Università degli Studi di Milano - Bicocca
Scuola di Scienze
Dipartimento di Informatica, Sistemistica e Comunicazione
Corso di laurea magistrale in Data Science

Comparative Analysis of Irony and Sarcasm Detection and Self Attention Approach Development

Relatore: Prof. Elisabetta Fersini
Correlatore: Prof. Paolo Rosso

Relazione della prova finale di:
Lorenzo Famiglini
Matricola 838675

Anno Accademico 2019-2020

Abstract

The detection of irony and sarcasm is one of the most insidious challenges in the field of Natural Language Processing. Over the years, several techniques have been studied to analyze these rhetorical figures, trying to identify the elements that discriminate, in a significant way, what is sarcastic or ironic from what is not. Within this study, some models that are state-of-the-art are analyzed. As far as Machine Learning is concerned, the most discriminating features such as part of speech, pragmatic particles and sentiment are studied. Subsequently, these models are optimized, comparing Bayesian optimization techniques and random search. Once, the best hyperparameters are identified, ensemble method such as Bayesian Model Averaging (BMA) is exploited. In relation to Deep Learning, two main models are analysed: DeepMoji, developed by MIT, and a model called RCNN-Roberta, which exploits the generalization power of Roberta Transformer. As soon as these models are compared, the main goal is to identify a new system able to better capture the two rhetorical figures. To this end, two models composed of attention mechanisms are proposed, exploiting the principle of Transfer Learning, using BERTweet Model and DeepMoji Model as features extractors. After identifying the various architectures, an ensemble method is applied on the set of approaches proposed, in order to identify the best combination of algorithms that can achieve satisfactory results. The development of all models is based on the following objectives: how well the classifiers are able to generalise over out-domain datasets and which models are able to identify the right patterns to achieve better performance.

Ringraziamenti

La sezione dei ringraziamenti risulta essere sempre quella meno importante rispetto al resto dei contenuti della tesi, ma dal mio punto di vista è di fondamentale importanza poter ringraziare chi di dovere e chi c'è sempre stato. In primo luogo, ringrazio profondamente la professoressa Elisabetta Fersini e il professore Paolo Rosso, in quanto guide principali di questo ambizioso progetto. Vorrei poi ringraziare due persone, amici e al tempo stesso colleghi, che in questi anni si sono rivelati di fondamentale importanza per il mio percorso nel mondo della Data Science: in particolare, mi sto riferendo a Giorgio e Pranav. Grazie a loro, ho potuto affrontare questa mia nuova esperienza di vita in maniera determinata raggiungendo traguardi importanti. Un'altra persona da dover menzionare è il mio caro amico Sergio, una forte amicizia fatta di fiducia e di reciproco aiuto. Inoltre, un immenso grazie anche a tutte le persone, amici e parenti, che fanno parte della mia vita quotidiana. Infine, per ultimi, ma non per importanza, vorrei porgere la mia più profonda gratitudine alla mia famiglia, a mia sorella Arianna, ma soprattutto ai miei due fantastici genitori Isabella e Massimo: senza di loro non sarei potuto essere qui, fornendomi tutto il necessario per poter affrontare una nuova vita lontano dalla mia città natale. Con tanti sacrifici, non mi hanno mai negato nulla, e per questo mi sento la persona più fortunata del mondo. Spero un giorno di poterli ripagare di tutta la fiducia riposta in me.

Gutta cavat lapidem non vi, sed saepe cadendo.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the last few years, social networks have taken on a fundamental role in each person's life. On one hand, consider Facebook and Twitter, which have become platforms where a massive election campaign is carried out by politicians from all over the world. On the other hand, people use social media as an instrument either to express their opinions and feelings or to share everything concerning their life. The data is mainly generated in the form of text, images and videos. How can data be analysed? From the point of view of unstructured data, such as text, Natural Language Processing (NLP) research plays a fundamental role in order to understand people thoughts. Especially, Sentiment Analysis (SA) makes it possible to carry out analysis of interactions between users, established in a given context and in a defined time frame. Specifically, it is the computational analysis of feelings and opinions expressed within generated texts. There are different techniques applied to this task, such as lexicon based, Machine Learning and hybrid approaches, [1]. The study is the result of an international collaboration between La Bicocca University and the University of Valencia, with the support of the NLP expert Professor Paolo Rosso. The main objective of this thesis is to analyse the problem of irony and sarcasm detection through the use of Machine Learning and Deep Learning techniques: a comparative analysis is applied to different state-of-the-art models, and the development of new methods based on the most recent NLP techniques, like Transformers as features extractors, convolutions with self attention layers and emotions embeddings, are proposed in order to identified the most accurate model for figurative language detection [2]. Starting from single evaluation to multiple evaluations, based on ensemble methods exploiting Bayesian

Model Averaging and other techniques, is the final propose of this related work together with the analysis of the true generalisation power of each model.

1.1 Figurative Language: Irony and Sarcasm

"Figurative language is one of the most arduous topics facing Natural Language Processing" [3]. It differs from the literal language, because figurative language is composed by intricate connotations like sarcasm, irony, metaphor etc. For example, the expression of irony through a sender of the message implies meta-information in order to understand the real meaning. This meta-information can be identified by the context, so the "physical" word isn't enough to capture the sense of the message. This cognitive process is complicated for a human, even more so for a machine.

The study addressed in this thesis mainly concerns the analysis of two rhetorical figures: irony and sarcasm. Linguists interpret irony in two ways: interactive irony and textual irony. The former is contextualized within an oral conversation, where it is expressed by intonation, facial expressions, gestures etc. Textual irony, is defined within either literary or non-literary texts. According to Didio (2007), textual irony is very dissimilar from interactive irony: it is planned and studied. Additionally, it is strongly related to the context of the topic, in the way that only the reader, that knows the context, can catch the figurative message [4]. Irony can be expressed as the opposite of the intended meaning, it has been widely investigated in order to define it in formal terms, but there isn't an exact definition on it because irony covers mainly two type of concept: verbal irony and situational irony. In general, according to E.Partiridge and J.Whitcut irony must not be confused with sarcasm. Irony can be seen as a big umbrella that covers also sarcasm. Indeed, Abrams and Harpham defined verbal irony "as a statement in which the meaning that a speaker employs is sharply different from the meaning that is ostensibly expressed. An ironic statement usually involves the explicit expression of one attitude or evaluation, but with indications in the overall speech-situation that the speaker intends a very different, and often opposite, attitude or evaluation". From this definition, verbal irony encloses sarcasm, but the main difference is that sarcasm is "aggressive humor that pokes fun" [5], it is sharp and more direct. Sarcasm is identified by tendency to aggression, and aimed at hurting. It is "a combination of the processes involved in both humor

and irony, but is hurtful and overtly mocks the target" [4]. According to Lars Elleström situational irony describes an opposition between the expected result and actual results in a certain situation. Lucariello (1994) and Shelley (2001) suggest that situational irony claims "the presence of an observer external to a situation or event that is perceived as ironic" [4]. To highlight how complex the definition of these rhetorical figures is, several definitions of the generic concept of irony are listed: in the study [6] is reported the hierarchical representation of irony. This division is taken from the study by Karoui et al., 2017, where the multi-layer scheme is described. Level 1 is about the irony or not ironic grouping of tweets. Level 2, irony activation: the annotators decide whether the form of contradiction which induces irony is explicit or implicit defined by a disagreement between the two items explicitly quoted in the text, or a contradiction in an external sense between the direct item cited and certain other objects. Level 3, irony categorization: there can be various ways to communicate both overt and implicit forms of activation. At this level, annotators are asked to figure out the ironic device with the following category tags: euphemism, comparison,context shift, false statement, oxymoron or paradox and hyperbole or exaggeration. Level 4 applies to a much finer-grained irony annotation which involves the inclusion of many clues, for example emoticons, punctuation marks, named entities, interjections and punctuation negation words [6].

1.2 Sentiment Analysis, Irony and Sarcasm in Twitter

From the point of view of social media, the focus of this study mainly concerned Twitter platform. The purpose of this website is to connect people through "SMS" where friends can write, see and share a specific message (tweet) with a predefined maximum length (including also images and videos). Every day 500 milion tweets, on average, are shared within the web, and this makes Twitter as a channel for different purpose like Social and Web Analytics for commercial use, behavioural analysis, etc. According to Kaplan and Haenlein, social media have special features: social media are interactive Web 2.0 applications, content is generated by users, such as text posts or comments, digital photos or videos and other data generated through online interactions, users create specific service profiles for the website or app designed and managed by the social media organization. Social media facilitate the development of online social networks by linking a user's profile

with those of other individuals or groups. User-Generated-Content (UGC) denotes any form of user-generated content. There are two types of incentives: implicit, that are not directly monetised: they allow the user to feel good, they can include user relationships, sharing media and experiences. Other common social incentives are the achievement of a status or a particular within the social media, obtaining possible additional privileges. In figure 1.1 are shown two examples of UGC in Twitter, one for irony and one for sarcasm tweet.

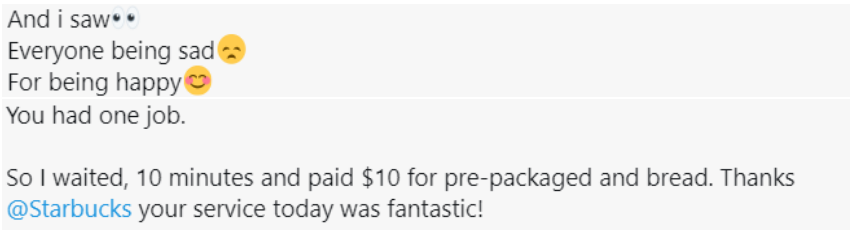

Figure 1.1: Irony tweet vs Sarcasm tweet

From the example above, it is clear how the use of emoticons and punctuation, determine certain understandings related to figurative language. The first case shows a contradiction of sentiment positive and negative, supported by the sad and happy face. In the second one, there is a direct attack to Starbucks due to the cost of a pre-packaged food (bitter message). The monitoring, collection and analysis of data is carried out in a context where social interactions take place. They include sentiment analysis, social network analysis, text analysis, predictive models and recommendation systems. The main problem, in social media, is the presence of ambiguous structure of text: grammatical errors, slang, acronyms, punctuation, emoticons, figurative language, etc. In order to classify a corpus of texts, there are different techniques: lexicon-based approach, supervised Learning, semi-supervised learning and unsupervised learning. In the Lexicon-based approach, once the words are pre-processed, reference vocabularies are taken, where most of the words of a language are present, where values representing the polarity for each word are associated. In the supervised case, models with labelled data are trained, so that by moving from a qualitative to a quantitative representation, it is possible to capture relationships that exist between words, and to define measures (e.g. similarity metrics) that help us to label more accurately whether a sentence is positive, negative or neutral. This is certainly a more precise approach than simply using lexicon based, but the biggest limitation is the availability of labelled data. When only a sample of the data is labelled,

semi-supervised techniques can be used to enrich the dataset. On one hand, if the word fair is labelled in the training, while its neighbour no, legitimate, and are linked by a conjunction like 'and', then the second word is most likely going to have the same polarity as the first. On the other hand, the conjunction but is present between the two words, it means that the second word is most likely going to have the opposite polarity as the first. Another technique is to search in search engines, for example, "was nice and" to see which other words are positively related to nice and then enrich the dataset with new tagged words. Another way for semi-supervised learning is to apply pre-trained word-embedding models, in order to exploit the relationships that have been found between the various words, and use them to label the unknown terms. How can polarity be measured? Find a positive word if it appears most of the times with the word 'excellent'. A negative word, if it appears more with the word 'poor'. Point-wise mutual information is one of the techniques that is used for discovering the co-occurrence between terms: in other words, it tells how two events x and y co-occur if they had been independent:

$$PMI(X|Y) = log_2 \frac{P(x,y)}{P(x)P(y)} \tag{1.1}$$

From the equation 1.1 the probability of words x and y is calculated as follows:

$$PMI(word_1, word_2) = log_2 \frac{\frac{1}{N} hits(word_1 \ near \ word_2)}{\frac{1}{N} hits \ (word_1) \frac{1}{N} hits \ (word_2)}$$

The polarity score is strongly biased from the presence of ambiguity and figurative language. In the recent years, researchers have tried to investigate features that can discriminate better text from these issues. In order to distinguish what is sarcasm and irony from what is not, linguistic characterizations are used such as part of speech, pragmatic particle (like emoticons), n-grams, word case, etc.. Nowadays, there are more sophisticated approaches in order to detect irony and sarcasm that are based on deep learning techniques like recurrent neural networks (RNN) for words dependencies, and convolutional neural networks (CNN) for finding locality information linked to the embeddings terms representations. The most recent state-of-art technique is called "Transformer" model that achieves very high performance in terms of classification tasks and for natural language generation.

1.3 Problem Statement

The main goal of various algorithms, that are used for a classification task, is to map an input X, a matrix of features, to an output Y. For the classification task, referring to unstructured data such as text, is the same. In this case, the matrix is a numerical representation of the text. Given M examples of documents, once the different features are extracted, the objective of a model is to identified relationships between the Y variable, defined as ground truth, among X features.

Sarcasm and Irony detection is defined as a classification problem, where the ground truth is a dichotomy variable 0 and 1, where 0 means that text is not a rhetorical figure, otherwise is an ironic or sarcastic statement (depending on the task). The mathematical formulation can be defined as follows:
The main idea is that each tweet can be represented by vectors where every value is a feature associated to a word within the text. Taking into account M tweets, X is a matrix of dimension

$$M_{sentences} \times N_{words} \times P_{features}$$

and Y is a vector composed by 0 and 1 of dimension $M_{sentences}$. The set of M examples is identified as $\{(x_1, y_1), ..., (x_m, y_m)\}$ where $x \in \mathbf{R}^{N \times P}$.

The objective function of model is

$$F : X \rightarrow Y$$

Finding the most representative matrix X for identifying either sarcasm or irony is one of the most challenging task in NLP.

1.4 Research Questions, Objectives and Contributions

The decision to analyse the problem of irony and sarcasm is based on the interest of answering five research questions:

1. What are the most representative features for identifying sarcasm and irony patterns?

Extracting features for Machine Learning algorithms can be a good starting point for understanding what is representative or not. Furthermore, applying the most advanced embeddings, from Deep Learning models, could be another point of view for answering this question.

2. What is the generalisation power of the developed models?
   Using different training and test sets (in-domain & out-domain) are the basis for recognizing if a model is finding the right patterns or not.

3. How encoder outputs layers, from Transformer architecture, can be exploited for these tasks?
   Finding the right combination of the output encoder layers is a crucial point for gaining accuracy by the models.

4. Are emotional embeddings useful for recognizing either sarcasm or irony statements?
   Exploiting the DeepMoji model as a feature extractor, the analysis is conducted by applying this information to one of the models proposed as a new methodology.

5. Can Ensemble methods help the quality prediction? Developing different approaches ranging from Bayesian Model Averaging to Soft/Hard classification could take advantage of the models' variety.

In order to address these questions, the main objectives of this study are based on:

- Comparison between three different studies that are the state-of-the-art in irony and sarcasm detection. From Machine Learning applications to Transformer architectures.

- Developing different new methodologies in order to obtain best performances, enhancing the power of self attention layers and ensemble methods.

- Trying to understand the limits of these methodologies and the data quality.

The key points that emerged at the end of this research highlight several contributions:

- Comparative study of the state-of-the-art to understand the generalising power of irony/sarcasm detection patterns.

- The development of a new methodology, based on the combination of multiple output encoder layers of the BERTweet model, for creating a more contextualized sentence embeddings.
- The development of a model based on the emotional features of DeepMoji, built on the concept of self attention layer, called DeepMoji Features-based.
- The creation of Ensemble of ensembles method, where a series of models, trained on different aspects of the text, identify the various patterns that define the figurative language.

1.5 Structure of the Thesis

A brief outline of the organisation of the thesis is given in this paragraph.

Chapter 2: Theoretical Aspects

In this section all theoretical aspect are addressed. How text can be represented as vector and the state-of-the-art features used for identifying irony and sarcasm within the text. This is followed by an explanation of the theoretical foundations of the Machine Learning models used within the study. Subsequently, the basics of neural networks are explained with a greater focus on recurrent layers, convolution, and BERT's architecture.

Chapter 3: Proposed Methods

In this chapter are shown the new methodologies that are the main core of the this research. Three types of approaches are discussed and developed: two Deep Learning models with convolution and self attention layers, where the input features are based on encoder output layers and emotional embeddings extracted from DeepMoji model. The last approach is referred to an ensemble method: simultaneously exploiting the Bayesian Model Averaging and soft/hard classification approaches, which allow the use of different models analysing multiple aspects of the text such as pragmatic particles and the part of speech to emotional embeddings.

Chapter 4: Dataset and Performance Measures

This chapter is a part of the experimental set up. The datasets are shown divided by the task. There is training set and test set sections where all the sources are defined. Furthermore, there is a paragraph where the tweets lengths distribution and summary statistics are illustrated divided by sarcasm and irony. Finally, the evaluation metrics used to compare the different models in these two tasks are shown.

Chapter 5: Experiments

This section summaries and compares all the results obtained from the comparative analysis to the new methodologies developed. The main metrics used are precision, recall, F-measure and accuracy.

Chapter 6: Conclusions

The last chapter summaries all the results obtained enhancing the most important information. Limits and next step for continuing the methods implemented are illustrated.

Chapter 2

Theoretical Aspects

2.1 Text Representation

In Natural Language processing there is an important pipeline to follow in order to get right information from the text:

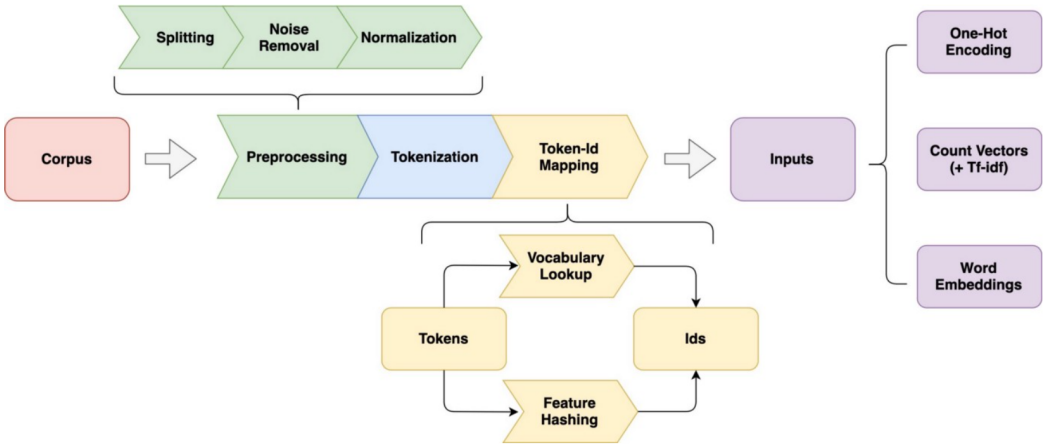

Figure 2.1: Preprocessing steps for NLP input, source [7]

The preprocessing phase is the most critical part, it differs depending on the type of task to be performed. The first step is to remove noise such as stopwords, that are those commonly used words that do not bring any useful information to the text. Typical examples of stopwords are conjunctions, adverbs, prepositions, pronouns, commonly used verbs. Lemmatization is the process that reduces words from their flexed form to their canonical form, called lemma. Stemming: what stemming does is to remove the ending of a word leaving only the root, it is a simple process that consists of truncating the final

part of the word according to a set of rules. The purpose of stemming and lemmatisation is the same: trying to reduce the size of the vocabulary. However, lemmatisation is a more sophisticated technique that leads to better results, even if it is more expensive on a computational level. Other steps can be lower case normalization, extend words contractions, punctuation and emoji removal. In the case of tweets, mentions and hashtags are usually eliminated.

Once the corpus is cleaned, after the tokenisation process, there are several ways to represent it in numerical form, usually vectors or matrices, which allow to make "measurements" from textual data. The most common model for representing text is the Bag of Words (BOW): vector representation does not consider the ordering in words. There are different ways to identify a BOW model:

- Binary term-document incidence matrix: each document can be represented by a set of terms or by a binary vector [0,1]. If it is 1, the term is present while 0 the term is missing. There is a limit: the frequency of each term is lost, indeed in this case a document is represented by a vector of 0 and 1.

- Term-document count matrices: consider the number of occurrences of a term in a document, where each document is a count vector in N.

This type of representations produce sparse matrix form. It is just a waste of memory and it is computationally expensive, especially if there is a large collection of documents. Another bottle neck is that the positional information of terms are lost: there are different ways to obtain the relations within words, such us n-grams (a contiguous sequence of N tokens from a given piece of text). In order to circumscribe the problem of the sparsity within matrices, these type of representations can be seen as a more advanced way to represent text through a BOW model:

- Term Frequency - Inverse Document Frequency (tf-idf): the general idea comes from Luhn's Analysis (1958): Luhn notes that "the frequency with which some words appear in a text provides an important indication of the significance of words". Starting from this concept, jointly together with the Zipf's curve, that describes the discriminating power of significant words, the general idea is to use a weighting schema based on the frequency terms. Tf-idf weighting of a term is the product of

it is term frequency weight and it's inverse document frequency weight:

$$w_{t,d} = \frac{tf_{t,d}}{max_{ti} tf_{ti,d}} \times \log_{10}(\frac{N}{df_t}) \tag{2.1}$$

The values obtained by this weighting schema are replaced within the matrix in order to reduce the sparsity and to gain more information on each word.

There are several ways for creating dense matrices: the most recent technique, which has produced better results compared to more traditional methods such as those mentioned above, is based on the construction of words embeddings. It indicates a set of techniques in NLP where words or phrases from the vocabulary are mapped to vectors of real numbers. It involves a mathematical embedding from space with many dimensions per word to a vector space with a much lower dimension. Models to generate this mapping include: Count-based (distributed semantic models) and Predictive (Neural Network) models. The basic concept refers to the fact that words that have the same meaning have a similar representation. The main idea is that each term has a dense vector representation, where each value can describe a specific information of the associated word: these values are called word features. Assuming a model is trained on the basis of completing a sentence by entering a word: for example, during the training phase it learns that after "I want a glass of orange ....." it most likely enters the word juice. But if the sentence is constructed differently, like "I want a glass of apple.....(juice)", the model is unable to define a relationship between orange and juice. This is the reason why the distance within vectors is always the same, in case the one hot encoding representation is used. Embeddings allow to solve this limit using the "featurized representation". In word embeddings, the set of words that make up a sentence are represented as follows:

$$\begin{array}{c} \\ I \\ love \\ NLP \\ course \end{array} \begin{array}{c} \begin{array}{cccc} f_1 & f_2 & \dots & f_n \end{array} \\ \begin{pmatrix} 0.435 & 0.191 & \dots & -0.934 \\ 0.768 & -0.123 & \dots & 0.169 \\ -0.232 & 0.944 & \dots & 0.212 \\ 0.991 & 0.810 & \dots & 0.021 \end{pmatrix} \end{array}$$

In the matrix is shown the structure of the words representations. Each row is associated to the terms, and each column corresponds to a specific feature linked to the

i-th word. In order to create this featurized representation, different models are used: count-based models, where the complexity of the term-context matrix is reduced by factoring it into a denser matrix where each line contains the vector representation of words. The most used techniques are Latent Dirichlet Allocation (LDA), Singular Value Decomposition (SVD) and GloVe Predictive models: directly try to predict a word from its neighbours in terms of learned small dense embedding vectors (considered parameters of the model). These are mainly neural networks (which directly try to predict a word and in the training phase the parameters, which represent the word embeddings, are optimized) and Word2Vec. From the point of view of the Glove: it is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space. It is training only on the non-zero elements in a word-word co-occurrence matrix, and this produce less space in terms of matrix sparsity. GloVe is essentially a log-bilinear model with a weighted least-squares objective. The main intuition underlying the model is the simple observation that ratios of word-word co-occurrence probabilities have the potential for encoding some form of meaning. Predictive models are slightly different: taking word2vec as an example, this model is based on the following assumption: the words that have a similar context they have a similar meaning. To achieve a good performance, requires very large texts in the learning phase (>10M of word). The main two models for word2vec are Continuous Bag of Words (CBOW) and Skip-Gram. The aim of CBOW and Skip-Gram is the same: creates a language model that from the input the context the output is the word that follows the context.

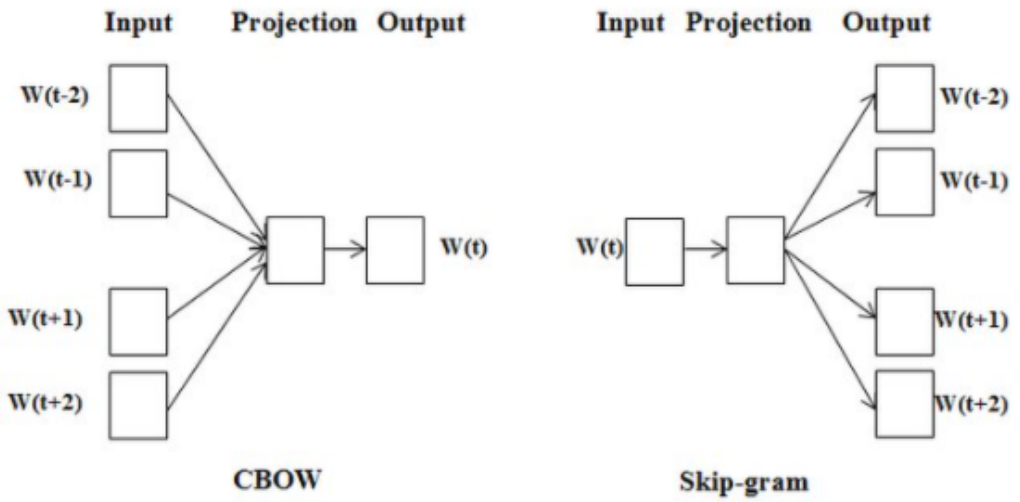

Figure 2.2: CBOW and Skip-Gram architectures, source Nailah Al-Madi.

From figure 2.2 it is clear the difference between the two models. For Skip-Gram the main task is to predict the surrounding words (context words), based on the current word (the center word). While, CBOW tries to predict the current word (the center word) based on the surrounding words. In the section 2.3.4 the most recent technique for creating word and sentence embeddings based on transformer architecture is explained.

2.1.1 Extract Features from Text

Once the data has been prepared and cleaned, the next step is to extrapolate the various features that can better discriminate the text. This section shows the variables that are most commonly used to solve the problem of identifying irony and sarcasm.

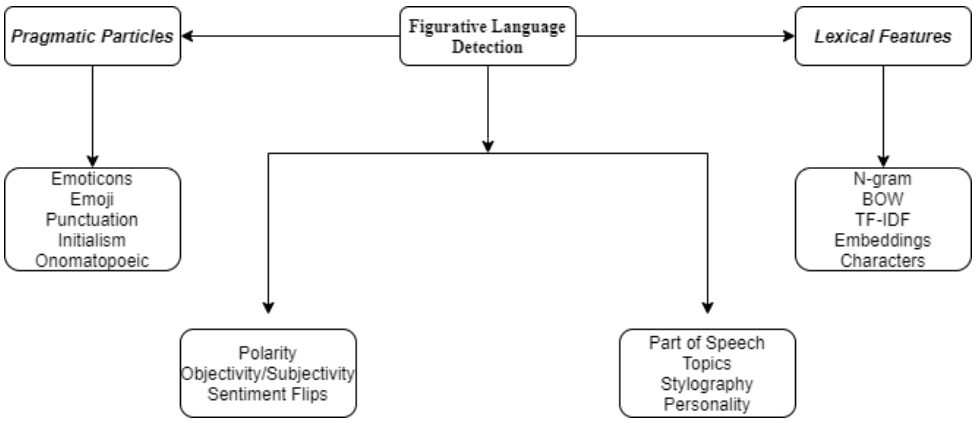

Figure 2.3: Most used features for irony and sarcasm detection

In figure 2.3 are shown the most salient features: the researchers try to answer which features are the best by looking at different aspects of the text. As can be seen, in addition to lexical analysis, pragmatic particles are considered. Twitter users make a lot of use of various pragmatic particles to express their opinions, feelings or to send a specific message. For example, emoticons can be useful for identifying rhetorical figures. A message such as "Today is a nice day :(" would seem to be a positive message at first glance. Thanks to the emoticon an opposite sentiment emerges, which leads to the understanding that the tweet is ironic. Punctuation can be useful to mark a message and intensify thoughts: "I'm really angry!!!!!!!". As for the use of n-grams, they are useful when two, or more, words taken together can express a different message than when analysing the individual ones. When a double polarity is expressed, a document is defined as neutral: to better identify this concept the term sentiment flip is used. This information is useful, especially for the identification of irony. Another variable is the level of subjectivity/objectivity of the message itself. The relationship between onomatopoeic figures and initialism with rhetorical figures are identified by various scholars, specifically an examples of onomatopoeic words are: 'haha', 'pop', 'bang', 'ugh', 'rush', 'hack', while for initialisms: 'LOL', 'OMG', 'LMAO', 'STFU'. All these features listed are derived directly from the 'explicit' analysis of the text. A very important aspect is what lies behind the text: the identification of the various topics is one of the approaches that seem to be useful in better discriminating what is ironic and what is sarcastic. When discussing hot topics such as politics, it is easy to think that many messages may contain ironic or sarcastic phrases referring to situations that define the political context. While if talking about economics it is much more difficult to find a document that uses rhetorical figures. Two other elements to be taken into account are aspects relating to the user's personality and stylography. For example, when analysing a user's profile, it emerges that he/she very often uses ironic/sarcastic messages. This implies that when a tweet related to the user is identified, there is a higher probability of encountering a post with these characteristics. In 2.3 the part of speech is present. Words are divided into lexical categories: the same word can belong to different classes, as an adjective or noun, but this depends solely on the sentence it belongs to. Several models are used in the literature, where the most commonly used is Penn Treebank. A treebank is a parsed content corpus that explains syntactic or semantic sentence structure. Specifically, the Penn Treebank was developed

using documents from the Wall Street Journal. Peen Treebank is developed for general

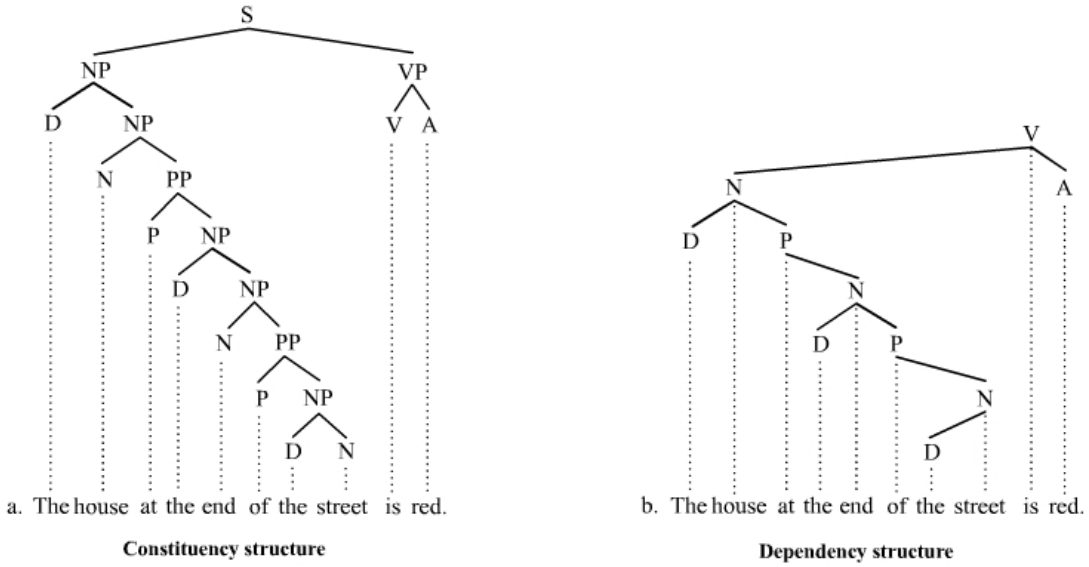

Figure 2.4: Penn Treebank example, source Wikipedia

purpose: labelling execution corrupts on out-of-domain information, and Twitter posts extra challenges due to the conversational nature of the content and the lack of ordinary orthography [8]. In this study, a different tagger model is applied, the Tweebank. This treebank is created on the basis of the structure of Twitter posts: this tagger is built for English data from Twitter, where the model's accuracy achieves 90%. This tagger is a conditional arbitrary field, empowering the joining of arbitrary neighbourhood features in a log-linear algorithm. The base features incorporate: a variable for each word type, a set of features that check if a word contains numbers or hyphens, suffix features up to length 3, and variables that identifying capitalization designs within the term [8]. There are new additional tags for different specific words: hashtag, mentions, url, emoticons, abbreviations etc.. These tags are very useful information in order to analyse in a granular scope different tweets. In figure 2.5 are displayed some of the tags built in the Tweetbank model. Especially, this algorithm is applied for the Machine Learning section in order to replicate one of the state-of-the-art methods used for the comparative analysis.

Twitter/online-specific			
#	hashtag (indicates topic/category for tweet)	#acl	1.0
@	at-mention (indicates another user as a recipient of a tweet)	@BarackObama	4.9
~	discourse marker, indications of continuation of a message across multiple tweets	RT and : in retweet construction RT @user : hello	3.4
U	URL or email address	http://bit.ly/xyz	1.6
E	emoticon	:-) :b (: <3 o_O	1.0
Miscellaneous			
$	numeral (CD)	2010 four 9:30	1.5
,	punctuation (#, $, ' ', (,), ,, ., :, ``)	!!! .... ?!?	11.6
G	other abbreviations, foreign words, possessive endings, symbols, garbage (FW, POS, SYM, LS)	ily (*I love you*) wby (*what about you*) 's ♫ --> awesome...I'm	1.1

Figure 2.5: Tweebank, additional tags, source [8]

In the recent years, academics have studied different features combinations as input for statistical classifier in order to understand the irony and sarcasm phenomena within the text [9]:

- Tsur et al 2010 define as markers for sarcasm punctuations and sarcastic patterns by applying semi-supervised techniques. With reference to punctuation based features, sentence length number of "!", "?", quotes and capitalized words are taken into account.

- Gonzalez-Ibanez et al. 2011 apply user mentions, emoticons, unigrams and sentiment lexicon based features.

- Reyes and Rosso 2012 distinguish six features for irony characteristics: funny profiling, part of speech, positive/negative information, n-grams, pleasantness and affective profiling.

- Riloff et al. 2013 introduce a set of patterns related to sentiment flip, specifically positive verb and negative situation.

- Liebrecht et al 2013 exploit intensifier words such as so, such etc.. and discover that hyperbole is an import mark for sarcasm.

- Buschmeier et al 2014 identifies hyperbole as Liebrecht. In addition, they take advantage of interjection and ellipsis.

- Barbieri et al 2014 focus on the gap between rare and common words, written spoke style uses, intensity of the adverbs and adjectives, structure of the text such as length, punctuation, emoticons and links. Sentiments features based (positive and negative words) and the measure of possible ambiguities.

- Bouazizi and Ohtsuki 2015 distinguish sentiment features and word information.

- Joshi et al 2016 exploit for the first time word embedding similarity for assessing these tasks.

These features are used as input either for statistical classifier or Deep Learning algorithms. In the next sections are explained different methodologies, that are based on distinct features. This is important for comparing what strategy is better and if the variables used are enough for a real understanding of what lies behind irony and sarcasm.

2.2 Machine Learning Perspective

Tom M. Mitchell defines Machine Learning as: "A programme is said to learn from experience E with reference to some class of task T and with performance measure P, if its performance in task T, as measured by P, improves with experience E". Starting from this citation it is clear what is the main purpose of Machine Learning (ML). Thanks to the high data availability, Machine Learning models have become more accurate, especially in Natural Language Processing task which needs a very high amount of textual data to understand what lies behind the structure of the text. Machine Learning can be divided into Supervised Learning, Unsupervised Learning, Semi-Supervised Learning and Reinforcement Learning. In this thesis Supervised Learning is applied: text is associated to ground truth such as irony or not irony and sarcasm and not sarcasm. In subsection 2.2.1, the study carried out by [10] is illustrated and then the theories underlying Machine Learning models used to reproduce this research are shown.

2.2.1 Detecting Irony and Sarcasm in Microblogs: The Role of Expressive Signals and Ensemble Classifiers

The study [10] evaluates an ensemble approach called Bayesian Model Averaging (BMA) that considers different Machine Learning models based on their the reliability and the marginal likelihood of the predictions. Various combinations of features are studied from term frequency matrix as baseline to part of speech information and pragmatic particles. The concept behind an ensemble mechanism is to take advantage of multiple different classifiers' properties by integrating them so as to achieve better performance than the best individual algorithm. This paper compares two main different ensemble methodologies: Majority Voting (MV) and BMA. One of the most common ensemble method is MV, which is defined by a collection of "experts" that classifies each observation by considering the vote of each model as equally significant and decides the final prediction by choosing the most frequent label prediction. The MV technique can be defined as hard classification because it refers to the label given by each model, without using the estimated probability. Another ensemble technique is called soft classification, where the marginal probability distributions of each model are used to make the label choice. The following formula 2.2 is used to combine the various probabilities of each algorithm belonging to the ensemble method:

$$l_{ensemble}(m) = argmax_{l(m)} \sum_{i \in C} P(l(m) \mid i) \tag{2.2}$$

where i is the classifier belonging to the set of classifier C and m is the label associated. The most significant constraint proposed by MV is that the models to be used in the combination have uniform distributed weights independently of their reliability. In the case of irony and sarcasm detection, referring to tweets, it is important to consider the reliability of the models. Irony and sarcasm comments are far less common than the other tweets: it is important to measure how reliable each algorithm is on minority class prediction (with reference to figurative language). To this end, the complexity generated behind by data and algorithms may be processed by considering the Bayesian method. In particular, BMA considers the information resulting from reliability and marginal probability predictions of each algorithm: suppose to have a collection of models C and a

dataset D, the BMA associates a tweet m a label $l(m)$ which it maximises:

$$P(l(m) \mid C, \mathcal{D}) = \sum_{i \in C} P(l(m) \mid i, \mathcal{D}) P(i \mid \mathcal{D}) \tag{2.3}$$

where $P(l(m) \mid C, \mathcal{D})$ is the estimated marginal distribution on a model i associated to the label predicted. The posterior probability $P(i \mid \mathcal{D})$ is expressed as:

$$P(i \mid \mathcal{D}) = \frac{P(\mathcal{D} \mid i) P(i)}{\sum_{j \in C} P(\mathcal{D} \mid j) P(j)} \tag{2.4}$$

in 2.4 $P(\mathcal{D} \mid i)$ is the classifier likelihood and $P(i)$ is the prior probability of the model i. The denominator of 2.4 is assumed to be a constant value indeed it can be ignored. The label assignment strategy of the BMA is based on the following formulation:

$$\begin{aligned} l^{\text{BMA}}(m) &= \arg\max_{l(m)} P(l(m) \mid C, \mathcal{D}) \\ &= \arg\max_{l(m)} \sum_{i \in C} P(l(m) \mid i, \mathcal{D}) P(i \mid \mathcal{D}) \\ &= \arg\max_{l(m)} \sum_{i \in C} P(l(m) \mid i, \mathcal{D}) P(\mathcal{D} \mid i) P(i) \end{aligned} \tag{2.5}$$

The equation 2.5 assigns to a tweet m the label $l^{\text{BMA}}(m)$. The study [10] suggests to replace the measure $P(\mathcal{D} \mid i)$ with the F_1-measure (more details in the section 4.3) calculated for each model i during preliminary investigation. To be specific, in order to obtain the reliability of the models, the F_1 score metric, for each classifier, is taken into account. The metric is estimated by considering the mean value obtained in ϕ-folds cross validation. This procedure is useful for understanding the generalization error of the classifiers: this means that overfitting problem is contained. $P(\mathcal{D} \mid i)$ is computed separately for each cross validation fold generated. The conditional probability can be approximated as:

$$P(\mathcal{D} \mid i) \approx \frac{1}{\iota} \sum_{\iota=1}^{\phi} \frac{2 \times P_{i\iota}(\mathcal{D}) \times R_{i\iota}(\mathcal{D})}{P_{i\iota}(\mathcal{D}) + R_{i\iota}(\mathcal{D})} \tag{2.6}$$

where ϕ is the number of folds, in particular for this study $\phi = 10$. $P_{i\iota}(\mathcal{D})$ is the precision and $R_{i\iota}(\mathcal{D})$ is the recall estimated for each model in the validation set. An important aspect to take into account is the search of the optimal set of classifiers to be consider

in order to find global minima. The set must be based on minimising three main errors that cause models to not generalise well: bias, variance and noise. One one hand, when the number of individual model grows the variance in the ensemble reduces. On the other hand, the variance increases due to the number of classifiers used, especially when classifiers are dependent and positively correlated. In order to find an optimal set of classifier, the paper introduces the concept of contribution derived by each model:

$$r_i^S = \frac{\sum_{j \in \{S \setminus i\}} \sum_{q \in \{0,1\}} P(i = 1 \mid j = q) P(j = q)}{\sum_{j \in \{S \setminus i\}} \sum_{q \in \{0,1\}} P(i = 0 \mid j = q) P(j = q)} \tag{2.7}$$

referring to 2.7 $P(j = q)$ is the prior prediction of model j that either properly and wrongly associated to m a label. Specifically, $P(j = 1)$ indicates the percentage of m observations that are properly identified, while $P(j = 0)$ denotes the ratio of m observations that are wrongly predicted. After classifier's contribution has been calculated, the paper proposes a backward elimination algorithm in order to find the right models' combination. This strategy is modified, and an efficient brute-force approach is developed in order to evaluate all the possible combinations and ensure a possible global optimal set.

2.2.1.1 Features Extraction and ML models

The second part of the study is mainly focused on the choice of variables to be used such as the BOW representation based on the words frequency. In addition to that, part of speech and pragmatic particles are taken into account. Specifically, for pragmatic particles the following are used: emoticons, initialism, onomatopoeic expression and punctuation. From the classifiers' point of view, different models are analysed: Multinomial Naive Bayes, Support Vector Machines, Bayesian Networks and Decision Trees.
In the reproduction of this study, several changes are made:

- The BOW representation is replaced with sentence embeddings extracted from Bert-Tweet. In order to have a better training speed, Principal Component Analysis is applied to reduce the dimensionality of the embeddings, while maintaining 95% variance. The vectors length is reduced from 768 to a size of 172.

- Additional features are taken into account: polarity information, subjectivity and

objectivity of each tweet by applying an intermediate model from [11] library.

- Emoticons and emojis are represented in two different ways: term frequency matrix of all the emoticons/emojis and a 2 dimensional matrix composed by counting positive and negative emojis.

- Different models are applied: Random Forest, Hist Gradient Boosting, Adaptive Boost, X Gradient Boosting and Logistic Regression. The choice of these models is justified on a simple need: rapid convergence during training.

All these models are optimized using a 2-steps tuning: the first one is regarding to the hyper-parameters of each model. This optimization phase is based on 10-folds cross validation using two different strategy: Random Search and Bayesian Search. Due to the limitation of computing power, the optimisation budget is set at 10. The second phase is based on the estimation of the classifiers' reliability and the application of BMA. These optimization techniques are applied in order to compare which of those are better given a limited training's budget. In the next pages, these methodologies are explained in detail, section 2.2.4. All the mathematical aspect of ML models is covered in depth in the section 2.2.3.

2.2.2 Principal Component Analysis

As mentioned above, the features baseline are referred to dimensionally reduced embeddings by applying the principal component analysis (PCA) technique. For this reason, this section is devoted to providing an overview of how this methodology works. PCA is a dimensionality-reduction approach frequently used to decrease the dimension of large data sets by creating latent variables that syntetise the original features space. It is clear that the decrease of the number of variables in a data set is at the cost of precision, but it is useful in order to process all the information in a faster way, with the aim of keeping the variance as high as possible. The first step is to compute the standardization for the input variables: it is important to highlight this step for the reason why features with large ranges impact the projection in the space by dominating over the features with small values, in formula: $z = \frac{x-\overline{x}}{\sigma}$. The second step is to estimate the covariance matrix $p \times p$ of the input features: this phase is crucial in order to determine the relationships

within the variables, indeed features that are highly correlated are grouped together. Once the covariance matrix has been calculated, for identifying the principal components, eigenvectors and eigenvalues are estimated.

1. Compute dot product matrix: $\mathbf{X}^T\mathbf{X} = \sum_{i=1}^{N} (\mathbf{x}_i - \boldsymbol{\mu})^T (\mathbf{x}_i - \boldsymbol{\mu})$

2. Eigenanalysis: $\mathbf{X}^T\mathbf{X} = \mathbf{V}\boldsymbol{\Lambda}\mathbf{V}^T$

3. Compute eigenvectors: $\mathbf{U} = \mathbf{X}\mathbf{V}\boldsymbol{\Lambda}^{-\frac{1}{2}}$

4. Keep specific number of first components: $\mathbf{U_d} = [\mathbf{u}_1, \ldots, \mathbf{u}_d]$

5. Compute d features: $\mathbf{Y} = \mathbf{U_d^T}\mathbf{X}$

The covariance matrix of $\mathbf{Y}$ is $\mathbf{Y}\mathbf{Y}^T = \mathbf{U}^T\mathbf{X}\mathbf{X}^T\mathbf{U} = \boldsymbol{\Lambda}$ and the projection matrix correspond to $\mathbf{W} = \mathbf{U}\boldsymbol{\Lambda}^{-\frac{1}{2}}$. For the sake of clarity refer to [12].

2.2.3 BMA Machine Learning Classifiers

Models such as Random Forest, X Gradient Boosting, Adaptive Boost and Hist Gradient Boosting are defined as tree based algorithms. These methods are an evolution of Decision Tree model. Generally, these kind of models divide the variable space into subsets of quadrilaterals, where for each of them a function is estimated that interpolates the points within that space. To understand what is behind the logic of trees, a well-known model is explored: Classification and Regression Trees (CART) is simple binary tree with powerful predictive performance used for classification and regression task. The main difference lies in the decision function for dividing the dataset within a tree. For classification task two different impurity measures are applied: Gini or Entropy function. While, for regression task the splits are identified by minimizing the residual sum of squares.

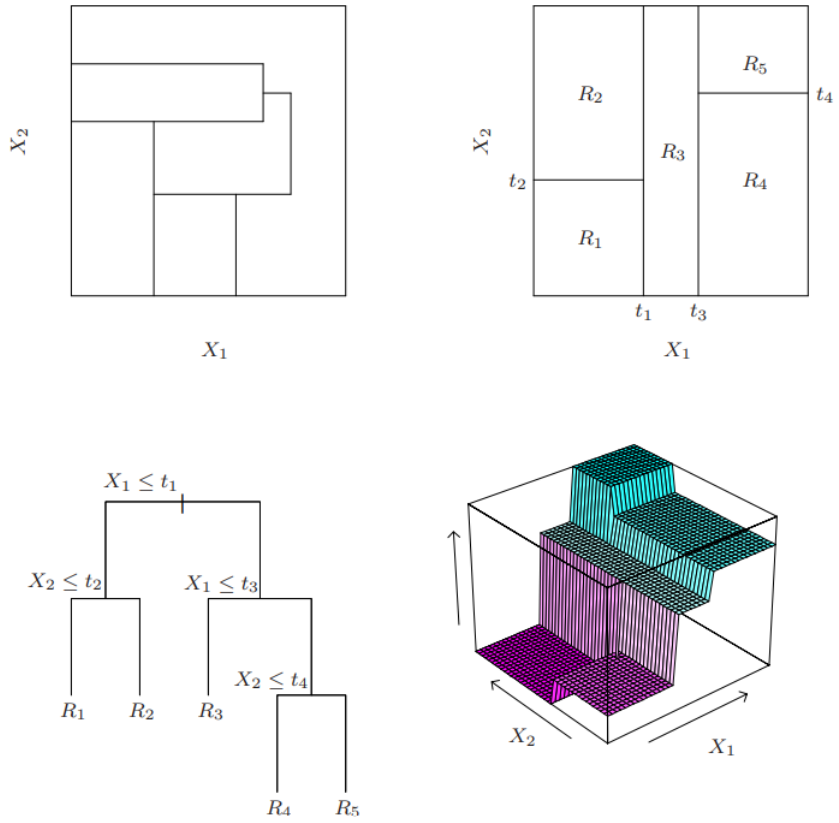

Figure 2.6: CART example, source [13]

In 2.6 is represented how a Decision Tree split the variables space of X_1 and X_2 features given a continuous response Y. In each region a function $\hat{F}(X)$ is estimated based on a split-point in order to achieve the best fit [13]:

$$\hat{f}(X) = \sum_{m=1}^{5} c_m I\left\{(X_1, X_2) \in R_m\right\} \tag{2.8}$$

In the equation 2.8 $i = 1, ..., 5$ referring to the number of rectangles identify, since the stopping criterion has been reached. The recurring binary tree's interpretability is a major advantage. A single tree describes the variable space partition.

The application of this method for a classification task is quite similar. As mentioned above, the adjustments to be made in the algorithm are: splitting criteria leaves and the pruning. The residual sum of squares doesn't fit well for this kind of task. Let's consider $\hat{p}_{mk}$ as the proportion of label k instances in leaf m:

$$\hat{p}_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} I\left(y_i = k\right) \tag{2.9}$$

The model associates the observations to majority class $k(m)$ in node m. There are three main impurity measures $Q_m(T)$ where T is the subset Tree, and m is the associated node:

$$\frac{1}{N_m} \sum_{i \in R_m} I\left(y_i \neq k(m)\right) = 1 - \hat{p}_{mk(m)} \tag{2.10}$$

$$\sum_{k \neq k'} \hat{p}_{mk} \hat{p}_{mk'} = \sum_{k=1}^{K} \hat{p}_{mk}\left(1 - \hat{p}_{mk}\right) \tag{2.11}$$

$$-\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk} \tag{2.12}$$

where 2.10 is the misclassification error, 2.11 is referred to Gini index and 2.12 is identified as Cross-entropy or deviance. The Decision Tree suffers from high variance. This is due to the fact that slight changes in the observations cause instabilities in the splits. This problem arises from the hierarchical structure of the tree, where an error at the top of the tree propagates to the bottom. Thus, the tree suffers from instabilities. To overcome this problem, several techniques are used, including bagging, which reduces this variance. Another issue is the definition of a single Decision Tree, which in some cases produces weak results [13]. To overcome all these limitations, boosting technique is introduced such as Adaptive Boosting and bagging method to limit the variance problem like Random Forest.

Boosting technique pooled the effects of various "poor" models in order to create a strong ensemble. Freund and Schapire (1997) develop a boosting method called "AdaBoost.M1": it takes into account a binary classification task, where $Y \in \{-1, 1\}$. The aim is to training a model $G(X)$, where X is the vector of independent variables, based on an error rate defined as:

$$\overline{\text{err}} = \frac{1}{N} \sum_{i=1}^{N} I\left(y_i \neq G\left(x_i\right)\right) \tag{2.13}$$

The goal of the boosting is essentially passing series of poor classifier to different set of the training data, in order to produce a sequence of m models $G_m(x), m = 1, 2, ..., M$. The function $G(x)$ is defined as

$$G(x) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m G_m(x)\right) \tag{2.14}$$

where $\alpha_1, ..., \alpha_M$ are the weights of each model m, based on the reliability, estimated by the boosting procedure. In 2.7 is shown the AdaBoost schema.

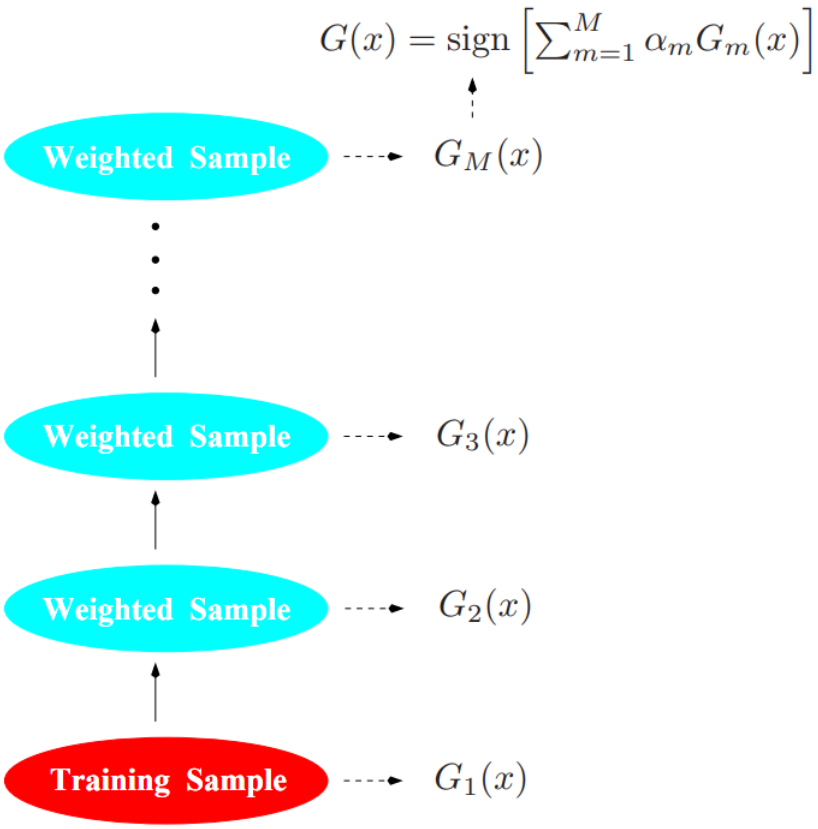

Figure 2.7: Training schema AdaBoost.M1, source [13]

The sign function is used in order to attribute the value in the set $\{-1, 1\}$. In each iteration, the training set is weighted by $w_1, ..., w_N$ constants, in order to introduce variance and build a stronger ensemble model. For the sake of clarity, the pseudo code is reported hereby:

1. The weights $w_i = 1/N, i = 1, 2, ...N$ are set
2. Starting from m = 1 to M:
   - Fit each model $G_m(x)$ to the training data set re-weighted by w_i
   - Calculate
   $$\text{err}_m = \frac{\sum_{i=1}^{N} w_i I\left(y_i \neq G_m\left(x_i\right)\right)}{\sum_{i=1}^{N} w_i}$$
   - Estimate $a_m = log((1 - \text{err}_m)/\,\text{err}_m)$
   - Set $w_i \leftarrow w_i \cdot \exp\left[\alpha_m \cdot I\left(y_i \neq G_m\left(x_i\right)\right)\right], i = 1, 2, \ldots, N$
3. Calculate $G(x) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m G_m(x)\right)$

For more details refer to [13]. As far as the application of boosting is concerned, this leads, as mentioned above, to an improvement in performance. However, this technique causes limitations, especially when it comes to decision trees: interpretability and speed. Especially, for AdaBoost, there are limitations on robustness when class distributions overlap. To solve these problems, a new class of models has been developed: gradient boosted models (GBM). They are a generalisation of boosting tree [13], and incorporate three components: an objective function such as loss function to be minimized, a poor learner for predicting instances and an additive algorithm to append poor models for minimizing the loss function. On one hand, Decision Tree is defined as a poor learner because reaches a limited number of hyper-parameters during the training phase, such as depth, number of leaves and splits. On the other hand, trees are the right learners regarding the definition of additive model. They can be appended one at time, without modifying the existing learners. In order to minimizing the loss function, the gradient descent algorithm is applied. For further information refer to [14].

In this thesis, different applications of boosting, based on tree, are used: Hist Gradient Boosting (Hist GBoost) and Extreme Gradient Boosting (XGBoost) that are an efficient implementation of the gradient boosting algorithm. Additionally, another meta-learning algorithm based on trees is applied, Adaptive Boosting SAMME. Hist Gradient Boosting is a powerful application of the LightGBM. It implements a modern one-sided sampling gradient (GOSS) strategy to select observations for determining a split value. XGBoost utilises pre-ordered algorithms and histograms to calculate the most effective split value. GOSS is a new concept for sampling observations dependent on gradients. In other words, the algorithm filters out observations that have a small gradient, because they have a small error term, in the training phase. Considering only instances with large gradient values implies that changes in the distribution of the training data are introduced. Therefore, the algorithm considers all observations with a high gradient and applies downsampling on instances with a small gradient. HistGBoost abd XGBoost are based on the histogram of features for finding the splits node, in order to speed up the process. Another difference lies in the nodes growth: XGBoost is based on the trees' depth, and it splits based on the pruning in backward step and delete nodes if the gain doesn't increase. HistGBoost applies best-first methodology, and it increases the number of nodes based of the minimization of the loss. XGBoost doesn't use the GOSS algorithm. For the sake of clarity

refers to [14].

From the point of view of Adaptive Boosting SAMME, it differs from the original implementation of Adaptive Boosting developed by Freund and Schapire. This model is an extension of the previous one, where the difference lies in the fact that it can be generalised to a multi-class problem. Generally speaking, Stagewise Additive Modeling using a Multi-class Exponential loss, SAMME works as follows:

1. The weights $w_i = 1/N, i = 1, 2, ...N$ are set
2. Starting from m = 1 to M:
   - Fit a tree model $G_{(m)}(x)$ to the training data set re-weighted by w_i
   - Calculate:
   $$\text{err}_m = \frac{\sum_{i=1}^{N} w_i I\left(y_i \neq G_m\left(x_i\right)\right)}{\sum_{i=1}^{N} w_i}$$
   - Estimate: $a_m = log((1 - \text{err}_m)/\text{err}_m) + log(K - 1)$
   - Set $w_i \leftarrow w_i \cdot \exp\left[\alpha_m \cdot I\left(y_i \neq G_m\left(x_i\right)\right)\right], i = 1, 2, \ldots, N$
   - Re-normalize w_i
3. Calculate $G(\boldsymbol{x}) = argmax_k \sum_{m=1}^{M} \alpha_{(m)} \cdot I\left(G_{(m)}(\boldsymbol{x}) = k\right)$

SAMME model is quite similar to the original implementation, except for a few points, such as for the estimate α_m the term $log(K - 1)$ is added, the weights are re-normalised and ultimately the final function changes, no longer using the sign function but $argmax$ [15]. The fourth trees based algorithm used for this study is the Random Forest model. As mentioned above, in order to limit the issue of the high variance in Decision Tree model, bagging is the methodology that reduce the variance within the estimated function $\hat{f}(X)$. This technique computes the mean of the prediction of learners, in order to have a lower variance. Bremain, develops Random Forests model: it's a alternation of the bagging based on building a set of de-correlated trees[13]. The main idea of the Random Forests model is the following:

1. For b = 1 to B, compute:
   - Apply a bootstrap sample Z^* of size N from the training set.

- A Random Forest T_b is added to the sampled data, by recursively repeating the following steps for each terminal leaf of the tree, until the minimum leaf size n_min is reached.
  - Choose randomly a subset m of features from p features.
  - Take the best feature/split-point among the m variables.
  - Split the leaf into two daughter leaves.

2. Output the ensemble of trees $\{T_b\}_1^B$

3. Classification: Let $\hat{C}_b(x)$ be the class predicted by bth Random Forest tree. Then apply the majority voting technique for the final class predicted.

The main idea in Random Forest is to increase the variance reduction in bagging by reducing the relationship between trees. This is accomplished by randomly choosing the input variables in the tree development process [13].
The last model used for the application of the BMA, is not part of the family of trees, but comes from the class of generalised linear models: Logistic Regression. The main reason for the development of this model stems from the fact that it is necessary to define the a posterior likelihoods of K groups, by a linear function in x and have values between [0,1]. Logistic Regression is defined as follows:

$$\begin{aligned} \log \tfrac{\Pr(G=1|X=x)}{\Pr(G=K|X=x)} &= \beta_{10} + \beta_1^T x \\ \log \tfrac{\Pr(G=2|X=x)}{\Pr(G=K|X=x)} &= \beta_{20} + \beta_2^T x \\ &\vdots \\ \log \tfrac{\Pr(G=K-1|X=x)}{\Pr(G=K|X=x)} &= \beta_{(K-1)0} + \beta_{K-1}^T x \end{aligned} \tag{2.15}$$

The last class of the group is used as denominator. From 2.15 is clear that this model is based on the $log(odds)$ between the target classes [13]. The use of the logarithm is important because, projected onto a Cartesian plane, the function must be continuous. Therefore the use of the logarithm allows such a projection in a space between $-\infty$ and $+\infty$. The conditional probability $\Pr(G = K \mid X = x)$ is defined as sigmoid function, and

it can be expressed as follows:

$$\begin{aligned}\Pr(G=k \mid X=x) &= \frac{\exp\left(\beta_{k0}+\beta_k^T x\right)}{1+\sum_{\ell=1}^{K-1}\exp\left(\beta_{\ell 0}+\beta_\ell^T x\right)}, k=1,\ldots,K-1,\\ \Pr(G=K \mid X=x) &= \frac{1}{1+\sum_{\ell=1}^{K-1}\exp\left(\beta_{\ell 0}+\beta_\ell^T x\right)}\end{aligned} \tag{2.16}$$

To abbreviate the notation of conditional probability, $\Pr(G=K \mid X=x) = p_k(x;\theta)$. For sarcasm and irony detection, K is set to 2. This implies that within the logistic regression a single linear function is applied. The parameter β is estimated using the maximum likelihood of $P(G|X)$. These probabilities are based on the multinomial distribution. Indeed, the log-likelihood for N instances is the following:

$$\ell(\theta) = \sum_{i=1}^{N} \log p_{g_i}\left(x_i;\theta\right) \tag{2.17}$$

more specifically, for a binary classification $y_i = [0,1]$, the log-likelihood is calculated as follows:

$$\begin{aligned}\ell(\beta) &= \sum_{i=1}^{N}\left\{y_i \log p\left(x_i;\beta\right) + \left(1-y_i\right)\log\left(1-p\left(x_i;\beta\right)\right)\right\}\\ &= \sum_{i=1}^{N}\left\{y_i\beta^T x_i - \log\left(1+e^{\beta^T x_i}\right)\right\}\end{aligned} \tag{2.18}$$

where the aim is to maximize the log-likelihood function, indeed the derivates are set equal to zero. In the mathematical form:

$$\frac{\partial \ell(\beta)}{\partial \beta} = \sum_{i=1}^{N} x_i\left(y_i - p\left(x_i;\beta\right)\right) = 0 \tag{2.19}$$

The optimization process is based on the Newton-Raphson method, where the second derivate is estimated:

$$\frac{\partial^2 \ell(\beta)}{\partial\beta\partial\beta^T} = -\sum_{i=1}^{N} x_i x_i^T p\left(x_i;\beta\right)\left(1-p\left(x_i;\beta\right)\right) \tag{2.20}$$

Indeed, the new parameter β is updated in this way:

$$\beta^{\text{new}} = \beta^{\text{old}} - \left(\frac{\partial^2 \ell(\beta)}{\partial\beta\partial\beta^T}\right)^{-1}\frac{\partial \ell(\beta)}{\partial\beta} \tag{2.21}$$

In this research, a variant of logistic regression is used, which is based on the addition of a regularisation term l1 or l2, which allows a kind of features selection. The least important coefficients tend to zero, and if l1 is applied, the variables with modest influence are eliminated. This technique is defined as coefficient shrinkage. For example, if the l1 (or Lasso) is applied, the function takes the following form [13]:

$$\max_{\beta_0,\beta} \left\{ \sum_{i=1}^{N} \left[y_i \left(\beta_0 + \beta^T x_i \right) - \log \left(1 + e^{\beta_0 + \beta^T x_i} \right) \right] - \lambda \sum_{j=1}^{p} |\beta_j| \right\} \tag{2.22}$$

for the ridge regularization or l2, the added term is elevated to the quadratic form:

$$\lambda \sum_{j=1}^{p} \beta_j^2 \tag{2.23}$$

All models presented in this section are optimised on the basis of two types of procedures: Random Search and Bayesian Optimization. In section 2.2.4 we go into the details of these methodologies.

2.2.4 Random Search and Bayesian Search Optimization

When dealing with a task that has to be solved with Machine Learning or Deep Learning techniques the main steps are: learning approaches, model selection (or designing the architecture of a neural network) and hyper-parameter optimisation. The latter requires the use of techniques called Grid Search or Random Search (the most common ones) to find the best combination of hyper-parameters that maximise or minimise an objective function, e.g. accuracy or loss function. These approaches imply a great waste of computational resources and time. Moreover, they do not guarantee the identification of the absolute minimum/maximum point. The set of hyper-parameters to be searched is called the search space within a number of iterations called budget. In other words, the budget is the number of experiments fixed during the optimization process. The main difference between Random Search and Grid Search is that Grid Search tries all hyper-parameters within the search space, whereas Random Search chooses randomly from a uniform distribution in the search space. The sampling method of these two techniques is not sample efficient, because given a budget constraint, one cannot test all possible configurations of

the hyper-parameters of a model. An optimisation process is defined as sample efficient when a sequential, model-based optimisation method is used. In this case, it is called sequential model based optimization: this methodology is the main core of Bayesian Search. Suppose a number of pairs of hyper-parameters are tested, where various results in terms of accuracy are recorded. Then a table is obtained, composed by a certain value for a hyper-parameter that corresponds to a certain level of accuracy. This constitutes a real data set: therefore the idea is to develop a Machine Learning model, called surrogate model, which is able to estimate the accuracy of another model on the basis of the tested hyper-parameters. There are two types of approaches: a deterministic one (for example the accuracy values) and a probabilistic one, where a confidence level for the prediction, hence accuracy, is provided, e.g. estimates obtained in k-fold cross validation [16]. For this thesis, a probabilistic surrogate model is used: the Gaussian Process (GP).

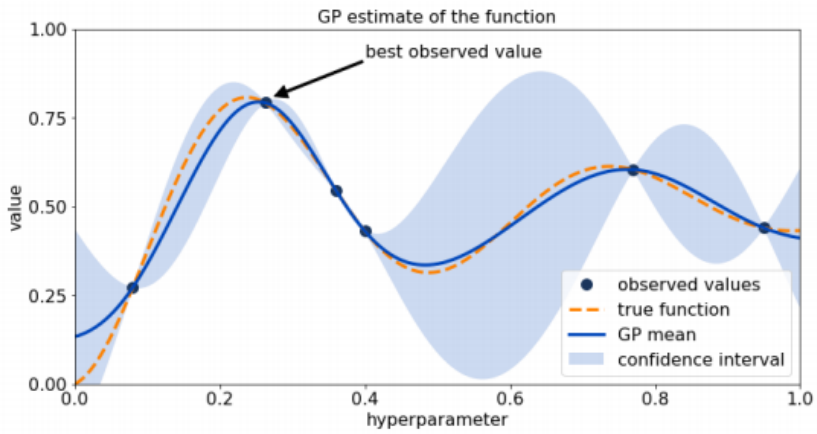

Figure 2.8: Gaussian Process example, source [16]

Given the set of hyper-parameters, a function $f(x)$ is estimated. In this case 2.8, the loss function, such as error rate, is calculated. The idea is to find the x^* hyper-parameters that $x^* = argmin_{x \in X} f(x)$. The most important thing to highlight from the figure 2.8 is that, the information obtained from the standard deviation estimation is exploited in order to find the optimum minimum of the error function. In details, when the surrogate model is a GP, it is defined as Bayesian optimisation. The advantage of using this GP is that it also provides confidence bands on the model estimate. One of the simplest mechanisms to select the next point is called a Lower Confidence Bound (in the case of (in the minimisation case as in 2.8). Otherwise, for maximization it is called the Upper Confidence Bound. The concept is to be optimistic about the variance: in other

words, in the case of minimisation, it is assumed that the curve, to be minimised, is not the one defined by the mean, but the one defined by the difference between mean and variance. The most complicated aspect is to understand the covariance function. There are two types of information when the GP is trained: the x^* configurations tested and y which corresponds to the estimated objective function on those configurations. What is expected is that the covariance in the values in y can be remapped into the covariance of the values in x. Particularly, if two configurations are very close in x, i.e. if the hyper-parameters of the two configurations are very close, and thus if there is a high covariance between the x, then the covariance of the respective y is assumed to be high. As we attempt to map the covariance between two points in y through their proximity in x, we make assumptions about the form of the objective function (e.g. accuracy or error). The covariance function can also be expressed with kernels, so covariance function and kernel are often used synonymously. The most used kernel function in the GP are [17]:

$$\text{Gaussian:} \quad = k\left(\mathbf{x}, \mathbf{x}'\right) = k(\tau) = \sigma_f^2 \exp\left(-\frac{\tau^2}{2l^2}\right)$$

$$\text{Matern } 3/2: \quad k\left(\mathbf{x}, \mathbf{x}'\right) = k(\tau) = \sigma_f^2 \left(1 + \frac{\sqrt{3}\tau}{l}\right) \exp\left(-\frac{\sqrt{3}\tau}{l}\right)$$

$$\text{Matern } 5/2: \quad k\left(\mathbf{x}, \mathbf{x}'\right) = k(\tau) = \sigma_f^2 \left(1 + \frac{\sqrt{5}\tau}{l} + \frac{5\tau^2}{3l^2}\right) \exp\left(-\frac{\sqrt{5}\tau}{l}\right)$$

$$\text{Exponential:} \quad k\left(\mathbf{x}, \mathbf{x}'\right) = k(\tau) = \sigma_f^2 \exp\left(-\frac{\tau}{\tau}\right)$$

Specifically, the kernel defines the smoothness of the estimated function. A few algorithms have only continuous hyper-parameters, such as GP. This gives rise to the need to introduce alternative methods for treating discrete and categorical variables as continuous. The implementation used for Bayesian Optimization of Scikit-Optimize exploits the following concept: the discrete variables are treated as continuous and if the GP identifies that there are 1.6 hidden layers (for example), in the case of the neural networks, then it approximates to 2. For the categorical variables one-hot encoding is carried out. Each of these one-hot variables are associated with a value of the loss function, so at the end the one with the best objective function value is identified. A very important aspect to consider is the acquisition function used within the surrogate model. This function

depends on what kind of strategy one wants to follow: exploration or exploitation. In figure 2.8 the acquisition function is named Lower Confidence Bound (LCB). Within the Scikit-Optimize algorithm, an hybrid acquisition function called "hedge" is used. It uses three different acquisition functions: Expected Negative Improvement, Lower Confidence Bound and Probability Negative Improvement. At each iteration of cross validation, all three functions mentioned above are estimated simultaneously. For each of them, the gain ϕ is estimated independently. All identified points are re-weighted according to the gain and a softmax function is applied to identify the best X^*. Using such functions means minimising the loss function, and makes it possible to combine the two principles of exploration and exploitation.

2.3 Introduction to Deep Learning and Transformer Architectures

The aim of this section is to provide an introduction to the Deep Neural Networks (DNN) and an in-depth explanation of new architectures used in the NLP field, such as self attention layers and Transformers. The foundations of Deep Learning are laid in 1943 by Warren McCulloch and Walter Pitts. The two researchers develops a system of algorithms to reproduce the way the human brain process works. In those years, the bottleneck lay in computational power. During that years one of the problems is how to optimise the weights to be estimated from the models efficiently. In 1960, Henry J. Kelley introduces the Back Propagation system. In 1962, this algorithm, is further improved by Stuart Dreyfus, developing a more efficient version based on a derivation rule called chain rule. Only in the 70's and 80's, more attention is paid to this topic: after several developments thanks to researchers such as Kunihiko Fukushima, with the Neocognitron model and Yann LeCun with an algorithm based on convolution operations to recognize handwritten numbers, neural networks starts to be important for the world of artificial intelligence. Only in the 2000s, however, the development of neural networks starts to become less computationally expensive, thanks to the introduction of hardware components for model training called GPUs [18].

2.3.1 Deep Neural Networks

The basis of Deep Neural Networks resides in the concept of Multi-layer perceptrons (MLPs). Consider an MLPs model with an intermediate layer:

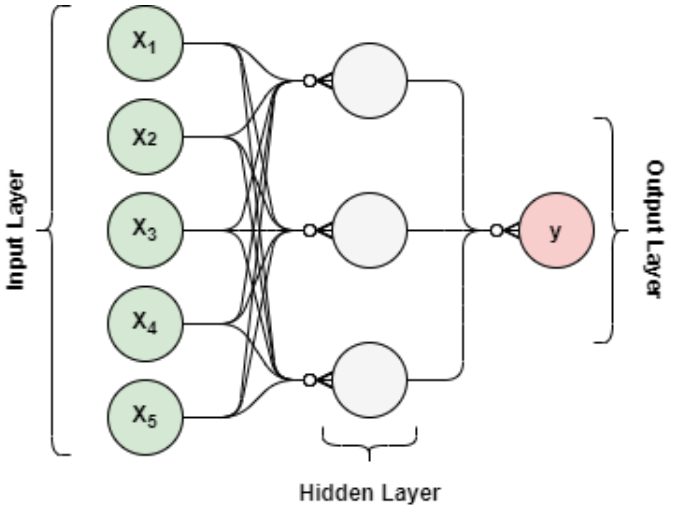

Figure 2.9: Multi-layer Perceptrons example

In figure 2.9 there is an input layer, a hidden layer and an output layer. The former consists of the input data and on the edges the corresponding weights. Within the hidden layer lies an activation function. The idea is to mapping the input features vector x to a specific output y. This mapping function is defined as:

$$y = f(x, \theta) \tag{2.24}$$

in 2.24, *theta* identifies the model's weights, called parameters, that the neural networks learns during the training phase, based on the estimation of the gradients and the minimisation of loss function. The training process is divided in two different phases: forward phase, where the input X is fed to the network, where it flows from the hidden layer to the output layer. During the feedforward phase two activation functions are applied (referring to 2.9):

$$y = f^2(f^1(x)) \tag{2.25}$$

where the f^1 function is applied to the hidden layer, and f^2 is calculated from the output layer. Once, $\hat{y}$ is estimated, starts the backward propagation phase, where θ weights are updated. The update is done through the gradient descent method where, exploiting the

chain-rule principle, the weights are updated. Partial derivatives are estimated based on the loss function calculated at the i-th iteration. The chain-rule formula is as follows:

$$f(x) = f(g(x)) \tag{2.26}$$

$$\frac{df}{dx} = \frac{df}{dg}\frac{dg}{dx} \tag{2.27}$$

While for updating the weights and the bias terms, the following function is applied:

$$\frac{dJ}{dw} = \frac{dJ}{d\phi}\frac{d\phi}{dx}\frac{dx}{dw} \tag{2.28}$$

$$\frac{dJ}{db} = \frac{dJ}{d\phi}\frac{d\phi}{dx}\frac{dx}{db} \tag{2.29}$$

where J is the cost function, ϕ is the activation function applied, x is the net input vector and b is the bias terms. As a result of the use of different activation functions and the inclusion of different layers within a model, neural networks are defined as universal function approximators. With regard to the cost function J, two different formulations are used depending on the activation functions applied in the output layer of each model. The two activation functions, used for the classification task, are as follows:

$$\textbf{Softmax: } \phi(\mathbf{x})_j = \frac{e^x}{\sum_{k=1}^{K} e^{x_k}} \text{ for } j = 1, \ldots, K \tag{2.30}$$

$$\textbf{Sigmoid: } \phi(x) = \frac{1}{1 + e^{-x}} \tag{2.31}$$

Their corresponding loss functions J are:

$$\textbf{Cross Entropy: } -\sum_{i}^{M} y_i \log\left(\hat{y}_i\right) \tag{2.32}$$

$$\textbf{Binary Cross Entropy: } -\left(y \log(\hat{y}) + (1-y)\log(1-\hat{y})\right) \tag{2.33}$$

where $\hat{y}$ is the predicted value and y is the ground truth. The equation 2.32 is associated to

the activation function 2.30 while 2.33 is linked to the function 2.31. The most commonly used activation functions in the intermediate layers are as described below:

- **Tanh:** $\tanh(x)$
- **ReLU:** $\max(0, x)$
- **Leaky ReLU:** $\max(0.1x, x)$

The training phase is based on the concept of optimisation: the objective of the training algorithm is to minimise the loss function. This phase is very critical since it is important to define the right regularization terms to avoid the problem of overfitting and underfitting. So it is important to find the optimum point between the loss and a regularization factor.

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}}(\overbrace{\frac{1}{n}\sum_{i=1}^{n} L\left(f\left(x_i;\theta\right), y_i\right)}^{\text{loss}} + \overbrace{\lambda R(\theta)}^{\text{regularization}}) \tag{2.34}$$

where $R(\theta)$ corresponds to the scalar that reflects complexity and it is therefore kept low for efficiency issues. λ is a hyper-parameter that controls the balance between bias and variance. This step is closely related to which optimiser is used to minimise the 2.34 function, such as [19]:

- **Momentum Gradient Descent:** $v_t = \gamma v_{t-1} + \eta \nabla J(\theta; x, y)$, where $\theta = \theta - v_t$
- **Adagrad:** $\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \varepsilon}} \cdot g_t$ where g_t is the sum of squares of the t-past gradients.
- **RMSProp:** $\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{(1-\gamma)g^2 t - 1 + \gamma g_t + \varepsilon}} \cdot g_t$, where g_t is the moving average of the squared gradients
- **Adam:** $\theta_{t+1} = \theta_t - \frac{\eta \hat{m}_t}{\sqrt{\hat{v}_t + \varepsilon}}$ where $\hat{m}_t$ and $\hat{v}_t$ are the first and second moment corrected with respect to the bias initialization.

For the sake of clarity and thoroughness, consider [20]. Within this chapter some topics are emphasised as they are applied in the thesis: convolutional layers, recurrent layers, self attention layers and Transformer.

2.3.2 Convolutional & Recurrent Layers

Convolutional layers arise with the application of Deep Learning to image-related tasks. Each input image is described through a tensor, that can be multi-channel, with the associated pixel values. Instead of mapping a single pixel at a time, the image is examined in macro-areas, using specific filters that allow us to extract more significant features from the image, removing possible disturbing factors. Also in the NLP field, convolutions have become a valid application, thanks to the concept of locality that they exploit by applying convolutional operations to a matrix of word features, called embeddings. This operation can be described as follows:

$$\int_{-\infty}^{+\infty} f(x-t)g(t)dt \tag{2.35}$$

where in the case of a 2-D convolutions f and g are the corresponding matrix and the kernels considered. The models analysed in this study mainly exploit 1-dimensional convolutions. This means that the kernel slides in only one direction.

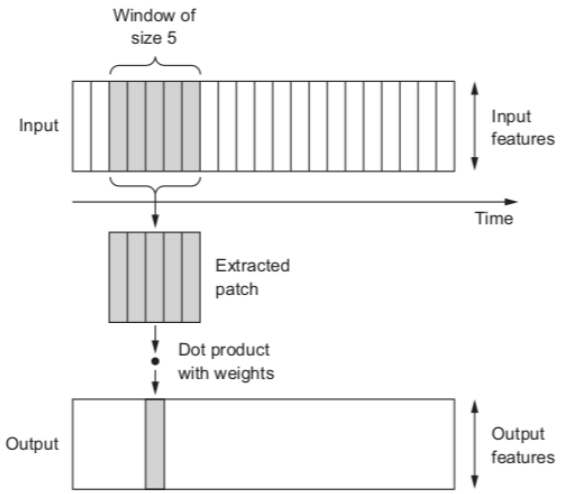

Figure 2.10: Conv1D operation example, source Francois Chollet

In 2.10 is described how 1D convolution works. This concept is applied to each embedding sequence of each tweet. This highlights the concept of locality and the consideration of temporal dependencies between embeddings. Indeed, it is important to underline the importance of the sequential dependence between words which, taken together, define a sentence. The result obtained from the convolution is a vector called feature map, that

has a dimension of:

$$\frac{DIMENSION\_INPUT - DIMENSION\_FILTER + 2 * PADDING + 1}{DIMENSION\_STRIDE} \quad (2.36)$$

where the term padding allows the original size of the input to be preserved. While, the term stride is a metric that regulates the movement of the convolutional filter above the input.

A bottleneck within the feature map is the fact that the estimated values are very dependent on the individual input values. This causes small variations to create a different feature map. One strategy to solve this limitation is downsampling: one way is to apply convolution with a larger stride, in order to summarize more generic information. A more robust technique is to use the pooling layer. The pooling layer operates independently on each feature map to generate a new collection of feature maps. There are several types of pooling operations to take into account, such as average pooling and max pooling.

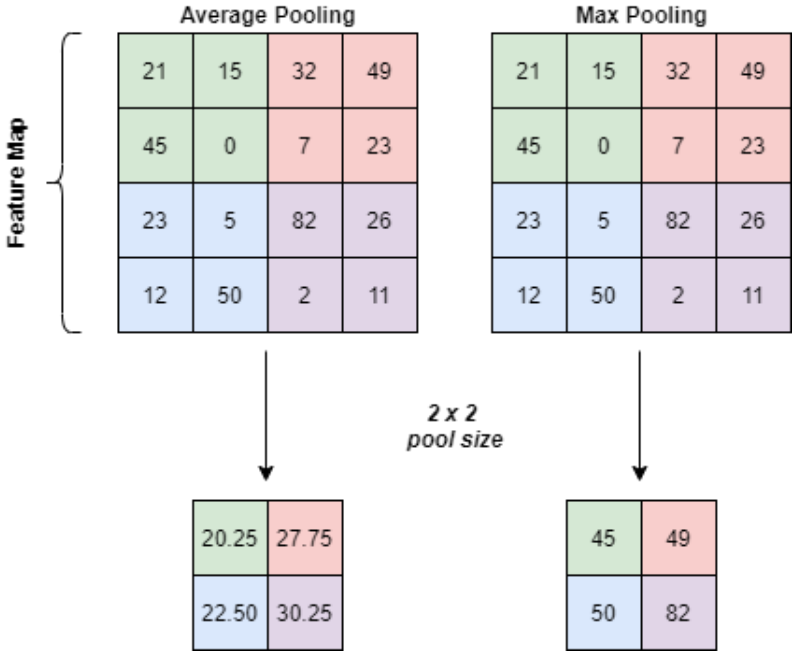

Figure 2.11: Max and Averaging Pooling example

The figure 2.11 shows two pooling operations on the same starting feature map.

More recent techniques exploit layers of convolution and recursion stacked to obtain more information from input data that have a time dependency. Recursive layers are used when there is a time dependency within the input vector or matrix. Consider, time series or, as

mentioned above, the words that make up a sentence. The novelty that the RNNs present, is the possibility of passing the output of the hidden layer not only to the final layer, but also to the hidden layer of the subsequent execution of the network. For example: suppose a Machine Translation problem has to be faced, the input of the network is a sentence in the language j and the output is a sentence in the language q. X_1 is associated to the first word, X_2 to the second and so on. So thanks to the recurrent link the output of the second network is influenced by the first word. The hidden layers are connected by another matrix of weights which defines the relationships between the words. This is propagated to the end of the sentence. The connection between these nodes makes it possible to define a time course between the various words and thus define a dependency between them. At each time step the same weight W is always used, so in this case we would have different $h_0, ..., h_t$ but always with the same weight. In backpropagation the gradient is calculated on each h. Once the gradient for each state is estimated, all values are summed to get the gradient for the weight w. This backpropagation procedure is defined as backpropagation through time (BTT). The figure 2.12 is an example of a Recurrent Layer works:

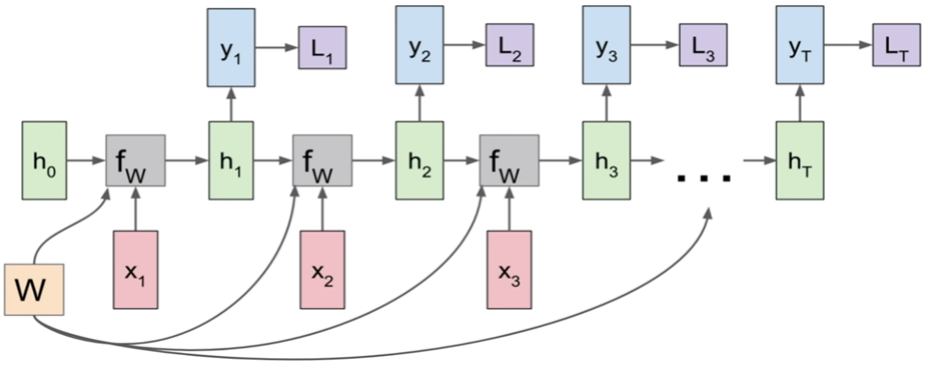

Figure 2.12: Recurrent Model structure Many to Many

A known problem, related to this type of architecture, is the vanishing gradient. Mainly it arises because, in the case of text classification for example, one has the necessity to construct deep neural networks. This implies the vanishing gradient because repeated multiplications carry out for values (phase of backpropagation) that tend to zero and therefore also the gradient disappears at zero. In the case of a Recurrent Neural Networks (RNN), it is sufficient simply to have only one hidden layer. To overcome this bottleneck, two different, more sophisticated recurrent layers are introduced: Long Short

Memory Term (LSTM) and Gated Recurrent Unit (GRU).

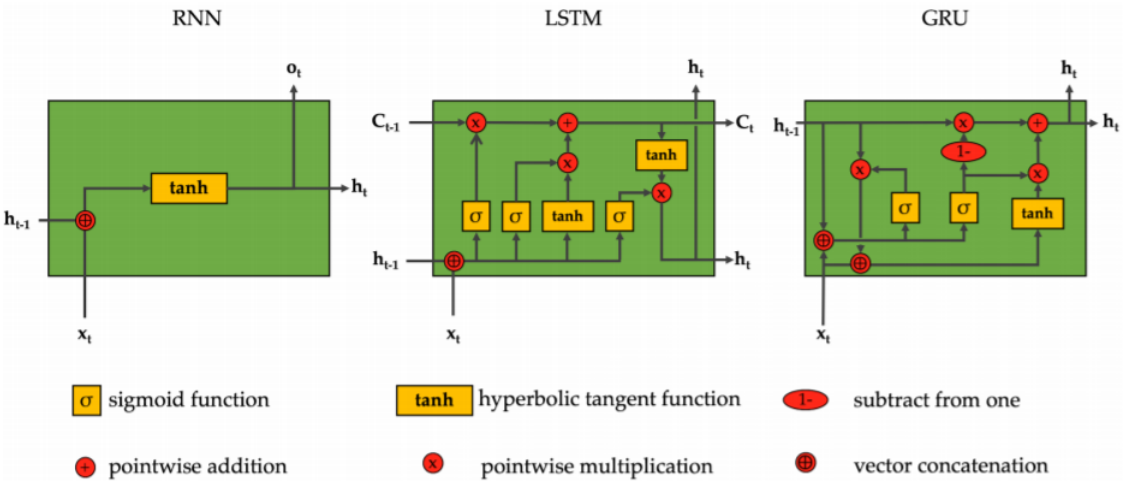

Figure 2.13: Differences between RNN, LSTM and GRU nodes

The central point of this type of architecture is the horizontal line connecting C_{t-1} to C_t. This is called the cell state (memory cell): it is the memory of the network that carries information from one execution to another, using linear functions that allow information to be added or removed to the cell state. The elements that control whether or not to modify information, within the cell state, are called Gates. Gates refer to the image 2.13 are the sigmoid functions. The first step determines whether or not the information in the cell state is to be changed. To do this, the first gate is used: Forget Gate,

$$f = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right) \tag{2.37}$$

It returns a value between 0 or 1 (typically it returns a value either very close to zero or very close to 1). For example, the cell state could carry information related to the subject of the sentence such as gender. The moment a new subject is encountered, a decision must be made as to whether or not to remove that information from the cell state in order to add the new information concerning the subject of the sentence. So the Forget Gate decides this. Next, the input gate:

$$i = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right) \tag{2.38}$$

It decides which information is to be replaced on the basis of a sigmoid function. The next step is to choose which information is to be added to the memory cell, the new candidate:

$$\widetilde{C}_t = \tanh\left(W_C \cdot [h_{t-1}, x_t] + b_C\right) \tag{2.39}$$

The new candidate allows to update the cell state:

$$C_t = f * C_{t-1} + i * \widetilde{C}_t \tag{2.40}$$

The last step is to calculate the actual output H_t and any Y_t. This is done by applying the output gate:

$$o = \sigma\left(W_0 \cdot [h_{t-1}, x_t] + b_0\right) \tag{2.41}$$

it defines which part of the cell state passes through the output h_t. o is multiplied by the cell state: this operation is called element wise multiplication.

$$\begin{aligned} h_t &= o * \tanh\left(C_t\right) \\ y_t &= \phi\left(h_t\right) \end{aligned} \tag{2.42}$$

Each value of the output gate is multiplied by each value of the cell state. Finally, the function of the last layer is applied to h_t. This type of architecture limits the problem of the disappearance of the gradient: through the cell state the most important information is stored. The GRU node is the simplified version of the LSTM node:

$$\begin{aligned} r &= \sigma\left(W_r \cdot [h_{t-1}, x_t] + b_r\right) \\ u &= \sigma\left(W_u \cdot [h_{t-1}, x_t] + b_u\right) \\ \tilde{h}_t &= \tanh\left(W_h \cdot [r * h_{t-1}, x_t] + b_a\right) \\ h_t &= (1-u) * h_{t-1} + u * \tilde{h}_t \\ y_t &= \phi\left(h_t\right) \end{aligned} \tag{2.43}$$

The novelty introduced by the GRU architecture is the relevance gate r. The difference is that GRU is less time consuming than LSTM. But, the main advantage of LSTM node is that usually leads to better results and remembers longer sequences. For more information refer to [20]. A limitation of these unidirectional structures is that they do not take into account relations that are far away: X_t receives residual information from X_1, but not vice versa. For this reason, bi-directional recurrent networks are implemented. In other words, a flow of information is created which moves both from left to right and right to

left: this methodology is based on an acyclic graph. In 2.14 is shown a typical structure of a Bidirectional LSTM:

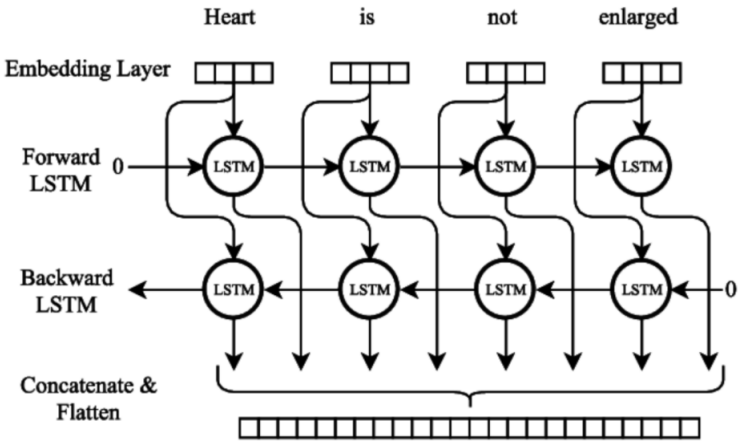

Figure 2.14: Bidirectional LSTM example

This type of architecture is used both in state-of-the-art models and in the new methodologies proposed for irony and sarcasm detection.

2.3.3 Transfer Learning

Transfer Learning is a method of Machine Learning in which a model developed for one task is reused as a starting point for a model that needs to be trained on a specific task. It is a popular approach in Deep Learning where pre-trained models are used as a starting point for solving computer vision and NLP tasks. Using pre-trained models on large data sets implies a large saving in computational resources. Instead of training a model from scratch, optimized weights are reused for a specific task. For example, exploiting models like Transformers trained on generic tasks, to solve binary classification problems such as irony and sarcasm. Transfer Learning can be defined as follows: a domain D which represents the features space X, the marginal distribution $P(X)$, and a task $T = \{Y, f(X)\}$, which is composed by a ground truth vector Y and a mapping function $f : X \rightarrow Y$. So, the main objective is to improve the mapping function $f_t(.)$ in task T_t by applying the information gained from another domain D_s and task T_s [21]. Transfer Learning is very useful when one does not have too many computing resources and the amount of data is relatively low. In the field of NLP, it is important to generate models that are capable of analysing the relationships between all words. The development

of accurate models implies having a lot of data available. In the case of the classification of irony and sarcasm, the availability of training data is very limited. A good strategy is to use the latent representation of embeddings, coming from models trained on other tasks, such as Word2Vec. Using such features implies having already very meaningful word representations. Obviously, these embeddings are not contextualised, and this implies limitations. As shown in section 2.3.4, extrapolating contextualised embeddings is possible using Transformers.

2.3.4 Self Attention Layer and Transformer Architectures

In order to understand how Transformers work, it is necessary, first of all, to analyse the so-called "Attention Mechanism". This mechanism allows Transformers to have all the necessary information about the dependencies of very long sequences. Compared to recurrent networks, this approach allows parallelisation of processes and has an infinite reference window. Indeed, it is able to use the whole context, especially when it comes to text generation, machine translation etc. To understand how it works, a practical example is used: suppose to have the phrase "I love NLP course": let's decompose it in tokens T where each one is associated with an embedding vector V. Once the embeddings matrix is obtained, it is inserted inside a layer called attention where its output is a contextualised vector.

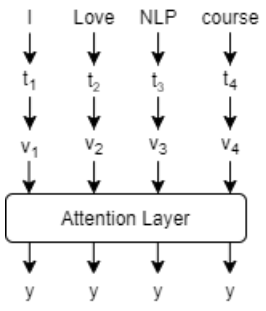

Figure 2.15: Input Attention example

The Attention Layer in 2.15 works as follows: starting with the four vectors representing each word, and assuming that the context information is added to the V_3 vector.

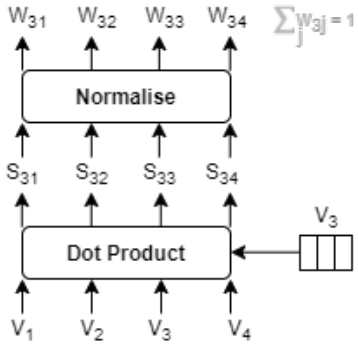

Figure 2.16: Attention Layer in details, part one

S_{3i} represents scores relative to the vector V_3 with respect to all the others. The dot product can be seen as a sort of calculation of similarity between the various vectors. As the values increase, the similarity between them increases. Obviously this output S can assume any value, for this reason the normalization of each vector S_{3i} is applied. What it is obtained are real weights for each word relative to the word V_3 taken into consideration. The sum of these weights must gives 1, usually the softmax function is applied. From these weights the "residual" block is added, i.e. the starting vectors are taken into account and multiplied by each other, to have a much more contextualised word embedding:

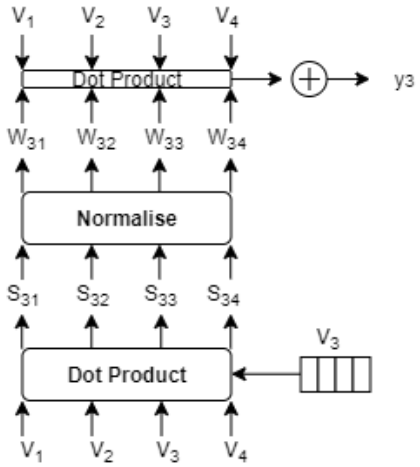

Figure 2.17: Attention Layer in details, part two

Y_3 is a contextualised vector of size 1 x 4 in this case. The whole process is done for all the tokens that are available. Obviously, in order to simplify the explanation, in this representation there are no weights to optimise. Adding weights makes the system more flexible. The weights are inserted 3 times, each time the V-vectors are used. From here they use the definitions of keys, queries, values. Given queries and keys the values are

obtained (the basis of information retrieval). Starting from a matrix M_k, remembering that V_i is of dimension $1 \times k$ and M_k the matrix $k \times k$, the result between product and matrix is a vector of dimension $1 \times k$. So, the operations remain the same even if a matrix of weights is inserted. Since, as said before, the associated weight matrices are: M_k for keys, M_q for queries and M_v for values:

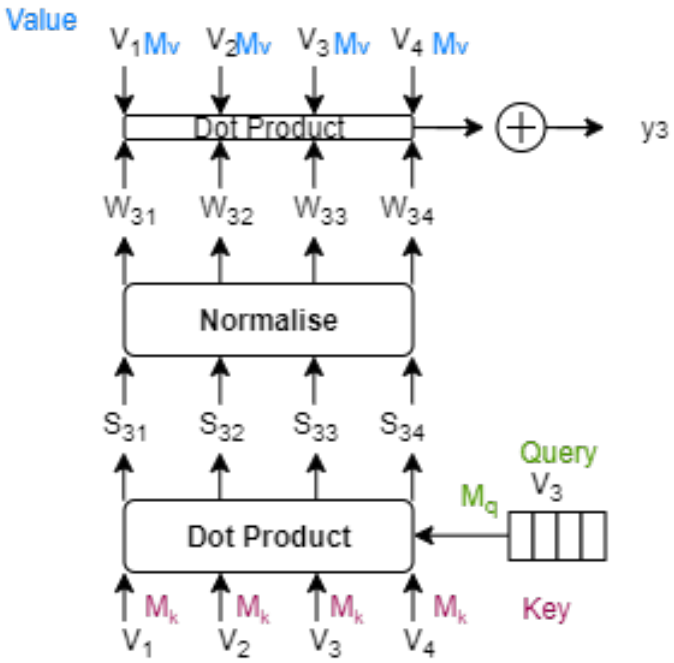

Figure 2.18: Attention Layer in details, part three

Now, that the weights to be optimised are introduced, one can represent this system as a neural network. Once the mechanism behind the Self Attention Layer is understood, Multi Head Attention is introduced. Using several attention heads at the same time allow us to analyse the relationships between words from different points of view, without having to completely saturate one of them. Each attention layer works in parallel and does not share weights with the others. In addition to adding further linear layers, without the bias term, other operations are applied:

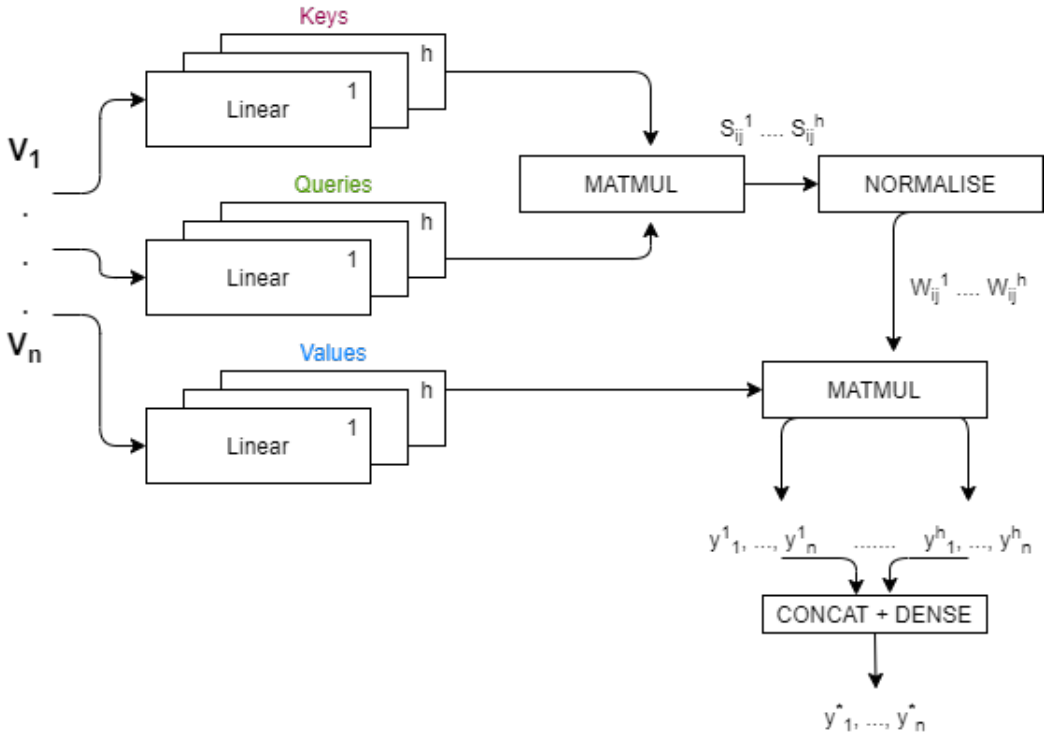

Figure 2.19: Multi Head Attention

As can be seen from the figure 2.19, compared to a simple attention layer, two operations are added to merge everything into a single output: concatenation and the addition of a dense layer create the final output of y^*. The application of Multi Head Attention layers is a tool that underlies the architecture of Transformers-based models. The importance of the development of the Transformer model lies in the fact that it makes it possible to capture the temporal relationships of sequences, without having to resort to recurrent networks. This has considerable advantages in terms of parallel computation: unlike RNNs, using multi head attention, parallel computations can be performed. In addition, this new approach can take into account longer temporal relationships. In RNN architectures, each element of the sequence must be processed sequentially. For example, given an input vector $T = [t_1, t_2, t_3, t_4]$, the recursive layers to estimate the relationship between t_1 and t_4, first compute the $n-1$ elements and then propagate the t_4 information. For self attention layers this condition does not arise. This has led to great advantages in terms of wasting computational resources. The first Transformer model is constituted by 6 stacked encoder-decoder structure [22]: The main objective of the encoder-decoder architecture is to map the input T into a continuous representation Z through the encoder. While the decoder allows to develop a combination Y in output, which corresponds to

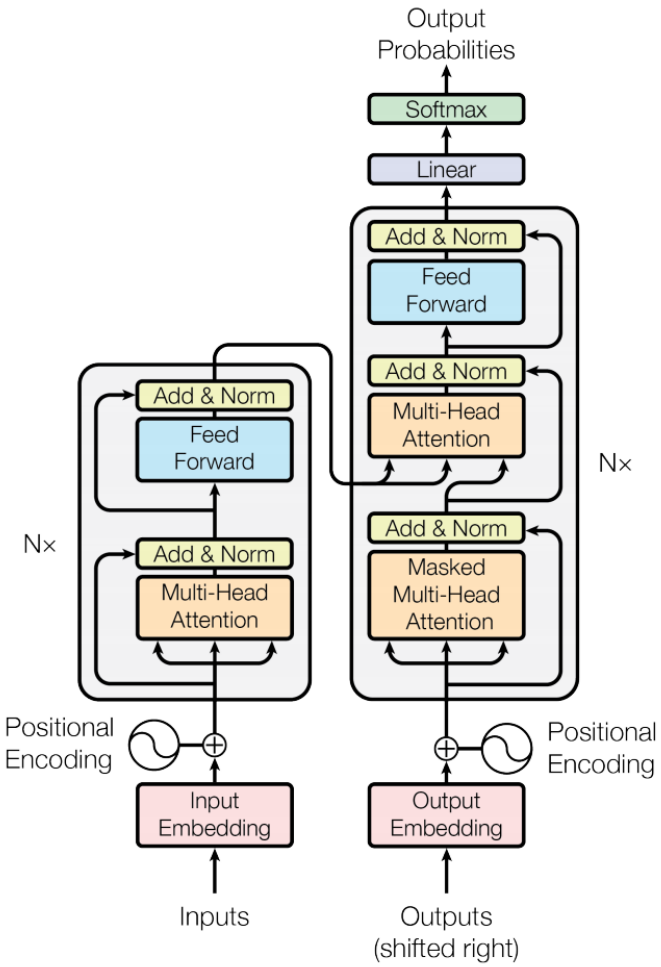

Figure 2.20: The Transformer architecture. On the left encoder, on the right decoder, source [22]

the decoding of the input sequence with respect to a certain target. For example, translating a sequence of English words into another language such as Spanish. Within the Transformers, a variant of the self attention layer is introduced and as a result of this, the multi head attention layer is also slightly modified:

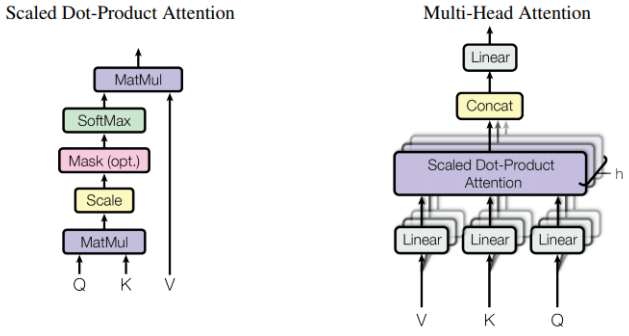

Figure 2.21: Scale Dot-Product Attention, source [22]

As can be seen in 2.21, some operations remain unchanged, except for a few elements: scaling, which allows greater stability for the calculation of the gradient: $scale = \frac{1}{\sqrt{d}}$, where d is the word-embeddings dimension. In addition, the masking operation that is optional. Consider the encoder part: this block can be stacked with other encoders, as can the decoder. Indeed, in the figure 2.20 $N\times$ means N blocks that can be added. The number of blocks, and the number of self attention represent hyper-parameters to be optimised. Analysing the structure in figure 2.20, we can see that the original input is reconnected to the output of the multi head attention layer. As also in the final part of feed forward there is the residual that is added to the output. These residuals are added for the same principle of the ResNet: at a numerical level there is a high risk relative to the vanishing gradient. Suppose of having N blocks, where N is a relatively large number. In the backward phase it leads to the vanishing of the gradient. So these conjunctions are added to cope with this problem. Since multi head attention does not take into account the order, but actually the order is relevant, positional encoding has been added. This is a kind of padding that forces the encoder to take more account of the position of each token. The same applies to the decoder block. For further clarification, please refer to [22].

In this study, two main Transformer-based model are taken into account: Pre-training of Deep Bidirectional Transformers (BERT) and Robustly Optimized Bert (Roberta). BERT is developed for learning words representation using unlabeled text applied to Transformer architectures. A single additional output layer may be used to finalise the pre-trained BERT model for a number of tasks, such as Q&A and Linguistic Inference, without significant task-specific improvements. The main difference with respect to Transformer architectures is based on the 'bidirectional' concept: in relation to the left-right language model, the masked language model goal, as BERT, enhances the relation between left and rigth context. At each iteration, one word of each input sentence is randomly masked, with the aim of predicting the original covered word. This brings out the relationship of the masked word with the context word, both to its right and left. BERT is divided in two phases: pre-training weights from unlabeled text applying unsupervised learning and a second step called fine-tuning for a supervised task, or downstream of the model's weights. An important element is the fact that the pre-trained architecture and the downstream architecture have minimalised variations. There are two different type of BERT model

based on different dimensions: $BERT_{BASE}$, with L 12 blocks, the dimension of each hidden size H is 768, 12 A attention heads, with a total parameters' number equal to 110 million. While, $BERT_{LARGE}$ is composed by L=24, H=1024, A=16 hyper-parameters, for a total of 340 million of parameters. Both model are pre-trained on the BooksCorpus composed by 800 million words and the all English Wikipedia website with a total of 2.500 million words. The BERT input words representation is as follows:

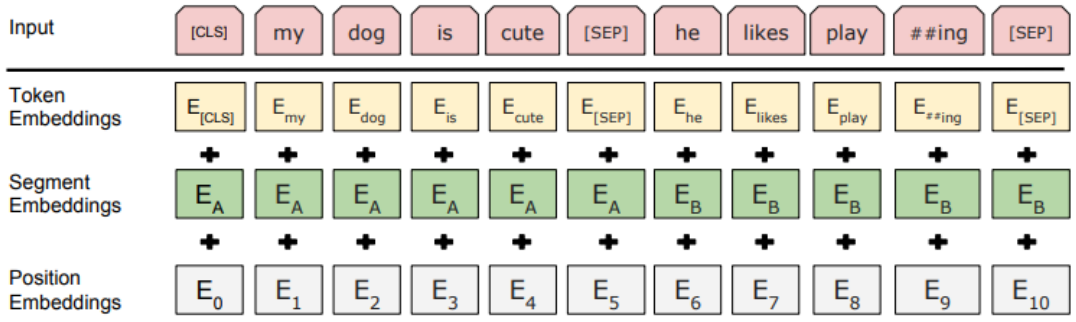

Figure 2.22: BERT input representation, source [23]

where the position embeddings are used to take into account the word position, since the attention layers use each word independently. The input representation of each token is the result of the sum between token embeddings, segment embeddings and position embeddings. In 2.22, CLS and SEP are special tokens used for aggregating each sentence. For the sake of clarity, refer to [23] paper.

From [24], the researchers highlight that BERT is under-trained, due to different factors. For this reason, Roberta model is developed: it is based on an almost identical architecture to BERT, with modifications to overcome the limitations that it suffers from. Roberta applies dynamic masking pattern instead of using static pattern, the training set is more or less 10 times bigger, 16 Gb for BERT and 161 Gb for Roberta. This model is trained on longer sequences, so it can take as input longer sentence and it exploits different training objective. These improvements lead Roberta to perform better, depending on the task, by $2 - 20\%$.

2.3.5 Using millions of emoji occurrences to learn any-domain representations for detecting sentiment, emotion and sarcasm

After analysing the application of Machine Learning with Bayesian Model Averaging, the second research object of the analysis applies emoji occurrences on a Deep Learning model. This study aims to generate any latent representation in the domain of sentiment, emotion and sarcasm detection [25]. The DeepMoji model is trained on 1246 million tweets dataset, where the task aims to predict 64 popular emoji by obtaining state-of-the-art performance in several benchmark dataset for sentiment, sarcasm and emotion detection using only a pretrained classifier. The baseline is based on the association between words and emoji in order to generate an emotional embedding for each input sequence:

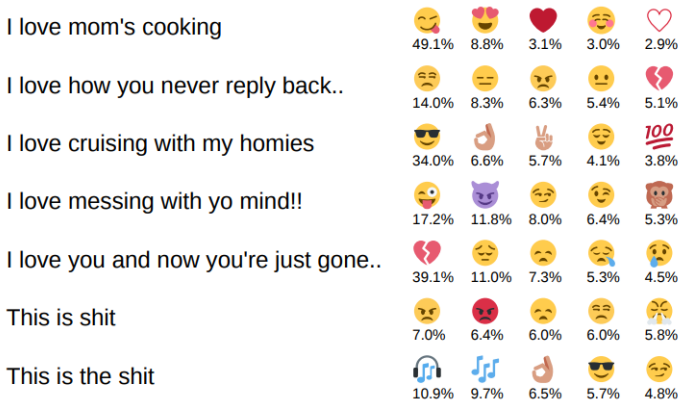

Figure 2.23: Emoji prediction example, source [25]

The study reveals that emojis can be applied to correctly identify texts' emotional content. For example, DeepMoji model identifies various uses of the term "love" as shown in 2.23. There are two main contributions to consider: the fine-tuning method called chain-thaw and the power of emotional embeddings representation with transfer learning applications. The model architecture is defined as follows:

- Input embeddings layer: each word is projected to a 256 features dense vector.
- Within the embedding layers the tangent activation function is applied in order to squeeze the values within $[-1, 1]$.
- Two Bidirectional LSTM hidden layers: with 1024 hidden units for each layer.

- Attention mechanism is exploited by applying skip-connections from the embedding output layer and the first bidirectional LSTM output layer. The weighting schema is the following:

$$\begin{aligned} e_t &= h_t w_a \\ a_t &= \frac{\exp(e_t)}{\sum_{i=1}^{T} \exp(e_i)} \\ v &= \sum_{i=1}^{T} a_i h_i \end{aligned} \tag{2.44}$$

  where v is the is the high-level encoding of the input tweets. Indeed, this embedding are extracted and used for a new method proposed in the fourth chapter.

- In the output layer a softmax function is applied.

- $L2$ regularization is used during the training phase

In summary, the model has the following architecture:

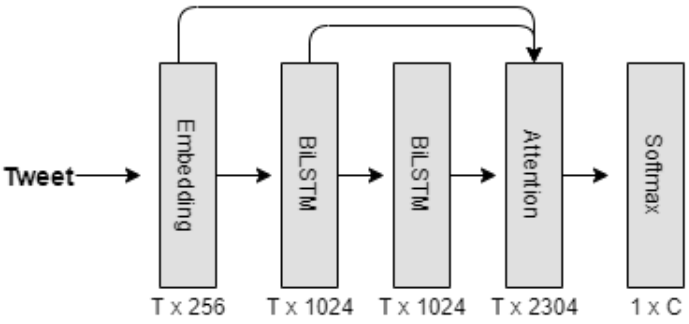

Figure 2.24: DeepMoji architecture

As mentioned above, one of the contributions of this study is the application of the concept of Transfer Learning on different tasks using only DeepMoji as a model trained on a more generic D domain. Three types of finetuning are analysed in the paper: training all layers simultaneously, training only the last layer and applying the chain-thaw method. The results show that the last methodology performs better than the others, and for this reason in the thesis DeepMoji is finetuned with the chain-thaw method. The new Transfer Learning method is based on a very simple concept: unfreeze the weights of each layer sequentially, one at a time. Each layer is trained independently of the others: we first start with the last layer, the one referring to the softmax activation function, by freezing the weights of the other layers. Once the number of epochs is reached, the remaining

layers are trained sequentially. So in this case, starting from the embedding layers up to the attention layer.

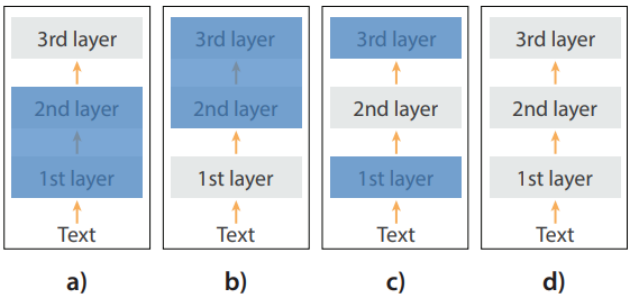

Figure 2.25: Chain-thaw procedure, source [25]

As can be seen from the figure, the first layer to be trained is the last (or new) one, proceeding sequentially, where at point d all layers are trained simultaneously. This method increases accuracy at the cost of additional computing resources for the finishing process.

2.3.6 A Transformer-based approach to Irony and Sarcasm detection

The third study analysed concerns the model developed by Potamias et al., which is declared to be the current state-of-the-art with regard to irony and sarcasm detection: Transformer-based approach using Recurrent Convolutional Neural Networks [26], called RCNN-Roberta. This perspective underpins RCNN-approach: Roberta's as pretrained networks are useful for multiple downstream activities, their performance may be improved further if other layers are added to the output of the pre-trained model. To achieve this, the researchers apply the principle of finetuning, adding an RCNN to the RoBerta output layer in order to pick up context information. This model is called hybrid, as it combines $Roberta_{BASE}$ (with the configuration of 12 hidden states and 12 attention heads), with a bidirectional LSTM layer, a pooling layer in order to produce a one-dimensional convolution, and a fully-connected layer that flows into the input of the output layer with the softmax activation function.

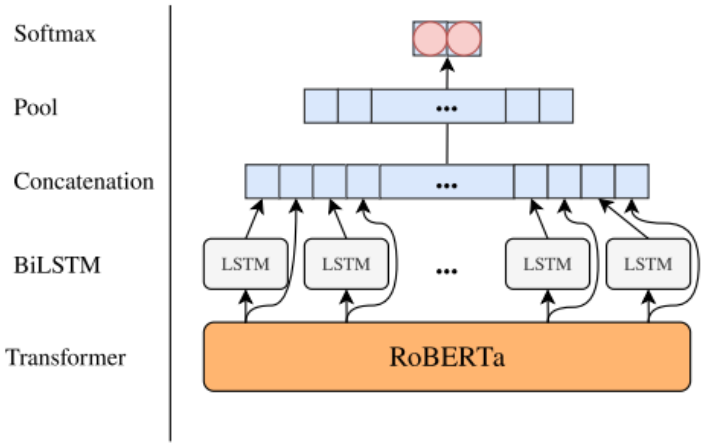

Figure 2.26: RCNN-Roberta architecture, source [26]

The hyper-parameters chosen for the RCNN-Roberta are the following: These hyper-

Hyper-parameter	**Value**
RoBERTa Layers	12
RoBERTa Attention Heads	12
LSTM units	64
LSTM dropout	0.1
Batch size	10
Adam epsilon	$1e-6$
Epochs	5
Learning rate	$2e-5$
Weight decay	$1e-5$

Table 2.1: RCNN-Roberta hyper-parameters

parameters are selected by applying a Grid Search optimisation approach evaluated in 5-folds cross validation. The model is compared in four separate reference datasets with all published approaches. The study also aims to reduce the pre-processing and features extraction phase. Indeed, information extracted with the feature engineering technique produce a high processing cost and especially produce large matrices, inducing many bottlenecks. Researchers claim that RCNN-Roberta, under six criteria over four benchmarks, achieve state-of-the-art results. In comparison, RCNN-Roberta outperformed all other state-of-the-art methods that have been evaluated under all metrics, including ELMo, XLnet and BERT [26]. In this case, the architecture is replicated from scratch, the source code not being available. Not all the information is available, so the results obtained in this study are to be taken into consideration with the limitations of the case.

Chapter 3

Proposed Methods

As mentioned in section 1.6, this chapter intends to highlight one of the contributions developed in this study. Besides, trying to analyse the problem of generalisation power, the development of new methodologies is one of the objectives. Several models are created in order to focus on different aspects of the problem such as trying to introduce new approaches of features extraction from hidden layers of encoder blocks and creating a model that analyses the emotional aspects of the text. Furthermore, it aims to develop a system that jointly exploits Deep Learning and Machine Learning models through different ensemble approaches. The first model is based on the use of the different output encoder layers of BERTweet. The choice of this specific Transformer stems from the need to exploit a contextualised model on data in a specific domain such as Twitter. Indeed, contextualising BERT implies large computational resources and significant amount of data. Therefore, in order to avoid such possible issues, this Transformer has been selected which is the first pre-trained Transformer based on English tweet [27]. In three different task based on Twitter data, BERTweet exceeds SOTA, $Roberta_{BASE}$ and $XLM-R_{BASE}$ models such as part of speech tagging, text classification and named entity recognition. Proving then the usefulness of the pre-trained large-scale and domain language model. BERTweet is trained on 850 million messages deriving from Twitter collected between 01/2012 and 08/2019, by reaching 16 billion word tokens. This Transformer is based on same architecture of $BERT_{BASE}$ and it is trained by applying a masked language modeling objective. While, the training procedure is the same as Roberta's, which makes the pre-training approach more efficient than BERT method.

The second proposed method is developed by exploiting the features extracted from DeepMoji model. This choice is justified by the fact that different information in the text, such as emotional embeddings, is to be exploited by developing a model that is based on attention mechanism and easily trainable.

The last approach is called "Ensemble of ensembles": once the two models using different features, designed on the basis of attention mechanisms, have been developed, a system is created in order to exploit different Machine Learning models together with the new proposals. Specifically, the BMA method is taken into account by mixing it with other ensemble methodologies.

3.1 BERTweet Features-based

In general, there are two ways to use Transformers: the former is to finetune the model on a specific task and the latter is to extract word embeddings from the different output encoder blocks, in order to feed them into a custom model developed for a specific task. In 3.1 is displayed the structure of each encoder outputs (of dimension $w \times 768$), and coloured based on the features representation level, starting from low representation (white-yellow) to high level embeddings representation (orange-red):

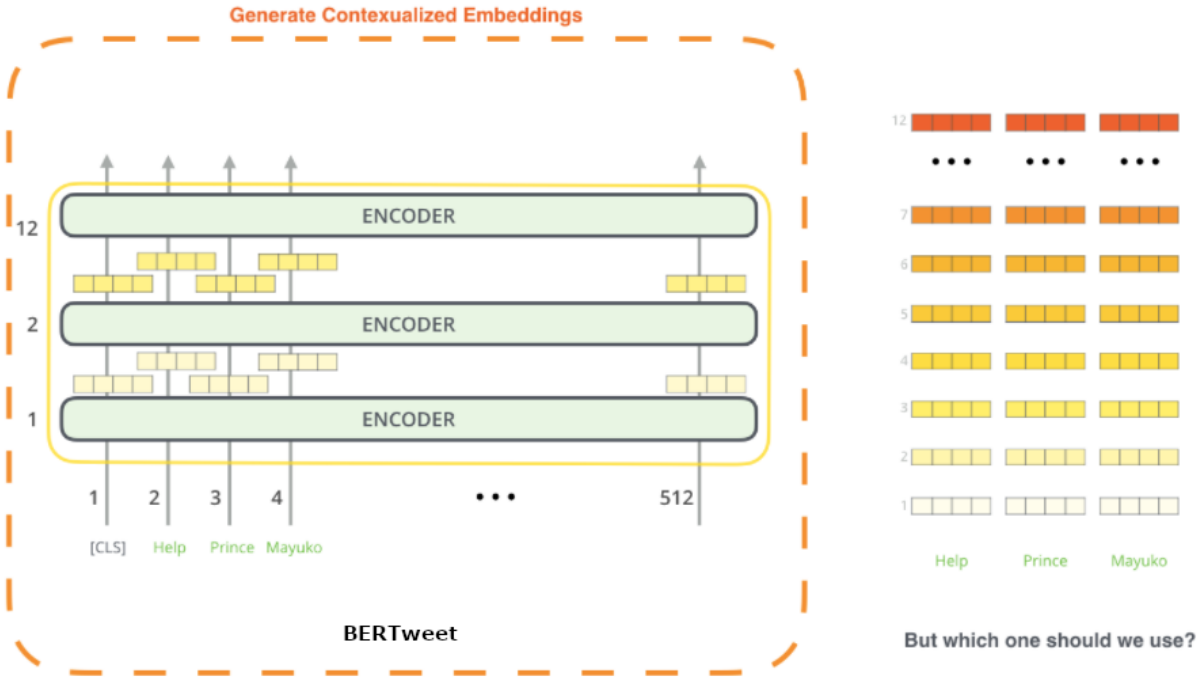

Figure 3.1: Output Encoder Layers of BERTweet, source [28]

The choice of which vectors to use has been the subject of analysis in several studies. It must be pointed out that a general solution for all tasks does not exist, where the best combination of embedding vectors must be found depending on the objective. The baseline, which is suggested by the authors of the paper [23], is estimated on a named entity recognition task by evaluating which combination would produce the highest F_1 score:

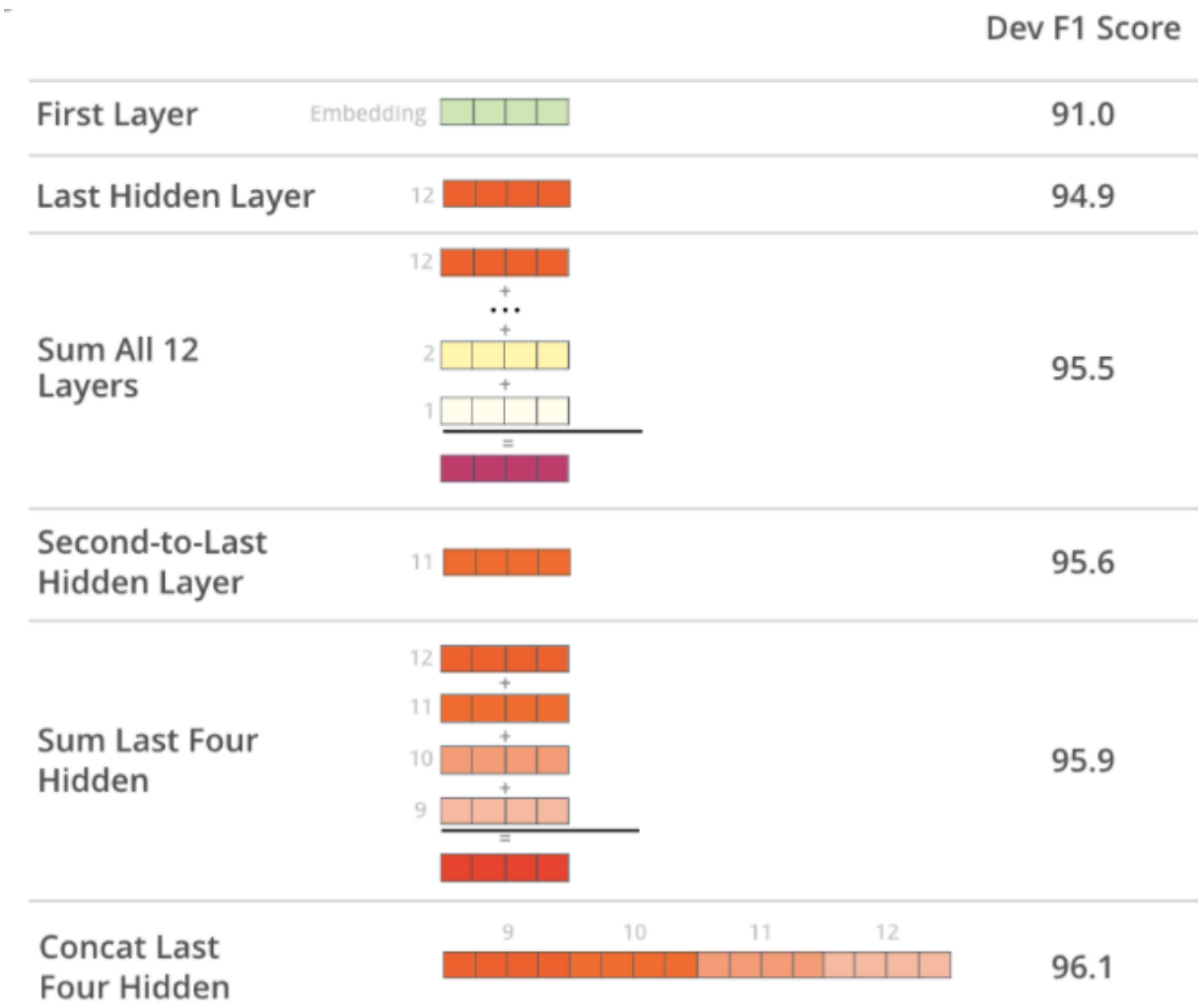

Figure 3.2: Contextualized embedding combinations of BERT, source [28]

As can be seen from the figure 3.2, the best combination is the concatenation of the last four embeddings associated with a high level of feature representation. Algebraic operations are also evaluated, such as the sum of the 12 layers or simply the last four. All these combinations introduce a concept of rigidity, being operations carried out upstream: for this reason, after in-depth analysis, a methodology is proposed that introduces the concept of flexibility in the combination of contextualised embeddings: weights that are

optimised within a custom neural network. In support of this thesis, in the same period as the development of this new combination methodology, a paper [29] is published proposing a dynamic system of joining all embeddings by using linear layers. This supports the thesis proposed for the new model combining the various embeddings layer. Compared to the paper, the proposed solution is different because it exploits 1-dimensional convolutions and residual connections that are fed to self attention layers. Due to computational limits, it is chosen to use sentence embeddings instead of word embeddings, as they are smaller in size. To estimate each sentence embedding, the strategy adopted refers to the baseline of their generation: it is the average of the word embeddings that compose the input sentence by generating a vector of dimensions 1×768. This thesis merely analyses the combination of the last four output encoder layers, laying the foundations for future developments in the generation of increasingly precise features. The first step is to concatenate the last four hidden encoder outputs on the third axis:

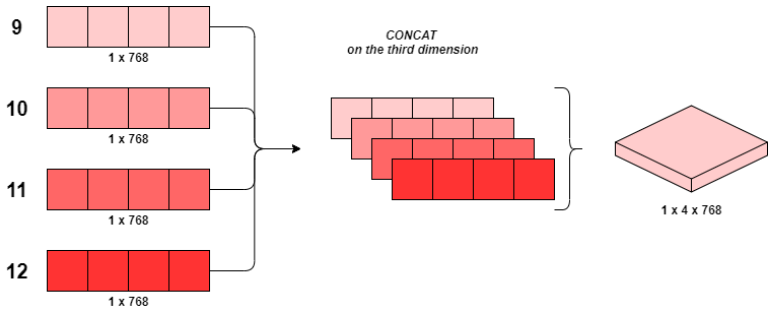

Figure 3.3: Building the input features

From 3.3 the last n-dimensional vector is created in order to generate a fourth channels representation of the sentence, similar to the image tensor structure (the notation is referred to the Pytorch library that is used in order to develop this model, where the channel's dimension is placed as second). The above proposal derives from the idea of concatenating vectors leads to take into account several information levels. Once the features are generated, they are fed to the following model:

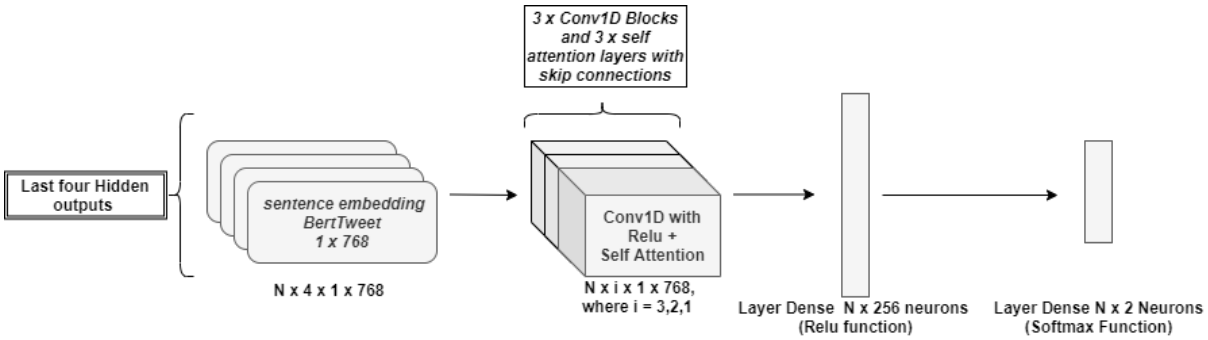

Figure 3.4: *BERTweet Features-based* architecture

In 3.4 is presented the Model 1 architecture that is developed for this thesis. In particular, the features vector is fed into a combination of one dimensional convolution and self attention layers. This block is created in order to synthesise the 4 output encoder layers into one by exploiting the skip connections between the output of each convolution and the various self attention layers, specifically the layer of self convolutional attention block is the following:

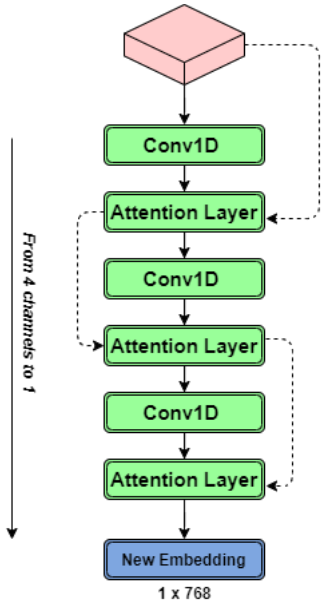

Figure 3.5: First block of BERTweet Features-based architecture

Once the new embedding vector is generated, the tensor is fed into a fully connected layer composed by 256 neurons. After applying the ReLu function, the information flows to the output layer where a Softmax is used in order to estimate the target probability distribution.

3.2 DeepMoji Features-based

An additional information to analyse, in the case of irony and sarcasm, is the emotional context as for example in irony one of the main markers is the use of double polarity within a sentence. In order to exploit this information, the DeepMoji model is chosen as feature extractor, where each emotional embedding is a vector of dimension 1×2304 that describes the emotional semantic of each tweet. Like the previously explained *Model 1, BERTweet Features-based*, *Model 2, DeepMoji Features-based* is also based on the concept of the self attention layer but with the addition of a Bidirectional GRU layer:

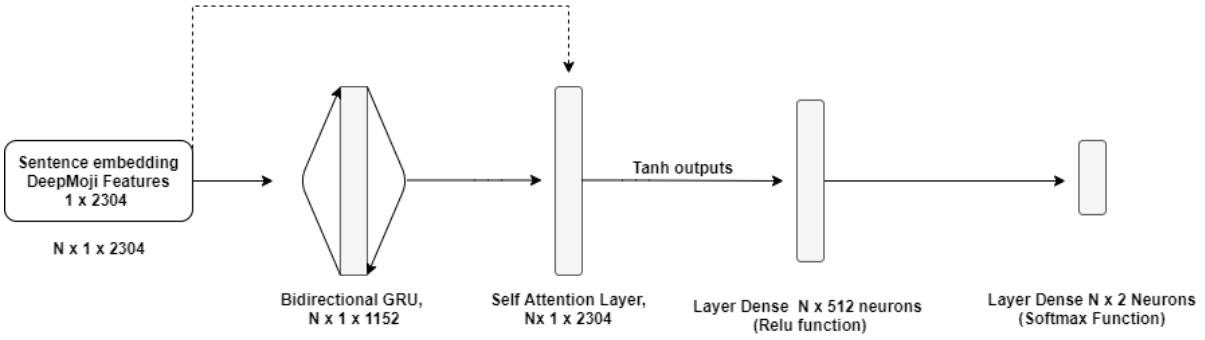

Figure 3.6: *DeepMoji Features-based* architecture

the implementation of Bidirectional GRU and the self attention layer aims to rebuild a more contextualized emotional embedding within these tasks. Indeed, the output of the attention layer has the same dimension of the emotional embedding input. From 3.6 is shown the skip connection that is used as the value input of the self attention layer. Once the information flows to the third hidden layer, a ReLu function is applied within a fully connected layer composed by 512 neurons. As the *Model 1*, the output layer is composed with a Softmax activation function (with 2 neurons) and both of them use the Categorical Cross-Entropy Loss.

3.3 Ensemble of Ensembles

The last proposed methodology is based on a simple concept: exploiting several models simultaneously to estimate the target variable based on the different marginal probability distributions of each classifier. The definition of the term "Ensemble of ensembles" derives

from the fact that within this system we exploit a latent model generated by the BMA ensemble method with *BERTweet Features-based* model and *DeepMoji Features-based*. The intuition comes from two needs: to exploit different models trained on different aspects of the text such as Machine Learning algorithms, which analyse the part of speech, pragmatic particles etc. together with other models like *DeepMoji Features-based*, which focuses on the emotional part of the test. The second need is based on obtaining a classifier where the performance improves thanks to the ensemble method. It must be underlined that the ensemble method does not guarantee an improvement in the performance of the model.

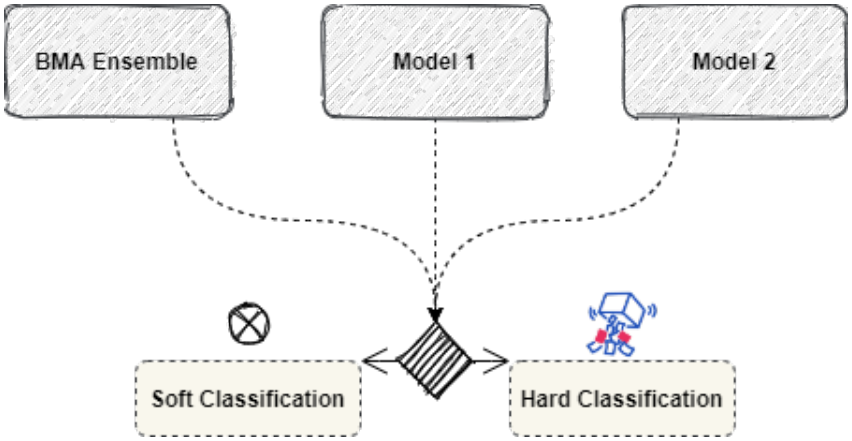

Figure 3.7: Ensemble of ensembles system

From 3.7 is clear how the system is designed, where three different models are merged together with two final choices: soft classification mentioned in the section 2.2.1 and equation 2.2, and hard classification with majority voting technique. The formulation of the former is the following:

$$l_{Ensembleofensemble}(m) = argmax_{l(m)} \sum_{i \in C} P(l(m) \mid i) \tag{3.1}$$

where the set $C = \{BMA, BERTweetFeatures - based, DeepMojiFeatures - based\}$. The choice of l(m) is based on the highest probability, between the two class, obtained by summing the i-th marginal probability on the class j of each classifier. From the point of view of hard classification the process for selecting the y outcome is slightly different from the soft classification. The term hard is referred to the fact that the choice of $l(m)$ is based on the label prediction and not on the probability prediction. The majority voting

algorithm is shown in the following figure:

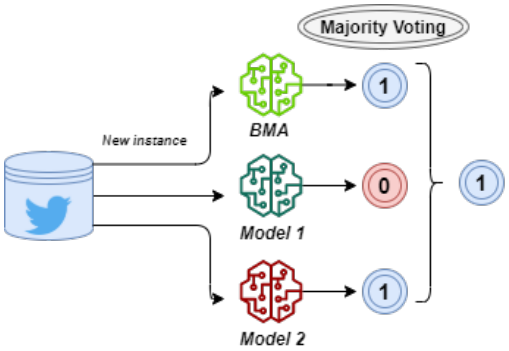

Figure 3.8: Majority Voting system

in other words, the class that obtains the majority of votes from the classifiers becomes the final label the i-th tweet is predicted.

All these approaches (BERTweet Features-based, DeepMoji Features-based and Ensemble of ensembles) are the result of ideas developed during the study phase of this thesis, proposing new methodologies to tackle the problem of text classification.

Chapter 4

Dataset and Performance Measures

4.1 Dataset gathering

In Machine Learning and Deep Learning the most important aspect is the dataset available for training the algorithms. Mainly, there are three fundamental aspects: representativeness (important for the inference phase), quality (garbage in garbage out), and sample size. For the sarcasm and irony detection task, the availability of datasets reflecting these three properties is very limited. Both because it is not easy to identify, in an automated way, documents with rhetorical figures (especially on Twitter and other social networks), the interpretation of what is sarcastic or ironic is very subjective and they occur infrequently. The models that are state-of-the-art for the irony and sarcasm detection task are very often based on small dataset sizes, also because of the reasons listed above. The commitment to identify multiple sources of quality data is the basis of this thesis. To answer the question, it is possible to produce models that can perform better with even more data available. Several datasets have been collected to solve the problem of identifying rhetorical figures, such as irony and sarcasm. Due to the limitations imposed by Twitter, the data collection is carried out through the use of the id of each tweet made available by the authors of the different studies. However, this implies strong limitations due to the tweets or accounts deleted on the social network. Indeed, it was not possible to collect all the data used by the authors of the replicated studies.

4.1.1 Training set

The first step of this study concerns dataset gathering for both tasks related to English tweets. Regarding sarcasm detection, three different source are used for building the training set: the first source is based on [30] studies, and it is composed by 14.070 sarcasm and 16.718 not sarcasm tweets (they are collected with distance labelling technique vased on the hashtags #sarcasm #not). The second source is referred to [10] research on using ensemble methods for addressing these tasks: a total number of 8000 tweets are collected of which 4000 sarcasm and 4000 not sarcasm. Also in this case, the tweets are gathered using distance labelling and after it, three researchers reviewed the data manually. "The inter-agreement between annotators has been computed according to the Fleiss' kappa statistics, which measures the reliability agreement of labeling over that which would be expected by chance. In our case, the inter-agreement statistics $\kappa = 0.79$ indicates a substantial agreement among annotators" [10]. Lastly, [31] the training set is selected: it is composed by 21292 not sarcasm and 18488 sarcasm tweets.

For irony detection, two main sources are identified for the training phase: the first one is referred to SemEval-2018 Task 3: Irony Detection in English Tweets [32], specifically task 3A, where the aim is to detect if a tweet is ironic or not. The training set with raw text is chosen with 2000 not irony and 2000 irony observations. The second source is based on Reyes-Rosso studies [33], where it is composed of 10,000 with irony label, and 30,000 as non-ironic divided by different topics: Politics, Humour, and Education.

	Positive Label	**Negative Label**	**Total**
Sarcasm	36557	42010	78567
Irony unconstrained	11899	31900	43799
Irony constrained	1898	1904	3802

Table 4.1: Training set used for this study

From 4.1 irony is divided into constrained and unconstrained definitions. In order to compare the results obtained with respect to the best models in this competitions, the two methodologies are adopted. This is useful for understanding what is the generalization power of the models developed using different training set distributions.

4.1.2 Test set

In order to produce conclusions regarding the performance of the implemented models, several test sets are chosen as benchmarks for these tasks. As far as sarcasm is concerned, two different test sets are selected. The first refers to the test set of [31], with a size of 1975 observations (975 labelled as non-sarcastic and 1000 labelled as sarcastic). While the second one is chosen to understand the actual inference power of the various models being formed by the whole training and test set of the [34] study, with a total number of tweets equal to 1956 (1648 labelled as non-sarcastic and 308 as sarcastic).

Concerning the irony task, due to the strong limitations of the available data, the test set of [32] task 3 A is chosen, with a total of 784 tweets, of which 473 as non-ironic and 311 as ironic.

4.2 Tweets length

In Natural Language Processing an important aspect to take into account is the length of a document. The length of a sentence is determined by the number of words within the document. The vector's dimension, that represent the sentence, may differ depending on the type of text representation chosen. In the next section different text depiction are described. There are pros and cons: in the case of natural language generation, if an algorithm is trained on very long documents it is able to detect even more granular patterns. This also applies to the text classification task. The more information the pattern has, the more precise it becomes. However, this has strong computational limitations, such as the generation of very large and in most cases, very scattered word matrices (this depends on what kind of representation is used). In addition to the computational limit, there is a bottleneck in the identification of dependencies between the beginning of a document and its end. However, thanks to the limitation imposed by Twitter for the number of characters within tweets, this issue can be exceeded. In table 4.2 is shown the summary statistics of tweets length divided by sarcasm and irony. They are useful when either word embeddings or sequence techniques are used for classification task. For example, when embeddings are generated it is mandatory to set a maximum for the token length, more details are shown in the next sections. The statistics within sarcasm and irony are pretty

	Sarcasm	Irony
min	1	1
max	58	60
mean	16.07	15.47
std	6.04	5.56
25%	12	11
50%	15	15
75%	20	19

Table 4.2: Tweets summary statistics in terms of token count (training set)

much the same. This is important for setting up the study pipeline; depending on which algorithm is chosen, these statistics are used for determining the model architecture.

In figure 4.1, the two distributions of the length of each tweet in the training set used for irony and sarcasm detection are shown.

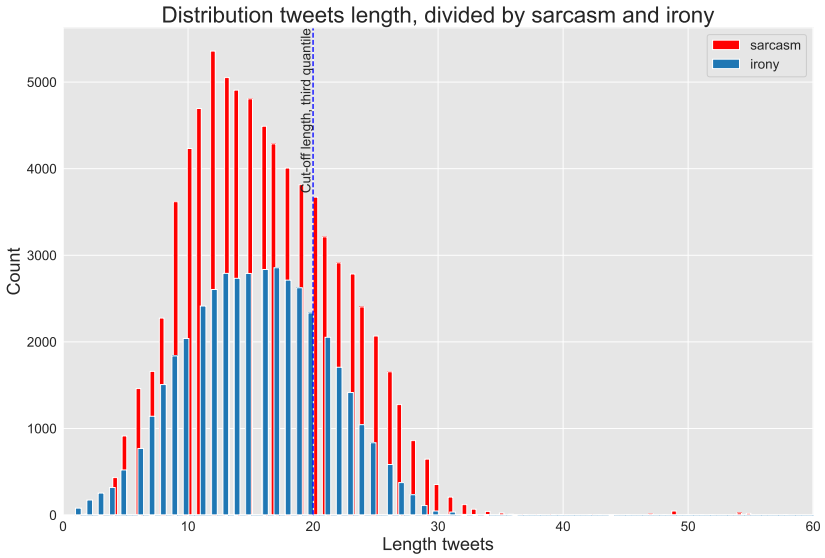

Figure 4.1: Irony vs Sarcasm tweets length distribution, training set

The threshold showed in figure 4.1 is referred to the third quartile of the distribution. This means that all the tweets have the same length: if the length is less then 20, a zero padding techniques is applied, otherwise the tweet is truncated at 20 tokens.

4.3 Evaluation Metrics

In binary classification, several metrics are used to analyse the performance of a model. The classification tasks of sarcasm and irony are defined as a binary problem, where class 0 denotes the absence of figurative language to the i-th tweet, while class 1 denotes the presence of it, specifically 1 is sarcasm in one task and in the other task 1 is associated with irony. These metrics are based on a key concept which is the confusion matrix:

		Predicted	
		+	-
Actual	+	$f_{++}(TP)$	$f_{+-}(FN)$
	-	$f_{-+}(FP)$	$f_{--}(TN)$

Table 4.3: Confusion Matrix for binary classification

The ideal situation is that the sum of TP and TN is 100% and the sum of FP and FN is 0%. The metrics used within this analysis are:

- True Positive Rate/Sensitivity/Recall:

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

- Positive Predictive Value/Precision:

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

- F_1 score:

$$2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

- Accuracy:

$$\frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

Precision determines the fraction of records that turn out to be positive in the group that the classifier has declared as a positive class. The higher the precision, the lower the number of false positive errors committed by the classifier ("how confident is a model in stating that an instance is positive"). Recall measures the fraction of positive instances

correctly predicted by the classifier. A large recall means that only a very small fraction of positive records have been misclassified as Negative class. It is often possible to build baseline models that maximise one metric but not the other. For example, a model that "declares" each record to the positive class has perfect recall but poor precision. A metric that takes both measures into account is the harmonic mean between precision and recall, called the F_1 score. Finally, accuracy expresses the measure of how well a model discriminates between the two classes. In the case of the unbalanced class in the thesis (related to SemEval's unconstrained task), other metrics need to be taken into account, because accuracy is insensitive to the zero rule. For this reason, it is useful to take into account the F_1 score, because it expresses how well a model discriminates between the two classes.

Chapter 5

Experiments

This chapter brings together all the results analysed in the course of this study, where the previously explained models are evaluated on three different datasets. For this reason, a section is devoted to each corresponding task. The experiment conditions are defined, which hyper-parameters are used, and on the basis of the results obtained, the main contributions stated in chapter 1 are brought to light. Indeed, several models are compared in order to state which of them is the best, under the same training conditions, and to analyse how much the different algorithms are able to generalise on different data distributions. The main frameworks used for the development of such models are: for Machine Learning and optimisation, Scikit-Learn and Scikit-Optimize are used. While, for Deep Learning, TensorFlow and Pytorch. Specifically, Scikit-Learn and Scikit-Optimize are applied to reproduce the [10] study. While for [25] and [26], TensorFlow 2.0 is used. For the proposed new models, frameworks like Pytorch and Hugging Face are exploited.

Figure 5.1: Frameworks used for this study

In order to be able to compare the several methodologies, the following chapter is divided in the following way: a section dedicated to each task (sarcasm, constrained irony and unconstrained irony), where the training phases of each methodology are illustrated, and a part dedicated to the comparative analysis and generalisation of the models. In the

next section, experiments on the sarcasm task are illustrated.

5.1 Sarcasm Detection

5.1.1 The Role of Expressive Signals and Ensemble Classifiers

Once all the features have been extracted, with the variations explained in chapter 3, and the models chosen, the study is reproduced in two parts: comparison of the various optimisation processes such as hyper-parameters and the reliability of each model, and the application of the best configurations by comparing the individual classifiers with the ensemble method on the various test set data. In the first phase, the two optimisation methods of Random Search and Bayesian Search are carried out, where with a budget of 10 and with 6-folds cross validation the following hyper-parameters range are optimised:

- **Logistic Regression**: C = $(0.1, 10)$, Penalty = (L_1, L_2), Solver = $(Liblinear, Saga)$
- **X Gradient Boosting**: Number of estimators = $(200, 500)$, Learning rate = $(0.01, 0.6)$, Depth = $(3, 30)$, Subsample = $(0.3, 0.9)$
- **Ada Boost**: Number of estimators = $(50, 500)$, Learning rate = $(0.001, 0.1)$
- **Hist Gradient Boost**: Depth = $(3, 30)$, Minimum samples leaf = $(2, 20)$, Learning rate = $(0.001, 0.1)$
- **Random Forest**: Number of estimators = $(50, 500)$, Depth = $(3, 30)$, Minimum samples leaf = $(5, 20)$, Minimum samples split = $(5, 20)$

Once the search space for each model has been identified, all the various combinations of features are evaluated in order to find the best ones for each classifier. All the possible combinations take into account the baseline i.e. embeddings reduced in size by PCA. This means that the basic features are the embeddings and gradually the others are added. The following combinations between Embeddings (PCA), Part of Speech (POS), Pragmatic Particles (PP) and Polarity & Subjectivity/Objectivity(POL) are evaluated:

1. Embeddings (PCA)
2. Embeddings (PCA) + POL

3. Embeddings (PCA) + POS

4. Embeddings (PCA) + PP

5. Embeddings (PCA) + POS + PP

6. Embeddings (PCA) + POS + PP + POL

7. Embeddings (PCA) + PP + POL

8. Embeddings (PCA) + POS + POL

All these features are assessed by estimating a confidence interval using the boostrap method, and according to which combination performs statistically better, they are applied to each model. The objective function chosen for optimisation is the accuracy: this choice is justified because the data is balanced and it is therefore considered significant to the problem. The following plot shows the results of the Random Search optimisation process.

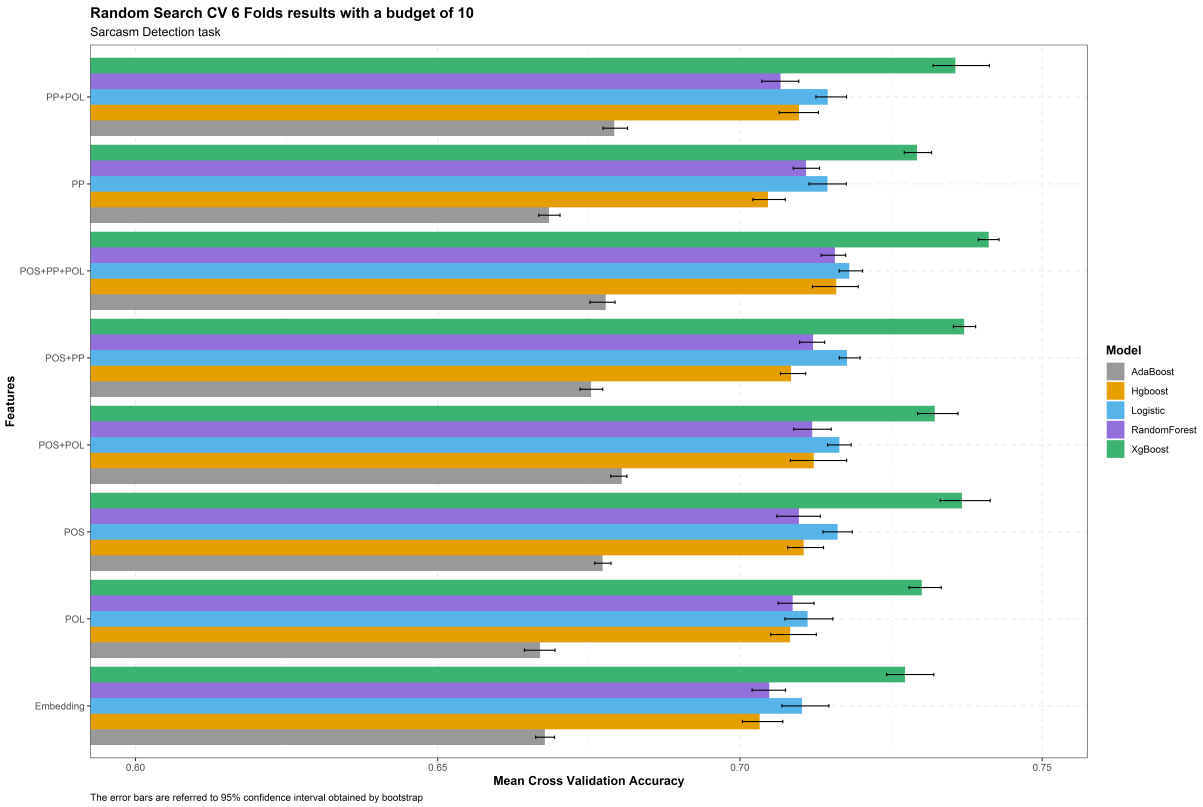

Figure 5.2: Random Search for Sarcasm

To determine which of the two optimisations worked best, with a very limited budget, a comparison is made in terms of results between 5.2 and the following:

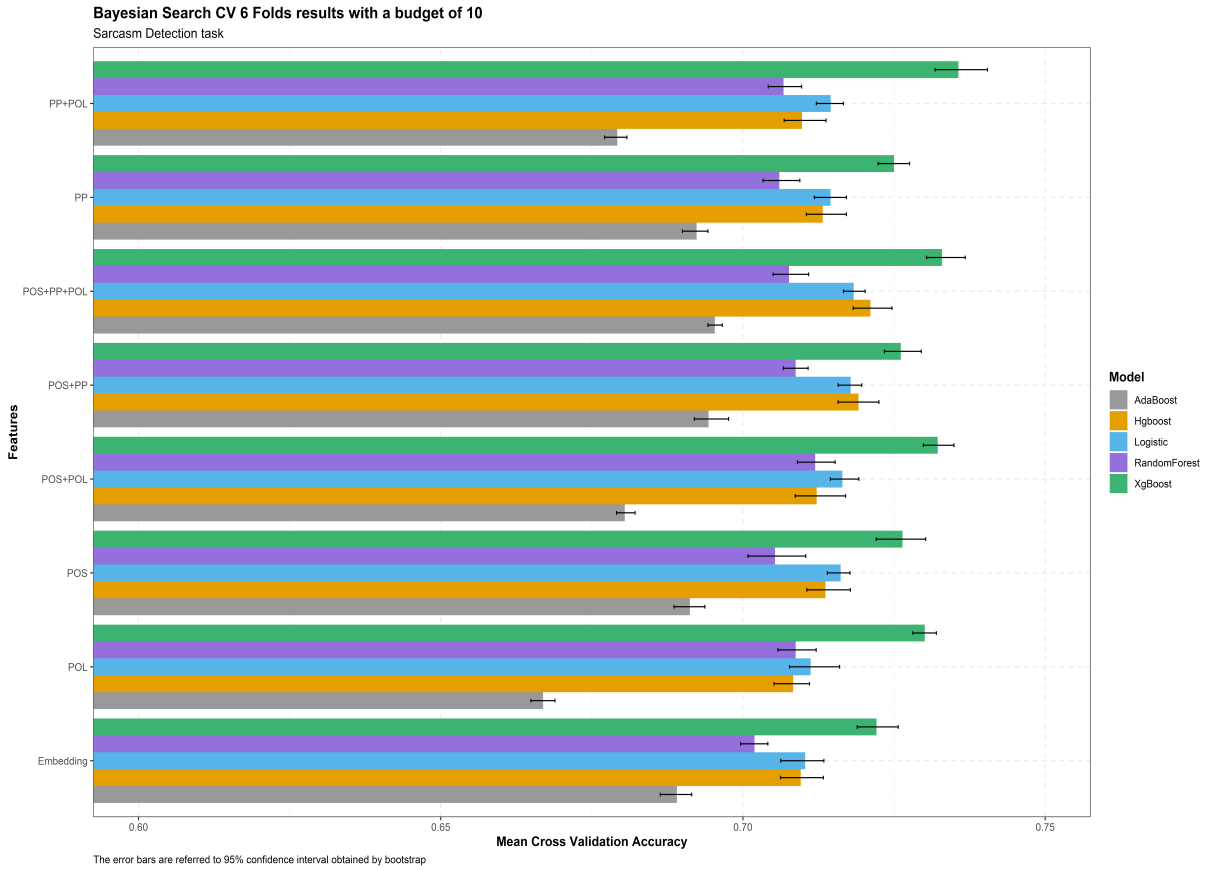

Figure 5.3: Bayesian Search for Sarcasm

What emerges in both optimisations is that the inclusion of certain features, in addition to embeddings, improves performance in a statistically significant way (with 95% confidence level). Indeed, as in the paper emerges that mixing information such as the pragmatic particles in the text and the part of speech, allows to better identify patterns related to sarcasm. Interestingly, the combination of these variables improves the performance of each model by an average of 4 percentage points. For this reason, the variables that have a significantly higher accuracy are identified. The combination of Embedding + PP + POS + POL and Embedding + PP + POS results equal as the lower bound and upper bound of each accuracy confidence interval overlap by an α of 0.05. The former refers to the best results obtained with Random Search approach, while the latter refers to the best accuracy obtained with Bayesian Search method. At the end, there isn't no significance evidence for declaring which optimization algorithm performs better. For this reason, the

hyper-parameters identified by the Random Search process are chosen as the confidence interval is narrower and the results, on average, are slightly higher than with the Bayesian Search method. This hyper-parameters based on Embedding + PP + POS + POL are the following:

- **Logistic Regression**: C = 6.38, Penalty = L_2, Solver = $Saga$, where C = 6.3 indicates a lighter regularization and the solver Saga is useful in order to speed up the convergence time for large dataset.

- **X Gradient Boosting**: Number of estimators = 173, Learning rate = 0.06, Depth = 9, Subsample = 0.83, where the number of estimator indicates how many gradient boosted trees are taken into account. In this case, the optimization process find out that 173 estimators are the right structure for this dataset with the constrain of depth fixed to 9.

- **Ada Boost**: Number of estimators = 158, Learning rate = 0.96

- **Hist Gradient Boost**: Depth = 23, Minimum samples leaf = 16, Learning rate = 0.09

- **Random Forest**: Number of estimators = 193, Depth = 18, Minimum samples leaf = 10, Minimum samples split = 12, these results such as the minimum samples split equals to 12 could lead to the overfitting problem, but it seems that due to the high dimensional features vector space and the number of instances the optimization algorithm find out this trade-off by maximizing the accuracy of each validation fold.

Once the hyper-parameters have been chosen, based on the same optimisation validation set, Bayesian Model Averaging is applied, which exploits multiple classifiers in search of performance improvements. As mentioned in chapter 2, to estimate the reliability of each model a 10 folds cross validation is applied and the average value of the corresponding F_1 scores for each model is estimated. After this estimation has been made, the best combination between the various models is evaluated on the basis of a second validation kept out of the training phase. The best ensemble includes: Logistic Regression, Random Forest and X Gradient Boost, resulting in the highest average accuracy value, which is

0.74. From the point of view of test set, the following tables shown the results between the single classifiers and the ensemble method:

	Accuracy	F1 Score	Precision	Sensitivity
Logistic Regression	0.63	0.46	0.77	0.65
X Gradient Boost	0.83	0.64	0.89	0.73
Ada Boost	0.64	0.45	0.76	0.64
Hist Gradient Boost	0.69	0.49	0.79	0.66
Random Forest	0.85	0.67	**0.90**	0.75
BMA	**0.87**	**0.68**	0.76	**0.90**

Table 5.1: Riloff Test Set Results, out-domain distribution

The best performances in the table 5.15 are highlighted in bold. What emerges, for Riloff data set, is that the best models are Random Forest and X Gradient Boosting, but the ensemble method performs even better, suggesting that this approach helped generate a more accurate model, thanks to the use of the reliability factor and the combination of multiple models. While, for the Ghosh test set the results are reported in this table:

	Accuracy	F1 Score	Precision	Sensitivity
Logistic Regression	0.78	0.79	0.78	0.78
X Gradient Boost	0.82	0.84	0.81	0.83
Ada Boost	0.75	0.76	0.74	0.75
Hist Gradient Boost	0.76	0.77	0.76	0.76
Random Forest	0.82	0.84	0.82	0.83
BMA	**0.84**	**0.85**	**0.85**	**0.84**

Table 5.2: Ghosh Test Set Results, in-domain distribution

In this case, BMA model outperforms in all the metrics. The interesting thing, which is explored later, is the fact that the performance in the case of an out-domain dataset (Riloff data set) worsens significantly, suggesting that probably the analysed models are not enough to identify a generic pattern defining sarcasm except for BMA ensemble method, Random Forest and XGBoost.

5.1.2 DeepMoji for detecting sentiment, emotion and sarcasm

The DeepMoji model is implemented following the authors' guidelines. For this reason, the training phase for the specific task is based on the fine-tuning methodology called chain-thaw. There are no hyper-parameters to define but only the number of epochs that

the model has to perform for each layer and the batch size. Moreover, as explained in chapter 3, the length relative to the third quartile of the distribution of tweets is 20. For this reason, during the preprocessing phase, each tweet is mapped with a maximum length of 20. The total number of epochs set is equal to 1 for each layer, since it was necessary to limit the overfitting problem. Indeed, setting the number of epochs higher, the model starts to have a high variance with respect to the training data. As far as batch size is concerned, for computational reasons, it is fixed at a size of 24. The only regularisation hyper-parameter used concern the dropout in the embeddings layer and the dropout in the penultimate layer: the former is set at 0.8 and the latter at 0.7. During the training phase the data are split into training and validation with an 80 and 20 proportion, so that it is possible to compare whether the model can generalise to a subset of data on which it has not been trained. The results obtained during training are shown in the following graphs, where the accuracy and the loss function are evaluated.

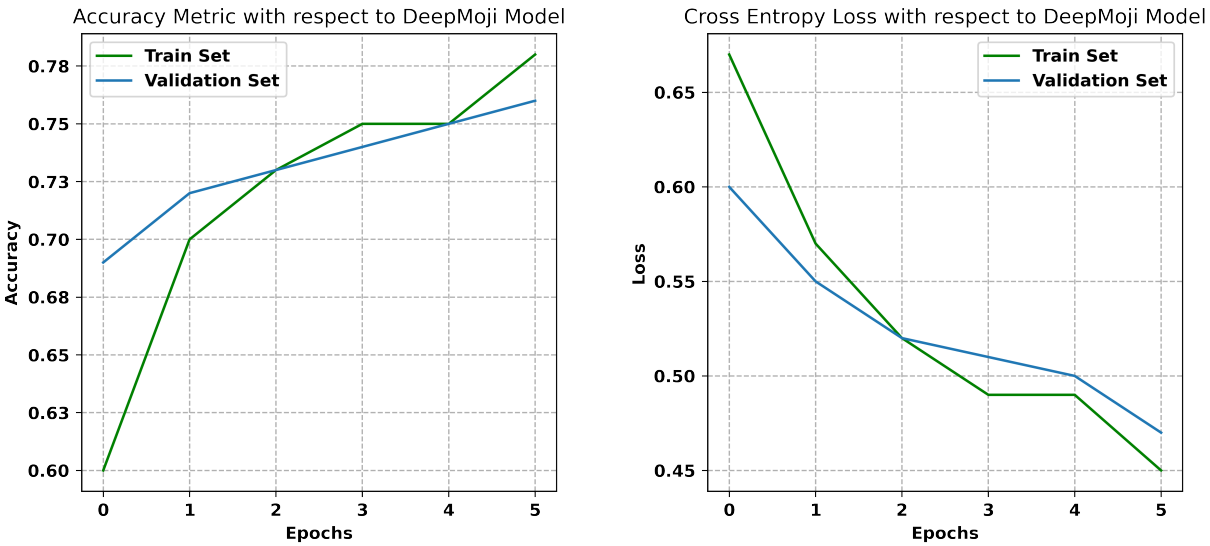

Figure 5.4: Training phase DeepMoji fine-tuning, Sarcasm

In 5.4 are displayed the metrics related to the training phase of DeepMoji model. After the sixth epoch, early stopping is applied since the validation curve starts to diverge from the training curve. This is ascertained as the epochs increase, indeed in the research phase it is stated that with two epochs for each layer, the model begins to overfit.

	Accuracy	F1 Score	Precision	Sensitivity
Training Set	0.81	0.81	0.82	0.81
Validation Set	0.78	0.76	0.78	0.78

Table 5.3: Training and Validation set DeepMoji, Sarcasm

The results shown in 5.3 suggest that there is an acceptable balance between model bias and variance with respect to the validation data.
As regards the inference analysis on out-domain and in-domain datasets, the results are less satisfactory than the methodology based on Machine Learning models.

	Accuracy	F1 Score	Precision	Sensitivity
Riloff Set	0.71	0.50	0.80	0.66
Ghosh Set	0.78	0.80	0.77	0.79

Table 5.4: Test set results from in-domain/out-domain distribution, DeepMoji model

Also in this case, it emerges how the results are affected by the in-domain and out-domain problem. Indeed, Ghosh test set metrics are in line with the performance of the validation set. While for the Riloff dataset, the results worsen considerably, especially the sensitivity and the F_1 Score metrics.

5.1.3 A Transformer-based approach to Irony and Sarcasm detection

For the RCNN-Roberta model changes have been made with respect to some hyper-parameters: regarding the Roberta block, the dropout of the last layer and the attention dropout are fixed at 0.4. The authors state that they set the length of each vector equal to the highest number of tokens within a tweet. In this case, as in DeepMoji, it is set to 20, for computational reasons. The contextualisation part of Transformer and fine-tuning are not clearly defined in the paper. So, besides not contextualising Roberta (too high dimensionality), the RCNN-Roberta model is trained by frizzing Roberta's layers and letting other layers be optimised for a total of 25 epochs. The fixed batch size is 50 and the categorical cross-entropy is applied as loss. Instead, the optimiser is Adam setting a low learning rate equal to 0.00002 and an epsilon equal to 0.001. The last hidden layer is composed by 128 neurons with ReLu activation function. Also in this case, the split

between training and validation is based on the proportions 80 and 20.

In order to optimise the training time, this phase is divided in the following ways: use Roberta as a feature extractor and once all the data has been processed, train the remaining part of the neural network. In the following figures, the accuracy and loss metrics in the model training phase are shown.

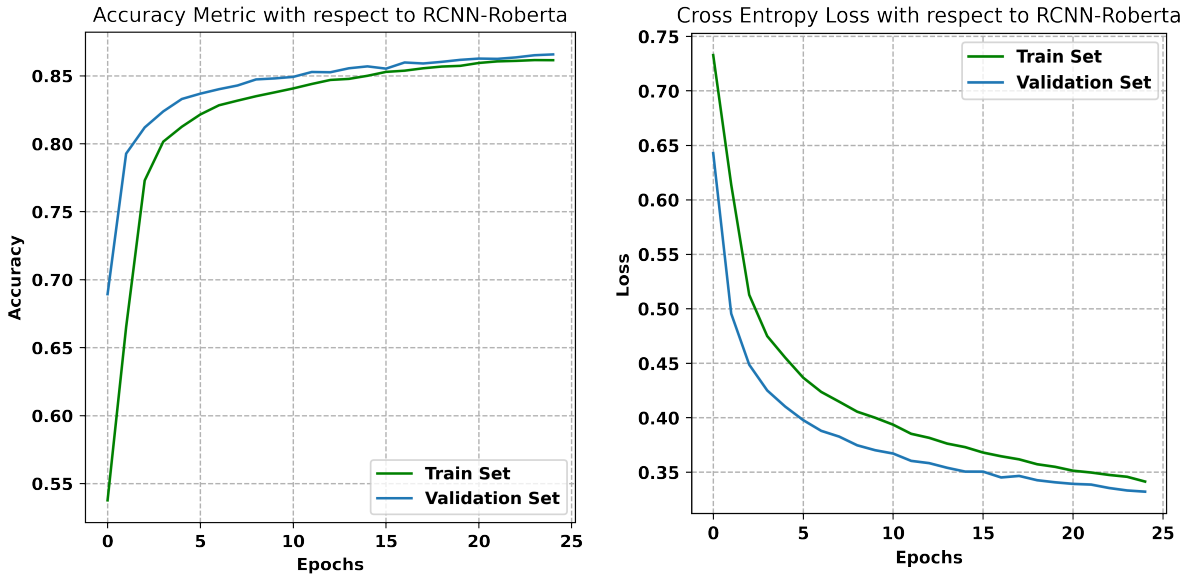

Figure 5.5: Training phase RCNN-Roberta model, Sarcasm

The figure 5.5 shows that after 25 epochs the model converges, generating balanced estimates between the training and validation data.

	Accuracy	**F1 Score**	**Precision**	**Sensitivity**
Training Set	0.87	0.86	0.87	0.87
Validation Set	0.86	0.85	0.86	0.86

Table 5.5: Training and Validation set RCNN-Roberta, Sarcasm

From 5.5 the results show that the model achieves good performance between the training set and the validation. This implies that should not be overfitting within the model and the external results. With regard to the external validation dataset, the results are as follows:

	Accuracy	F1 Score	Precision	Sensitivity
Riloff Set	0.67	0.41	0.70	0.61
Ghosh Set	0.83	0.83	0.83	0.82

Table 5.6: Test set results from in-domain/out-domain distribution, RCNN-Roberta

The following metrics show, that the two distributions are strongly affected by problem of bias in the patterns identified by the model for the in/out domain dataset. Moreover, the results obtained between the validation and the Ghosh test set are in line as it is an in-domain dataset for the model.

5.1.4 Proposed Methods

The development of new methodologies implies having to create models from scratch while trying to achieve levels of accuracy with the constraints of computational power and data availability. For this reason, embeddings are extracted using models trained on a large amount of data. The first model analysed in chapter 3 is based on the use of sentence embeddings extracted from BERTweet. This classifier presents the following configuration: for each convolutional layer a kernel of dimension 1 is applied, where the first one takes as input 4 channels and as output produces 3, this is replicated until reaching the dimension of a tensor with one channel. For each convolutional layer is applied the batch normalization and ReLu activation function. As seen before, between a convolution and another, there is a layer of self attention with a number of neurons equal to 768. The function of activation hyperbolic tangent is applied to the output of each attention layer. Within the block between convolution at 1 dimension and attention layer, the dropout with a value equal to 0.3 is used. Once reached the last attention layer, the tensor is flattened and fed into a dense layer with 768 neurons, with ReLu activation function. Finally, the output layer of the model presents a number of neurons equal to 2 with Softmax activation function. The weights of each layer with ReLu function are initialised through the uniform Xavier distribution with a gain equal to 5/3. During the training phase the learning rate is set to 0.0001 based on the AdamW optimizer. This optimizer exploits the reduce on plateau function (with patience equal to 5) in order to restrict the overfitting issue and the problem of local minima. The number of epochs is set to 80, if this number increases then the model starts to overfitting. Finally, the batch

size is set at 64. As in the previous models, the training and validation set was split on the basis of the proportion 80 and 20, and the same applies to the model analysed below (DeepMoji features based).

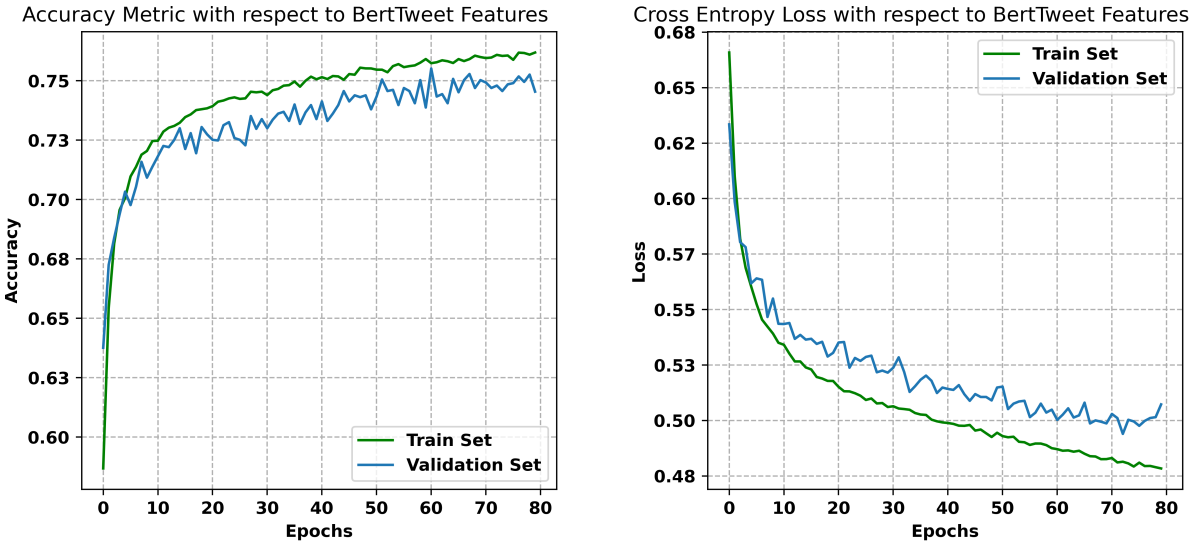

Figure 5.6: Training phase BERTweet Features-based, Sarcasm

After 80 epochs the metrics between training and validation set start to diverge, indeed early stopping is applied. The training phase metrics are reported below:

	Accuracy	F1 Score	Precision	Sensitivity
Training Set	0.78	0.77	0.75	0.78
Validation Set	0.76	0.78	0.77	0.79

Table 5.7: Training and Validation set BERTweet Features-based, Sarcasm

As can be seen from the table 5.7, the use of early stopping made it possible to develop a model that is parsimonious with respect to training and validation estimates. Regarding the inference phase, the results are as follows:

	Accuracy	F1 Score	Precision	Sensitivity
Riloff Set	0.65	0.47	0.78	0.65
Ghosh Set	0.79	0.81	0.79	0.80

Table 5.8: Test set results from in-domain/out-domain distribution, BERTweet Features based, Sarcasm

The results in the in-domain test set are satisfactory as the values estimated in the training phase correspond to the evaluation data in the test. This is not for the case with the outdomain dataset, Riloff, where there is a difference of almost 10% in terms of accuracy and even more on the F1 score where the difference (compared to validation) is about 30%. As the F1 score refers to the positive class, this could imply that the model is not able to identify sarcastic tweets well.

The second model, based on DeepMoji features, presents a different and simpler architecture than the BERTweet Features-based algorithm: the input tensor of dimension 1×2304 is fed into the bidirectional GRU that contains 1152 neurons ($\times 2$ for the bidirectional structure). In this layer the dropout is set to 0.55, and the output of GRU layer flows into an attention layer (2304 neurons) that estimates the attention scores between this output and the residual connection of the input layer. The last hidden layer presents a dense structure with 512 neurons and a ReLu activation function. In this layer the dropout is set to 0.55, and the output flows into the last layer that contains 2 neurons and a Softmax activation function. During the training phase, the learning rate is set to 0.000001 by using Adam optimizer with a weight decay equals to 0.01. The batch size is equal to 32, and the number of epochs is set to 25. During this training phase, the metrics in the training set tend to have a high variance, for this reason the criterion of early stopping comes into operation at the 25th epoch:

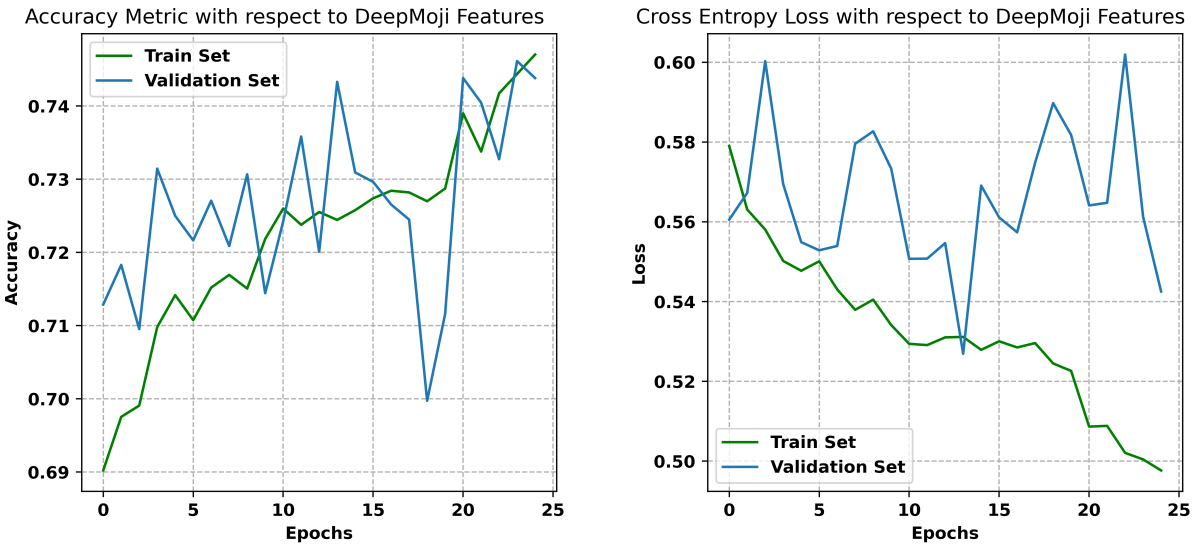

Figure 5.7: Training phase DeepMoji Features-based, Sarcasm

In figure 5.7 a strongly irregular trend emerges: different architectures, different learning rates and optimisers are analysed in order to limit the overfitting issue, however the best result is obtained with this trend in the plot.

	Accuracy	**F1 Score**	**Precision**	**Sensitivity**
Training Set	0.77	0.74	0.78	0.79
Validation Set	0.75	0.71	0.74	0.76

Table 5.9: Training and Validation set DeepMoji Features-based, Sarcasm

As in the other models, the use of weight decay and dropout provides a good trade-off between bias and variance.
The metrics in test sets are as follows:

	Accuracy	**F1 Score**	**Precision**	**Sensitivity**
Riloff Set	0.74	0.52	0.80	0.67
Ghosh Set	0.82	0.83	0.82	0.82

Table 5.10: Test set results from in-domain/out-domain distribution, DeepMoji Features based, Sarcasm

The results that emerge are slightly different from the conclusions drawn in previous models. In fact, exploiting the concept of emotion, contextualised with a model created from scratch, led to better results in the case of the out-domain dataset. Although the results of the Riloff test set of accuracy and specificity improve, the other metrics have significantly lower values. While for the case of the in-domain test set the results continue to be in line with the training metrics. Regarding Ensemble of ensembles model, Two different approaches are evaluated, soft classification and hard classification. In both methodologies better results are achieved compared to all other models:

	Accuracy	**F1 Score**	**Precision**	**Sensitivity**
Riloff Set	0.72	0.52	0.68	0.82
Ghosh Set	0.83	0.85	0.84	0.83

Table 5.11: Test set results from in-domain/out-domain distribution, Ensemble of ensembles (soft classification), Sarcasm

	Accuracy	F1 Score	Precision	Sensitivity
Riloff Set	0.79	0.60	0.71	0.86
Ghosh Set	0.85	0.86	0.86	0.85

Table 5.12: Test set results from in-domain/out-domain distribution, Ensemble of ensembles (hard classification), Sarcasm

It is interesting to see how the hard classification system achieves better results than the soft classification method. Combining the various models that focus on various aspects of the text would seem to improve the ability to generalise in the case of out-domain datasets with respect to Deep Learning models.
To summarise and compare the various models, tables comparing the various metrics are provided, so as to state which system performs best in the case of in-domain and out-domain datasets.

	Accuracy	F1 Score	Precision	Sensitivity
BMA	**0.87**	**0.68**	0.76	**0.90**
DeepMoji	0.71	0.50	**0.80**	0.66
RCNN-Roberta	0.67	0.41	0.70	0.61
BERTweet Features-based	0.65	0.47	0.78	0.65
DeepMoji Features-based	0.74	0.52	**0.80**	0.67
Ensemble of Ensembles (soft)	0.72	0.52	0.68	0.82
Ensemble of Ensembles (hard)	0.79	0.60	0.71	0.86

Table 5.13: Summary Riloff test set results, Sarcasm

The approach based on feature extraction and machine learning models makes it possible to develop systems that are better able to capture the patterns of sarcasm. Indeed, the BMA outperforms all other models on 3 metrics: accuracy, F1 score and sensitivity. The fact of using more explicit features such as part of speech, pragmatic particles and information of polarity and objectivity, allows to create models that seem to be more stable in the out-domain case. The only metric that is lower than other models is precision, i.e. of those declared positive how positive they actually are. In fact, the two models linked to DeepMoji reach the highest value of 0.80. This suggests that deep learning models, which are claimed to be state-of-the-art, are indeed able to produce good results in the case of in-domain distribution but do not capture what is the common pattern among them all. While for the in-domain case the results are as follows:

	Accuracy	F1 Score	Precision	Sensitivity
BMA	0.84	0.85	0.85	0.84
DeepMoji	0.78	0.80	0.77	0.79
RCNN-Roberta	0.83	0.83	0.83	0.82
BERTweet Features-based	0.79	0.81	0.79	0.80
DeepMoji Features-based	0.82	0.83	0.82	0.82
Ensemble of Ensembles (soft)	0.83	0.85	0.84	0.83
Ensemble of Ensembles (hard)	**0.85**	**0.86**	**0.86**	**0.85**

Table 5.14: Summary Ghosh test set results, Sarcasm

In this case the model that outperforms on all metrics is the ensemble of ensembles with the hard classification system. What has emerged before is that adding Deep Learning models to the BMA system, produces a slight worsening in the results in the case of out-domain, while in the case of in-domain there is an improvement. This could reinforce what previously said, where Deep Learning models like these, manage to reach good standards of quality only in the case of in-domain, if features like embeddings and emotional information are used.

5.2 Irony Detection

Regarding the irony task, two aspects are mainly analysed based on the competition of SemEval 2018 task 3A: the unconstrained and constrained case. In addition to assessing the performance of the proposed methodologies against the official ranking of the paper, the aspects of including an out-domain dataset in the training phase, compared to simply using the in-domain training set, are analysed to see whether or not the generalisation power of the various models improves.
As far as the Machine Learning based methodology is concerned, for both the constrained and the unconstrained case the identical optimisation procedures are applied, with the same search space of the hyper-parameters, the same budget, etc. For this reason, the optimisation criteria carried out as in section 5.1.1 are not introduced again.

5.2.1 Unconstrained Task

For the unconstrained case, several datasets are used for the training phase: the inclusion of different sources has produced an unbalanced dataset in which the positive class

represents 27 % of the total distribution of the training data. For this reason, the undersampling technique is used in the various models, where within a training batch, all positive observations are taken into account and the negative ones are sampled, creating a balanced subset.

5.2.1.1 The Role of Expressive Signals and Ensemble Classifiers

The only two differences, with respect to the optimisation for sarcasm detection, are the following: the use of the undersampling technique and the objective function to be maximised, specifically the F1 Score is used in order to better evaluate the performance of each model taking into account the problem of unbalanced classes. Indeed, if accuracy is used, the optimisation could follow the zero rule, creating models of very poor quality for the positive class. As for the optimisation based on Random Search, the following results are obtained, based on the boostrap methodology with a confidence level of 95%:

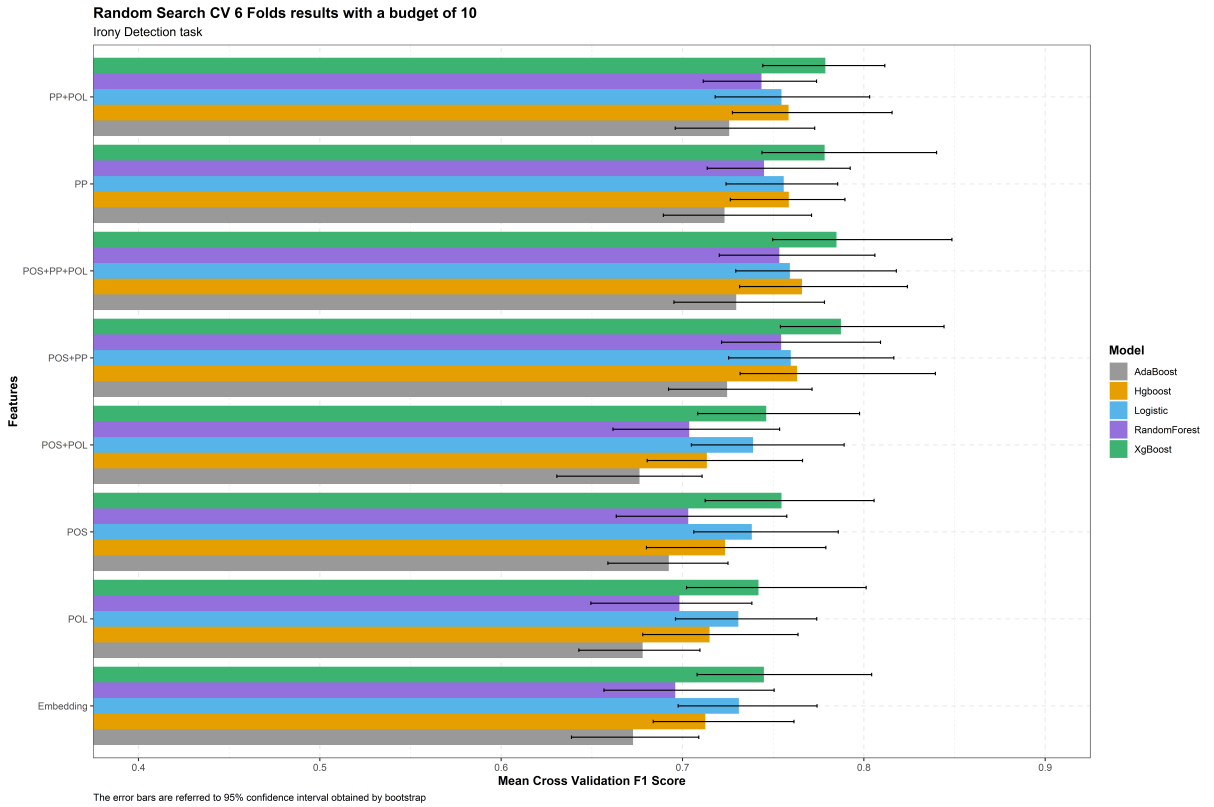

Figure 5.8: Random Search for Irony, unconstrained task

While with regards to Bayesian optimisation:

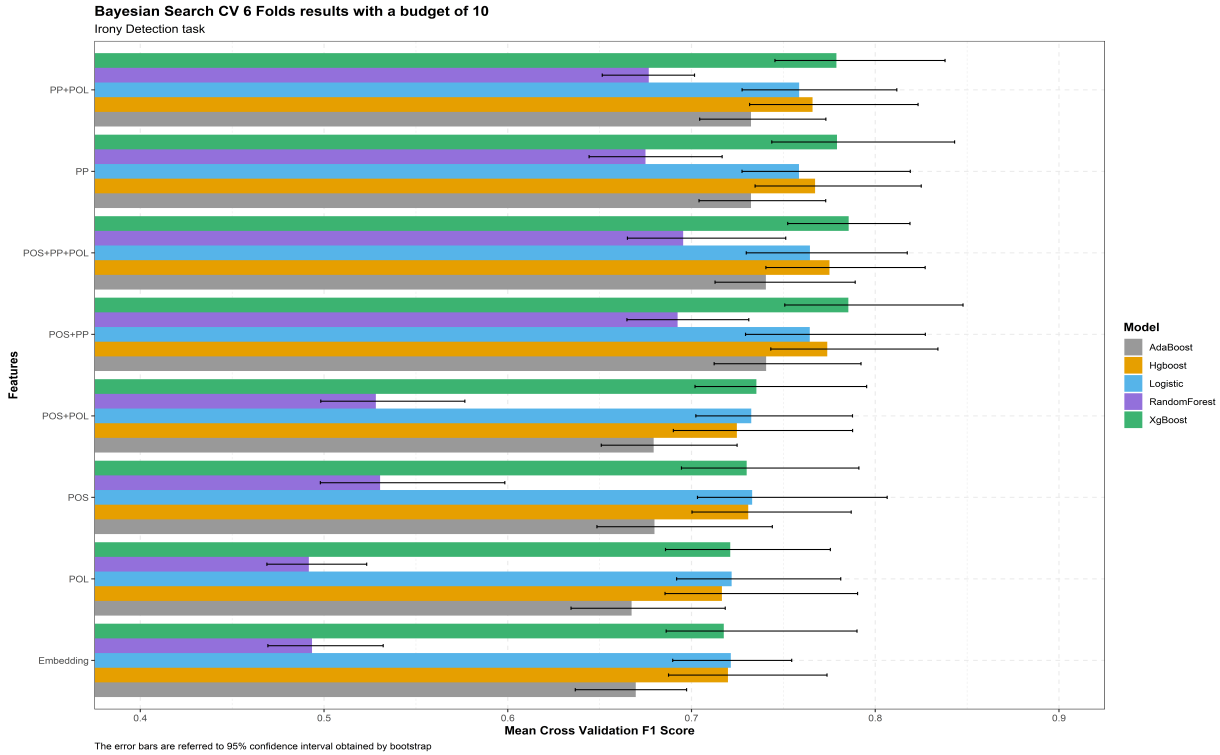

Figure 5.9: Bayesian Search for Irony, unconstrained task

Looking at the two graphs, on average, the Random Search method produces better results, e.g. by looking at the values obtained for the Random Forest on certain combinations of variables. However, for the subset: POS+PP, POS+PP+POL, PP, PP + POL, there is no significant difference either between the optimisation methods or between the different combinations of variables, which is different for sarcasm. These conclusions, regarding which features are more discriminative for irony, are based on Random Search, having found slightly better hyper-parameters on the various combinations.
The best hyper-parameters, based on Random Search, are the following:

- **Logistic Regression**: C = 6.80, Penalty = L_2, Solver = *Saga*
- **X Gradient Boosting**: Number of estimators = 173, Learning rate = 0.07, Depth = 9, Subsample = 0.82
- **Ada Boost**: Number of estimators = 180, Learning rate = 0.45
- **Hist Gradient Boost**: Depth = 19, Minimum samples leaf = 13, Learning rate = 0.12
- **Random Forest**: Number of estimators = 110, Depth = 18, Minimum samples leaf = 10, Minimum samples split = 12

The features used for the models are: POS + POL + PP, since there is no significant difference with the others, the choice is justified by the fact that this combination has the highest upper bound based on F1 Score. Except for the Hist Gradient Boost which is based on the following subset, POS + PP. Also with regard to the estimation phase of the BMA ensemble method, the same configurations as for sarcasm are used: the best combination is found by combining the Logistic Regression and Hist Gradient Boosting models (with an accuracy of 0.68). In the following table, the individual models are compared to the ensemble method:

	Accuracy	**F1 Score**	**Precision**	**Sensitivity**
Logistic Regression	0.61	**0.61**	**0.64**	**0.64**
X Gradient Boost	0.55	0.57	0.58	0.59
Ada Boost	0.56	0.56	0.58	0.59
Hist Gradient Boost	0.56	0.58	0.60	0.60
Random Forest	0.57	0.55	0.59	0.58
BMA	**0.62**	0.60	0.63	**0.64**

Table 5.15: SemEval task 3A Test Set Results, unconstrained

Contrary to sarcasm, the BMA methodology does not outperform in all metrics, but even worsens in some. Indeed, for accuracy the highest value is obtained at 0.62, while for sensitivity it has a value equal to Logistic Regression. For F1 Score and precision, Logistic Regression obtains the best results. In this case, one could conclude that the latter model is more robust than the others. Especially if one analyses the F1 score, which is the reference metric of the SemEval task.

5.2.1.2 DeepMoji for detecting sentiment, emotion and sarcasm

The DeepMoji model is trained using the undersampling technique: as in the previous case, during the training phase, each batch is balanced by considering a balanced distribution between the two classes. As said before, the maximum length of the input vector is fixed at 20 and the dropout is fixed at 0.7 and 0.8 for the respective layers: embedding and the last layer of the network. The batch size is fixed at 24, and for each layer, through the chain-thaw methodology, the number of epochs is fixed at 2, so in total the model is finetuned on 12 epochs. As in the previous cases the split between training and validation is fixed in the proportion 80 and 20. In the following figures, the accuracy metric and the cross-entropy loss, obtained in the training phase, are shown:

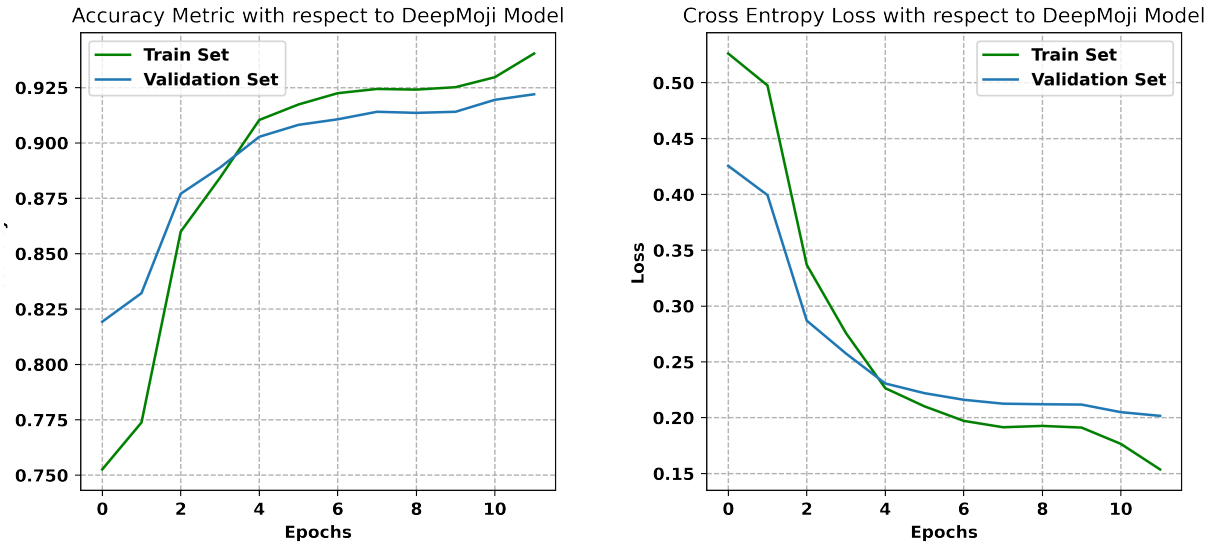

Figure 5.10: Training phase DeepMoji Features-based, Irony unconstrained

As can be seen from the two figures, after the 12th epoch onwards, the metrics between the training set and the validation set begin to diverge, which is the reason why training is stopped. The performance between the training set and the validation set seems to be in line with each other: the following table compares the metrics between the two subsets of data:

	Accuracy	**F1 Score**	**Precision**	**Sensitivity**
Training Set	0.94	0.92	0.94	0.94
Validation Set	0.93	0.86	0.91	0.91

Table 5.16: Training and Validation set DeepMoji, Irony unconstrained

From the results in 5.17 there would appear to be a good trade off between training and validation, however the results in testing deteriorate considerably:

	Accuracy	**F1 Score**	**Precision**	**Sensitivity**
SemEval Test set 3A	0.60	0.59	0.62	0.62

Table 5.17: SemEval test set DeepMoji, Irony unconstrained

The fact of having introduced several out-domain datasets seems not to have helped the model to generalise better on the SemEval test set. Indeed, the metrics drop by 30 percent,

suggesting that the model has learned some aspects of the out-domain training set but not those of the in-domain. There could be a bias here, where trivially the model learns only the topic aspect and not the discriminative patterns of irony.

5.2.1.3 A Transformer-based approach to Irony and Sarcasm detection

The same configuration is used as for sarcasm, the only changes concern the learning rate which is fixed at 0.0002 and an epsilon value (for the Adam optimiser) of 0.002.

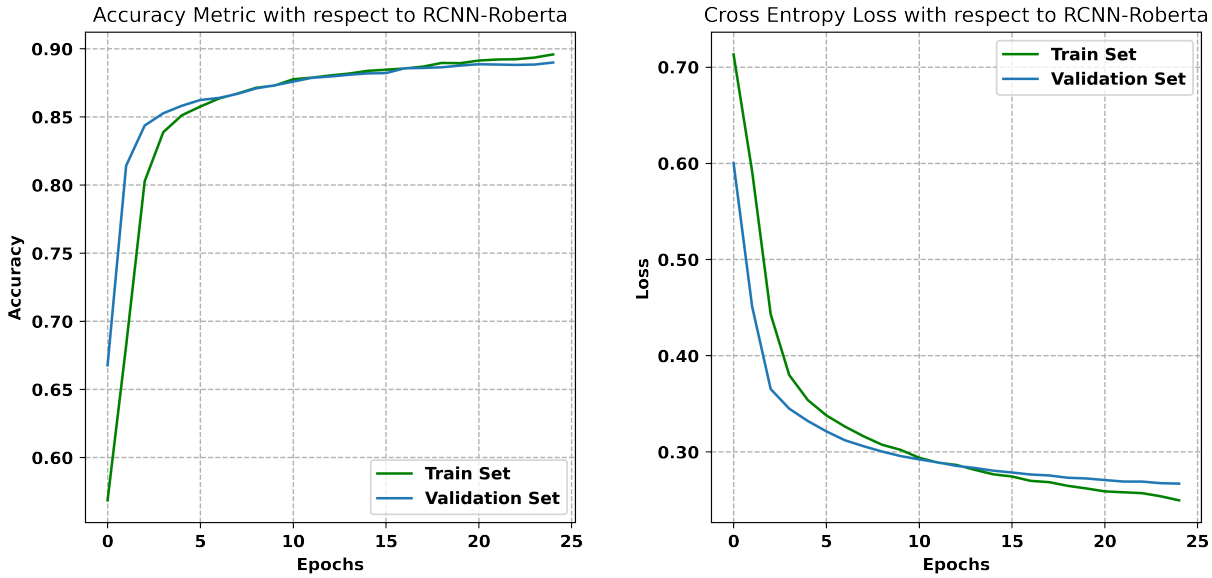

Figure 5.11: Training phase RCNN-Roberta, Irony unconstrained

The results obtained in the training phase are as follows:

	Accuracy	F1 Score	Precision	Sensitivity
Training Set	0.91	0.84	0.92	0.90
Validation Set	0.88	0.81	0.86	0.85

Table 5.18: Training and Validation set RCNN-Roberta, Irony unconstrained

The hyper-parameters used seem to produce satisfactory results in terms of overfitting and underfitting. This reasoning does not apply to the inference phase, which as also seen above, the metrics perform poorly:

	Accuracy	F1 Score	Precision	Sensitivity
SemEval Test set 3A	0.59	0.58	0.61	0.61

Table 5.19: SemEval test set RCNN-Roberta, Irony unconstrained

The interesting thing is to see that the behaviour of Deep Learning models is similar to each other, and what emerged is the fact that Machine Learning models seem to provide more stable estimates. Indeed, both Deep Learning models lose their effectiveness by an average of 30%, as in the case that the various tweets come from different topics, the Deep Learning models mainly detect this pattern while the Machine Learning models, thanks to feature engineering, are not affected by this problem.

5.2.1.4 Proposed Methods

The proposed methodologies are replicated in the same way for each single task, except for some modifications, in order to understand if similar structures could work on different figurative languages. For the case of unconstrained irony task the undersampling technique is applied to the majority class, in order to provide more accurate estimates for the minority class. In the case of the BERTweet Features-based model the differences, with respect to the model trained for sarcasm, are as follows: the number of epochs is equal to 100, the learning rate is set to 0.00002 and the batch size is 32.

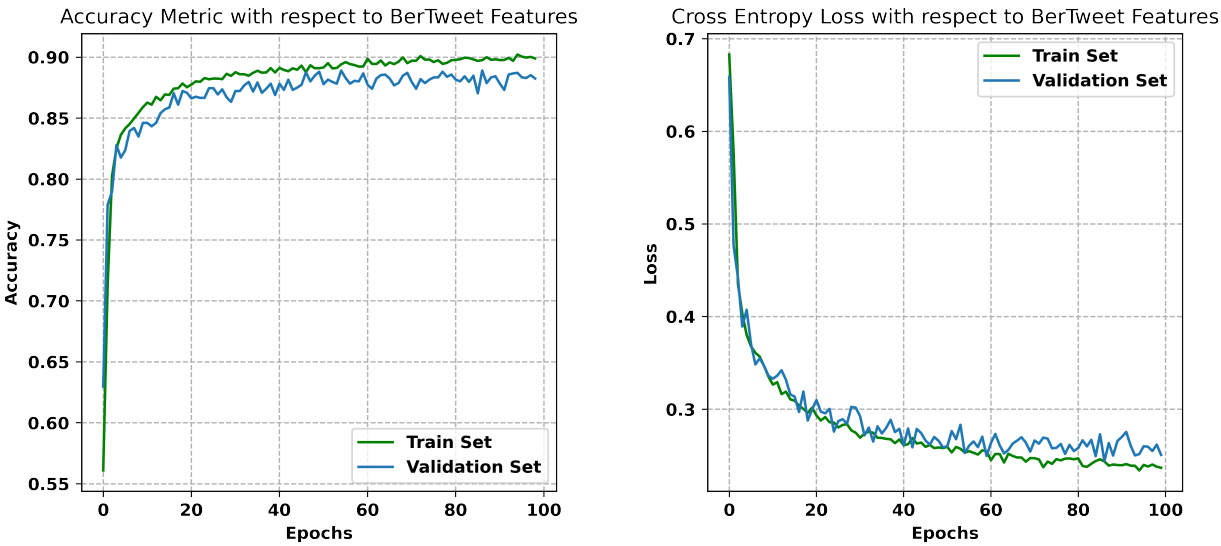

Figure 5.12: Training phase BERTweet Features-based, Irony unconstrained

The training phase metrics are reported below:

	Accuracy	**F1 Score**	**Precision**	**Sensitivity**
Training Set	0.89	0.82	0.90	0.85
Validation Set	0.88	0.81	0.89	0.85

Table 5.20: Training and Validation set BERTweet Features-based, Irony unconstrained

Analysing the trend in loss and accuracy during the training phase shows that the model converges and presents optimal results in terms of trade-off between bias and variance. However, the results in the inference phase do not reflect the values estimated in the training phase:

	Accuracy	**F1 Score**	**Precision**	**Sensitivity**
SemEval Test set 3A	0.53	0.61	0.60	0.66

Table 5.21: SemEval test set BERTweet Features-based, Irony unconstrained

This phenomenon of overfitting against the test set is found in all Deep Learning models, including those proposed. Indeed, from the results in 5.28 we have metrics that vary between about 20% and 35% less than training set.
The following changes are made for the DeepMoji Features-based model: the dense layer with the ReLu activation function is composed by 256 neurons instead of 512. The dropout is set to 0.4 except for the last hidden layer, that is equal to 0.5. The learning rate is set to 0.0002, the batch size is equal to 32 and the number of epochs ended at 35 thanks to the early stopping method.

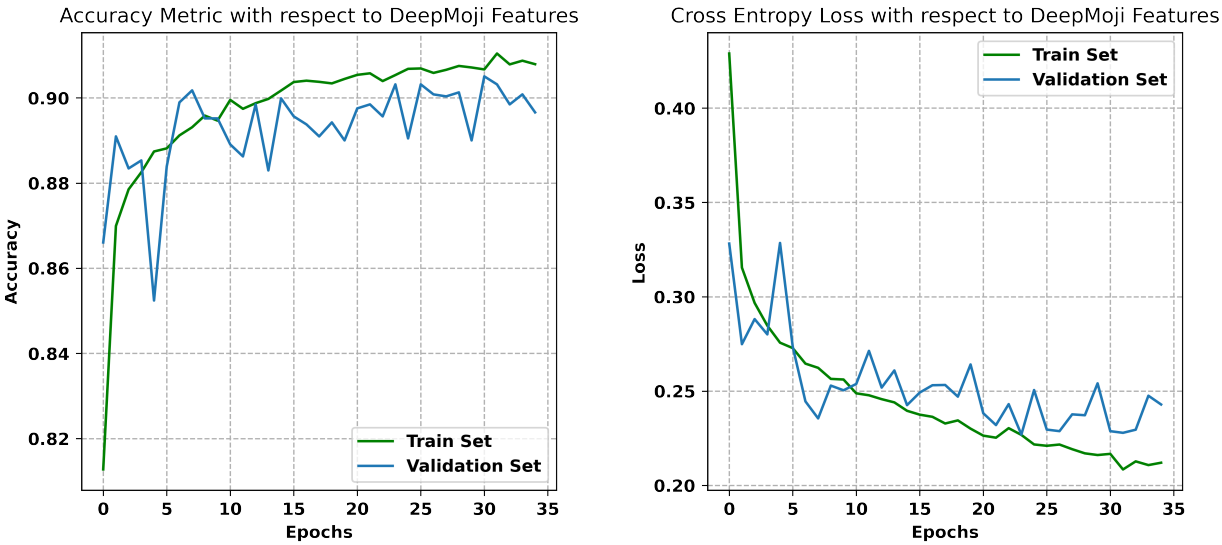

Figure 5.13: Training phase DeepMoji Features-based, Irony unconstrained

The estimated training metrics are as shown in the table below:

	Accuracy	F1 Score	Precision	Sensitivity
Training Set	0.92	0.86	0.91	0.89
Validation Set	0.90	0.85	0.90	0.90

Table 5.22: Training and Validation set DeepMoji Features-based, Irony unconstrained

While for the test set of SemEval task 3A:

	Accuracy	F1 Score	Precision	Sensitivity
SemEval Test set 3A	0.59	0.62	0.63	0.64

Table 5.23: SemEval test set DeepMoji Features-based, Irony unconstrained

As in the BERTweet Features-based model, it also suffers a decrease in performance on the test set. However, compared to the former, it obtains better values on average. Finally, for the last proposed model, Ensemble of ensembles, results in test sets are directly evaluated:

	Accuracy	F1 Score	Precision	Sensitivity
Hard Classification	0.58	0.62	0.65	0.63
Soft Classification	0.61	0.63	0.66	0.65

Table 5.24: Test set results SemEval, Ensemble of ensembles, unconstrained

As demonstrated above, combining several models, and applying different methods of choosing y, allows for a gain in results, especially if soft classification is applied.
In order to determine which model is the best, a table summarising the results of all the models analysed is provided:

	Accuracy	**F1 Score**	**Precision**	**Sensitivity**
BMA	**0.62**	0.60	0.63	0.64
DeepMoji	0.60	0.59	0.62	0.62
RCNN-Roberta	0.59	0.58	0.61	0.61
BERTweet Features-based	0.53	0.61	0.60	**0.66**
DeepMoji Features-based	0.59	0.62	0.63	0.64
Ensemble of Ensembles (soft)	0.61	**0.63**	**0.66**	0.65
Ensemble of Ensembles (hard)	0.58	0.62	0.65	0.63

Table 5.25: Summary SemEval task 3A test set results, Irony unconstrained

In general, it can be concluded that the Ensemble of ensembles model, with soft classification methodology, is the one that provided the best results in terms of F1 Score, precision and sensitivity, except for accuracy, where the BMA model reaches one percentage point higher. Since the SemEval competition takes into account the F1 Score as an evaluation metric, the Ensemble of ensembles is used as a comparison to the official ranking of the unconstrained case.

5.2.2 Constrained Task

SemEval offers two rankings based on which type of dataset is used for the training phase. In the previous section, the unconstrained dataset is analysed, while in this paragraph only the competition training set is used. This choice is justified for two reasons: the first is to analyse the in-domain out-domain case, while the second is to compare the developed models with those that participated in the SemEval competition, based on the constrained and unconstrained cases.

5.2.2.1 The Role of Expressive Signals and Ensemble Classifiers

Unlike the previous case, the optimisation objective function of each model used is accuracy. In this case, using only the training data, the ground truth classes are balanced. This is the only difference from the previous case, everything else remains unchanged. The two optimization methods are compared in order to find the best hyper-parameters for each model based on the several features combinations: The Random Search and Bayes Search approaches are shown below:

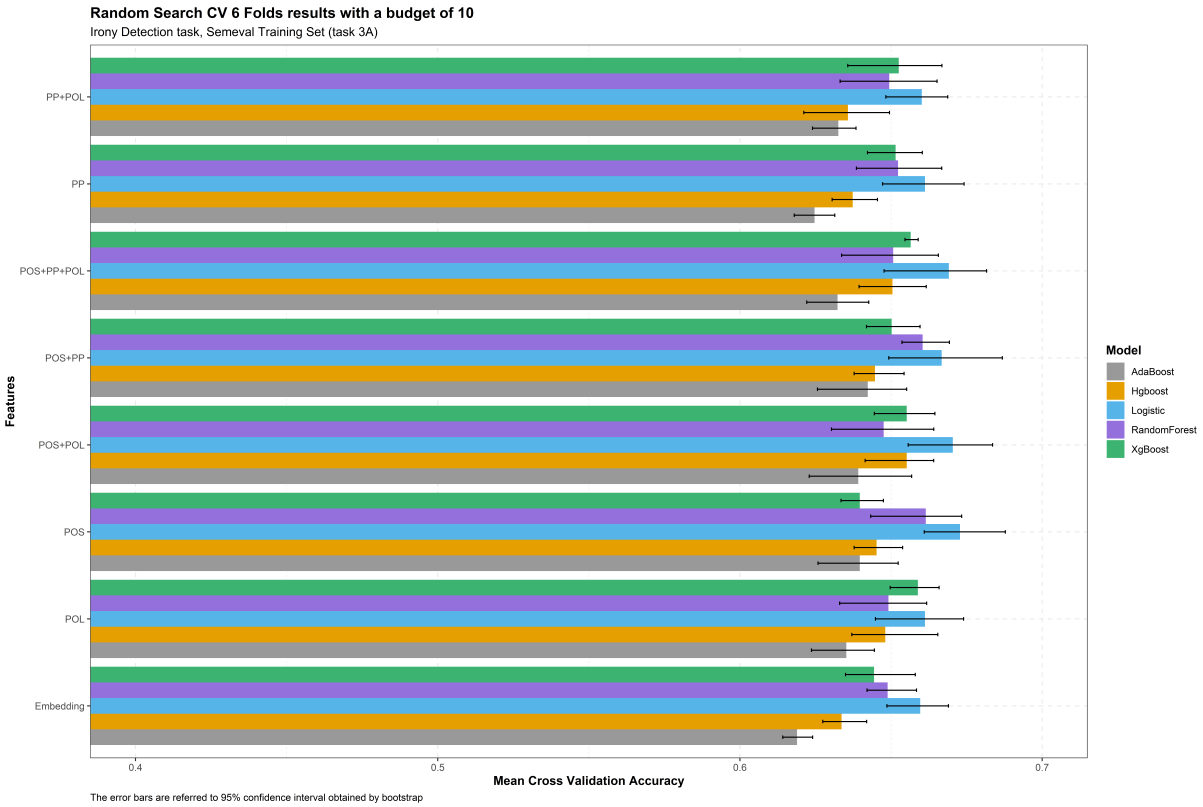

Figure 5.14: Random Search for Irony, constrained task

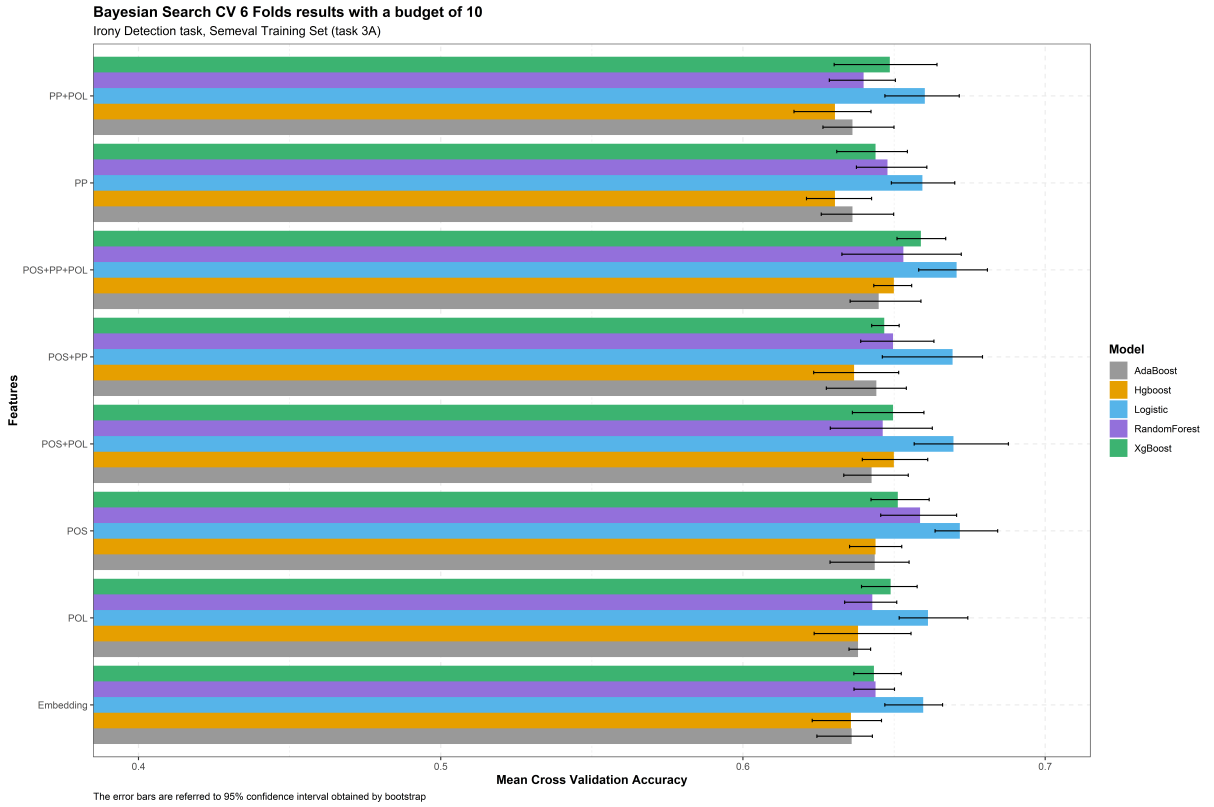

Figure 5.15: Bayes Search for Irony, constrained task

As in the unconstrained case, a different phenomenon, with respect to sarcasm, emerges from the optimisation analysis: by using the different features, the models do not have a significant gain in performance. Just observe the overlapping confidence intervals (with an alpha of 0.05). Even between the same optimisation criteria, there is no significant difference, so Random Search is chosen as the average accuracy is slightly better than that obtained by Bayes Search, especially for the combination embeddings + POS.

- **Logistic Regression**: C = 0.89, Penalty = L_2, Solver = *Saga*, in this case the regularization term tends to shrinkage the coefficient to zero.
- **X Gradient Boosting**: Number of estimators = 109, Learning rate = 0.06, Depth = 9, Subsample = 0.83
- **Ada Boost**: Number of estimators = 72, Learning rate = 0.68
- **Hist Gradient Boost**: Depth = 14, Minimum samples leaf = 13, Learning rate = 0.11
- **Random Forest**: Number of estimators = 51, Depth = 15, Minimum samples leaf = 9, Minimum samples split = 16

The second phase concerns the development of the BMA ensemble method: the best

configuration identified in this optimisation phase are X Gradient Boost, Hist Gradient Boost, Logistic regression and Ada Boost (with an accuracy of 0.72).

	Accuracy	**F1 Score**	**Precision**	**Sensitivity**
Logistic Regression	0.67	0.62	0.67	0.66
X Gradient Boost	0.67	0.62	0.67	0.67
Ada Boost	0.67	0.62	0.67	0.66
Hist Gradient Boost	0.66	0.60	0.66	0.66
Random Forest	0.58	0.50	0.57	0.57
BMA	**0.69**	**0.63**	**0.68**	**0.69**

Table 5.26: SemEval task 3A Test Set Results BMA, constrained

From table 5.26 the model that outperforms the others is the BMA ensemble method. The thing to highlight is to see how the models, trained only on the in-domain dataset, achieve much better results than those previously developed with a larger training set. The interesting thing is that the results are in line with the confidence intervals estimated in the optimisation phase, also justified by the fact that the training distribution is more homogeneous with the test distribution.

5.2.2.2 DeepMoji for detecting sentiment, emotion and sarcasm

The only difference with the DeepMoji model trained on the contrained case concerns the number of epochs which is fixed at 3 for each layer of the model (recalling the chain-thaw methodology), for a total of 18 epochs. The choice is justified by the fact that if we choose 4 as the number of iterations for each layer to train, the model starts to overfit significantly.

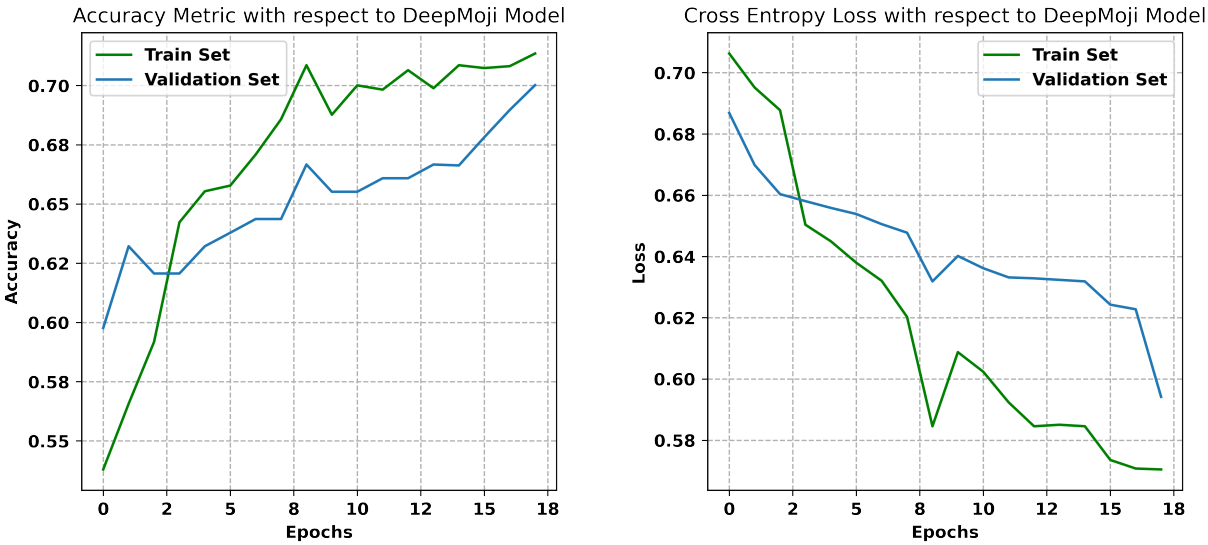

Figure 5.16: Training phase DeepMoji model, Irony constrained

The metrics obtained during the training phase are as follows:

	Accuracy	F1 Score	Precision	Sensitivity
Training Set	0.74	0.70	0.75	0.75
Validation Set	0.72	0.67	0.74	0.73

Table 5.27: Training and Validation set DeepMoji model, Irony constrained

While for the test set task 3A, the results are reported below:

	Accuracy	F1 Score	Precision	Sensitivity
SemEval Test set 3A	0.71	0.64	0.70	0.70

Table 5.28: SemEval test set DeepMoji Features-based, Irony unconstrained

As seen in Machine Learning, using only the in-domain dataset in the training phase, produces representative estimates between the validation and test set. Indeed, the results of the test set are similar to those of the validation, which is different from the unconstrained case (since the difference reaches an average of 30 percentage points).

5.2.2.3 A Transformer-based approach to Irony and Sarcasm detection

The RCNN-Roberta model for the costrained case is based on the same experimental settings as the unconstrained case, except for the number of epochs: with early stopping

the model terminates the epochs after 65 epochs.

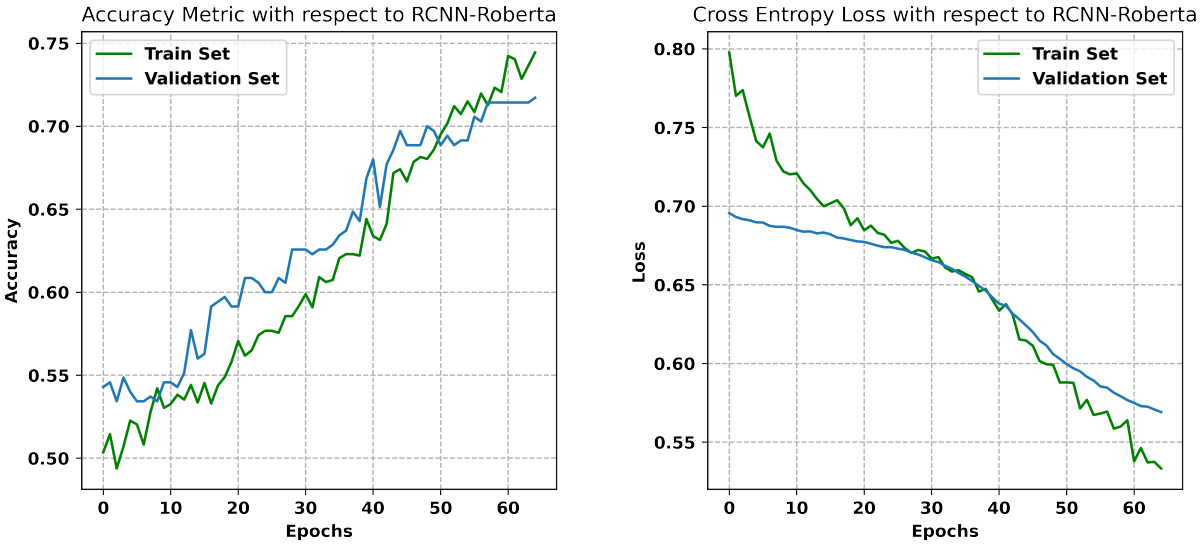

Figure 5.17: Training phase RCNN-Roberta model, Irony constrained

The RCNN-Roberta model obtains the following metrics between training and validation:

	Accuracy	F1 Score	Precision	Sensitivity
Training Set	0.74	0.71	0.76	0.75
Validation Set	0.72	0.70	0.71	0.72

Table 5.29: Training and Validation set RCNN-Roberta model, Irony constrained

The model performs satisfactorily in the test set, where performances is, on average, higher than the other classifiers:

	Accuracy	F1 Score	Precision	Sensitivity
SemEval Test set 3A	0.74	0.72	0.76	0.75

Table 5.30: SemEval test set RCNN-Roberta, Irony constrained

In the RCNN-Roberta paper, the researchers claim to achieve even higher performance, such as the F1 Score of 0.80, on the SemEval test set, which can be justified by the fact that the replication conditions of the model are not exactly the same, due to lack of information.

5.2.2.4 Proposed Methods

The experimental settings of BERTweet Features-based and DeepMoji Features-based are very similar to the previous task. Specifically, for BERTweet Features-based there are a few changes made for the training phase: batch size is equal to 64, the optimiser is the Adam with a weight decay set to 0.01, learning rate equal to 0.00002 and the number of epochs set to 85 (early stopping technique).

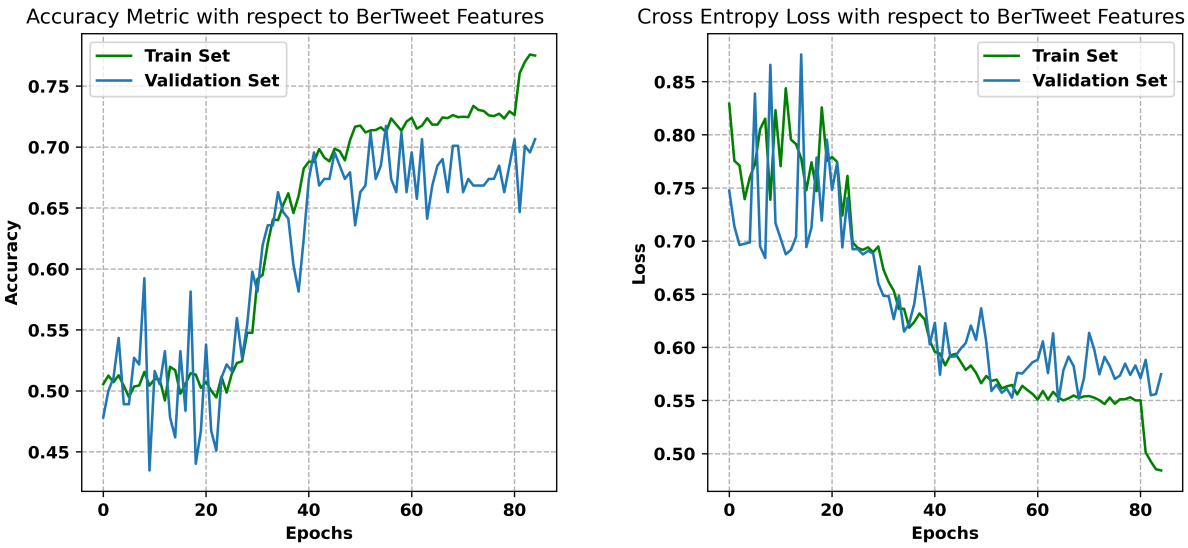

Figure 5.18: Training phase BERTweet Features-based, Irony constrained

The metrics for the training and validation set of BERTweet Features-based are the following:

	Accuracy	F1 Score	Precision	Sensitivity
Training Set	0.76	0.73	0.77	0.76
Validation Set	0.71	0.69	0.70	0.71

Table 5.31: Training and Validation set BERTweet Features-based, Irony constrained

The results obtained in the test are slightly below the validation set, maintaining a sufficient power of generalisation:

	Accuracy	F1 Score	Precision	Sensitivity
SemEval Test set 3A	0.71	0.65	0.71	0.70

Table 5.32: SemEval test set BERTweet Features-based, Irony constrained

As for the DeepMoji Features-based model, the changes made compared to the experimental settings of the unconstrained task are as follows: learning rate equal to 0.0001, weight decay set to 0.01 based on Adam optimizer, and the number of epochs equal to 200:

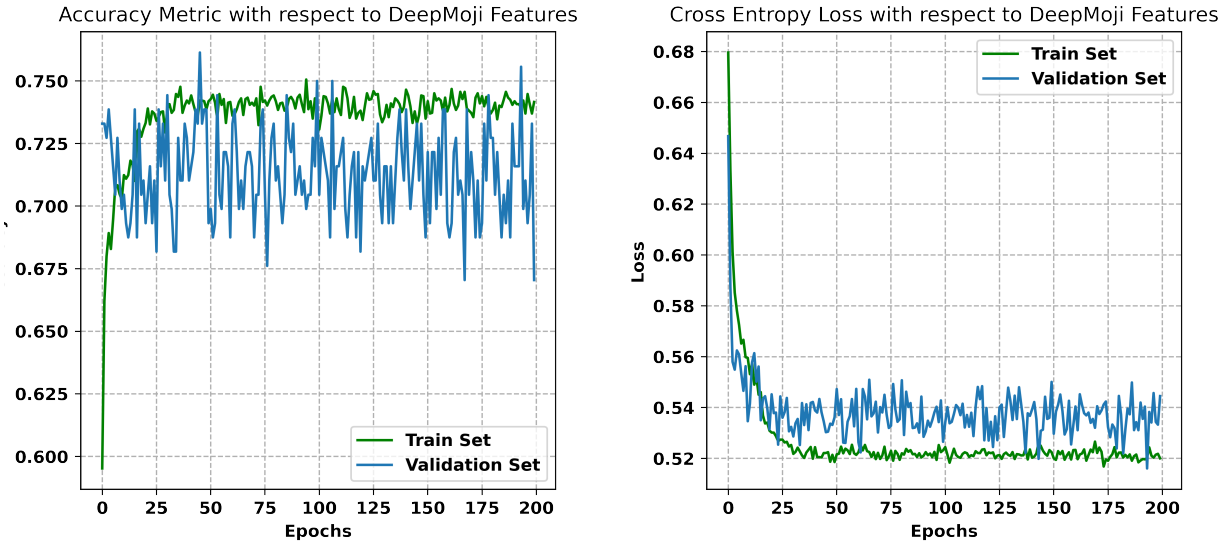

Figure 5.19: Training phase DeepMoji Features-based, Irony constrained

Compared to the previous model, the measurements with respect to the train and the validation set are slightly less stable, but there is still a good trade-off:

	Accuracy	F1 Score	Precision	Sensitivity
Training Set	0.74	0.73	0.74	0.75
Validation Set	0.67	0.66	0.67	0.67

Table 5.33: Training and Validation set DeepMoji Features-based, Irony constrained

While on the test set of SemEval task 3A the results obtained are shown in the table below:

	Accuracy	F1 Score	Precision	Sensitivity
SemEval Test set 3A	0.70	0.63	0.69	0.69

Table 5.34: SemEval test set DeepMoji Features-based, Irony constrained

As it continues to emerge, these models trained only on the in-domain set, despite the much smaller sample size, provide more robust results than models trained with several sources.

Finally, the last methodology Ensemble of ensembles combines these last two models together with the BMA based on four Machine Learning models: for the corresponding methodologies, soft and hard classification, the results on the test set are the following:

	Accuracy	F1 Score	Precision	Sensitivity
Hard Classification	0.73	0.67	0.72	0.72
Soft Classification	0.74	0.69	0.73	0.74

Table 5.35: Test set results SemEval, Ensemble of ensembles, constrained

The table that summarise all the performance of each developed model is reported below:

	Accuracy	F1 Score	Precision	Sensitivity
BMA	0.69	0.63	0.68	0.66
DeepMoji	0.71	0.64	0.70	0.70
RCNN-Roberta	**0.74**	**0.72**	**0.76**	**0.75**
BERTweet Features-based	0.71	0.65	0.71	0.70
DeepMoji Features-based	0.70	0.63	0.69	0.69
Ensemble of Ensembles (soft)	**0.74**	0.69	0.73	0.74
Ensemble of Ensembles (hard)	0.73	0.67	0.72	0.72

Table 5.36: Summary SemEval task 3A test set results, Irony constrained

The model that outperforms, especially for the F1 Score is the RCNN-Roberta model, this also confirms what is stated in the model paper as the state-of-the-art on this task. It should be emphasised that the Emsemble of ensemble model also achieves good results, for example, an accuracy equal to that of RCNN-Roberta. For this reason, in the next paragraph the best 5 models of the competition are shown, both in the constrained and in the unconstrained case, together with the last proposed methodology Ensemble of ensembles.

5.2.3 SemEval 2018 Competition Ranking Task 3A

To provide an index of comparison between the latest developed model and the other classifiers developed by the competition teams, a table with the top 5 highest performing models is provided. In order to provide as much detail as possible, various metrics are described, but the reference one is the F1 score calculated on the positive class (the order in the tables depends on the value obtained in F1 Score) [35].

	Accuracy	Precision	Sensitivity	$\mathbf{F}_1$
UCDCC	**0.797**	**0.788**	0.669	**0.724**
THUNGN	0.735	0.630	0.801	0.705
Ensemble of Ensembles (soft)	0.693	0.681	0.692	0.690
NTUA-SLP	0.732	0.654	0.691	0.672
WLV	0.643	0.532	**0.836**	0.650

Table 5.37: Ranking SemEval Task 3A, constrained

As for the proposed model, in the constrained case, based on F1 Score, it ranks third. The metric where it outperforms the others is sensitivity, while the team that ranks first is UDCC. Furthermore for the unconstrained case:

	Accuracy	Precision	Sensitivity	$\mathbf{F}_1$
Ensemble of Ensembles (soft)	0.612	**0.661**	0.653	**0.631**
NonDicevo-SulSerio	**0.679**	0.583	0.666	0.622
INAOE-UPV	0.651	0.546	0.714	0.618
RM@IT	0.649	0.544	0.714	0.618
ValenTO	0.598	0.496	**0.781**	0.607

Table 5.38: Ranking SemEval Task 3A, unconstrained

This time the results are much better, because the proposed model outperforms the other classifiers on the F1 Score by ranking first. To summarise the Ensemble of ensembles, due to its combination of models, which are based on different features, would seem to have a better power of generalisation to the models developed by the other teams (referring to the unconstrained case). While in the other case, the results remain valid but not the best. The thing that emerges, and that is underlined in the next chapter, is that all the models, both those developed in this thesis and those of the competition, suffer a deterioration in performance when taking into account the in-domain set and the out-domain set. This suggests that the models do not seem to be robust when different types of ironies are used, as they cannot capture a generic pattern.

Chapter 6

Conclusions

After an in-depth study of the different techniques used to recognise irony and sarcasm, several insights emerged. Specifically, comparing the techniques of Machine Learning with those of Deep Learning, it has emerged that, for the case of sarcasm, the models that have provided the best performance of generalization are the former: it would seem that, thanks to the techniques of features engineering, the models of Machine Learning are more stable in the case of out-domain. In addition to this, some combinations of features such as POS, pragmatic particles, sentiment (plus baseline embeddings reduced in size) allow a significant gain in performance measures. Among other things, the BMA model outperforms all other models in the case of the out-domain dataset, while for the in-domain it ranks second, after the ensemble of ensembles, which in turn exploits the BMA. What needs to be highlighted is the fact that the models analysed are not able to generalise well if automatic feature extraction techniques are used, as is the case in Deep Learning. Explicit information, such as POS, emoji count, punctuation, etc., help simpler models, such as Machine Learning, to identify a more significant pattern for sarcasm. As far as emotional embeddings are concerned, they are on average slightly more discriminative than textual embeddings. In fact, for the case of sarcasm for both the in-domain and the out-domain test set, the model based on DeepMoji features obtains on average higher metrics than the model that exploits BERTweet embeddings. While for irony, only in the unconstrained case it seems better than the latter.

In terms of irony, the different features used for Machine Learning classifiers do not show a significant difference in the optimised metrics. This suggests that, with respect to

irony, the combinations of the various features do not appear to have a significant impact as seen above: for a 95% confidence interval, it is concluded that the variables do not describe the patterns of irony differently. To analyse the different models, in the case of irony, the out-domain dataset is used only for the training part, and the test set consists only of the in-domain. When comparing the various classifiers, both in the comparative study and in the SemEval ranking (unconstrained case), as the sample size increases, the performance of the models under test deteriorates significantly. This shows that the structures of these models are not able to identify general information related to irony, but only focus on small aspects. Suffice it to say, that one of the training datasets used for irony, presents a subdivision of tweets by topics, where the political one is composed of ironic tweets. The inclusion of this bias highlights the fact that using only these features is not efficient in producing a good model for recognising irony. The BMA and Ensemble of Ensembles models are the most robust for the case of sarcasm, both for the in-domain and the out-domain test set. The study shows that certain models claimed to be state-of-the-art, actually are only when using the in-domain dataset, where they are trained on specific distributions. Just for irony, in the constrained task, the only classifier that is confirmed as state-of-the-art turns out to be RCNN-Roberta, as stated in the paper [26].

For what concerns the model based on exploiting the different encoder output layers, a new way of understanding how to combine the various embeddings (in order to obtain as much information as possible on different feature levels) opens up. However, this approach has been limited to sentence embeddings, losing the granularity of word embeddings, due to the limitation of computational resources. The development of a methodology to flexibly combine the outputs of encoders lays the foundation for exploiting different flexible approaches to generate more representative word embeddings. Indeed, the future development of this work is to use the word embeddings and to deepen the study on all 12 outputs layers.

What can be said is that the joint use of different models, which are based on different aspects of the text, allows to better capture the aspects of the figurative language. However, the analysis shows that the quality of the data and the available information, has limitations in terms of understanding what is sarcastic and what is ironic. To better explain this concept, we can consider the field of image recognition. When it comes to latter, it is important to generate a good representative training set that is robust to

variation. If a model is trained on dark images only, the classifier is not able to generalize well to light images. This is also true for text, where light tones, this time, do not depend on the text itself, but on the writer. The subjective element is one of the keys to solve this problem and, exploiting only part of the information available, the models will never be able to generalise over different distributions.

From what stated above, it is possible to conclude that the use of models based on self attention layers and ensemble methods such as BERTweet Features-based, DeepMoji Features-based and Ensemble of ensembles, produced results that can be declared satisfactory. As a matter of fact, it is sufficient to look at the rankings of the SemEval competition, where in one case third place is reached 5.37, and in the other first place 5.38.

Bibliography

[1] Carlos A.Iglesias Fernando Sánchez-Rada. *Social context in sentiment analysis: Formal definition, overview of current trends and framework for comparison*. 2019. URL: https://www.sciencedirect.com/science/article/pii/S1566253518308704.

[2] Ronaldo Cristiano Prati Leila Weitzel and Raul Freire Aguiar. "The Comprehension of Figurative Language:What Is the Influence of Irony and Sarcasmon NLP Techniques?" In: *Springer International Publishing Switzerland* (2016). DOI: https://www.researchgate.net/publication/299470588_The_Comprehension_of_Figurative_Language_What_Is_the_Influence_of_Irony_and_Sarcasm_on_NLP_Techniques.

[3] Davide Buscaldi Antonio Reyes Paolo Rosso. "From humor recognition to irony detection: The figurative languageof social media". In: *Elsevier* (2012). DOI: https://www.academia.edu/3032533/From_humor_recognition_to_irony_detection_The_figurative_language_of_social_media.

[4] Moriceau Karoui Benamara. *Automatic Detection of Irony*. Wiley, 2019.

[5] R A Martin. "The Psychology of Humor: An Integrative Approach". In: *Elsevier Academic Press* (2007).

[6] Viviana Patti Mirko Lai Alessandra Teresa Cignarella Cristina Bosco. "Application and Analysis of a Multi-layered Scheme for Irony on the Italian Twitter Corpus TWITTIRO". In: (2018).

[7] Jiawei Hu. *An Overview for Text Representations in NLP*. URL: an-overview-for-text-representations-in-nlp.

[8] Brendan O'Connor Dipanjan Das Daniel Mills Jacob Eisenstein Michael Heilman Dani Yogatama Jeffrey Flanigan Kevin Gimpel Nathan Schneider and Noah A. Smith. "Part-of-Speech Tagging for Twitter: Annotation, Features, and Experiments". In: (2013).

[9] MARK J CARMAN ADITYA JOSHI PUSHPAK BHATTACHARYYA. "Automatic Sarcasm Detection: A Survey". In: (2016).

[10] Enza Messina Elisabetta Fersini Federico Alberto Pozzi. "Detecting Irony and Sarcasm in Microblogs: The Role of Expressive Signals and Ensemble Classifiers". In: (2015).

[11] Pete Keen et al. *TextBlob: Simplified Text Processing*. URL: `https://textblob.readthedocs.io/en/dev/`.

[12] Dr. Stefanos Zafeiriou. "Notes on Implementation of Component Analysis Techniques". In: (2015).

[13] Trevor Hastie Jerome H. Friedman Robert Tibshirani. *Elements of Statistical Learning*. Springer, 2001.

[14] Jason Brownlee. *A Gentle Introduction to the Gradient Boosting Algorithm for Machine Learning*. URL: `/machinelearningmastery.com/gentle-introduction-gradient-boosting-algorithm-machine-learning/` (visited on 09/09/2016).

[15] Hui Zou Trevor Hastie Ji Zhu Saharon Rosset. "Multi-class AdaBoost". In: (2006).

[16] Joaquin Vanschoren Frank Hutter Lars Kotthoff. *Automated Machine Learning, Methods, Systems Challenges*. Springer, 2019.

[17] C K Williams and Rasmussen. *Gaussian processes for machine learning*. MIT Press, 2006.

[18] Kamalika Some. *THE HISTORY, EVOLUTION AND GROWTH OF DEEP LEARNING*. URL: `https://www.analyticsinsight.net/the-history-evolution-and-growth-of-deep-learning/`.

[19] Renu Khandelwal. *A Gentle Introduction to the Gradient Boosting Algorithm for Machine Learning*. URL: `https://medium.com/datadriveninvestor/overview-of-different-optimizers-for-neural-networks-e0ed119440c3` (visited on 02/03/2019).

[20] Aoron Courville Ian Goodfellow Yoshua Bengio. *Deep Learning*. MIT Press, 2017.

[21] *Transfer learning*. URL: https://en.wikipedia.org/wiki/Transfer_learning#cite_note-Lin,_Jung_2017-15 (visited on 01/14/2021).

[22] Ashihs Vaswani et al. "Attention Is All You Need". In: (2017).

[23] Jacob Devlin Ming-Wei Chang Kenton Lee Kristina Toutanova. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: (2019).

[24] Yinhan Liu et al. "RoBERTa: A Robustly Optimized BERT Pretraining Approach". In: (2019).

[25] Anders Søgaard Iyad Rahwan Sune Lehmann Bjarke Felbo Alan Mislove. "Using millions of emoji occurrences to learn any-domain representations for detecting sentiment, emotion and sarcasm". In: (2017).

[26] Andreas Georgios Stafylopatis Rolandos Alexandros Potamias Georgios Siolas. "A Transformer-based approach to Irony and Sarcasm detection". In: (2019).

[27] Dat Quoc Nguyen, Thanh Vu, and Anh Tuan Nguyen. "BerTweet: A pre-trained language model for English Tweets". In: (Oct. 2020), pp. 9–14. DOI: 10.18653/v1/2020.emnlp-demos.2. URL: https://www.aclweb.org/anthology/2020.emnlp-demos.2.

[28] Jay Alammar. *The Illustrated BERT, ELMo, and co. (How NLP Cracked Transfer Learning)*. 2018. URL: http://jalammar.github.io/illustrated-bert/.

[29] Hai Ye et al. *Feature Adaptation of Pre-Trained Language Models across Languages and Domains with Robust Self-Training*. 2020.

[30] Ivan Habernal Tomas Ptacek and Jun Hong. "Sarcasm Detection on Czech and English Twitter". In: (2014). DOI: https://www.aclweb.org/anthology/C14-1022.pdf.

[31] T. Veale A. Ghosh. "Fracking Sarcasm using Neural Network". In: *Proceedings of NAACL-HLT* (2016).

[32] Els Lefever Cynthia Van Hee and Veronique Host. "SemEval-2018 Task 3: Irony Detection in English Tweets". In: (2018).

[33] P. Rosso A. Reyes and T. Veale. *A multidimensional approach for detecting irony in twitter*. 2013.

[34] Prafulla Surve Lalindra De Silva Nathan Gilbert Ruihong Huang Ellen Riloff Ashequl Qadir. "Sarcasm as Contrast between a Positive Sentiment and Negative Situation". In: (2013).

[35] Els Lefever Cynthia Van Hee and Veronique Hoste. "SemEval-2018 Task 3: Irony Detection in English Tweets". In: (2018).