classification**: it is a specific task of supervised learning, where the data points are related to *categorical* labels and the aim of the learner is to optimize the function that links the data to their specific labels.
- **Clustering** it is the task of grouping the data instances in such a way that the points with similar characteristics belong to the same group, which are called *clusters*; in this scenario the data has no labels, so it falls in the unsupervised learning field.
- **Regression**: it is a sub-task of the supervised learning, since the labels attached to the data are real continuous values; in this case the task of the agent is to estimate a function that can approximate the relationship between the outcome of the data (the label, that is defined as *dependent variable*) and the features (which are defined as *independent variables*) based on a mathematical criterion.
- **Dimensionality reduction**: it is a process that concerns the reduction of the feature space, so the number of characteristics that a single data point has, in order to obtain a smaller set of *principal components* in which the data instances are distributed; this goal can be achieved through extraction, elimination or elaboration of the feature fields. The most popular mathematical tool in this sense is the PCA.

2.2.1 Supervised Learning

Since this work is focused on *text classification*, a more precise overview of supervised learning is provided. A typical supervised learning setup is composed by an unknown distribution of data, which is divided in *training and test set*

$$D = \{(x_1, y_1), ..., (x_n, y_n), (x_{n+1}, y_{n+1}), ..., (x_m, y_m)\}$$

where the samples from 1 to n are the ones belonging to the training set (x_{tr}, y_{tr}) and the others from $n+1$ to m are set as test set (x_t, y_t). So, it is a finite sequence of pairs $\mathbf{x} \in X$, $\mathbf{y} \in Y$ in the $X \times Y$ space, where X is the feature space, while Y is the set of possible labels or outcomes. Assuming that the data distribution D is defined as independently and identically distributed (i.i.d.), the main aim of the supervised learning model is to find a function f, called *hypotesis function*, that:

- belongs to the set of possible functions F ($f \in F : X \to Y$)
- learns a relationship between the features and the outcomes such that $y_i = f(x_i)$ $\forall i \in D$

- minimize a function that depend on the **loss**

The *loss function* is a relationship that defines the difference between the true labels and the predicted ones, in such way it describes how well the model make predictions on the data: $L(\mathbf{x}, \mathbf{y}, f(\mathbf{x}))$. The loss can be of different types depending on the type of task to be solved (binary, squared, hinge, etc.).
The minimized and optimized function that depends on the previously defined loss is the **Risk**, or **Generalization error**:

$$R_{L,D}(f) = \mathbb{E}[L(x, y, f(x)] = \mathbb{P}_{x \sim D}[h(x) \neq f(x)]$$

where $h = \arg\min_{h \in F} R(h)$ and the expectation is taken with respect to D. This means that since the data distribution is unknown and i.i.d., it can be defined as a random variable, consequently also the risk function is defined as such; so, in a probabilistic sense, the risk is the *expected value* of the loss function. The minimization of the risk is not a trivial problem, since the function h is not accessible, together with the distribution D, because the model has access only to the training data. In order to solve this optimization, different resolutions may be adopted: one of these is the *Empirical Risk Minimization*. The ERM concerns another definition of risk, defined as **Empirical Risk** or **Empirical Error**:

$$R_{\text{emp}}(f) = \frac{1}{n} \sum_{i=1}^{n} L(f(x_i), y_i)$$

As can be seen, the amount of available data - belonging to the training set - is used to compose an approximation of the risk that has to be optimized by finding the appropriate function f that minimize the ERM:

$$f^* = \arg\min_{f \in F} R_{\text{emp}}(f)$$

This optimization problem may often lead to a small generalization, since it will tend to *overfit* over the training data; **overfitting** means to learn too closely to the training data - even the noise - so, for future and unknown samples, the model will fail to fit them because of a very low generalization. In order to avoid this, some restrictions are often applied to some parameters, such as the solution space F; these limitations are called *generalization bounds*. The opposite problem, called **underfitting**, occurs when the model cannot capture the structure of data in a proper way due to a lack of parameters.
This situation can be explained also in error terms: once the solution of the optimization problem f^* is found, we can compute its risk as

$$R_D(f^*) = \epsilon_{app} + \epsilon_{est}$$

where ϵ_{app} is the *approximation error*, which defines the **bias** and it is due to wrong assumptions about the learning model and the data distribution; it is strongly related to the restriction on the complexity of the solution space F and consequently a high value of bias leads inevitably to underfitting, provoking a low value for the other error. This last one, denoted as ϵ_{est}, is the *estimation error*, which represents the **variance**. It depends on the model sensitivity to small fluctuation of the training samples, so it is strictly correlated to the data distribution D and its high value corresponds to low bias and leads to overfitting. The goal of the optimization problem is also to find a proper *bias-variance tradeoff* that can lead to high prediction performances, therefore to risk minimization, without generalize too much or too little.

2.2.2 Classification metrics

Before diving into the study of the data and its divisions, it is important to ascertain how the learning models will be evaluated, thus how we will assess the distance between the true labels and the predicted ones. All measures presented here are based on the binary concepts of positives and negatives, which can in turn be true or false, in short:

- True Positive (**TP**): number of samples for which the prediction is positive and the true label is positive
- False Positive (**FP**): number of samples for which the prediction is positive but the true label is negative
- True Negative (**TN**): number of samples for which the prediction is negative and the true label is negative
- False Negative (**FN**): number of samples for which the prediction is negative but the true label is positive

Given these parameters, we can define the *confusion matrix*, which namely is a table layout that allows the visualization of a classification algorithm; it consists in a 2×2 matrix where each row represents the cases of a true class, while the columns represent the instances of the predicted classes. Through this structure we can compute the main measures:

- **Accuracy**: it is the proportion of correct prediction over the data cardinality.

$$\text{Acc} = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.3}$$

- **Recall** (or **True Positive Rate**): it is the proportion of correctly identified positives over the actual positives.

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

- **Precision**: it is the proportion of correctly identified positives over all samples identified as positive.

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

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

$$\text{F1} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{2.6}$$

- **AUC score**: it is a measure based on the *ROC curve* (Receiver Operating Characteristic curve); the ROC curve is a graph that shows the relationship between the TPR and the FPR (False Positive Rate) at different classification threshold. The FPR is the proportion of the wrongly identified negatives over all the actual negatives:

$$FPR = \frac{FP}{FP + TN} \tag{2.7}$$

  The ROC space concerns a [0,1] square, so the *perfect prediction* is a function that covers the point in the upper left corner or coordinate (0,1) of the ROC space, while a diagonal line corresponds to a random guess of the prediction. Through this function it is possible to compute the classification goodness, namely the degree or measure of separability among the classes; this means computing the area below the function (*Area Under the Curve*). Figure Figure 2.3 shows some examples of ROC curves.

- **MCC** (Matthews Correlation Coefficient): it is a correlation coefficient between the observed and predicted binary classifications; it is considered an equivalent to a χ-square statistics for a 2×2 contingency table (such as the confusion matrix). Unlike the other measures which compute values between 0 and 1, it returns a value between -1 and +1: a result of +1 represents a perfect prediction, 0 loosely indicates a random prediction and -1 indicates total disagreement between prediction and reference value.

$$\text{MCC} = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{2.8}$$

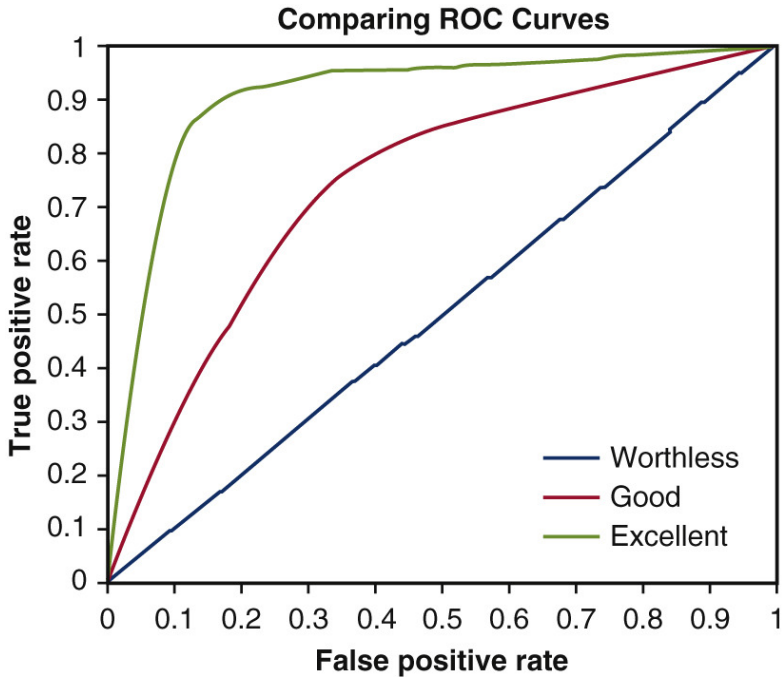

Figure 2.3: ROC curves

2.2.3 Training and validation method

The methods concerning the training and validation steps are divided into three parts: *dataset splitting*, *cross validation* and *hyperparameter tuning*.
The first step, the dataset splitting, is performed so that the model can better generalise over the data. The datatset is divided in two parts which we have already referred to above: the *training set* and the *test set*. In this way, the model is trained on the former and then tested on the latter, i.e. the algorithm parameters are adjusted based on the first, then it tries to predict the labels of the latter and its outcomes are then compared with the true labels, from which the performance is measured. The splitting is usually applied using 80% of the data as training, while the rest of it is used as test set. The split is performed in a *stratified* way, this means that the distribution of labels in the original dataset is also maintained in the resulting splits.
Despite this approach, the data that falls in the test set is unknown and the performance is strongly dependent on which data belong to this set, so the variance is high. In order to reduce this characteristic, the *cross validation* method is adopted: the training set is split again in training set and *validation set* in a stratified way. This last element is the part which is used to have a first glimpse of the model goodness, since the model is trained on the first set and then evaluated

on the second. Usually a specific version of this approach is used, called *K-fold cross validation*, which consists in several steps:

1. Split the training set in K subsets, which are called *folds*

2. The classification paradigm is trained on $k-1$ folds, while the k-th fold is used as validation set

3. The process is repeated changing the k-th fold in order to change the validation set

4. Eventually the results of the previous K validations are averaged

This approach allows a reduced variance and a more precise evaluation of the measure. Figure 2.4 represents a visualization of this approach.

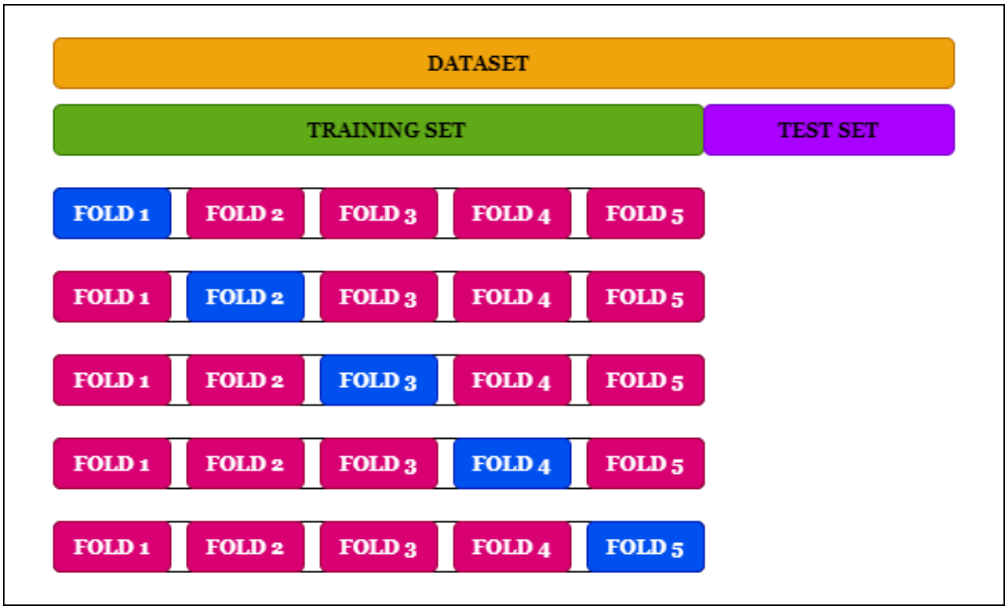

Figure 2.4: Visual of K fold cross validation

Moreover, this method is also used to choose the best model. A model is changed by modifying its *hyperparameters*: a hyperparameter is a variable specific to the model which is set before the algorithm processes the data and their value deeply affect its performance. The K fold validation is repeated several times in order to find the best values for the hyperparameters: this process is called *hyperparameter tuning*.

2.3 Machine Learning models

In this section we introduce some algorithms that have been used during the challenges with a brief description of their logic and parameters.

2.3.1 Tree-based algorithms

The tree-based methods are specific types of ML paradigms which involve the segmentation the feature space into regions through a set of splitting rules: this concept allows their representation as a *tree-like* graphic. These algorithms are really simple in terms of usage and representation, but they can have less accuracy performances respect to more sophisticated learning approaches; indeed, they are the more prone to overfit.

Decision Tree

The *Decision tree* is one of the tree-based methods that can be used both for regression and classification. The goal of this paradigm is to iteratively split the feature space in subregions R_i in order to minimize an objective variable. Displayed, trees are usually represented uspide-down, which means that the root is on top, then any points along the tree where the prediction space is split are defined as *internal nodes*, while the leaf nodes, where the regions are defined, are called *terminal nodes*.
The decision tree process can be described as:

- **Greedy**: at each step the node is split according to the best locally optimal decision, independently from the previous and future steps; thus, we have no certainty of reaching a global optimum.
- **Top-down**: it begins at the top of the tree with the root node and at each step it generates recursively two branches that correspond to splits of the feature space.

This greedy approach is strongly related to the *Hunt's algorithm*, which is the basis of the many decision tree models. In general the paradigm can be summarized in two steps:

1. It divides the predictor space into J distinct and non-overlapping regions R_1,R_2,...,R_J
2. For every observation that falls into the region R_j, it makes the same prediction: it assigns the most occurring label related to the observations that lie in that region

As stated previously, the locally optimal splits are made in order to minimize an object variable, which is linked to the concept of *impurity*. A node is considered *pure* if all of its data belongs to a single class, while it is *completely impure* if the node can be split evenly 50/50 based on a feature. There are several measurement of purity or impurity, but the most commonly used are:

- **Classification error rate**: the fraction of the training observations in that region that do not belong to the most common class

$$E = 1 - \max_k(\hat{p}_{mk}) \tag{2.9}$$

  where $\hat{p}_{mk}$ represents the proportion of training observations in the m-th region that are from the k-th class

- **Gini index**: it is a measure of total variance across the K classes. It can be seen as a measure of node purity, a small value indicates that a node contains predominantly observations from a single class

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}) \tag{2.10}$$

- **Cross validation**: it is a variation of the Gini index

$$D = -\sum_{k=1}^{K} \hat{p}_{mk} \log(\hat{p}_{mk}) \tag{2.11}$$

Bagging & Random Forest

Bagging (or *Bootstrap aggregation*) is a general-purpose procedure for reducing the variance of a statistical learning method. Since deeper trees are characterized by low bias and high variance, it is particularly useful and frequently used in the context of decision trees. The concept is statistical: if we have n observations with variance σ^2, the mean value of the observation will have a lower variance $\frac{\sigma^2}{2}$. So, averaging a set of observations reduces variance. Unfortunately, we do not always have access to multiple datasets; to overcome this problem, to solution is to *bootstrap* our training dataset, which means taking repeated samples from the single set. After this, having B bootstrapped datasets, we train our method on the b-th dataset, computing an outcome $\hat{f}^{*b}(x)$, which is a prediction of sample x. Then, in case of regression, an average of all the prediction is computed:

$$\hat{f}_{\text{bag}}(x) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}^b(x)$$

While, in case of classification, a *majority vote* among all predictions is performed. **Random forest** is an ensemble learning method for classification, regression and other tasks that make use of a set of uncorrelated decisions trees to construct a model during the training phase. In particular, it is based on the previously described bootstrapping technique, but it also introduces also an improvement that decorrelates the trees. As in bagging, we build a number of decision trees on bootstrapped samples, which are constructed over all the predictors; conversely, in this scenario a randomly subset of the features is selected, which cardinality is often $m \approx \sqrt{p}$, where p is the number of features in the dataset. The final prediction is chosen through majority voting, as shown in Figure 2.5.

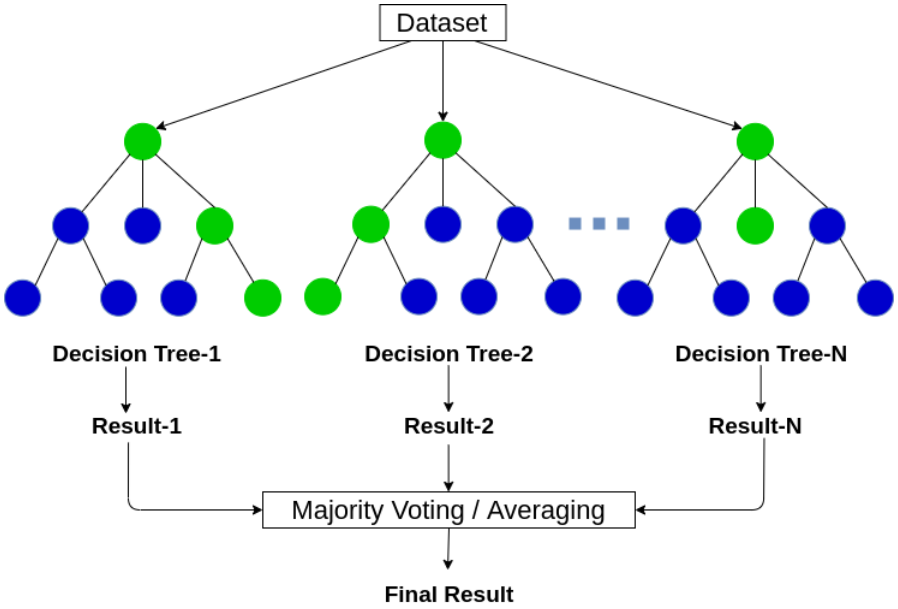

Figure 2.5: Bagging and random forest

Boosting & AdaBoost

Boosting is a general purpose method similar to bagging, but with the main difference that the approach is *sequential*: specifically, the concept is train weakly classifier in a sequential way such that every model will correct the misclassification of its predecessor. This approach allows to reduce variance taking into account the bias too, developing a procedure based on the bias-variance trade-off.
AdaBoost is a meta-algorithm based on boosting method which often use decision trees as weak classifiers: it takes as an input a distribution of data $S = \{(x_1, y_1), ..., (x_m, y_m)\}$, where $\forall i$ x_i are the features, y_i are the labels, m is the cardinality of the set and the weak learners are functions such that $y_i = f(x_i)$;

in addition, the procedure consists in T rounds. At each round a distribution of weights D_t is assigned to the data, and it is related to the learner; initially, denoting w_i^t as the weight for sample i at round t, all the weights are set equally, but they are updated at each round such that the misclassified samples' weights are increased, while the correctly classified ones are decreased. In such way, the algorithm forces the following learner to focus and fix the previously misclassified data points. Visualization of the process is shown in Figure 2.6.

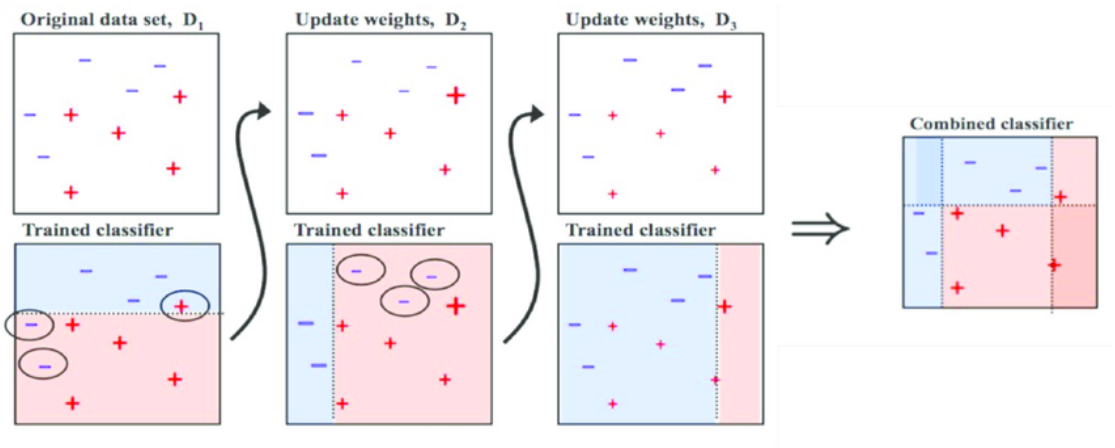

Figure 2.6: AdaBoost procedure

2.3.2 Support Vector Machines

Support Vector Machines are classification methods that aim to find the best hyperplane which can separate the data belonging to different classes, while maximizing the margin, which is defined as the distance between the separating hyperplane and the closest data points of any class, called *support vectors*.

Linear SVM

Linear SVM is sub-type of classifier that aims to find a linear separation among the classes and it is based on the *maximal marginal classifier*, in which we can distinguish *hard margin classifier* and *soft margin classifier*.
The first considers the *hard margin problem*, which is an optimization problem that can be solved if the data is linearly separable; thus, having a dataset $S = \{(x_1, y_1), ..., (x_N, y_N)\}$ where $\forall i\ x_i \in \mathbb{R}^d$ and $y_i = \pm 1$, the algorithm aims to find a linear hyperplane

$$\mathbf{w} \cdot \mathbf{x} + b = 0$$

or, in other terms

$$f(X) = \beta_0 + \beta_1 X_1 + ... + \beta_p X_p = 0$$

where $\vec{X} = x_i$, $\beta_0 = b$ and $\vec{\beta} = \mathbf{w}$, which is the unitary vector orthogonal to the hyperplane. The hyperplane sought must be such that it maximises the margin M and, at the same time, separate the data points belonging to different classes; this last objective can be achieved by adding the condition that $y_i f(x_i) > 0$, but since we are in the hard margin setting, the formula must be greater than M. Having these elements, we can formulate the optimization problem in its *primal form*:

$$\begin{aligned} & \max_{\beta_0,\beta_1,...,\beta_p} M \\ \text{s.t.} \quad & \sum_{j=1}^{p} \beta_j^2 = 1 \\ & y_i(\beta_0 + \beta_1 x_{i1} + ... + \beta_p x_{ip}) \geq M \\ & \forall i = 1, ..., N \end{aligned} \tag{2.12}$$

where N is the total number of samples in the training set. Figure 2.7 summarizes the method.

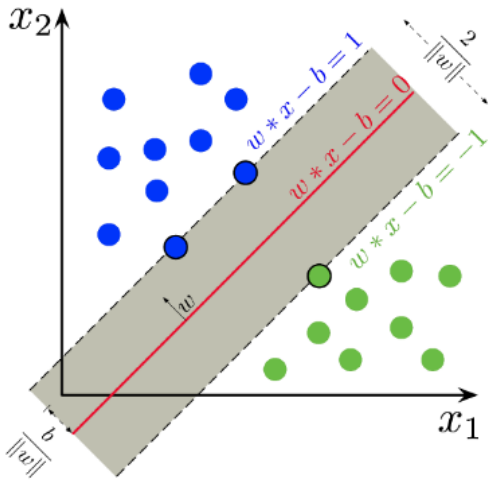

Figure 2.7: Linear Support Vector Machine

Despite this, most of real life datasets have no points that are linearly separable, so it is a very rare situation to correctly classify all the samples. In these cases, the aim is to focus on the trade-off between maximizing the margin and minimizing the misclassification rate: the algorithm can be adjusted by injecting in the marginal classifier a sort of error "*tolerance*" for the margin zone. This paradigm is called *soft margin classifier*. The tolerance for misclassification is inserted through some

non-negative variables, defined as *slack variables* ξ_i.; they allow some errors inside the margin and measure how much the constraint is violated. The optimization problem is structured as

$$\begin{aligned} & \max_{\beta_0,\beta_1,...,\beta_p} M \\ \text{s.t.} \quad & \sum_{j=1}^{p} \beta_j^2 = 1 \\ & y_i(\beta_0 + \beta_1 x_{i1} + ... + \beta_p x_{ip}) \geq M(1 - \xi_i) \\ & \sum_{i=1}^{N} \xi_i \leq C \\ & \xi_i \geq 0 \\ & \forall i = 1, ..., N \end{aligned} \tag{2.13}$$

C is defined as *cost variable*, it is an hyperparameter and regularization term, it defines how much the classifier is keen to "pay" to allow samples in the margin area. Higher values of C means higher slack variables, which consequently means that the classifier will allow more misclassification points, increasing the variance and the probability to overfit. On the other hand, if we decrease C, it will result in a narrower margin (smaller M), increasing the bias.

Kernel function

It may happen that the data is non-linearly separable: this means that regardless of error tolerance level, the classes cannot be separable through a linear hyperplane. In this scenario, SVMs can be used also to generate non-linear functions that help classifying the samples: in particular, the concept is to map the sample points into a higher dimensional space where there is at least one hyperplane that can linearly separate the classes. Figure 2.8 visualizes this concept.
Thus, having a dataset $S = \{(x_1, y_1), ..., (x_N, y_N)\}$, there exists a mapping function $\phi : X \to \mathbb{R}^n$ that maps the points creating the image sequence $\hat{S} = \{(\phi(x_1), y_1), ..., (\phi(x_N), y_N)\}$. Despite this, computing the transformation function may be computationally expensive, so a process called *Kernel trick* is adopted: instead of calculating all data projections, only the inner products between them are computed, using the *Kernel function*. To better understand this process, we define an inner product as:

$$\langle x_i, x_j \rangle = \sum_{k=1}^{p} x_{ik} x_{jk} \tag{2.14}$$

On the other hand, the Kernel function is defined as

$$K(x, x') = \langle \phi(x), \phi(x') \rangle \tag{2.15}$$

which is the inner product between the projection of two data samples. Using the *representer theorem*, we can write $\mathbf{w}$ as

$$\mathbf{w} = \sum_{j=1}^{N} \alpha_j y_i \phi(x_j) \tag{2.16}$$

Consequently:

$$\begin{aligned} \|\mathbf{w}\|^2 &= \langle \sum_i \alpha_i \phi(x_i), \sum_j \alpha_j \phi(x_j) \rangle \\ &= \sum_{i,j} \alpha_i \alpha_j \langle \phi(x_i), \phi(x_j) \rangle \\ &= \sum_{i,j} \alpha_i \alpha_j K(x_i, x_j) \end{aligned} \tag{2.17}$$

Thus, eventually the final function is dependent on Kernel functions; moreover, most of the α_i will be zero, since only the ones related to the support vectors will have non-zero values. The function is solved by replacing the inner products with a Kernel function: the type of separation depends on the chosen function.
Usually a Kernel function : $X \times X \to \mathbb{R}$ is symmetric and must obey to the *Mercer's theorem*, which defines that the matrix related to the Kernel function, called *Gram matrix*, must be *positive semidefinite*

$$G_{ij} = K(x_i, x_j)$$

The most popular Kernel function is the **Radial Basis Function** (**RBF** kernel):

$$K(x_i, x_j) = e^{-\gamma \|x_i - x_j\|^2} \tag{2.18}$$

2.3.3 Naive Bayes classifiers

Naive Bayes classifiers are a subset of ML models, intended as probabilistic classifiers, which are based on the strong naive assumption of independence between the features, so the presence of one of them does not affect the others. The classifier is based on the *Bayes rule*:

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

Having a dataset $S = \{(x_1, y_1), ..., (x_N, y_N)\}$, we can define $X = \{x_1, ..., x_N\}$ and $y = \{y_1, ..., y_N\}$; through these elements and exploiting the naive assumption of independence we can rewrite the Bayes function as

$$\begin{aligned} P(y|X) &= \frac{P(X|y)P(y)}{P(X)} \\ &= \frac{P(x_1|y)P(x_2|y)...P(x_N|y)P(y)}{P(x_1)P(x_2)...P(x_N)} \end{aligned} \tag{2.19}$$

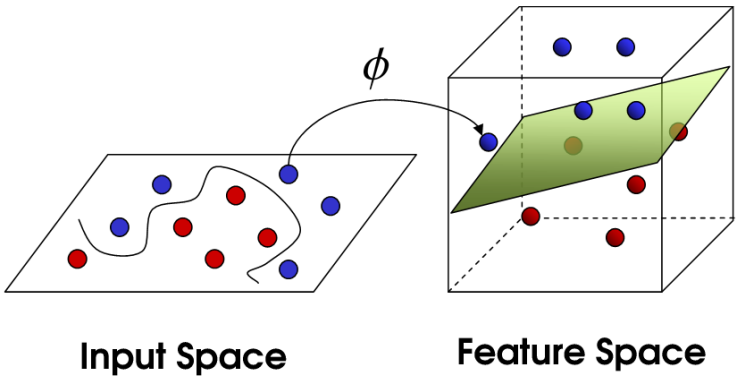

Figure 2.8: Kernel trick

Again, for the naive assumption, the denominator of the fraction can be overlooked by introducing a proportionality relationship, since it will never change:

$$P(y|x_1, x_2, ..., x_N) \propto P(y) \prod_{i=1}^{N} P(x_i|y) \tag{2.20}$$

After this, the classifier combines this with a decision rule, which usually it is to pick the most probable hypothesis i.e. label. This approach is known as *maximum a posteriori* decision rule, thus it needs to find the $\hat{y}$ with maximum probability:

$$\hat{y} = \arg\max_{y} P(y) \prod_{i=1}^{N} P(x_i|y) \tag{2.21}$$

By assuming or estimating the feature distribution, called *event model*, one can change the likelihood probability and replace it with a known distribution, depending on the type of features (continuous or discrete). The most popular ones are:

- **Gaussian**: the continuous values associated with each class are distributed according to a normal distribution with μ_y as mean value and σ_y^2 as variance

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}$$

- **Bernoulli**: it is a multivariate event model where features are independent binary variables describing inputs; this model is popular for document classification tasks.

$$P(x_i|y) = \prod_{i=1}^{N} p_{iy}^{x_i}(1 - p_{iy})^{x_i}$$

2.3.4 K-Nearest Neighbor

K-nearest neighbor is a non-parametric ML algorithm used both in classification and regression. The basic concept is to assign a label or class to a new data point based on the majority voting of its K neighbors, where K is an user-defined hyperparameter.
In order to return a proper classification, it needs:

- A set of **stored records**
- The **distance metric**, in order to compute the distances from the neighbors; this measurement can be user-defined, but the most used one is the *Euclidian distance*
- A **value** for K, which can affect the model: specifically, if it is too small, the classifier is more sensitive to noise and outliers, if on the other hand it is too large, the neighborhood may not be precise and include elements with other labels

Keeping these elements in mind, and having a new unseen data point, the algorithm can be decomposed in few steps:

1. Compute the distance between the new record and all the other ones
2. Identify the K nearest elements
3. Take the majority vote of class labels among the k-nearest neighbors
4. Weight the vote according to the distance d (with a weight factor of $\frac{1}{d^2}$)

Despite its simplicity and its versatility, it is very sensitive to noise and outliers. Moreover, it is computationally expensive due to the great amount of distances to compute for every new record; that is why it usually works better with small datasets.

2.3.5 Logistic Regression

Logistic regression is a generalized linear model used for classification tasks which is graphically structured as an S-shaped function between 0 and 1. It is structured in such way since it tries to assign a label by exploiting the probability that a sample has of belonging to a certain class: it computes if the probability related to one sample is greater or smaller than a certain threshold. This calculated probability is a conditioned one, so defining

$$p(X) = P(Y = 1|X)$$

as the conditional probability that x belongs to class 1 having seen the data x, logistic regression identifies this probability as

$$p(X) = \frac{e^{\beta_0+\beta_1 X}}{1 + e^{\beta_0+\beta_1 X}} \tag{2.22}$$

and it is used to assign a label to a sample. Then, we make a *logit transformation* of the previous equation obtaining

$$\log \frac{p(X)}{1 - p(X)} = \beta_0 + \beta_1 X \tag{2.23}$$

The final goal is to estimate β_0 and β_1 such that we can maximize the maximum likelihood estimator of observing the given class on the observed sample:

$$L(\beta_0, \beta_1) = \prod_{i:y_i=1} p(x_i) \prod_{i:y_i=0} (1 - p(x_i)) \tag{2.24}$$

2.3.6 Linear Regression Models

Linear regression models are a subset of models used in regression paradigms and often in classification ones too. The most basic type is the simple linear regression, which tries to analyse the probabilistic relationship between a set of predictors and their response. This model can be generalised through the following formula

$$Y = X\beta + \varepsilon \tag{2.25}$$

where

- **Y** is the vector of *response variable*, so a vector of random variables with dimensions $n \times 1$.
- **X** is the matrix of row vectors x_i containing the values of the p predictors, which are called *independent variables*, so its dimensions are $n \times p$.
- β is the vector of parameters, called *regression coefficients*, which are the goal of the statistical inference; each element refers to a predictor, so it has $p \times 1$ dimensions.
- ε is the set of non-observable random variables called *errors*: they take into account the uncertainty, the dimensions are equal to Y.

The learning step involves the resolution of the problem by finding the most appropriate vector of coefficient β such that the vector of error ε is at its minimum. The most popular way to solve this optimization problem is called *Ordinary least*

squares estimation: the solution consists in an optimization problem structured as follows

$$\hat{\beta} = \arg\min_{\beta} \|y - X\beta\|^2 \tag{2.26}$$

Solving the previous we obtain this estimated solution

$$\hat{\beta} = (X^T X)^{-1} X^T y \tag{2.27}$$

Despite the simple solution, it is based on the assumption that the features are independent of both features and coefficients, but this is not always true in real world problems. When features are correlated and dependent on each other, the elements are in a situation called *multicollinearity* and it may give problems in coefficient estimation and standard errors. In order to solve this situation, some regularization techniques can be adopted, such as *Ridge* and *Lasso*. Since only the first is adopted in this challenge, a brief explanation of it is provided.

Ridge

Ridge regression is a regularization technique injected in linear regression, in particular it uses an *L2* regularization term: it means that it adds an L2 penalty to the optimization problem, obtaining

$$\hat{\beta} = \arg\min_{\beta} \|y - X\beta\|^2 + \alpha \left\| \beta^2 \right\| \tag{2.28}$$

where α is an hyperparameter that concerns the coefficient shrinkage: the more α is near zero, the more similar the problem is to the ordinary least squares one; on the other hand, if α tends to higher values or to $+\infty$, then the coefficients tend to zero, since the weight of the penalty is higher.
Ridge paradigm is popular also in classification problem: in particular, for binary classifications, the labels are converted in $y_i = \pm 1$ and then the regression task is performed; the final class depends on the sign of the predicted value.

2.4 Introduction to Deep Learning

Deep Learning is a sub-field of Machine Learning that concerns *artificial neural networks*; these networks uses multiple layer entities that progressively extract high-level features from an input which can be an image, text, audio or other kind of data.
The underlying element of NNs is the *artificial neuron*: since NNs were conceived as inspired by the neural networks of the animal brain, i.e. interconnected entities that exchange and process electrical signals, an NN is composed by artificial neurons which are connected by edges and their signals are called *weights*. In order to better understand the underlying logic, an explenation of the *perceptron* is provided.

Perceptron

It is a basic classifier that artificial neurons use to elaborate inputs. The elements that characterize a perceptron are:

- A set of **inputs** $\mathbf{x} = \{x_1, x_2, ..., x_n\}$: these are the independent variables
- A vector of **weights** $\mathbf{w} = \{w_1, w_2, ..., w_n\}$: they are the coefficients which values are the goal of the optimization
- A **bias** b
- An **activation function** $\phi(\cdot)$: this is particularly important since it is the one function that allows the transmission of the signal, together with a *threshold*; if the input has enough probability to happen, it will activate the neuron and transmit the output. Different types of function can be used as activation function.
- The **output** y, which for the i-th perceptron is defined as:

$$y_i = \phi(\sum_{j=1}^{n} w_{ij}x_j + b_i) \tag{2.29}$$

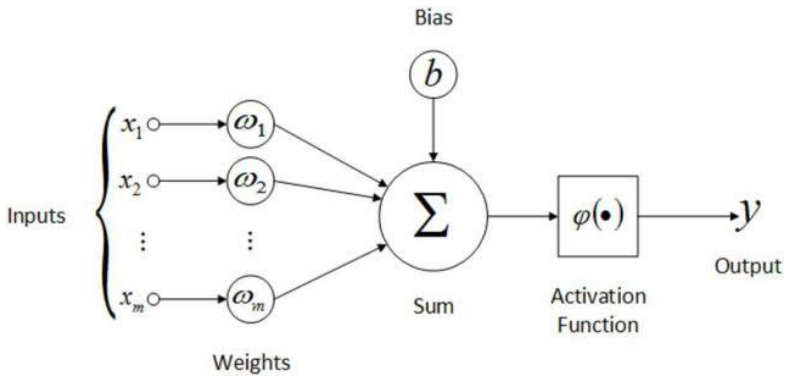

Figure 2.9: Basic structure of a perceptron

The basic concept is structuring a model that performs a weighted linear combination of its inputs and then applies an activation function: if the weighted input activates it, then it has enough probability to fire the unit and pass the output to the following perceptron. Figure 2.9 presents its basic structure. The goal is to have a threshold that allows the important part of the input - the signal - to pass through it and not the noise, and that makes it tolerant for small variations. Since the activation function is a non-linear one that has also the task to normalize

the output, its choice deeply affect the task performances and there are several different functions that can be adopted. Specifically, keeping in mind that in the following formulas $x = \sum_j w_{ij}x_j + b_i$, some options are:

- **Linear function**: it is the simplest function, since it returns exactly the value of the input:
$$\phi(x) = x$$
- **Step function**: it is the most popular function, if the linear combination overcomes a certain threshold θ, than the output is 1, otherwise is 0
$$\phi(x) = \begin{cases} 0 & \text{if } x < \theta \\ 1 & \text{if } x \geq \theta \end{cases}$$
- **Sigmoid function**, structured as:
$$\phi(x) = \frac{1}{1 + e^{-x}}$$
- **Hyperbolic tangent function**, defined as:
$$\phi(x) = tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$
- **ReLU function** (Rectified Linear Unit function), defined as:
$$\phi(x) = \max(0, x)$$

Figure 2.10 shows and summarizes the most popular functions.

2.4.1 Multilayer Perceptron

After presenting the basic element, the most popular and supporting structure is the *multilayer perceptron*: it is often used in classification frameworks where data is non-linearly separable. Regarding this structure, it is composed by *layers*, which are set of neurons, they are usually *fully connected*, i.e. each output or hidden unit takes as input all the outputs from the units at the previous layer. A single structure may have different number of layers depending on the task and its level of complexity: usually they are always composed by an input layer, which accept the input and it consists in a number of neurons equal to the number of dataset features, one or more hidden layers and a classification layer, which is the output one and it has a number of neurons equal to the number of classes. It is common to combine the output layer with a *softmax layer*, which is the layer in charge of

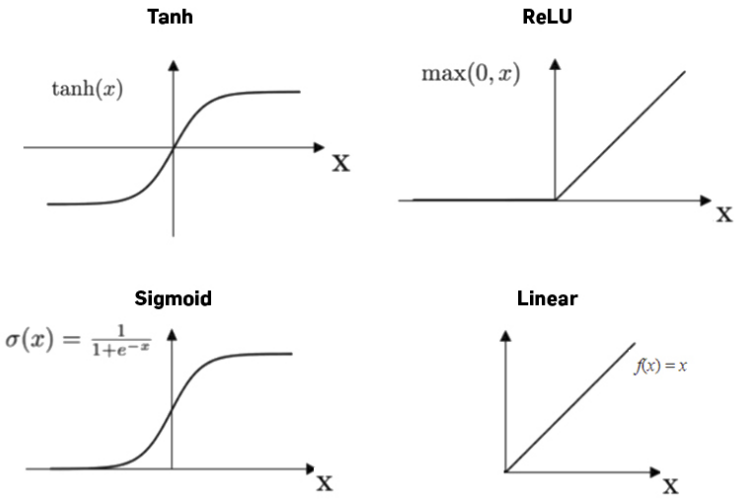

Figure 2.10: Examples of activation function

normalise the output in order to generate a distribution probability of the classes: in this way the output of the NN can be interpreted as a vector of probabilities, where the i-th element with highest probability is the class prediction.
It is noteworthy to say that there is a distinction between *shallow* and *deep* neural networks based on the task performed by the *hidden layers*:

- **Shallow architectures**: they are networks composed by feature extractors and a trainable classifier, which usually embeds a generic ML algorithm, so *features are not learned*.

- **Deep architectures**: they are networks which usually consists in several number of hidden layers and the ability to learn features through filters at different levels of semantic abstraction depending on the layer level, structuring in such a way a *feature hierarchy*.

The training phase of the MLP, or NN in general, consists in several steps, called *epochs*; the number of steps is an user-defined hyperparameter that depends on the task and on the performances. In each step there is the production of the output and a way to make the network learn, which is performed through the loss computing. In short we can identify two different phases:

- **Feed-forwarding** (or *forward propagation*): it is the process of computing the output of a network given the input; so, the input is fed to the first layer which consequently activates its nodes and the ones of the following layers until it reaches an output, which is used to compute the loss.

- **Back-propagation**: it is a chain rule-based process to compute the gradients of the loss with respect to parameters in a multi-layer network, in order to penalize the NN's behaviour, since the network *learns* if it minimizes the loss (plus some regularization terms) with respect to parameter over the training set. In case of classification tasks, the loss is a *Cross-entropy loss* defined as a negative log-likelihood

$$L(x, y; \mathbf{w}) = -\sum_j y_j \log p(c_j|x) \tag{2.30}$$

where y_j is the ground truth, $\mathbf{w}$ is the vector of parameters and $p(c_j|x)$ is the conditional probability related to the c_j class. In such way, the learning optimization problem becomes

$$w^* = \arg\min_\theta \sum_{n=1}^{N} L(x^n, y^n; w) \tag{2.31}$$

In order to solve this problem, the backpropagation technique is adopted, where the gradient of the loss is computed and it is used to adjust the value of the parameter (weight) vector through a Stochastic Gradiend Descent rule. To better understand, pseudocode for backpropagation (Algorithm 1) is provided in case of 4-layer neural network like the one in Figure 2.11.

Algorithm 1 Backpropagation

Initialize all the weights w to small random numbers
repeat
    Input the training examples and compute output o
    for each output unit k **do**
        $\delta_k \leftarrow o_k^2(1 - o_k^2)(t - o_k^2)$ ▷ t is the target output
    end for
    for each hidden unit h **do**
        $\delta_h \leftarrow o_h^1(1 - o_h^1) \sum_{k \in outputs} w_{h,k}\delta_k$
    end for
    $w_{ij} \leftarrow w_{ij} + \Delta w_{ij}$ where $\Delta w_{ij} = \eta \delta_j x^j$ ▷ η is an user-defined hyperparameter called *Learning rate*
until convergence

2.4.2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a type of deep NN with a specialized connectivity structure and multiple stages of feature extraction, often used in

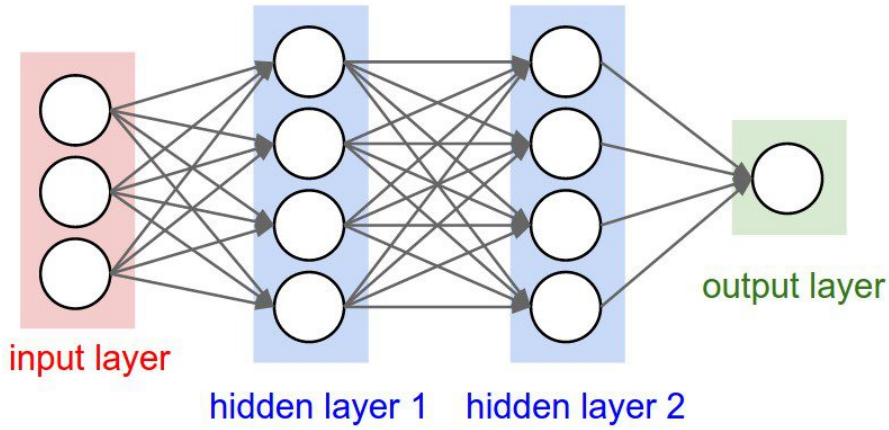

Figure 2.11: Neural network with 2 hidden layers

computer vision and NLP tasks, since it allows to embed relationships among elements at different positions; they are characterized by:

- **Convolutional layers**: as stated previously, feed-forward NNs usually have *fully connected layers*, where every neuron takes as input every output from the previous layer, and this implies a great amount of parameters for deep networks; on the other hand, CNNs have convolutional layers, where every neuron is in charge of a specific sub-window of data which "stride" through the set. Subsequently, the layer apply the same activation passage as a Fully connected layer and then it performs a dot-product between the extracted sub-window and a matrix called *kernel* or *filter*, obtaining a value what will compose the *feature map*.

- **Pooling layer**: after generating the convolutional output, a *pooling layer* may be insterted, which has the task to divide the output in other sub-windows and then collapse each of them in a single data point with a specific function (usually averaging, sum or max), giving robustness to the network.

- **Normalization**: Usually the output of the convolutional layers is always normalized through a *normalization layer*.

- **Non-linearity functions**: while in the fully connected layers a sigmoid or a tanh function is applied to data, in convolutional layers there is always a non-linear function, such as $ReLU$ or $Leaky\ ReLU$, the latter defined as

$$\phi(x) = \begin{cases} x & \text{if } x > 0 \\ 0.01x & \text{otherwise} \end{cases}$$

These are used in order to solve problems like *output saturation*, *collapsing of subsequent linear Fully connected layers* and the *vanishing problem* in the backpropagation phase.

Figure 2.12 shows a simplified structured performance of the convolutional layer.

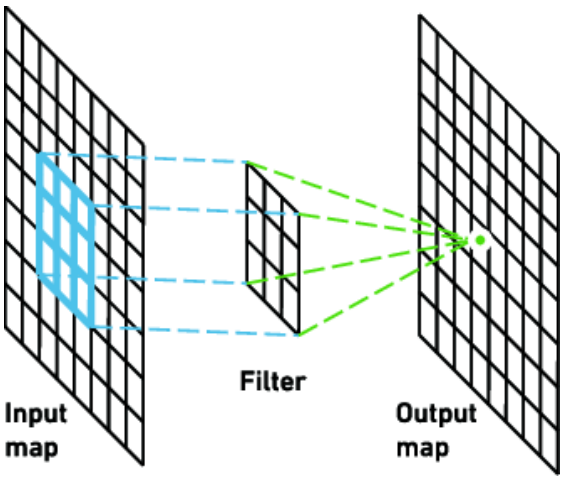

Figure 2.12: Actions of a convolutional layer

2.4.3 Recurrent Neural Networks

Recurrent Neural Networks are a class of deep neural networks that mainly works with sequential data, thus data with temporal characteristics. Several different types of RNNs can be identified depending on the number of inputs and outputs; the most popular is the *Vanilla RNN*, which is a one-to-one network.
The main feature of the RNN is to elaborate the inputs while maintaining an *internal state* which is recurrently elaborated with the input from the sequence at the time unit t. In formula:

$$h_t = f_{\mathbf{w}}(h_{t-1}, x_t) \tag{2.32}$$

where

- h_t is the current state at time t
- $f_{\mathbf{w}}(\cdot)$ is the activation function which also applies the product with the vector parameter $\mathbf{w}$
- h_{t-1} is the immediately previous state at time $t-1$
- x_t is the input vector at time t

The training phase of an RNN is equal to the one related to MLP or CNN, but the backpropagation phase is subject to a modification: the basic idea is to compute the loss through the entire sequence and then backpropagate it, but this is a technique that could be computationally expensive. So, the adopted solution is a division of the input: the sequence is divided in *chunks*, thus the loss and the backpropagation are computed and performed only for that specific chunk and the process is repeated for each of them. This technique is called *truncated backpropagation*.
In some tasks RNNs are often combined with *attention layer*, in order to identify the most important areas in an image or elements in a text through an attention distribution. Evolution of the RNNs are the *Long-Short Term Memories* (LSTM): they are specialized architectures characterized by two hidden state and 4 different gates with specific function; they have been created in order to solve the *vanishing and exploding problems* of parameters and to reduce the computational slowness of RNNs.

2.5 Transformers

In the field of NLP and for specific tasks such as machine translation, sequence modeling and text classification, research has always implemented complex versions of NN, in particular *Convolutional neural networks* and *Recurrent neural networks*. Despite they have been used as basic building blocks for state-of-art network, they had several drawbacks e.g. the number of operation performed to relate signals from two positions in the input sequence increases as the distance between positions grows; because of this, learning the dependencies among elements becomes more and more difficult for complex nets. Since then, state of art evolved due to the implementation of **encoders-decoders** models: an *encoder* is a network (FC, CNN, RNN, or other) that takes the input, and generates a feature vector as output; these feature vector holds the information that represents the input. On the other hand, the *decoder* is again a network (usually the same network structure as encoder but in opposite orientation) that takes the feature vector, and gives the best closest match to the actual input or intended output.
The current state of art for NLP is based on this model, since it is the basic block for **Transformers**. A *Transformer*, first presented in the famous work [2], is an encoder-decoder model based solely on attention mechanisms, specifically the *self-attention* (also called *intra-attention*), which is "an attention mechanism relating different positions of a single sequence in order to compute a representation of the sequence", as stated in [2].
Figure 2.13 shows the model structure. As can be seen it is divided in two parts: encoder and decoder. The general process involves two steps:

- The encoder maps the input sequence $\mathbf{x} = (x_1, ..., x_n)$ of symbols to a sequence

of representations $\mathbf{z} = (z_1, ..., z_n)$

- Having $\mathbf{z}$, the decoder takes it as an input and generate an output sequence $\mathbf{y} = (y_1, ..., y_n)$ of symbols one element at a time.

At each step the decoder part of the model is fed with the previously generated symbols as additional input. On the other hand, for what concern the structure:

- The **encoder** consists in $N = 6$ identical layers, which each of them is in turn composed by 2 sub-layers: the first is a *multi-head attention layer*, while the second is a feed-forward fully connected layer. A layer normalization and residual connections (i.e. blocks that allows data to reach latter parts of the neural network by skipping some layers) are added to each sub-layer, so the output of a single sub-layer can be summarized as

$$o(x) = \text{LayerNorm}(x + \text{Sublayer}(x))$$

- The **decoder** consists in $N = 6$ layers too, but in this case each layer is composed by 3 sub-layers: one of them is a multi-head attention layer, the last one is a feed-forward network, while the second in-between layer is another attention layer which is fed with the true output (offset by one position) and the encoder output. Moreover, it is modified with a masking technique in order to prevent positions from attending to subsequent position: this approach ensures that the prediction for a position i depends only on the known information at positions less than i. All the 3 sub-layers are provided with residual connections and normalization layers.

Multi-head attention layer

This layer and its approach is the basic block of the Transformer, so it is worth to delve into how it works. The layer works with the *attention function*, which is described as a mapping of a query and a set of key-value tuples to an output, which are all extracted components of the input embedding. A simplified structure is shown in Figure 2.14. This mapping process is performed by computing a weighted sum of the values, where the weights in object are computed through a compatibility measure between the key related to the value and the query.
In practical sense, the attention function is computed through the *scaled dot-product attention*: it performs a dot-product between queries and keys, with a normalization term that depends on the key vector dimension. The result of this multiplication is then fed to a softmax function, which output is the weight related to the values. The additional scaled factor is used to counteract a no-convergence situation: for small d_k values, the layer performs similarly to an additive attention layer, but

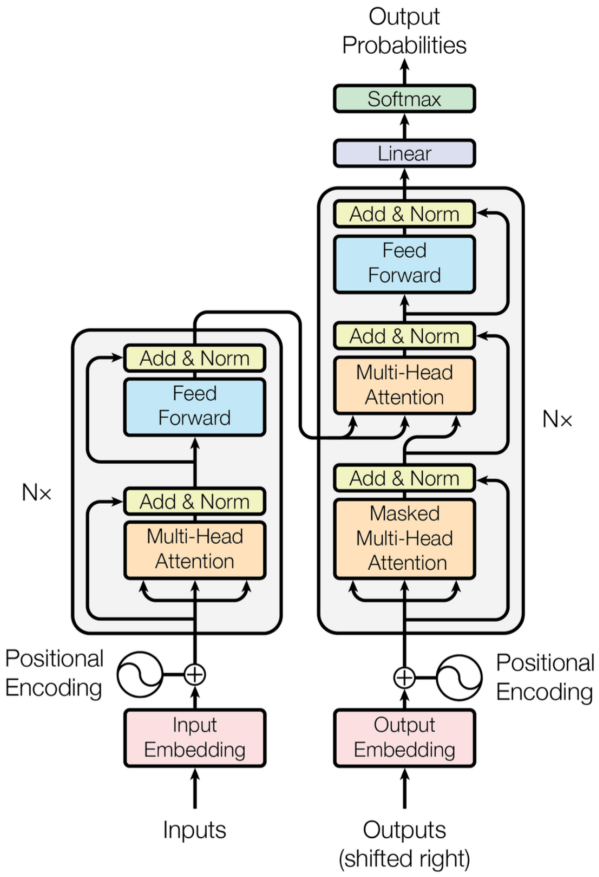

Figure 2.13: Transformer structure [2]

for larger values, the following softmax function may push the multiplication into region with small gradient values, which may provoke a slow convergence or no convergence at all. In formula:

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \tag{2.33}$$

where Q, K and V are the matrices related respectively to queries, keys and values; the first two have d_k dimensions while the latter has d_v dimensions. These matrices are extracted from the input d_{model}-dimensional embedding through a linear projection layer with proper weight matrices.
Eventually this process is repeated in the multi-head layer: the heads allow the model to jointly receive information from different representation sub-spaces at different position. The outputs of the several heads, which in the original work [2] are $h = 8$, are then concatenated and agaijn projected in a d_{model}-dimensional

space:

$$\begin{aligned} \text{MultiHead}(Q, K, V) &= \text{Concat}(head_1, ..., head_h)W^O \\ \text{where head}_i &= \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \end{aligned} \tag{2.34}$$

The multi-layer attention layers are used in various parts of the model with different inputs:

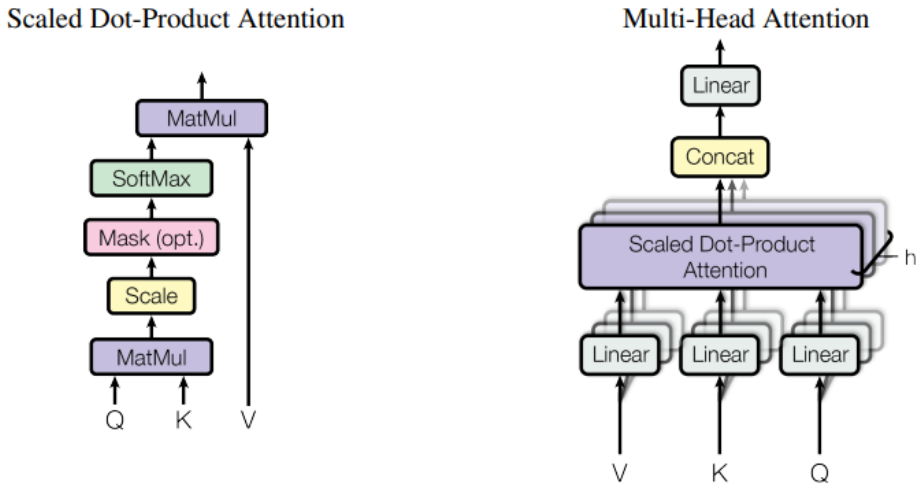

Figure 2.14: Scaled-dot product and Multi-head attention layer structures from [2]

- In the "encoder-decoder attention layers": they are placed in the decoder, but the queries are the outputs of the previous decoder layer, while keys and values are received from the output of the encoder. This allows every position of the decoder to elaborate all the position from the encoder, related to the input sequence.

- In the encoder: these attention layers receive all the matrices from the previous layer in the encoder itself.

- In the decoder: these layers are modified through a masking technique in order to make accessible only the positions up to the one that the decoder is currently elaborating; thus, queries, keys and values are received from the right-shifted embedding (the first solution token related to the first decoder step is always a padding one), and the masking is performed by setting to $-\infty$ all the values related to the illegal positions.

Other elements

Since the Transformer was born as sequence transduction model, it uses **learned embeddings** to convert the input and output tokens into d_{model}-dimensional vectors; on the other hand at the end of the decoder there are a **linear transformation layer** and a **softmax** layer in order to obtain a distribution probability about the most probable word prediction.
At the beginning of both encoder and decoder there is a **positional encoding** layer which add its output to the input embedding: this additional process is performed in order to inject the information about the relative and absolute position of the tokens. The value for the i-th dimension is computed though sinusoidal functions, which depend both on the position of the token and the dimension of the element in positional array, in this way both types of positions are encoded:

$$\begin{aligned} \mathrm{PE}_{\mathrm{pos},2i} &= \sin \frac{\mathrm{pos}}{10000^{2i/d_{model}}} \\ \mathrm{PE}_{\mathrm{pos},2i+1} &= \cos \frac{\mathrm{pos}}{10000^{2i/d_{model}}} \end{aligned} \tag{2.35}$$

Figure 2.15 shows how the values of the positional encoding sinusoid functions can be visualized on scale. Essentially, two tokens that are nearby in a sentence will have similar values when the sinusoidal frequency is low and when it is high, but their difference in position will be noticed at very higher values; on the other hand, two distant tokens are more likely to have different positional encoding values at much lower frequency values. In this way every element of the positional encoding array is computer for each token and than added to the related embedding vector.

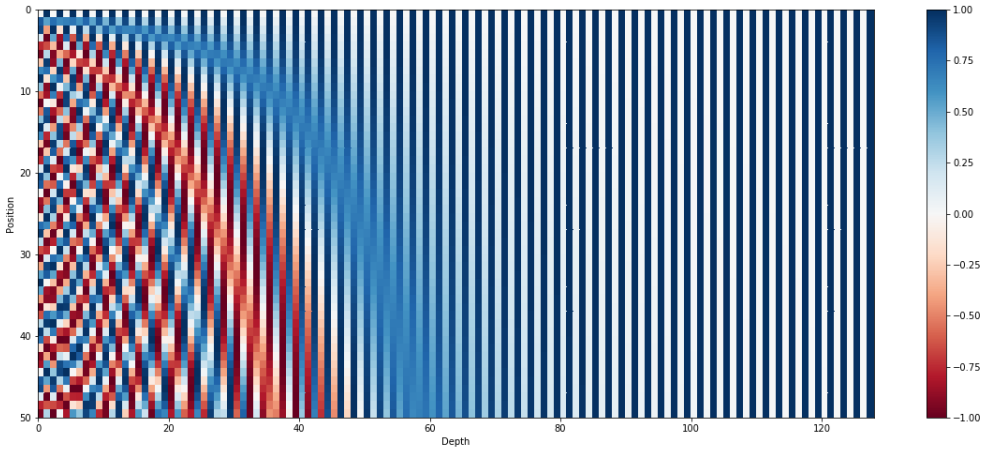

Figure 2.15: Positional embedding curves on scale

2.5.1 BERT

As stated previously, until the Transformers, language neural networks have been based on recurrent neural nets and LSTM, which despite reaching a decent accuracy, they have still some drawbacks, such as their slowness in training (words are passed and generated sequentially), and in particular, they are not truly bidirectional. These problems have been solved thanks to the introduction of Transformers, which are faster and deeply bidirectional, so they are able to understand the word and sentence context both from left to right and from right to left. Moreover, both of the core elements of the Transformer (encoder and decoder) have a general understanding of semantics and language, even though they have different tasks in the model.
The basic idea of the new NLP state-of-art, first introduced in [3], is to stack multiple times together one of this core elements: in case of **BERT** (Bidirectional Encoder Representations from Transformers), it is composed by several encoders stacked together. This architecture falls into the field of the *fine-tuned approach*, which divide the training phase into two sub-phases: a *pre-training* step that introduces tasks and specific parameters, and a *fine-tuning* phase, where the model is trained by fine-tuning all the pre-trained parameters with a downstream task. Before delving into its phases, we summarize its structure. One can refer to Figure 2.16 to better understand structures and tasks.

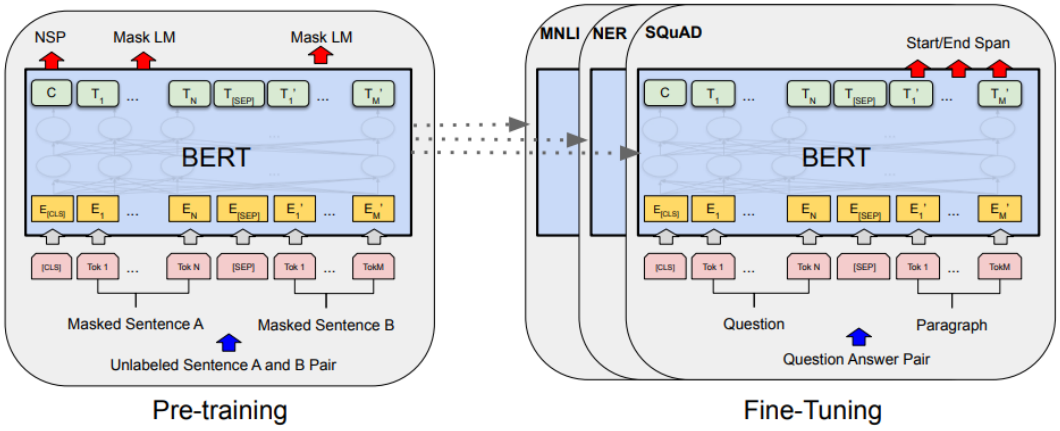

Figure 2.16: BERT structure from [3]

Model structure

The main feature of BERT is its unified structure even if it is trained on different tasks, so the differences between the pre-trained architecture and the final

downstream ones are minimal. As stated, its structure is derived from the encoder model from the Transformer structure, thus it is composed by several multi-layer bidirectional Transformers encoders. Depending on the number of Transformer blocks, we have different BERT configurations: denoting the number of stacked blocks as L, the hidden size as H and the number of attention heads as A, [3] presents two different architectures:

- $BERT_{BASE}$: $L = 12, H = 768, A = 12, \text{parameters} = 110M$
- $BERT_{LARGE}$: $L = 24, H = 1024, A = 16, \text{parameters} = 340M$

For what concern the *input/output representation structure*, we recall that the model must be trained over different tasks. Because of this, the input structure has to represent different elements always composed by tokens: the input can represent both a single sentence, but it can also represent a pair of sentences as a token sequence interspersed with some special tokens that defines specific entities. Its structure depends on the sub-task which the model is trained on. In order to convert the word into tokens, [3] uses WordPiece embeddings with a 30,000 token vocabulary. The special tokens are the first of the sequence `[CLS]`, which is a special token used for the binary classification task with one sentence input, while the second is a token that defines a separator `[SEP]`, in order to distinguish first and second sentence. Moreover, based on Figure 2.17, when it is codified the input structure involves different types of embeddings: the first is the *token embedding*, taken from the previously mentioned vocabulary, the second is the *sentence embedding*, which is a set of labels that define the first from the second sentence, and eventually, a *positional embedding* like in Transformer structure, explained in [2].

Training

Since BERT is based on a fine-tuning approach, we already stated how the training phase is sub-divided, we can differentiate two phases, which are also the reason why the model is so easy and quick to train and use: *pre-training phase* and *fine-tuning phase*.

Pre-training phase

BERT is trained through two unsupervised tasks simultaneously:

- **Masked Language Model** (MLM): nowadays language models can be trained in a bidirectional way in order to make them able to understand the context; despite this, this approach may allow each word to "see itself" and the task of word prediction could become trivial. In order to avoid this problem,

the solution adopted in BERT is to randomly mask a percentage of the token representations through a token mask defined as `[MASK]`. The model will have a vector representation of all the token words, then only the ones related to the masked tokens will be fed to a softmax layer with a number of classes equal to the number of tokens from the vocabulary; finally, the predicted token is evaluated through a *cross-entropy loss*. So, its final task is to predict the masked words instead of reconstruct the entire input. Nevertheless, this approach creates a discrepancy between the two training phases, since the masked tokens do not appear during fine-tuning. In order to attenuate this, the mask may be replaced: the training data generator considers 15% of the tokens at random, if a token is chosen, it is replaced with:

  - The `[MASK]` token the 80% of the time
  - A random token the 10% of the time
  - The same unchanged token the last 10% of the time

  Eventually, the original chosen token is used to evaluate the performance.

- **Next Sentence Prediction** (NSP): this task mainly concerns the relationship between sentences, which needs to be recognized through a binary output. The prediction task needs two sentences A and B as inputs and it has to return the label `IsNext` if B is the actual sentence that follows A, the label `NotNext` otherwise. The datasets analysed in [3] are composed from pairs of sentences where 50% of the time B actually follows A, in the other 50% of the cases B is a random sentence taken from the corpus. Specifically, the corpuses are the BooksCorpus with 800M words and English Wikipedia (only the unstructured text parts) with 2,500M words.

The introduction of these tasks with the related parameters reduce the need for a heavily-engineered task-specific architectures, using a single paradigm that can be fine-tuned on a specific objective.

Fine-tuning phase

The fine-tuning phase is more fast and less computationally expensive compared to the previous pre-training part: for general tasks, usually involving a single sentence, the model needs only a modification of the input and the output structures, which have to fit the task properties, and then the parameters can be fine-tuned end-to-end. When BERT is used for task involving more than one sentence, the self-attention mechanism become extremely useful to elaborate in a single step a pair of sentences and understand their relationship: this is done concatenating the text pair with self-attention, performing in such a way a *bidirectional cross attention* between the

sentences. Then, at output level, the token representation are fed to an appropriate output layer, while the `[CLS]` token is fed to a softmax layer for classification.

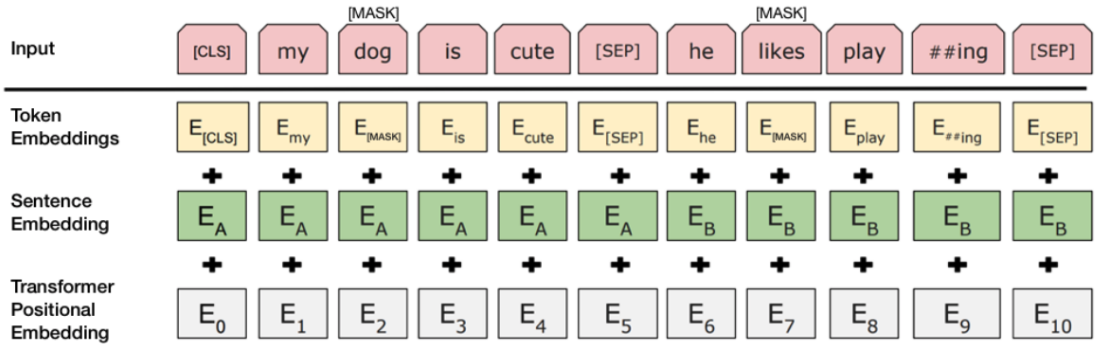

Figure 2.17: BERT input embeddings from [3]

2.5.2 RoBERTa

RoBERTa (Robustly optimized BERT approach), presented in [4], is a replication study of BERT pre-training involving the modification of hyperparameter tuning, training set size and task approaches, since the authors found that the previous Transformer-based model was significantly under-trained and they managed to improve the performances on several tasks by changing some parameters and paradigms. Moreover, the same paper [4] collected as much more data through five different corpora which concern different domains and consist in different sizes.
The training procedure experimentation involves different aspects of the model:

- Masking
- Model input format and NSP task
- Mini-batches size and text encoding

Next sections will delve in each of these elements.

Masking

The original implementation of BERT and the modified approach differ in how they mask the input tokens in order to pre-train the model for the MLM task: the original architecture makes use of *static masking*, which means that it masks a random token with a certain probability. To avoid the selection of the same masked token for each instance in every epoch, the data is duplicated 10 times, so that the sequences are masked in 10 different ways over the overall 40 epochs; but this

implies that the same mask is seen 4 times by the model.
The previous approach is substituted by the *dynamic masking*, which implies the generation of a masking pattern every time the model is fed with a sequence. Results in [4] shows that the re-implementation performs slightly better than the original one.

Model input format and NSP

In the original BERT configuration, the model is fed with two segments which could have come with 50% probability from the same document, or otherwise from different documents; moreover, they were concatenated through a separator token. The authors of [4] tried different configurations of input structures and they studied the effects of the Next Sentence Prediction loss, which is the one used to make the model able to learn. The different experimented configurations are:

- *Segment-pair with NSP loss*: this is the original BERT configuration
- *Sentence-pair with NSP loss*: in this case each input consists in a pair of natural sentences extracted from the same document or different documents; the pair is usually shorter than 512 tokens in its entirety.
- *Full sentences*: the input is composed from full sentences sampled contiguously from a single documents or more than one, but with the overall length at most equal to 512 tokens. The sampling can be cross document: if during the sampling process a document ends, the following sample is taken from the immediately following document. In this case the NSP loss is removed.
- *Doc-sentences*: the input is similar to the full sentences case, but with the exception that the sampling is not cross document boundaries.

The results show how extracting only natural sentences instead of segment worsens performance, while the NSP loss removing increase it. Overall the Full sentence paradigm is the best, but in the [4] experiments the full sentences approach is chosen because it is less computationally expensive.

Batches size and text encoding

Usually the increment of mini-batch size improve optimization and task performance and this is experimented also for BERT: the $BERT_{BASE}$ architecture is originally trained with a batch size of 256 sequences for 1M steps; this is changed to 2K sequences with 125K steps, since it is equal in terms of computational burden. For what concern the last point, the *Byte-Pair Encoding* (BPE) is an encoding representation that falls between the character and the word-level, and it allows

the representation of units in a large vocabulary from 10K to 100K sub-word units. The original BERT implementation uses a character-level BPE vocabulary with 30K units. In [4] the byte level of the vocabulary is increased up to 50K sub-word units; despite it is a feature that during experiment decreases the performance, the authors believe that benefits outweigh the accuracy discrepancy.

Eventually this combination of modifications is applied to BERT_{LARGE} architecture, obtaining RoBERTa, and the results are compared: the latter configuration provides a great improvement with respect to the performances stated in [3] in different fine-tuning settings.

Chapter 3

General background

In this chapter we introduce the concept of "Fake news", a broad argument that will be briefly described in order to understand the context and the motivation of the tackled challenges. Since this phenomenon has its inevitable consequences, we delve into the reasons why we should implement current technology to curb them, along with the challenges the scientific community is currently facing and the solutions they are implementing to solve them.

3.1 Definitions and motivations

The advent of the World Wide Web and in particular social networks has allowed individuals to freely express their opinions, while at the same time allowing everyone to access a nearly infinite range of information and sources: today, everyone has the potential to open an online communication channel and reach a wide audience as never before in history.
This has led on one hand to an exponential increase of freedom of expression, allowing everyone to be a *news medium*, leading to the birth of so-called *citizien journalism*. On the other hand, this allowed the rapid and wide dissemination of misleading and often dangerous information that have serious repercussions on both a social and political level. This had as a first consequence a *trust crisis*, where society found itself helpless in front of the manipulation of a large amount of information and secondly, the *Post-Truth Era*, where emotions have a preponderant power over facts. Notable examples include the emergence of fake news dissemination during the 2016 U.S. presidential campaign, during Brexit, and in the aftermath of the COVID-19 pandemic health emergency.
Furthermore, [5] showed how misinformation persists at a social level, even after it has been discredited: this social attitude is due to the causal inferences that individuals make based on available information about a certain event. However, the

same review also showed that it is possible to combat this behaviour by providing a causal explanation for the unexplained outcome. The emergency is not only worsened by individuals, but also by the media channels themselves, which often pick up news from citizens (and netizens), acting as external agents that intensify the echo of inaccurate information, acting complementarily to internal individuals, as demonstrated in [6].
These conditions have increased the interest of the worldwide scientific community in the topic of disinformation and in the ways to use modern technology to face the problem. This is not the first time that technology has been implemented to solve social problems: in fact, data from micro-blogging platforms have often been exploited to implement algorithms and databases to address relevant issues. For example, [7] shows how internet communication channels are particularly useful during mass convergences and emergency events, as they are often filled with information during these mass events; this data has been collected and can be used to implement autonomous systems that induce collective awareness and other humanitarian response efforts. Some solutions have been implemented even in the medical field: another noteworthy case is that of [8], where they present the implementation of a framework to predict possible influenza and H1N1 outbreaks in advance through mentions of the diseases on Twitter.

3.1.1 Fake news

The "Fake news" term has been widely used for years in both journalistic and everyday language to indicate a piece of news that is declared false by an authoritative source. It was elected Word of the Year 2016 by Macquaire Dictionary[1] and of the Year 2017 as well by Collins English Dictionary[2]. Despite this, the term is still extremely vague and it is used, especially in the media, as a buzzword to indicate a current socio-political problem. [9] has defined this term very precisely; first, the authors refer to the phenomenon as *misinformation* and they defined it as a misleading information that has two fundamental characteristics:

- The information is *false* (*Factuality*)
- The information spreads *deliberately to deceive and harm others* (*Harmfulness*)

Often when people talk about misinformation they refer exclusively to the first characteristic, the veracity of the information. However, we often overlook the most serious consequence of misleading information, namely the danger and harmfulness of its dissemination. The experiments presented here aim at limiting the spread of

[1] www.macquariedictionary.com.au/resources/view/word/of/the/year/2016

[2] blog.collinsdictionary.com/language-lovers/collins-2017-word-of-the-year-shortlist/

fake news considering also this second aspect.
Misinformation can be spread in a variety of modes such as text (e. g. viral claims, tweets), images (e.g. memes, fauxtografics), video (e.g. deepfakes) and sound (e.g. public speeches, political debates) in a multi-modal manner: each media channel has features and metadata of various kinds that can be extracted and used to predict their factuality or harmfulness, including features regarding the network in which the information spreads and temporal characteristics. Figure 3.1 summarises how multi-modal fake news is structured. Nevertheless, in the challenges presented in this work, only textual channels will be considered. Regardless of the type of media, [10] has shown that fake news spreads six times faster, deeper, and more broadly than the true ones in all categories of information, with a specific stress on political news.

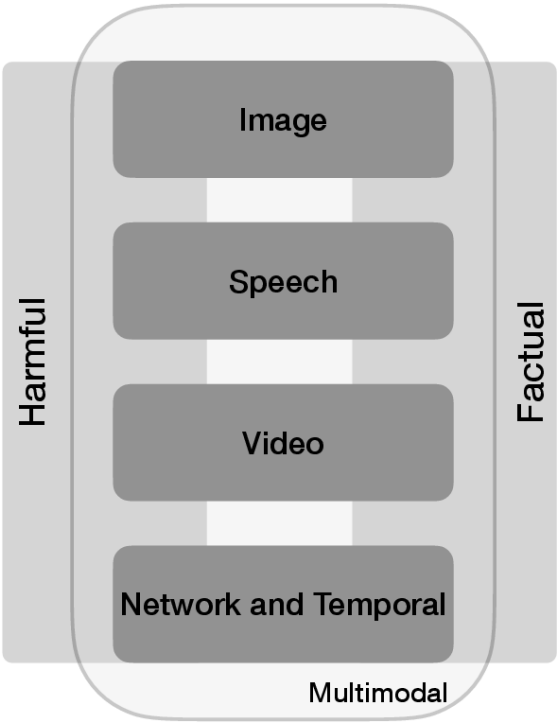

Figure 3.1: Envision of misinformation multi-modality

3.1.2 Fact-checkers

The misuse of the media that has led to the spread of misinformation has led to the emergence of a new profession: the *Fact-checker*. [11] presents an excellent definition of this new professional together with a survey of his/her work, in particular the FC is defined as "a professional whose main aim is to examine claims using

available evidence to assess their veracity". [12] presents a typical pipeline of FCs in assessing the veracity of a claim, divided into four steps: selecting check-worthy claims and news, constructing the appropriate questions to retrieve the correct information, obtaining evidence (e.g. previously verified claims, general evidences) from reliable sources and reaching a final verdict.
Given the widespread use of the web and social media, the fight against misinformation has become increasingly complex as time has passed in parallel with the complexity of the internet. This has led to the creation of public and private fact-checking organisations, some of the most relevant are *FactCheck.org*[3], *Snopes*[4], *PolitiFact*[5], *FullFact*[6], *Pagella Politica*[7], *EUfactCheck*[8], *Google Factcheck*[9] and *Hoaxy*[10].

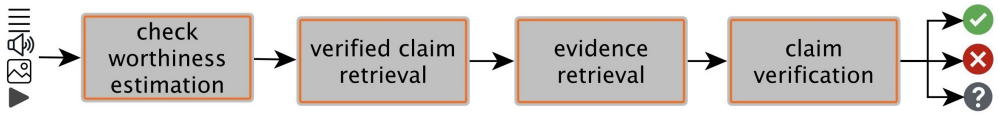

Figure 3.2: Fact-checking pipeline

Despite the presence of a lot of organizations which fight against modern misinformation is significant, their workers still have to face several issues about fact checking. Most of the difficulties for FCs are time-related and related to the resources that current organisations have. First of all, their activity is very time-consuming and this characteristic is in contrast with the very nature of the internet: as previously written, fake news spreads faster than normal news and moreover, focusing on the more general field of viral content, [13] shows that half of the spread of viral claims happens in the first ten minutes after their publication on social media. Secondly, the role of the fact checker has to be free of any bias in order to assess a claim or news item correctly: this, besides being time-consuming, implies that each step of the pipeline cannot be implemented on all data from the online world. Thus, many claims remain unchecked and/or unverified, as the amount of text data is much higher than what can be processed by human FCs.

[3] www.factcheck.org/

[4] www.snopes.com

[5] www.politifact.com

[6] fullfact.org

[7] pagellapolitica.it

[8] eufactcheck.eu

[9] toolbox.google.com/factcheck/explorer

[10] hoaxy.osome.iu.edu

This is where modern technology, and in particular AI, comes to the rescue: currently there are algorithms that can implement certain steps in the pipeline autonomously or can create a support system for human FCs (e.g. human-in-the-loop). However, these systems are not infallible or problem-free: in fact, many claims are not simply true or false, but their factuality may change or is highly dependent on the context from which they were extracted. For this reason, an autonomous fact-checking system is not considered as possible in the near future. To summarise, AI has to prove firstly that it can offer suitable solutions to the problems faced by FCs, and secondly its reliability as a support or as an autonomous system.

3.2 State of art

In this section we analyse the state-of-art solutions and projects that have been developed in recent years for the different steps of the fact-checking pipeline, together with the datasets often released together with them. In this regard, the analysis is divided according to the task for which the project was developed, thus differentiating the generated datasets which are freely provided to the scientific community, from the application solutions that focus mainly on the various steps of the pipeline described above: check-worthy claim and previously fact-checked claim detection, evidence retrieval and veracity prediction. Nevertheless, many of the projects mentioned cover more than one step in the pipeline, as they are strongly interrelated.

Datasets

The most important datasets used by the scientific community in the field of NLP and fake news detection are generally data collected by organisations working in the field of fact-checking, such as the ones mentioned above: *Snopes*[4], *PolitiFacts*[5] and *FullFact*[6]. The data collected by the aforementioned organizations is often used to create new datasets merging them with other textual data: the first part of [14] presents an example where sentences from political debates and politically oriented tweets are merged with sets of verified claims and annotations from *PolitiFacts* and *Snopes*.
Among the most important projects, it is worth mentioning the GDELT[11] project on which numerous studies and scientific papers have been based. GDELT is an international project that aims to monitor and collect data and events from news media from all over the world. Among the works based on the databases they provide the previously mentioned [7] and [15] are worthy of note. In the

[11] https://www.gdeltproject.org/

latter, a system for efficient in-memory analysis of data is presented, making use of high-performance computing hardwares and analysing correlation among GDELT data.
In other examples, data generation and/or its retrieval are integral elements of the project: this is the case of CLEF *CheckThat!* projects [16], [17], [18], [19] which focus mainly on the first two steps of the pipeline, but they also generate datasets by web scraping from social networks, often in English and other languages such as Arabic. In particular, [16] was the first edition in which a dataset corpus was created for the aim of the work, using both the transcription of political debates from U.S. presidential campaigns and the related annotations from *FactCheck*[3].
The broad topic of datasets does not only include the free provision of data to the scientific community, but also their collection and the consequent difficulties encountered during this process, such as complex APIs and data scraping limitations. For this reason, many projects include or focus exclusively on lowering these virtual barriers to allow teams of scientists to legally access more data and information. In this sense two examples are [20] and [21]: they present two different approaches to retrieve a sufficient amount of data from the social network Twitter. The first example is FACT (Framework for Analysis and Capture of Twitter Graphs) framework: it is a scalable framework for capturing and analysing information from Twitter, creating graphs of retrieved follower networks, posts and profiles in a multi-experiment environment; in this scenario an experiment is described as an user-defined code prescribing the data to be captured and the analysis to be run. Each experiment interacts with the Twitter API through a Proxy in order to retrieve the data, with a passive storage section where the structured graphs are stored and with a third party analysis component. All these elements are managed by an experiment wrapper. The second project is a technique designed to improve the FACT framework, in particular the management of many data-dependent experiments in a single closed research environment: the authors presents a improved Proxy in order to overcome the Twitter API limitations; in this setting, the Proxy server injects authentication information into the requests, managing all the quotas (rate limits for data retrieval) through a centralised API endpoint.

Check-worthy claim detection

As explained above, FCs often find it difficult to check and verify claims that proliferate online because of both the great amount of information that is generated, even in short time windows, and the speed with which claims, especially the ones derived by fake news, spread through the interconnected communities of the internet. So, the algorithms that can detect which claims are more worth fact-checking respect to others are of great help in this situation. In this sense, the problem is tackled as a ranking one, where the AI solution has to return a rank of the analysed claims with

the related check-worthiness score. This latter element is particularly important in order to generate a human-in-the-loop system, since the FC must be the last evaluator for the purpose of prioritise or filter claims.
The aforementioned CLEF *CheckThat!* projects include this first task in their objectives, often reformulated differently with different and innovative solutions for each edition. However, the basis of the task has always been constant, i.e. a list of politically oriented sentences or tweets: the best approach in the 2018 edition [16] constructed pseudo-discourses as a concatenation of all sentences and represented them through embeddings, part-of-speech tags and semantic dependencies. In 2019 [17] the best model was a neural network that learned domain-specific word embedding and syntactic dependencies whose output was fed to a LSTM classifier, performing in such way an individual classification. Finally, in 2020 edition [18] they used the state-of-art transformers model such as BERT [3] and RoBERTa [4]. As a final mention, we mention the work of FullFact[6] [22], which uses a BERT model to classify phrases from posters of various political parties, based on the type of claim e.g. a quantitative claim, a prediction about the future etc. This allows FCs to identify the most important and worthwhile claims based on their type.

Previously fact-checked claim detection

When fake news spreads, it often occurs in multiple channels and often undergoes a process of paraphrased repetition. This is closely linked to the speed with which they spread, which is greater than the speed with which normal news circulates, as mentioned above via [10]. Due to their speed and the time the fact-checking process requires, detection often comes too late: identifying whether a news item has been previously checked can both decrease the time spent in the process and at the same time anticipate the spread and the dangerous consequences of misinformation. From a FC point of view, the huge amount of claims that spreads online and on the newspapers increase the likelihood to find a claim that have been already fact-checked from a trusted source or organization. While, from a journalistic perspective, solving this task may be revolutionary since it can ensure that journalists are able to put politicians on the spot in real time during speeches and debates.
The world of research has only recently begun to take an interest in this issue, with one of the first to address this problem being the aforementioned [14]; the authors define the problem as follows: "*Given a check-worthy input claim and a set of verified claims, rank those verified claims, so that the claims that can help verify the input claim, or a sub-claim in it, are ranked above any claim that is not helpful to verify the input claim*". Again, as the previous task, the problem is tackled as a ranking one where an input claim is needed along with the previously

checked textual dataset. As stated previously, the authors in [14] created and made freely available to the scientific community two datasets consisting of tuples of verified claim-sets. In addition to this, the input claims are strictly correlated to an article (often it is extracted by it) with related features, composed by Title, Verified claim and Article Body. Exploiting the similarity analysis between the input claim and these elements (or a combination of them) they tried different models and paradigms to retrieve a ranked list for each input. The models tested are the BM25 [23],[24] and various variations of the BERT model [3]. The first is a classic information retrieval approach: BM25 produces a rank through a similarity score which is computed for each tuple based on exact matching between the words in the query and the words in a target document, that in this case may be simply the verified claim or a representation of it through the combination of two or more of its features. On the other hand, the BERT models are used as a sentence encoder to obtain a representation, then a cosine similarity between the claim and a representation of the verified claim is computed to obtain a score. A third model is taken into account by combining the previous two ranked list through a re-ranking algorithm, which learns to rank using a pairwise loss.
The task was also included in the aforementioned CLEF project [18], where the best solution in 2020 was a specific variation of the BERT model (top-ranked fine-tuned RoBERTa).

Evidence retrieval

Although it is difficult for FCs to check the volume of claims available online, evidence retrieval can be an excellent tool that can be used to facilitate the process. This step is particularly useful when the evidence to be retrieved is found in very long documents, audio-visual recordings or simply information in languages different from the ones which the FCs are familiar with. Combining this tool with other factors such as summaries or machine translation can create a supporting structure. The aim of the task is to find external evidence in order to support or deny the claim under examination. So, the inputs are the claim and a data collection of evidences, meanwhile the process must produce a ranked list of the relevant data or a specific piece of data from the collection: in this sense the task can be categorized as a ranking problem.
Initial work in this direction has purely focused on a similarity calculation at document and sentence level in order to retrieve the most relevant documents: [25] applies this approach of a clickbait detection scenario, in particular exploiting a TF-IDF matching between the title of the article and the body of the article itself to identify its stance and determine whether the two elements are related to each other.
This piece of work is also included in the multi-task project CLEF in 2019 and

2020: in the first, [17] shows that the retrieval of evidences had to be applied on a set of Web pages e.g. "*Given a claim and a set of potentially relevant Web pages, identify which of the pages are useful for assisting a human in fact-checking the claim. [...]*": the task has been solved using a textual entailment. The following year the proposed solution shown in [18] concerns "*ranking a set of verified claims, so that those that verify the input claim (or a sub-claim in it) are ranked on top*"; the most effective approach used a fine-tuned RoBERTa.
Finally, one of the most comprehensive works dealing with both the current task and automatic verification is the FEVER project shown in [26]: in this work, the authors focused on the extraction of unstructured evidence (e.g. sentences) from a corpus containing the set of articles written on Wikipedia, in order to use them to verify the veracity of claims. The evidence retrieval phase was developed using a cosine similarity between TF-IDF encoding of the claim and the pages in order to retrieve a ranked list of them and then the same score computation is performed to obtain a ranked list of sentences from the previously selected articles. The evolution of this project has resulted in the baseline system described in chapter 4.

Automated verification

This last task is the ultimate goal of fact-checking, i.e. to speed up the work of the FCs in order to develop a system that can pre-emptively identify false claims. As mentioned before, there are several obstacles to overcome: firstly, current technologies do not allow us to develop a system that is able to identify false claims with absolute accuracy; secondly, many claims can be true, false or partially true/false, or misleading or decontextualised, which increases the complexity of task automation. In addition to this, fact checking organisations need to be trustworthy out of respect for their readers, and unfortunately AI solutions alone do not yet enjoy this trust. Despite this and the imperfections of the applications, it is not excluded that a support system for FCs could be developed, since it may be useful in presenting evidences, reasoning and a possible conclusion regarding a claim under examination.
In this sense, [11] differentiates between two types of approaches: *explainable* and *non-explainable*. The former are also referred to as *reference-based approaches* and are generally the most widely used, as they seek to generate a verdict on the claim by exploiting a reliable resource, such as tables or knowledge graphs, or a database. This approach also includes two techniques analysed in the previous paragraphs, i.e. the search of previously checked claims as in [14], where the authors perform a search within a dataset starting from a single claim, and fact-checking through external databases, as occurs in [26], where following the retrieval of evidence, a Transformer gives as output a label that defines whether the evidence *supports*,

rejects or does *not have enough information* on the claim under examination. Non-explainable approaches, on the other hand, make a prediction based on documents retrieved from social media or the web in general by studying the content of the claim and its dissemination. An example is presented by [27]: in fact FANG, despite being a project focused on representation, manages to create an inductive graph learning framework that effectively captures social structure and engagement patterns; together with this, the representations learned by FANG manages to generalise to the related task of predicting the factuality and veracity of a media.

Chapter 4

FEVEROUS challenge

The FEVEROUS (*Fact Extraction and VERification Over Unstructured and Structured information*) challenge[1] is a competition held in 2021 based on fact news detection: the aim of the project is to to evaluate the ability of a system to verify information using unstructured and structured evidence from Wikipedia. The challenge is based on the project presented in [28]. The present chapter deals with a brief description of the task objectives, then it presents the structures of FEVEROUS dataset and corpus, followed by the explenation of the benchmark and our solution, together with the results. Our work and results are presented in [29].

4.1 Task description

In fake news detection through evidence retrieval, previous benchmarks and experiments have mainly focused on textual data, so unstructured information, rather than more structured formats such as table, lists, etc. This new project highlights the importance of exploit structured data in NLP tasks and text classification field, given also the increasing diffusion of datasets with structured information of this kind. [28] presents a project in which a novel dataset and a baseline benchmark are introduced, in order to tackle the fake claim detection task by selecting and retrieving unstructured and structured evidences from a corpus based on the English Wikipedia. The ultimate objectives of the challenge are:

- Properly retrieve a set of evidences E from the Wikipedia corpus
- Assign a label y to the claim in exam: a record can be labelled as "*Supports*", "*Refutes*" or "*Not Enough Information*" (or NEI), so $y \in \{\text{Supports}, \text{Refutes}, \text{NEI}\}$|

[1] https://fever.ai/task.html

Each claim may need one or more evidences to retrieve, which can be structured, unstructured or both in the case of multiple information. The evidences needed for a single claim could be recovered from a single Wikipedia page or multiple web pages from the same corpus.

4.2 Dataset

Claims

The FEVEROUS dataset consists in 87,026 claims divided in training, development and test set with a ratio respectively of 0.8, 0.1 and 0.1. As will be explained, each group of three claims is generated by an highlight from a Wikipedia page: the division is made in such a way as to assign each group to the same split. Since there is an evident lack of NEI samples with respect to the other two classes, a slight more balanced split has been performed for the test set, both regarding the classes and the type of evidence needed to predict claim factuality (some claims need only unstructured evidence, others only structured evidence, others a mix of the previous two). The split is summarized in Table 4.1.

Labels	**# of claims**	**Train**	**Dev**	**Test**
Supports	49,115 (56%)	41,835 (50%)	3,908 (50%)	3,372 (43%)
Refutes	33,669 (39%)	27,215 (38%)	3,481 (44%)	2,973 (38%)
NEI	4,242 (5%)	2,241 (3%)	501 (6%)	1,500 (19%)
Total	87,026	71,291	7,890	7,845

Table 4.1: FEVEROUS dataset splitting per label

Moreover, a data augmentation concerning the NEI samples was performed: as can be seen from Table 4.1 the dataset results unbalanced, only 5% of it are NEI, so the authors of [28] sampled additional NEI instances for training by modifying annotations that contain both structured and unstructured evidences and then removing either a sentence or a structured information (table or list) from it.
The generation of the dataset was done through a group of annotators hired by the authors themselves and it consists in three phases: *generation*, *verification* and *quality control*. The annotators were provided with a dedicated interface built on Wikipedia software called Mediawiki[2] in order to facilitate the search for information. As mentioned before, in the *generation phase*, each annotator is provided with a *highlight* from a Wikipedia page, which can be a set of four

[2]www.mediawiki.org/wiki/MediaWiki

sentences or structured information. For each provided highlight, they had the task of writing three different claims with the following rules leading to three types:

- **Claim using highlight only** (Type I): the claim must be generated using only information from the highlighted evidence and their context, possibly combining information and characteristics from the four sentences/tables.

- **Claim beyond the highlight** (Type II): this type must be based on the highlighted information and the ones beyond them. The claim can be generated either by modifying the previous type by adding other information or creating an unrelated claim which must still include insights from the highlights, enforcing the information retrieval from other Wikipedia pages in half of the cases.

- **Mutated claim** (Type III): this last type involves the manipulation of one of the previous two claims using one of the proposed 'mutations', *More specific*, *Generalization*, *Negation*, *Paraphrasing* or *Entity substitution* with probabilities 0.15, 0.15, 0.3, 0.1, 0.3.

Following the generation, the claims are subject to a second phase of *verification* where annotators are asked to determine the label of the claim based on the evidence they can find on Wikipedia. For supported claims, every part of the claim has to be verified through evidences, while for refuted claims, they only need to refute at least one part of it. A claim is considered unverifiable (NEI) if the information to precisely label the data is not found or if the information found is insufficient.
Finally, the last phase of *quality control* is performed simultaneously to the previous steps. This process consists in evaluating the quality of different factors that contributed to the dataset creation process:

- **Annotators**: each claim were generated and verified by different annotators, all US-English speaker or at least language-aware, which are in turn supervised by other managers.

- **Calibration**: annotation is a complex process, so training and selecting annotators are two phases performed together with a calibration procedure.

- **Quality assurance**: after the generation, each claim were checked by claim verification annotators in order to identify and report the ones that do not meet the claim requirements.

- **Dataset artifact and biases**: the association between several claim-related variables are measured in order to avoid dataset artifacts and no co-occurrence has been found between the verdicts and the claim words.

Corpus

As stated previously, the corpus from which the algorithm have to retrieve the pieces of evidence is a dataset of about 60GB based on the English Wikipedia collection of pages. The evidences can be structured as:

- **Sentences**: the unstructured information are any given piece of text from a Wikipedia page, considering also the sentences that refers to other pages (hyperlinks are maintained)
- **Tables**: the structured information considered in the corpus have various formats, they can be tables, lists and infoboxes, even with complex structures such as multi-headers. A general structured evidence consists in a set of cells denoted as c_{ij} where i and j corresponds to the row and column indices where the cell is placed; a caption q is related to the cell, it is usually the content of the cell, which can be a word, a numerical value or an entire sentence. To retrieve the evidence, the algorithm has to extract the cell which the related caption can help evaluate the claim, instead of extracting the table in its entirety.

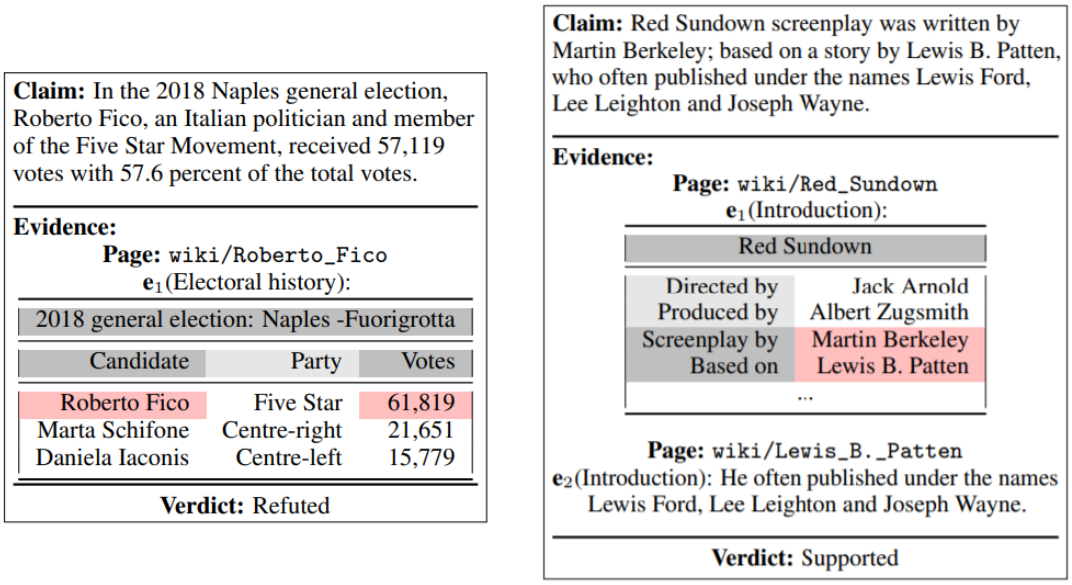

Figure 4.1: Evidence examples from [28]

Despite the concept of evidence allows the presence of hyperlinks (only to other Wikipedia pages), it does not include the references that are common in the Wikipedia site. Examples of claim-evidences pairs are shown in Figure 4.1.

Each piece of evidence is also addressed with its *context*: this consists in principal elements extracted from the Wikipedia page which the evidence belongs to; usually these elements are the article's title or the section title, including the sub-sections. In the case of structured information such as tables or lists, the context includes the nearest row and column headers; if the selected row/column is preceded by another header, the latter is also included in the context too. This is a particularly important piece of information, since it helps the model to understand relevant features of the evidence, even though it is not considered as evidence by itself.

4.3 Baseline

Before presenting the main baseline, the authors of [28] experimented some simpler models in order to compare the performances with the [26] ones: they developed a *claim-only baseline* and *claim-only evidence type model*. The first baseline is a model which is fed only with the claims and it aims to predict the verdict label: this is implemented through a fine-tuned BERT with a classification layer on top of it and the accuracy results similar as the one implemented in [26]. The second architecture is instead a model which takes as input only the claim, but it has to predict whenever the sentence requires a sentence, cells or a combination of both as evidence types. They developed it in the same way as the previous model and it reached a 62% of accuracy.
The real baseline model implemented for the main objective of the challenge is shown in Figure 4.2 and it has a structure based on two elements:

- **Retriever**: this element has the task to retrieve evidences from the Wikipedia corpus, firstly recovering pages and secondly retrieving specific elements such as sentences and cells from the previously selected pages. The extraction of the k pages is performed through an *entity matching* between the claim and the Wikipedia page's title. If it receive less than k pages, the rest of them is retrieved through a *TF-IDF matching* between the first sentences of the article and the claim. After obtaining the pages, the top l sentences and q tables are retrieved via TF-IDF again. k, l and q are user-defined hyperparameters which are set respectively to 5, 5, and 3 in the baseline model. Once the evidences are retrieved, the q tables are linearized, concatenated to the input claim and fed to a RoBERTa model, aiming to extract the most important cells through a binary classification for each cell.

- **Verdict predictor**: the label prediction is performed through a RoBERTa model with a linear layer on top of it. The input are the concatenation of claim, sentence evidences and cells, enabling in such a way the cross attention between the evidences and the input claim. It is worth noticing that every context information is concatenated to the related evidence in this phase.

To reproduce the results in [28], the model has been trained on a set of labelled claims (71,291 samples) with their associated evidence.

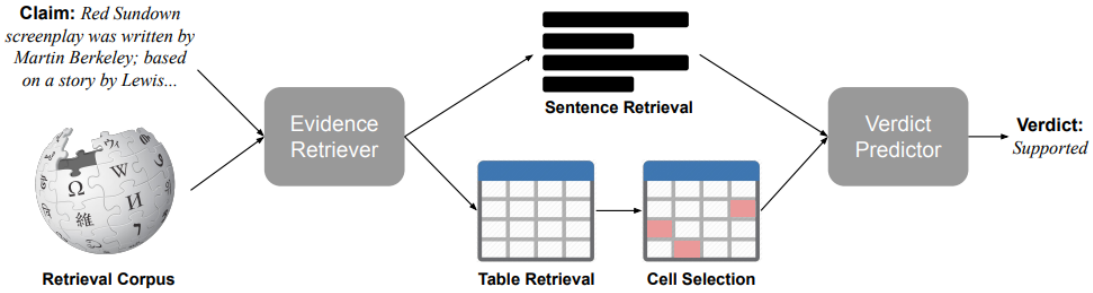

Figure 4.2: Baseline model from [28]

Evaluation

The evaluation of the performance for this challenge was not based on the assigned label only, but it must take into consideration also how well the algorithm retrieve the correct evidences, since they are fundamental to provide a justification for the final verdict. [28] introduces a new evaluation score, called *FEVEROUS score*, and defined as

$$\text{Score}(y, \hat{y}, \mathbb{E}, \hat{E}) = \begin{cases} 1 & \exists E \in \mathbb{E} : E \subseteq \hat{E} \wedge \hat{y} = y \\ 0 & \text{otherwise} \end{cases} \tag{4.1}$$

where $\hat{y}$ and $\hat{E}$ are the predicted label and evidence, $\mathbb{E}$ is the set of gold evidences. Hence, the FEVEROUS score returns 1 if there is at least one predicted evidence that belongs to the golden set and its label prediction is correct, otherwise it scores 0. On the other hand, the task needs also another score to measure the goodness of evidence prediction: in this sense we measures precision, recall and F1 scores of the evidence retrieval.

4.4 Method and proposed solution

Studying the architecture, since the verdict predictor part uses basically a state-of-art model, we assumed that the part which we should focus on is the retrieval part, since it is trivial to find the accurate evidence in order to enhance the label prediction. Thus, the next step is to find a suitable method that can better identify the correct Wikipedia pages and relative elements within it. We started by analysing the novelties in the information retrieval (IR) community, where the neural ranking models have been widely exploited.

Our proposed solution is based on a standard IR pipeline, which consists in a two-step ranking process where:

- Given a query, a large number of documents are retrieved from a corpus through some standard mechanism: for this step we decided to keep the original method by using a mix of entity and TF-IDF matching, while increasing the number k of pages extracted from the Wikipedia corpus.
- The documents are scored and re-ranked using a non-standard and more computationally expensive method: in this case, since the popularity of neural re-rankers in the IR community, our choice was a pre-trained BERT which was fine-tuned on a specific re-ranking task of the MS MARCO dataset, explained in [30], and it has to minimize the binary cross-entropy loss

$$L = -\sum_{i \in I^+} \log(s_i) - \sum_{i \in I^-} \log(1 - s_i) \tag{4.2}$$

  where I^+ and I^- are respectively the relevant and non-relevant pages.

In the representation of the experiments we will define the first step as $BL_{page}(k)$ (Baseline Page extractor) where k are the number of relevant extracted pages, while the second model step will be defined as $PR(m)$ (Page Ranker) where it takes the k relevant pages extracted previously as input, it scores them and then it outputs the top m pages in the ranked list.

4.5 Experiments and results

As stated previously, our model relies on:

- Entity and TF-IDF matching for extracting the pages
- A BERT encoder fine-tuned on the MS MARCO Passage Ranking task for scoring, re-ranking and extracting the top pages
- a TF-IDF matching for extracting the sentences and the table evidences, while a RoBERTa model is used for the cell evidences
- A final RoEBRTa to predict the verdict

The best model has the hyperparameters set as $k = 150$ and $m = 5$, where in order to obtain a large number of document in the first instance, the model extract 150 pages through the matching mechanism; these pages are scored and re-ranked, then from these the top 5 pages are extracted. The remaining part of the pipeline is kept equal to the baseline ($l = 5$, $q = 3$). This configuration is

defined $BL_{page}(150) \rightarrow PR(5) \rightarrow$ tf-idf(5,3). The results are shown in Table 4.2. As made clear in the Github documentation[3] of the project the baseline model has been trained only partially on the training set, defined as $BL(5,5,3)$ obtaining the first line of results in Table 4.2. Our first experiment concerned the full training of the baseline and its test on the development set (defined as $BL(5,5,3)_{full}$: the results are quite similar and no improvement is worth noticing. For what concern our model, looking at Table 4.2, it can be seen an improvement between the implemented model (written in bold) and the previous explained configuration: this means that the injected re-ranker has increased the coverage of the documents compared to the basic TF-IDF technique.
Despite our model implements more time-consuming paradigms, it returns more accurate pages and evidences compared to the baseline model in most of the cases. An example is the input claim "*Family Guy is an American animated sitcom that features five main voice actors [...] and has appeared in 22 (out of 349) episodes [...] that has appeared in 90 episodes.*": in this case it is trivial that the main evidences should be in the *Family Guy* Wikipedia page. The baseline retrieval extracts pages that are formally similar to the claim, but semantically incoherent, like *Guy* and *American*, and in some cases even pages that have nothing in common with the input. On the contrary, our model succeeds in retrieving the correct page in the top 5, together with other related pages, such as *List of Family Guy episodes*. On the other hand, there are also cases in which the basic baseline mechanism is more precise: for the claim "*Seven notable animated television series, including Super Why!, a children's educational show created by Angela C. Santomero and Samantha Freeman Alpert, Phineas and Ferb and WordGirl, were released in September 2007.*" the re-ranker retrieve related pages, such as other TV shows created from Angela C. Santomero, but no meaningful page is shown in the top 5. However, the baseline can identify *Phineas and Ferb* as important page, but it fails in predicting the correct elements.
Nevertheless, despite these drawbacks, our team (EURECOM Fever) managed to reach the fifth position on the final leaderboard with a score of 0.2001 (measured with FEVEROUS score) on the test set, above the benchmark set by the original baseline. The final competition ranking can be seen on the FEVER shared task website[4]. Our code is freely accessible on GitLab[5].

[3]Code source: github.com/Raldir/FEVEROUS

[4]FEVER website: fever.ai/task.html

[5]Code: gitlab.eurecom.fr/saeedm1/eurecom-fever

	FS	**LA**	**EP**	**ER**	**E-F1**
BL(5,5,3)	0.19	0.54	0.12	0.29	0.17
$BL(5,5,3)_{full}$	0.186	0.533	0.119	0.289	0.168
$BLpage(50) \rightarrow PR(5) \rightarrow tfidf$(5,3)	0.129	0.468	0.120	0.201	0.151
$BLpage(150) \rightarrow PR(5) \rightarrow tfidf$(5,3)	**0.218**	**0.548**	**0.145**	**0.339**	**0.203**
$BLpage(150) \rightarrow PR(5) \rightarrow BM25$(5,3)	0.205	0.550	0.127	0.321	0.182
$BLpage(150) \rightarrow PR(5) \rightarrow SR(5) \rightarrow tfidf_{table}(3)$	0.184	0.501	0.130	0.283	0.179

Table 4.2: Results on the dev set showing the FEVEROUS Score (FS), the Label Accuracy (LA), the Evidence Precision (EP), the Evidence Recall (ER), and the Evidence F1-score (E-F1) of the different system variants.

Other attempts

We have attempted other configurations shown in Table 4.2 in order to enhance the performance of our model:

- We tried to reduce the number k of extracted paged by the re-ranker to 50, but this experiment worsened the score. This model is defined $BL_{page}(50) \rightarrow PR(5) \rightarrow$ tf-idf(5,3).
- After noticing the great result with the page re-ranker, we tried to apply this configuration in order to extract better sentence evidences, exploiting another BERT-based encoder. However, despite the previous scores, this implementation did not return better results compared to the ones obtained with the TF-IDF matching. This model is shown in the table as $BL_{page}(150) \rightarrow PR(5) \rightarrow SR(5) \rightarrow \text{tf-idf}_{\text{table}}(5,3)$.
- Regarding the retrieval of sentences and tables, we experimented with the Okapi BM25 scoring function [23], which is a ranking function based on the probabilistic retrieval framework. It is in great use in IR community, but in this case it did not return better result than TF-IDF matching function, probably because of the pre-processing, since we did not explore many pre-processing functions.

To participate to the challenge, we used the best model in Table 4.2. However, we continued to further experiment and improve our configuration after the deadline, by changing parameters in order to reach better results at least on the development set. Since we noticed that the increase of the hyperparameter k led to better results, we tried different amount of pages to select. The results can be seen in Table 4.3: as the number of pages increases, the performances are better as well,

	FS	**LA**	**EP**	**ER**	**E-F1**
$BLpage(150) \rightarrow PR(5) \rightarrow tfidf(5,3)$	**0.218**	**0.548**	**0.145**	**0.339**	**0.203**
$BLpage(200) \rightarrow PR(5) \rightarrow tfidf(5,3)$	0.216	0.543	0.146	0.338	0.204
$BLpage(300) \rightarrow PR(5) \rightarrow tfidf(5,3)$	**0.224**	**0.545**	**0.147**	**0.349**	**0.207**
$BLpage(500) \rightarrow PR(5) \rightarrow tfidf(5,3)$	0.219	0.541	0.147	0.344	0.206

Table 4.3: Results on the dev set showing the FEVEROUS Score (FS), the Label Accuracy (LA), the Evidence Precision (EP), the Evidence Recall (ER), and the Evidence F1-score (E-F1) of the different models changing the number of retrieved pages by the re-ranker.

until it reach a peak at 300 pages and the results are worsened if the number keeps growing. While the performances are more variable, the process becomes more time-consuming proportionally to the number of pages to be selected. Eventually, we reach a model that perform slightly better on the development set.

Chapter 5

MediaEval 2021 challenge

The MediaEval Multimedia Evaluation benchmark[1] is an organization that offers tasks and workshops which are related to multimedia retrieval, analysis, and exploration. In this chapter we delve into a specific challenge held during the 2021 edition called "*FakeNews: Corona Virus and Conspiracies Multimedia Analysis Task*"[2]. We already analysed how the digital wildfires provoked by fake news spreading can be harmful for public consciousness and can imply severe real-world implication: focal points for the spreading are the social networks, such as Twitter, Facebook, Reddit etc. The Covid-19 pandemic has affected the life of all the population worldwide and it has been a central topic in news media and in public debates both in real life and on the Internet. Thus, this has provoked consequently a great spread of misinformation in parallel, mostly due to the fact that the scientific community did not have all the information and details about the virus and the pandemic situation. This challenge aims at the development of methods capable of detecting such misinformation on tweets. Online misinformation provides very different narratives when it comes to Covid-19: for this specific task, the aim is focused on the pandemic-related conspiracy theories. Regarding this challenge, our work is presented in [31].

5.1 Task description

The task concern a dataset of tweets related to Covid-19 pandemic and conspiracy theories. It is divided in three sub-taks: the first sub-task concerns text-based fake news detection, the second sub-task aims to conspiracy theory-related detection,

[1]Website: multimediaeval.github.io/

[2]multimediaeval.github.io/editions/2021/tasks/fakenews/

while the third sub-task is a combination of the previous topic and conspiracy detection. The theories taken into consideration are usually the ones considering some reprehensible action committed by the governments, such as intentional spreading of the disease, or lying about the nature of the virus. Specifically, we can describe the sub-tasks as:

- **Text-Based Misinformation Detection**: the first sub-task concerns the detection of the topic-related discussion in a dataset composed by tweet text blocks in English related to Covid-19. In this case, a single tweet can *promote* or *discuss* at least one conspiracy theory; the third label is that the text does *not concern* any Covid-19-related topic. In the situation in which the tweet discusses a theory but supports another, the label refers to the "stronger" label ("support" is stronger than "discuss", which in turn is heavier than "not concerning"). The participants have to implement a *multi-class* classifier with three output labels.
- **Text-Based Conspiracy Theories Recognition**: in this sub-task the labels of every single tweet from the dataset are 9, each of them refers to a specific conspiracy theory. The participants are encouraged to create a *multi-label* classifier which can identify which of the conspiracy theories the tweet is focused on (or if it does not concern any theory at all).
- **Text-Based Combined Misinformation and Conspiracies Detection**: this last sub-task is a mix of the previous two steps; specifically, the classifier that the participants have to implement is a *multi-class multi-label* type. The model has to identify which of the conspiracy theories the tweet is about and moreover if the tweet is promoting or just discussing the theory (or theories) in exam.

In the testing phase, where the test set become available, the challenge is structured in 5 runs for each sub-task:

- **First required run**: it must concern an automated tweet content classification model based only on the provided text tweet dataset.
- **Second optional run**: this run requires an automated tweet content classification model that can be based on any publicly available pre-trained linguistic and NLP architecture which can re-trained or fine-tuned on the provided dataset, but any other external training data is prohibited for use.
- **From third to fifth optional run**: for these last runs the participants can use an automated tweet content classification model that can be based on any pre-trained linguistic and NLP architecture and/or on any external data scraped from any external sources.

Eventually, the evaluation is performed using the MCC (Matthews Correlation Coefficient), defined in Equation 2.8. In the first sub-task it is computed in its multi-class configuration; on the other hand, for the second and third sub-tasks, it is computed individually for each label and then the average of the values defines the measure. A detailed description of the task can be found in [32].

5.2 Dataset

The provided dataset contains 1554 tweets in English mentioning Coronavirus and various conspiracy theories. The texts are characterized by various long sentences with neutral, positive, negative, and sarcastic phrasing. The texts have been collected in a period of time from January 2020 to July 2021 by searching on Twitter Corona-virus-related keywords and hashtags, such as "corona", "COVID-19", etc. followed by another search related to conspiracy theories. The ground truth was created by a team of students and researchers using an overlapping annotation process. The dataset is not balanced with respect to the number of samples of conspiracy-promoting and other tweets, nor to the number of tweets per each conspiracy class.
The dataset is divided in three parts, one per sub-tasks, but they all share the same texts for the tweets and the related tweets IDs. The labels change depending on the sub-task which they refer to: the first two fields of the datasets are the tweet ID and the related tweet text, while for the first sub-task these fields are followed by a class identifier value which can have three values:

- **3 = Promotes/Supports conspiracy**: this class label is assigned to all those tweets promotes, supports, or insinuate some relationship between Covid-19 and various conspiracies, such as the 5G connection caused the virus pandemic, the intentional release of the virus from government for various purposes like a population reduction, the idea that the pandemic is completely made up and/or a hoax, etc.

- **2 = Discusses conspiracy**: this class contains all the tweets that do not support the conspiracies, but they mention the existing of these connections, or deny them, or they discuss them in a clearly sarcastic manner.

- **2 = Non-conspiracy**: this class covers all the tweets that are not labelled with the previous two scenarios, so generally those tweets that do not mention Covid-19 or they mention the pandemic situation without any hint on conspiracies.

The dataset related to the second sub-task instead contains 9 different binary flags that report the presence (1) or the absence (0) of that specific conspiracy theory they relate to in an individual tweet. The conspiracy theories they identify are

Suppressed Cures, Behaviour and Mind Control, Antivax, Fake Virus, Intentional Pandemic, Harmful Radiation, Population Reduction, New World Order, and *Satanism*. The final sub-task is related to the dataset that have, together with the ID and the text, 9 different class labels: each class labels i defines if the tweet promotes (3), discusses (2) or it is not about (1) that specific i-th conspiracy. The theories are the same mentioned before.
Figure 5.1, Figure 5.2 and Figure 5.3 show an approximate distribution of class and labels, stressing the unbalancing. The dataset is fully explained in [33].

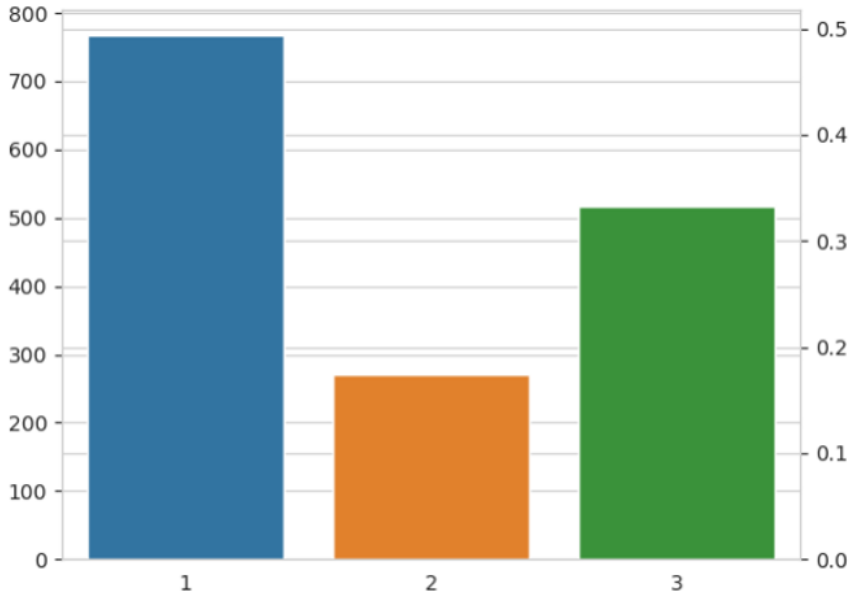

Figure 5.1: Class distribution for sub-task 1

5.3 Method and proposed solutions

Since this challenge did not provide any baseline or starting point, in order to tackle the task, we proceeded to develop three different kind of approaches. The first uses a combination of TF-IDF encoding and Machine Learning algorithms. The second experimented with language model models which were fine-tuned on NLI datasets and combined with some metadata scraped from Wikipedia. The third approach aimed to fine-tuning transformer-based models.

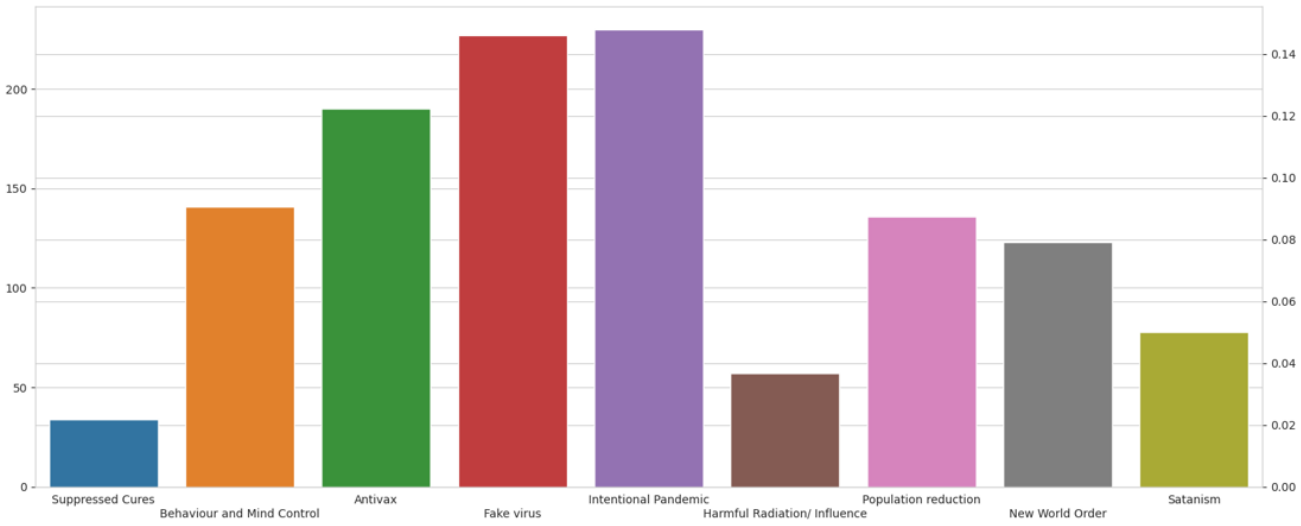

Figure 5.2: Label distribution for sub-task 2

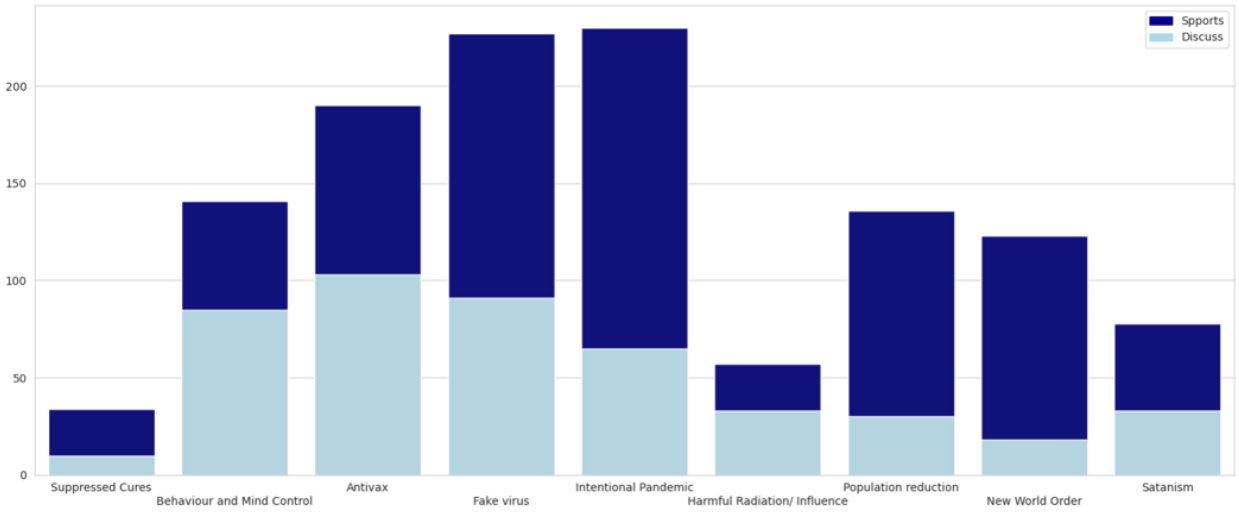

Figure 5.3: Class label distribution for sub-task 3

5.3.1 TF-IDF-based approach

TF-IDF is one of the first technique we experimented with, because, despite its simplicity, it allows us to have a decent baseline to compare more complex configuration. Moreover, it has been very useful during the first required runs of the challenge, where no pre-trained language models, nor any external data outside

the textual one extracted from the tweets were allowed.
Firstly, we elaborate the text of the tweets through a pre-processing step which concerned tokenization, capitalization and stop word removal. Secondly, the plain text has been fed to a TF-IDF vectorization and this method allowed us to obtain a sparse TF-IDF feature matrix which was fed to different supervised Machine Learning methods. Several algorithms have been tested: Decision Tree, Naive Bayes classifier (Gaussian and Bernoullian), AdaBoost, Ridge and Logistic Regression.
In the case of Task 1, these were used in a multi-class asset. In the multi-label case of Task 2, we used a multi-output classifier with the different methods listed above as estimators: in this scenario the algorithm instantiates a binary model estimator for each conspiracy theory and train the models with the same training data, interpreting the problem as a binary one. Finally, in the Task 3, only the strictly tree-based algorithms (Decision Tree, Random Forest and AdaBoost) were tested, since they are the only ones to allow a multi-label and multi-class output at the same time.

5.3.2 NLI-based approach

This approach relies on leveraging pre-trained language models that are then fine-tuned on databases based on the task of NLI. The NLI task has been explained in section 2.1. These type of models are trained to project text records that share similar features in the same area in their embedding space, so we expected to identify tweets that focus on the same topic/conspiracy theory in the same space using the common embedding space. The model used for these experiments is a specific type of NLI pre-trained sentence transformer-based BERT, which allows the computation of these embeddings. This model is deeply described in [34].
In the first task, the idea is to identify the stance of each tweet, which can support, discuss or being neutral about conspiracies, embedding the records in the same space: in this scenario, the tweets with similar stances should lay in the same sub-space. Thus, the first step is to generate the text embeddings, and then classify the different data points through a K-nearest neighbor. In the second task, the process is really similar, but is geared towards the differentiation of conspiracy theories, rather than stances. Moreover, we pre-trained the models by feeding them with definition of the conspiracy theories found on Wikipedia: thus, the model identify a tweet related a certain conspiracy if it predict an entailment between the Wikipedia definition (premise) and the tweet text (hypothesis). A mix of the previous processes is used in the third task: annotated tweets related to a specific conspiracy are used as a premise instead of a definition of the conspiracy, then the model classify them if an entailment is found.

5.3.3 Transformer-based approach

As stated multiple times previously, the Transformer-based models are the state-of-art for NLP task like the one in exam. In our scenario, we decided to try several experiments with a $\text{RoBERTa}_{\text{Large}}$ and a CT-BERT (Covid-Twitter-BERT). This last Transformer is presented in [35], a BERT-based transformer which has been pre-trained on a large corpus of Twitter messages on the topic of COVID-19. It has shown remarkable improvements in Covid-19-related task compared to $\text{BERT}_{\text{Large}}$. Before feeding the tweets into these models, we perform a basic pre-processing technique: we replaced all emojis with their textual meaning, and removed all the '#' characters (so that the hashtags remain in the text as simple words), since hashtags are often used also as part of sentences with their semantic and contextual meaning. The first task is approached as a multi-class task, so each transformer-based model is fine-tuned on the training set with a Cross-entropy loss function. The second task is performed in the same way experimenting with the models, but as a multi-label task using a weighted Binary Cross Entropy loss function. The weights are factors which depends on the inverse frequency of each class or sub-class they are related to.
Eventually, the third task can be approached with different strategies. The first we tried concerned the mixing of the results of the previous two tasks: we labelled the tweets with the level detected in the first task for the conspiracy theories detected in the second one. Despite its simplicity and its promising results, this technique fails to recognize tweets that supports a theory, but discusses another one. The second strategy is to train the two transformers in exam directly on the task: these models are fine tuned for nine different multi-class classification problems with nine Cross Entropy loss functions, one for each conspiracy theory. The final loss is the weighted sum of the nine losses with weight factors similar to the previously described ones. This approach simplify also the previous two task, since they are specific cases of the last one. Figure 5.4 shows an illustration of this configuration.

5.4 Experiments and results

Results are presented in Table 5.1: the values for the evaluation set are available for all the experimented models; on the other hand we had to select few models to be tested on the test set, due to the limited number of attempts available, as explained in section 5.1.
The dataset has been splitted in training and test set with a proportionality of 0.8 and 0.2 respectively. The evaluation has been made with a stratified 5-fold validation with a 0.2 of the training set used as validation set. As expected, the Transformers obtained the most competitive results. Specifically, the RoBERTa performed worse than the CT-BERT model, surely because it has been pre-trained

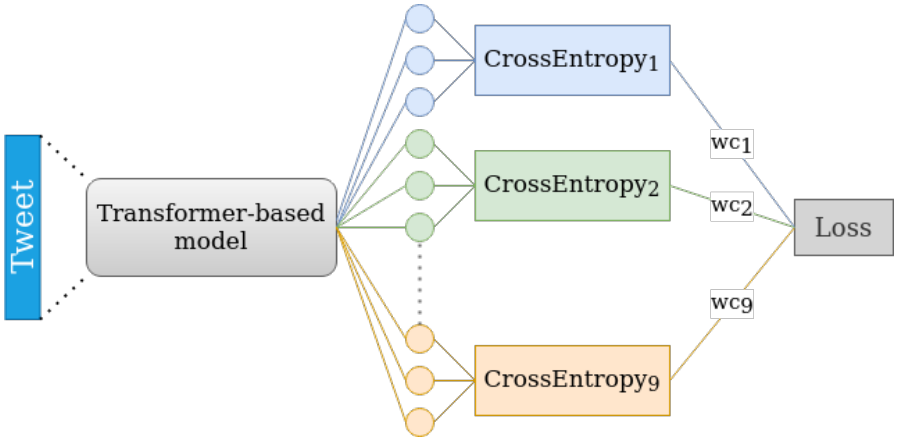

Figure 5.4: Illustration of the models for Task 3

on dataset that uses a strong Covid-19-related vocabulary, such as the dataset in exam in this challenge, that is hardly understood from the other model.
For what concern the TF-IDF approach, it gives decent baselines for all the three tasks: in the first a linear Support Vector Machine model gave the best performances with 0.461 MCC score; while for the other two tasks, the tree-based performed better, Decision tree in particular, with 0.565 and 0.497 MCC score. The linear and tree-based models worked better in this case probably because of the sparsity of the generated TF-IDF matrix. Unfortunately, the NLI approach did not measure up to the fully-trained models, but it may be an interesting alternative to explore in the case where annotated data is lacking: a few tweets per class/label or just the definition may give interesting results.
It is worth mentioning that all our models performed slightly worse in the most present label in task 2 and 3, which is *Intentional pandemic*, while they performed better with other labels such as *Harmful influence* and *New World Order*. This is probably due to the more specific vocabulary that these tweets tend to have: e.g. they mention the words "5G" and "NWO" several times. As a last experiment, we tried to create ensambling models by mixing the results of the Transformer-based models through majority voting. The results have increased significantly on the evaluation set, but they did not reflected the same improvement in the test set too. With these results we were able to win the challenge by placing first. The code is freely available on Github[3].

[3]Code: github.com/D2KLab/mediaeval-fakenews

	Models	Evaluation MCC	Test MCC
Task 1	TFIDF (SVC)	0.461	0.498
	NLI transformer	0.426	X
	RoBERTa	0.624	X
	CT-BERT	0.676	X
	RoBERTa-task3	0.667	X
	CT-BERT-task3	0.700	0.720
	Ensembling Models	0.716	**0.733**
Task 2	TFIDF (DT + Multi output clf)	0.585	0.317
	NLI Wikipedia	0.310	X
	RoBERTa	0.731	X
	CT-BERT	0.780	**0.774**
	RoBERTa-task3	0.734	X
	CT-BERT-task3	0.743	0.719
	Ensembling Models	0.781	0.768
Task 3	TFIDF (DT)	0.497	0.186
	RoBERTa-task1+task2	0.675	X
	CT-BERT-task1+task2	0.717	**0.775**
	RoBERTa-task3	0.690	X
	CT-BERT-task3	0.706	0.713
	Ensembling Models	0.726	0.676

Table 5.1: MCC results for each task, based on stratified 5-fold cross-validation set and then on the test set

Chapter 6

Conclusion

In this work we faced two different challenges concerning fake news detection. After a brief description of technical and social background, we implemented different solutions for the presented tasks. The implemented models allowed us to place fifth in the first challenge and first in the second one: these results reflect how language models can help create an excellent tool that may not replace the human work, but instead it can create an helpful environment to the human worker i.e. a human-in-the-loop tool, that can lighten the burden of fact-checking work, since it is an increasingly heavy workload due to the growing flow of information on the Internet.
For what concern our solution, in the first challenge we had a baseline model to start with: this was based on a mix of TF-IDF and entity matching, which works in parallel with transformer-based models that elaborated both structured and unstructured information. Our solution improved the previous baseline by adding a transformer-based neural re-ranker that was able to extract more accurately the Wikipedia pages, then another mechanism extracted the evidences from them. We managed to further improve the model by changing the number of selected top k pages, but its increase provoked a more time-consuming process. On the other hand, in the second project we experimented with different implementations: from TF-IDF encoding, resulting in large sparse matrices, combined with various supervised models, to state-of-art BERT-based models fine-tuned to specific datasets concerning the Covid-19 context.
This work has the main aim to give to the reader a glimpse of how the Natural Language models and the current state-of-art can help curb a social-political problem that human work hardly manage. Obviously these are not the only solutions one can implement: scientific community in data science and NLP field is continuing to grow while creating more and more efficient and complex models. From a transformer-based point of view, there is a continuous study on how (and why) present transformers work, as explained in [36]. Other studies focus on how

the data should be stored and how to interact with it: some examples are the Neural Databases, presented by [37], which are modular architecture capable of taking as input complex database-style queries, and making the information retrieval process more accurate and fast. On the other hand, Knowledge Base Graphs are deeply studied as data storing technique, since this type of data configuration can be mixed with state-of-art NLP technologies to improve performances in a wide range of fields.

Bibliography

[1] Kamran Kowsari, Kiana Jafari Meimandi, Mojtaba Heidarysafa, Sanjana Mendu, Laura E. Barnes, and Donald E. Brown. «Text Classification Algorithms: A Survey». In: *CoRR* abs/1904.08067 (2019). arXiv: 1904.08067. URL: http://arxiv.org/abs/1904.08067 (cit. on p. 8).

[2] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. *Attention Is All You Need*. 2017. arXiv: 1706.03762 [cs.CL] (cit. on pp. 34, 36, 37, 40).

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. «BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding». In: *CoRR* abs/1810.04805 (2018). arXiv: 1810.04805. URL: http://arxiv.org/abs/1810.04805 (cit. on pp. 39–42, 44, 51, 52).

[4] Yinhan Liu et al. «RoBERTa: A Robustly Optimized BERT Pretraining Approach». In: *CoRR* abs/1907.11692 (2019). arXiv: 1907.11692. URL: http://arxiv.org/abs/1907.11692 (cit. on pp. 42–44, 51).

[5] Brendan Nyhan and Jason Reifler. «Displacing Misinformation about Events: An Experimental Test of Causal Corrections». In: *Journal of Experimental Political Science* 2.1 (2015), pp. 81–93. DOI: 10.1017/XPS.2014.22 (cit. on p. 45).

[6] Chuang Liu, Xiu-Xiu Zhan, Zi-Ke Zhang, Gui-Quan Sun, and Pak Ming Hui. «How events determine spreading patterns: information transmission via internal and external influences on social networks». In: *New Journal of Physics* 17.11 (Nov. 2015), p. 113045. ISSN: 1367-2630. DOI: 10.1088/1367-2630/17/11/113045. URL: http://dx.doi.org/10.1088/1367-2630/17/11/113045 (cit. on p. 46).

[7] Muhammad Imran, Prasenjit Mitra, and Carlos Castillo. *Twitter as a Lifeline: Human-annotated Twitter Corpora for NLP of Crisis-related Messages*. 2016. arXiv: 1605.05894 [cs.CL] (cit. on pp. 46, 49).

[8] Harshavardhan Achrekar, Avinash Gandhe, Ross Lazarus, Ssu-Hsin Yu, and Benyuan Liu. «Predicting Flu Trends using Twitter data». In: *2011 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. 2011, pp. 702–707. DOI: 10.1109/INFCOMW.2011.5928903 (cit. on p. 46).

[9] Firoj Alam, Stefano Cresci, Tanmoy Chakraborty, Fabrizio Silvestri, Dimiter Dimitrov, Giovanni Da San Martino, Shaden Shaar, Hamed Firooz, and Preslav Nakov. *A Survey on Multimodal Disinformation Detection*. 2021. arXiv: 2103.12541 [cs.MM] (cit. on p. 46).

[10] Soroush Vosoughi, Deb Roy, and Sinan Aral. «The spread of true and false news online». In: *Science* 359.6380 (2018), pp. 1146–1151. DOI: 10.1126/science.aap9559. URL: https://www.science.org/doi/abs/10.1126/science.aap9559 (cit. on pp. 47, 51).

[11] Preslav Nakov, David P. A. Corney, Maram Hasanain, Firoj Alam, Tamer Elsayed, Alberto Barrón-Cedeño, Paolo Papotti, Shaden Shaar, and Giovanni Da San Martino. «Automated Fact-Checking for Assisting Human Fact-Checkers». In: *CoRR* abs/2103.07769 (2021). arXiv: 2103.07769. URL: https://arxiv.org/abs/2103.07769 (cit. on pp. 47, 53).

[12] Andreas Vlachos and Sebastian Riedel. «Fact Checking: Task definition and dataset construction». In: *Proceedings of the ACL 2014 Workshop on Language Technologies and Computational Social Science*. Baltimore, MD, USA: Association for Computational Linguistics, June 2014, pp. 18–22. DOI: 10.3115/v1/W14-2508. URL: https://aclanthology.org/W14-2508 (cit. on p. 48).

[13] Tauhid Zaman, Emily B. Fox, and Eric T. Bradlow. «A Bayesian Approach for Predicting the Popularity of Tweets». In: *CoRR* abs/1304.6777 (2013). arXiv: 1304.6777. URL: http://arxiv.org/abs/1304.6777 (cit. on p. 48).

[14] Shaden Shaar, Giovanni Da San Martino, Nikolay Babulkov, and Preslav Nakov. «That is a Known Lie: Detecting Previously Fact-Checked Claims». In: *CoRR* abs/2005.06058 (2020). arXiv: 2005.06058. URL: https://arxiv.org/abs/2005.06058 (cit. on pp. 49, 51–53).

[15] Konstantin Pogorelov, Daniel Schroeder, Petra Filkuková, and Johannes Langguth. «A System for High Performance Mining on GDELT Data». In: May 2020, pp. 1101–1111. DOI: 10.1109/IPDPSW50202.2020.00182 (cit. on p. 49).

[16] Pepa Atanasova, Alberto Barron-Cedeno, Tamer Elsayed, Reem Suwaileh, Wajdi Zaghouani, Spas Kyuchukov, Giovanni Da San Martino, and Preslav Nakov. *Overview of the CLEF-2018 CheckThat! Lab on Automatic Identification and Verification of Political Claims. Task 1: Check-Worthiness*. 2018. arXiv: 1808.05542 [cs.CL] (cit. on pp. 50, 51).

[17] Tamer Elsayed, Preslav Nakov, Alberto Barrón-Cedeño, Maram Hasanain, Reem Suwaileh, Giovanni Da San Martino, and Pepa Atanasova. *Overview of the CLEF-2019 CheckThat!: Automatic Identification and Verification of Claims*. 2021. arXiv: 2109.15118 [cs.CL] (cit. on pp. 50, 51, 53).

[18] Alberto Barron-Cedeno et al. *Overview of CheckThat! 2020: Automatic Identification and Verification of Claims in Social Media*. 2020. arXiv: 2007.07997 [cs.CL] (cit. on pp. 50–53).

[19] Preslav Nakov et al. «The CLEF-2021 CheckThat! Lab on Detecting Check-Worthy Claims, Previously Fact-Checked Claims, and Fake News». In: *ECIR*. 2021 (cit. on p. 50).

[20] Daniel Schroeder, Konstantin Pogorelov, and Johannes Langguth. «FACT: a Framework for Analysis and Capture of Twitter Graphs». In: Oct. 2019. DOI: 10.1109/SNAMS.2019.8931870 (cit. on p. 50).

[21] Luk Burchard, Daniel Schroeder, Konstantin Pogorelov, Sören Becker, Emily Dietrich, Petra Filkuková, and Johannes Langguth. «A Scalable System for Bundling Online Social Network Mining Research». In: Dec. 2020, pp. 1–6. DOI: 10.1109/SNAMS52053.2020.9336577 (cit. on p. 50).

[22] David Corney. *How we use AI to help fact check party manifestos*. 2019. URL: https://fullfact.org/blog/2019/dec/how-we-use-ai-help-fact-check-party-manifestos/ (cit. on p. 51).

[23] Stephen Robertson, S. Walker, S. Jones, M. M. Hancock-Beaulieu, and M. Gatford. «Okapi at TREC-3». In: *Overview of the Third Text REtrieval Conference (TREC-3)*. Gaithersburg, MD: NIST, Jan. 1995, pp. 109–126. URL: https://www.microsoft.com/en-us/research/publication/okapi-at-trec-3/ (cit. on pp. 52, 63).

[24] Stephen Robertson and Hugo Zaragoza. «The Probabilistic Relevance Framework: BM25 and Beyond». In: *Found. Trends Inf. Retr.* 3.4 (Apr. 2009), pp. 333–389. ISSN: 1554-0669. DOI: 10.1561/1500000019. URL: https://doi.org/10.1561/1500000019 (cit. on p. 52).

[25] Peter Bourgonje, Julian Moreno Schneider, and Georg Rehm. «From Clickbait to Fake News Detection: An Approach based on Detecting the Stance of Headlines to Articles». In: *Proceedings of the 2017 EMNLP Workshop: Natural Language Processing meets Journalism*. Copenhagen, Denmark: Association for Computational Linguistics, Sept. 2017, pp. 84–89. DOI: 10.18653/v1/W17-4215. URL: https://aclanthology.org/W17-4215 (cit. on p. 52).

[26] James Thorne, Andreas Vlachos, Christos Christodoulopoulos, and Arpit Mittal. «FEVER: a large-scale dataset for Fact Extraction and VERification». In: *CoRR* abs/1803.05355 (2018). arXiv: 1803.05355. URL: http://arxiv.org/abs/1803.05355 (cit. on pp. 53, 59).

[27] Van-Hoang Nguyen, Kazunari Sugiyama, Preslav Nakov, and Min-Yen Kan. «FANG: Leveraging Social Context for Fake News Detection Using Graph Representation». In: *CoRR* abs/2008.07939 (2020). arXiv: 2008.07939. URL: https://arxiv.org/abs/2008.07939 (cit. on p. 54).

[28] Rami Aly, Zhijiang Guo, Michael Schlichtkrull, James Thorne, Andreas Vlachos, Christos Christodoulopoulos, Oana Cocarascu, and Arpit Mittal. *FEVEROUS: Fact Extraction and VERification Over Unstructured and Structured information*. 2021. arXiv: 2106.05707 [cs.CL] (cit. on pp. 55, 56, 58–60).

[29] Mohammed Saeed, Giulio Alfarano, Khai Nguyen, Duc Pham, Raphael Troncy, and Paolo Papotti. «Neural Re-rankers for Evidence Retrieval in the FEVEROUS Task». In: *Proceedings of the Fourth Workshop on Fact Extraction and VERification (FEVER)*. Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 108–112. DOI: 10.18653/v1/2021.fever-1.12. URL: https://aclanthology.org/2021.fever-1.12 (cit. on p. 55).

[30] Payal Bajaj et al. *MS MARCO: A Human Generated MAchine Reading COmprehension Dataset*. 2018. arXiv: 1611.09268 [cs.CL] (cit. on p. 61).

[31] Youri Peskine, Giulio Alfarano, Ismail Harrando, Paolo Papotti, and Raphael Troncy. «Detecting COVID-19-related conspiracy theories in tweets». In: *MediaEval 2021, MediaEval Benchmarking Initiative for Multimedia Evaluation Workshop, 13-15 December 2021 (Online Event)*. Ed. by ACM. © ACM, 2021. This is the author's version of the work. It is posted here by permission of ACM for your personal use. Not for redistribution. The definitive version was published in MediaEval 2021, MediaEval Benchmarking Initiative for Multimedia Evaluation Workshop, 13-15 December 2021 (Online Event). 2021 (cit. on p. 65).

[32] Konstantin Pogorelov, Daniel Thilo Schroeder, Stefan Brenner, and Johannes Langguth. «FakeNews: Corona Virus and Conspiracies Multimedia Analysis Task at MediaEval 2021». In: *Multimedia Benchmark Workshop*. 2021 (cit. on p. 67).

[33] Konstantin Pogorelov, Daniel Thilo Schroeder, Petra Filkuková, Stefan Brenner, and Johannes Langguth. «WICO Text: A Labeled Dataset of Conspiracy Theory and 5G-Corona Misinformation Tweets». In: *Proceedings of the 2021 Workshop on Open Challenges in Online Social Networks*. New York, NY, USA: Association for Computing Machinery, 2021, pp. 21–25. DOI: 10.1145/3472720.3483617 (cit. on p. 68).

[34] Nils Reimers and Iryna Gurevych. «Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks». In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Nov. 2019. URL: http://arxiv.org/abs/1908.10084 (cit. on p. 70).

[35] Martin Müller, Marcel Salathé, and Per Egil Kummervold. «COVID-Twitter-BERT: A Natural Language Processing Model to Analyse COVID-19 Content on Twitter». In: *CoRR* abs/2005.07503 (2020). arXiv: 2005.07503. URL: https://arxiv.org/abs/2005.07503 (cit. on p. 71).

[36] Anna Rogers, Olga Kovaleva, and Anna Rumshisky. «A Primer in BERTology: What we know about how BERT works». In: *CoRR* abs/2002.12327 (2020). arXiv: 2002.12327. URL: https://arxiv.org/abs/2002.12327 (cit. on p. 74).

[37] James Thorne, Majid Yazdani, Marzieh Saeidi, Fabrizio Silvestri, Sebastian Riedel, and Alon Halevy. *Database Reasoning Over Text*. 2021. arXiv: 2106.01074 [cs.CL] (cit. on p. 75).