UNIVERSITAT DE BARCELONA

Bachelor's Thesis

Bachelor's Degree In Computer Engineering

Faculty of Mathematics and Computer Science
Universitat de Barcelona

Research and Analysis of Hate and Other Emotions in Social Media

Author: María Isabel González Sánchez

Director: Dra. María Salamó Llorente
Written in: Department of Mathematics and Computer Science
Faculty of Mathematics and Computer Science

Barcelona, June 12, 2022

Contents

Research and Analysis of Hate and Other Emotions in Social Media

Detection, analysis and research on Hate Speech, Offensiveness and numerous Emotions for certain languages in social networks.

Abstract

In the course of just a few years, with the massive introduction of social media, people have changed the way they communicate and share experiences dramatically. The global scale that this topic has reached, combined with its rapid expansion, is a historic landmark. However, what do social networks represent in our day-to-day lifestyles? The answer is a double life. Since their launch, a digital pseudo-reality has been created in which thoughts, emotions and privacy can be expressed in detail. This leads us to dump each of society's concerns into community applications, and if you add the factor of anonymity behind a screen, the result is incendiary.

Through this work, it is intended to identify, study and analyze the high level of emotions, mostly negative, that has been flooding social media thanks to the aforementioned anonymity. This process will be carried out by entering the Natural Language Processing field. For this purpose, a study of Hate Speech, Toxicity, Offensiveness and other emotions will be carried out on four datasets, each one with one of these tasks respectively. Using these datasets, three language models, based on Transformers and Deep Learning, will be trained and validated for their future comparison.

All of this is performed with the aim of finding the ideal framework for each of the featured tasks, which are based on true-to-life situations. Furthermore, it is intended to find the causes of the inconveniences that the models may present, in a concise and intuitive way for the reader.

Estudio y Análisis del Odio y demás Emociones en las Redes Sociales

Detección, análisis e investigación del lenguaje de odio, de la ofensividad y de varias emocionesés para ciertos idiomas en las Redes Sociales.

Resumen

En tan solo unos pocos años, la vida cotidiana como la entendemos ha sufrido un cambio radical con la llegada masiva de las redes sociales. La globalidad que han alcanzado duchas redes sociales, junto con su rápida expansión, es un hito histórico. Sin embargo, ¿qué suponen las redes sociales en nuestra vida diaria? La respuesta es una doble vida. Desde su introducción, se ha creado una pseudo-realidad digital en la que poder expresar nuestros pensamientos, nuestras emociones y nuestra privacidad en detalle. Esto nos lleva a volcar cada uno de los problemas de la sociedad en aplicaciones comunitarias y, si le juntas el factor de anonimidad tras una pantalla, el resultado es incendiario.

Con este trabajo de finde grado, lo que se pretende es detectar, estudiar y analizar el alto nivel de emociones, en su mayoría negativas, que ha inundado las redes sociales gracias a la ya mencionada anonimidad. Este proceso se llevará a cabo introduciéndonos en el mundo del Procesamiento de Lenguaje Natural. Para ello, se desarrollará un estudio del Lenguaje de odio, la Toxicidad, la Ofensividad y otras emociones en cuatro datasets, cada uno con una de estas tareas respectivamente. Con estos datasets se entrenarán y validarán tres modelos de lenguaje, cuya base son *Transformers* y *Deep Learning*, para su futura comparación.

Todo ello se realizá con el fin de encontrar el framework idóneo para cada una de las tareas presentadas, que están basadas en situaciones reales. Además, se analizarán las causas de los inconvenietes que presenten los modelos, en una forma concisa e intuitiva para el lector.

Chapter 1

Introduction

This section of the Bachelor's thesis will serve as a prologue to the rest of the project. It will begin with a brief description of the reasons for pursuing this topic, followed by a statement of the problem to be dealt with. Finally, the objectives to be achieved and the general organization of the document will be discussed.

1.1 Motivation: Literature and Computer Science can go hand in hand

For as long as I can remember, I have always been a bookworm. A child with one or two books in her backpack, who read in class instead of paying attention to teachers and preferred to get lost in fantasy realms and adventures rather than focus on real life. Eventually that girl decided to travel the world and pursued a career that was the complete opposite of her hobbies: a Double Degree in Computer Engineering and Mathematics.

Although one of these careers was put aside, what never disappeared was the love for reading. At this final point in her university life, unifying her two passions, reading and computer science, was not a terrible idea: that's how she came up with the option of getting into Natural Language Processing. Reading comprehension, cohesion and understanding of emotions and words was still present, but with a new twist. Now, the only thing left to do is to get lost in a new world of *Deep Learning*, *Transformers* and *Linguistics*.

1.2 Problem to be solved: fast and accurate detection of Hate and other negative Emotions

Nowadays, few people remember what life was like without uploading a story on Instagram or not looking at tiktoks when you are bored. That is how immersed today's society is in social media, as if we have copied and pasted our existence onto our devices and life goes on there. Therefore, all the real social experiences and problems have been moved to a new battlefield with a brand new factor to take into account: the possibility of

maintaining your anonymity by being behind a screen. As a consequence, our whole life is exposed to public view and emotions become more intense than in real life. Emotions as primary as joy, anger or disgust are magnified in social networks, leading to cases of lack of respect, harassment or even public trials.

Given the convulsive times we have been living in recent years, social media has become a hotbed of polarized emotions and the cradle of indiscriminate attacks against certain groups. Massive social movements such as #MeToo or #BlackLivesMatter, alongside historical events such as COVID-19, war and global instability make detecting hate speech, offensiveness and other negative emotions in posts vital [51]. Although public collaboration for reporting and, subsequently, removing these posts is a valuable help, it is not enough. This is where Natural Language Processing comes in.

To avoid misinformation and further polarization of such raw topics as the current ones, linguistics and Artificial Intelligence must gather forces to come up with algorithms capable of classifying such posts. However, it is not as simple as it seems. Reading comprehension plays a significant role, as not all posts with offensive language, such as swear words, are made with hatred and not all polite and proper posts have the best intentions. So, will human ingenuity and Data Science be enough to overcome this great challenge? The answer is yes.

1.3 Bachelor's Thesis Objectives

Throughout this work and as all the necessary concepts being exposed, a series of key objectives will be fulfilled. These goals will be the target of each one of the chapters of this bachelor's thesis. For this purpose, the following topics will be discussed in order of their importance and appearance in the project.

First of all, the **main objective** of the entire study, as the title of this thesis indicates, will be the investigation and analysis of several emotions in social networks. In other words, a study of emotions such as hate, joy or disgust will be carried out through 4 datasets of similar themes. On the one hand, we will look at Hate Speech for observing hatred in social media. Then, toxicity and offensiveness will be analyzed with the next two datasets and, finally, the fourth dataset will be used for studying several emotions at the same time depending on the language in which they are expressed.

However, this macro objective can be subdivided into small achievable ***milestones*** that can be reached chapter by chapter. Since this goal will be carried out using state-of-the-art NLP models, we will start with a straightforward **investigation** of these models, starting from the basics of the field.

Once the research part is over, the next milestone will be the understanding of the four available **datasets**. The idea is to carry out a statistical analysis alongside the development of a framework for pre-processing them for the aforementioned state-of-the-art models.

From this implementation, results and conclusions will be drawn which will require a detailed study of their behavior. Therefore, several milestones will come out of this section. The first one will be the massive **collection of training and evaluation results** of the chosen NLP models with each dataset, focusing on specific metrics and loss functions. Then, using all this data, an analysis of **advantages, inconveniences and problems** of each model regarding each dataset will be performed. Therefore, the last objective will be a **comparison** among models according to their performance using each dataset.

Finally, the last remaining milestone will be a combination of several previous goals. In other words, a small analysis of how the use of **transfer learning** affects the training and evaluation of the models will be carried out. This objective will entail a study and investigation of the **benefits, drawbacks and issues** that each model presents when trained with one dataset and evaluated with a different but similar thematic dataset. Furthermore, a **comparison** will be made to decide which model has the best performance.

1.4 Project Organization

This thesis will be subdivided into five parts, being this Introduction the first of them (1). Here, as can be seen above, the reasons for choosing this field of study as the subject of the research and the target problems have been explained. Its main purpose has been to start the report in the most bearable way possible and to introduce the reader to the subject.

If we continue to the following point of the project, we will enter the State-Of-The-Art (SOTA) section (2). As the name suggests, this section will discuss how to deal with the problems of hatred and polarization in social media with the leading-edge models available in the industry [32][69]. To do so, a brief contextualization will be made along with a general review of all types of techniques, making an incremental sweep until reaching the SOTA models themselves. Reading and understanding current papers will be crucial as well as making a detailed summary of their content.

Once this exhaustive analysis of latest Natural Language Processing (NLP) methodology is done, it will be time to put into practice what has been learned and summarized. Therefore, in the Implementation chapter (3), the reader will be taken to a detailed explanation of how to analyze our Datasets, built with real data, and how to train the chosen SOTA models. Therefore, the framework for retrieving analyses on hate and other emotions will be described.

Also, it is important to develop some techniques for their pre-processing, together with obtaining optimal parameters for training NLP models.

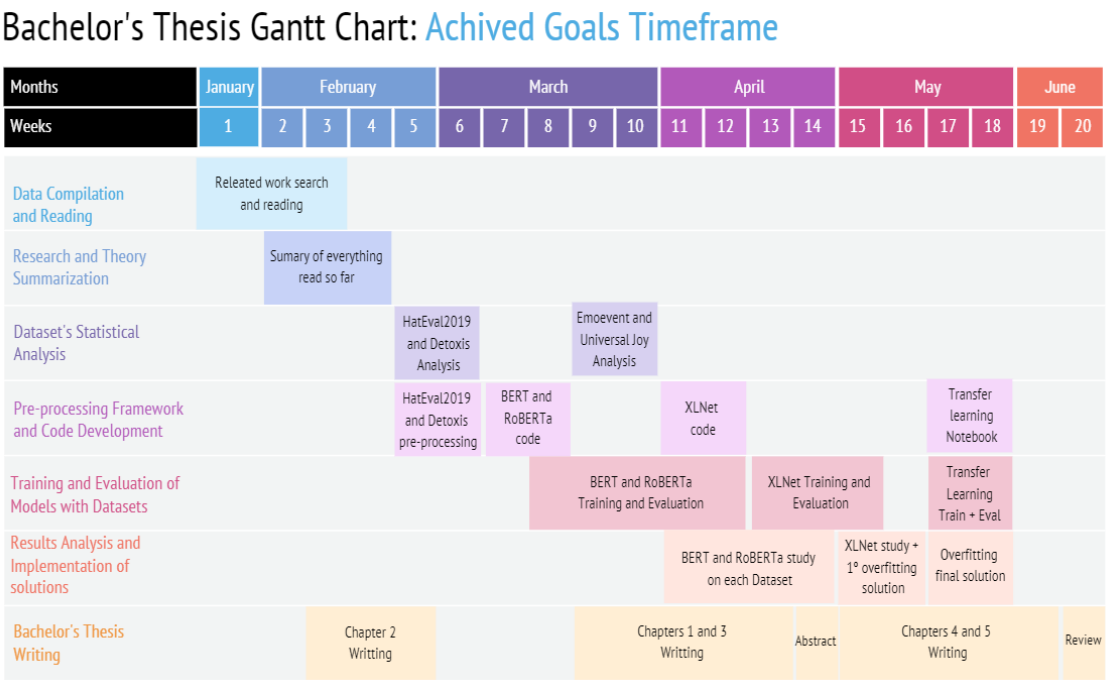
Consequently, the fourth chapter will consist of all the results (4) that we will gather from the different training sessions of the models. This Results section will aim to study the influence of certain preprocessing techniques and the chosen models on the resulting output, evaluated with several metrics.

As it is to be expected, comparisons will be made among techniques and models in order to find the optimal framework for the different tasks. Furthermore, it will be taken into considerations all the problematics found on the way, their causes and how could some of them be solved. By the end of this chapter, some transfer learning will be discussed together with what its output reveals. The aim is to study the effect of training the different models with one of the four available datasets, while evaluating them with one of the other three.

Finally, the last chapter will consist of the conclusions to the entire thesis (5). In this part there will be an observation of what has been learned during the course of the thesis, as well as an evaluation of the objectives achieved. Without further delay, let's start with the thesis.

1.5 Project Schedule

Last but nor least, I would like to detail how these almost 6 months of work have gone by in a visual way. For this purpose, the following Gantt chart will be used, where for each month, the week of work it contains is highlighted. In addition, you can see on the left side the different goals to be achieved during these weeks and how they have been subdivided into small milestones, expressed with text squares of the same color as the major achievement.



Chapter 2

State of the Art in Natural Language Processing

This chapter will explain the basic concepts for understanding the project as a whole, along with a brief introduction to the techniques that will be used for NLP data processing. However, the bulk of the chapter will focus on the comparison and study of current models and algorithms in this field.

2.1 Contextualization: Natural Language Processing through Social Media

Natural language processing (NLP) is a branch of Artificial Intelligence that helps machines understand, process and manipulate everyday language. As we can guess, NLP draws inspiration from disciplines such as Data Science and Computational Linguistics in its quest to bridge the gap between ordinary and computational language.

Therefore, if our mission is to study Hate Speech, Offensiveness and other Emotions in social media posts, where natural language is used, this field is the right one. This project will focus on the detection, analysis and study of several tasks, mainly distributed in four parts. These tasks will be:

1. ***Hate Speech* (HS) study, alongside *Aggressiveness* (AG) and *Target Rate* (TR).**
   These last two will be analyzed in function of the first one. In other words, if Hate Speech is detected in a publication, its level of Aggressiveness and its Target Rate will also be analyzed.

2. ***Toxicity* study together its *toxicity level*** Society and social media have an undeniable toxic relationship, as stated above. Therefore, through the provided real data, there is a solid opportunity to comprehend how much toxicity may be exhibited in theses networks and at what level it is.

3. ***Offensiveness* study through several emotions** As Hate Speech, offensiveness can be found in all kind of posts, independently of whether they show happiness or hatred or whether they are polite or they are disrespectful. Therefore, it is necessary to obtain a proper output, without the interference of emotions, like joy, fear or disgust.

4. ***Emotions* detection in several languages** Not all cultures show or express their emotions equally. Hence, their writing style may cause some misleading inputs which need to be taken into account. In other words, there is a clear need to analyze the impact of various emotions depending on the language in which they are expressed.

As previously mentioned, negative emotions on social networks are suffering an almost exponential increase over the years. Therefore, its investigation is crucial to tackle this issue and, subsequently, have more civilized communities for all audiences. So, this will become our main problem for the Bachelor's thesis: to investigate the hatred, toxicity and offensiveness of society in social media through Natural Language Processing models.

To do so, we will have to follow a structured plan and understand the basic concepts of this broad field.

2.2 Pre-processing Techniques

All Natural Language Processing research begins with a *corpus* [35], a set of data or collection of documents that needs to be processed. To prepare the text data for the model, it is necessary to perform some text pre-processing techniques. Hence, this will be the first phase to be carried out.

Data cleansing and pre-processing are as vital as building an excellent Machine Learning model. So, the reliability of your model is extremely dependent upon the quality of data. Consequently, the advantages they present are clear: faster training, clarity of input data for better results, elimination of noise and redundancy, etc.

Basic Pre-processing procedures

At a more basic level, we will perform text tokenization and certain word replacements. Firstly, **tokenization** is the process of breaking down a large chunk of text into smaller *tokens* [31]. In this case, tokens can be words, characters, or sub-words, called n-grams. This process can thus be divided into three categories: word, character, and n-gram tokenization, and the word-based one is typically used. Hence, it is an essential step in both traditional methods and Advanced Deep Learning-based architectures.

Secondly, **replacements** of certain words or natural language "expressions" are very common. URLs, hashtags, user mentions and emails are not processable as they appear, so certain modifications must be made. For example, hyperlinks and numbers are usually replaced by "url" and "number", respectively. This also happens for usernames (to "username"), but hashtags are treated differently. As they could have potential mean-

ing for recognizing HS, only the # is removed. Nevertheless, if none of these tokens are considered to be meaningful, they would be erased.

Sentiment-oriented Pre-processing procedures

Focusing on sentiment-oriented techniques, there are some word replacements and removals that might be helpful, like **stopwords** and **punctuation** [31]. These two are considered part of the removal group, as they are sets of words and symbols that do not contribute to the sentiment analysis. However, there is an exception: emojis. Emojis are made up of combinations of punctuation symbols and have been shown to withhold meaningful connotations about hate speech. So, sometimes they are allowed to remain as a token.

If we move on to replacements, in a very specific way, we can talk about **slang** and **contractions**. Both will be normalized to their original forms to extract their meaning easily and without duplicating words in the corpus vocabulary. Abbreviations like "shouldn't" or "doesn't" will change to "should not" and "does not" and slang, such as "dope", to "cool".

In addition, we will find techniques such as lower casing, stemming and lemmatization. **Lower casing**, as its name suggests, consists of converting each word to lowercase. The main reason for doing this conversion is because words with the same meaning, but different cases, are represented as different words in the vector space if they are not changed, resulting in more dimensions and cost.

Then, stemming and lemmatization are quite similar [14]. On the one hand, **stemming** algorithms work by slicing off the word's end or beginning, taking into account a collection of frequent prefixes and suffixes found in derived words. So, it reduces the inflected word to its original stem or root. However, on the other hand, **lemmatization** algorithms focus on the morphological analysis of the words, and they require deep linguistics knowledge. To do so, detailed datasets must be used as input for these algorithms to search through and resolve a word to its original lemma. Here, Figure 2.1 shows an example of these two concepts:

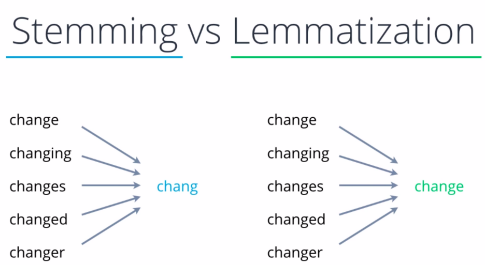


Figure 2.1: Example of stemming and lemmatization

2.3 Traditional Approached and Models

Once texts have been pre-processed to token sets, they have to be represented in a vector format in the feature space. In this way, we will get a reliable representation for the subsequent classification process.

2.3.1 Classical approach: Word Embeddings

Text representation that consists of assigning a vector to each word [2, 9, 41]. Therefore, it is generalizable: the algorithm created can be used for solving different types of problems. Words themselves cannot be processed by computer systems. Hence, they must be converted. This is where Word Embedding vectors come into play as their mathematical representations [23]. Some techniques are Bag of Words, TF-IDF or CountVectorizer.

Word Embedding traditional representations

First of all, **Bag of words**, or BoW for short [8, 13], is a simple text representation that counts occurrences of words within a document of the *corpus*. This model involves a *vocabulary*, which will be all the possible words in the *corpus*, and an occurrences vector for measuring the presence of known words. Here, texts will be called "bags", which contain words, so this representation is only concerned with whether known words occur in the document, not where in the document. Hence, there is a dimensionality issue, as the total dimension is the vocabulary size, and it can easily overfit. What's more, this representation does not consider semantic relationships among words.

Secondly, **Term Frequency - Inverse Document Frequency** (TF-IDf) is a statistical measure that evaluates how relevant a word is to a document in a *corpus* [50]. This is accomplished by multiplying two metrics: the Term Frequency (TF) and the Inverse Document Frequency (IDF). TF shall be the number of times a word appears in a document and IDF, the logarithm of the division of the total number of documents by the number of documents containing the word [56]. Even though TF-IDF is an improvement in word representation, it is based on the BoW model. Therefore, it captures neither word's positions in documents nor semantic relations nor co-occurrences.

$$w_{x,y} = tf_{x,y} \times \log\left(\frac{N}{df_x}\right)$$

TF-IDF
Term x within document y

$tf_{x,y}$ = frequency of x in y
df_x = number of documents containing x
N = total number of documents

Figure 2.2: TF-IDF formula for Word Embedding Representation [58]

Finally, Scikit's **Count Vectorizer model** or One-Hot Encoding is used for converting a *corpus* to a matrix of token counts [57]. This implementation produces a sparse representation of the counts, so it will present the same problems as the BoW model.

Once seen all three techniques, there should be mentioned that they have common drawbacks. These Word Embedding methods are not powerful enough to understand relationships among words in the same sentence. Therefore, they fail to solve the problems of continuity or sentence completion within NLP.

Neural Networks for word representations

As the main problem was semantic relationships, some new word models came along: Word2Vec, Glove, etc. **Word2vec** is a two-layered shallow Neural Network that generates a relative model of word embedding [28]. It works by taking a huge number of vocabulary-based datasets as input and generating a vector space in which each word corresponds to a single vector. We may now represent word's relationship in this way. This model has two different architectures for creating the word embeddings: CBOW and Skim-gram.

On one hand, **Continuous Bag of Words** model (CBoW) [49], in essence, tries to understand the context of the words and takes it as input. It tries to predict a masked target word by trying to understand the context of its surrounding neighbors. Therefore, this is unsupervised ML, which necessitates the use of labels in order to train the model.

On the other hand, **Skip-gram** learns by predicting surrounding words given a target [47, 65]. In other words, the architecture predicts words within a certain range before and after the current word in the same sentence, so it is the complementary model to CBoW and both perform the task of learning weights for their Neural Network's hidden layer [44]. Although Word2Vec can be helpful for semantic relationships, it has its drawbacks:

1. **Failure to deal with unfamiliar words**
   If a model has never seen a word, it won't be able to interpret it or build its vector.
2. **Difficult to train and to fine-tune**
   Large Datasets imply huge dimensionality. Hence, they are impossible to fine-tune.
3. **No shared representations at sub-word levels**
   Despite the fact that many words are morphologically similar, it treats each one as independent vectors.
4. **Scaling to new languages implies building new embedding matrices**
   It does not support parameter sharing, so the same model cannot be used across languages.

2.3.2 Neural Networks approach: RNN's and LSTM's

Some of Word Embedding's approach limitations can be solved by using Fully Connected Neural Networks (FCNN), like **Recurrent Neural Networks** (RNN) [26, 38]. They are, essentially, a FCNN that contains a refactoring of some of its hidden layers into a loop. That loop allows them to implement back propagation for accepting variable-length inputs. This is why they were better suited for processing textual input and predicting than Word Embedding-based models: RNNs maintain information in 'memory' over time.

These networks contain three hidden layers and the output will be a one-hot-encoded vector representing the predicted target word. Hence, the first layer will take a word's vector representation as input and its output will serve as the second hidden layer's input alongside with another word's vector. The third layer behaves exactly as the second one, so they could both use the same weight matrix, opening the opportunity of refactoring this into a loop to become recurrent.

Although back-propagation and recurrent refactoring helps with problems like training, fine-tuning and prediction, they have some downsides: RNNs have short term memory and suffer from vanishing and exploding gradient problems. So, this may not help with contextual issues. **Long Short-Term Memory neural networks** (LSTM) are able to solve it by introducing additional gates, such as input and forget gates, that allow for better gradient flow management and the storage of "long-range dependencies". Increasing the number of repeating layers in LSTM solves the long-range dependency in RNN.

However, traditional neural networks performance is surpassed by Deep learning methods, which are achieving SOTA results on challenging ML and NLP problems.

2.4 Deep Learning Approach with Transformers

For a long time, a large percentage of NLP approaches relied on shallow Machine Learning models and intensive hand-crafted features. As a result, issues such as the *curse of dimensionality* appeared [53, 38]. However, with the transition to Deep Neural Networks, a new world of possibilities opened up: the application of reinforcement learning, unsupervised methods and deep generative models to complex NLP tasks is at our fingertips.

Transformers: an introduction to the state-of-the-art model

One of the greatest advances in NLP recently has been **transformers**: a new encoder-decoder architecture that seeks to tackle sequence-to-sequence issues while also coping with long-range dependencies, proposed in the paper *Attention Is All You Need* [62]. Its main innovation is that it does not use sequence-aligned RNNs or convolutions to compute input and output representations. Instead, it focuses exclusively on stacked self-attention.

Transformers Architecture

As the transformers are based on traditional RNNs, each step taken by the model will be **auto-regressive**: layers consume the previously generated outputs as additional inputs when generating next representations. What's more, their architecture is divided into an encoder and a decoder. The first one, the **encoder** is formed by a six-layer stack where each layer has two sub-layers: a multi-head self-attention mechanism and a position-wise Fully Connected Feed-Forward RNN network. Therefore, the resultant output will be followed by a normalization layer. Regarding the **decoder**, it is almost identical to the encoder, but it inserts a third sub-layer to each layer of the six-layer stack: a multi-head attention mechanism which will be used on the encoder's resultant output. Here, Figure 2.3 summarizes what has been explained:

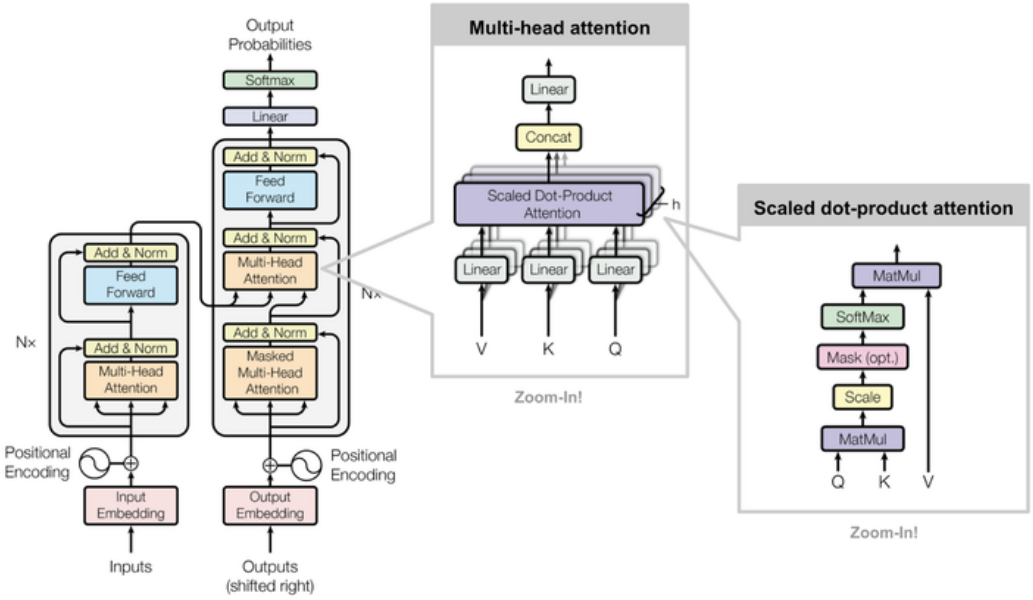


Figure 2.3: Transformer Architecture, focusing on Multi-head Attention Mechanisms [6]

Attention and Multi-head attention mechanisms

The previously described **attention mechanism** refers to a critical component of a Transformer's NN architecture [29]. It allows the model to dynamically highlight the most important features of the input, and it can be used on raw data or a higher-level representation of it. The basic concept behind attention is to compute a weight distribution on the input sequences, with larger values being assigned to more relevant elements.

$$Attention(Q, K, V) = SoftMax(\frac{QK^T}{\sqrt{d_k}})V$$

where d_k is the dimension of the Keys, K,V and Q are the matrices of the Keys, the Values and the Queries respectively.

All in all, attention is a function used by the FC Feed-Forward network for input translation, as its goal is to generate an output sequence $y = (y_1, ..., y_T)$ that is a translation of a given input $x = (x_1, ..., x_n)$. Hence, in a **multi-head attention mechanism**, instead of calculating a single attention function, queries, keys and values are projected h times to different dimensions. On each projection, this mechanism performs an attention function, in parallel with the rest. Then, the independent outputs are concatenated and linearly reprojected into the expected dimension.

$$Multi - head(Q, K, V) = Concat(head_1, ..., head_h)W^O$$

where $head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$, $W_i^Q \in \Re^{d_{model} \times d_k}$ are the projection parameters and h, the parallel attention layers.

2.4.1 Generative Pre-trained Transformers (GPT-2 and GPT-3)

OpenAI GPT-2 [5, 64] is a large transformer-based language model with 1.5 billion parameters, which surpasses its predecessor GPT-1 [55]. Its main goal is to predict the next token in a sequence given the previous context and, for this particular task, it has been trained with casual language modeling.

In terms of its architecture, it remained almost the same as in GPT-1 [60] but with an increase in the number of layers: from 12 to a 48-layer decoder-only transformer structure with masked self-attention. What's more, regarding the training phase, there was a transition from performing supervised and unsupervised training to only unsupervised training based on zero-shot learning. Consequently, GPT-2 outperformed 7 out of 8 datasets, becoming a SOTA model in 2019.

However, this model still needed fine-tuning to achieve its best results. For this reason and many more, OpenAI released in 2020 **GPT-3** [3, 12]: a 175 billion parameter autoregressive language model based also on a decoder-only transformer structure. Due to this enormous number of parameters and datasets the model has been trained on, GPT-3 performs well under the three NLP tasks it has been evaluated: zero-shot (0S), one-shot (1S) and few-shot (FS) learning. During these assignments, the model is given no demonstrations of the task (0S), only one (1S) or K examples (FS) of the context.

Regarding its differences with its predecessor GPT-2, GPT-3's architecture has 96 decoder layers, where each one contains a multi-head attention mechanism with 96 attention functions.

2.4.2 Bidirectional Encoder Representations from Transformers (BERT)

BERT [24], as the previous GPT's, is a transformer-based language model developed by Google AI researchers. As the name itself suggests, it implements bidirectional training of Transformers, a huge innovations in the NLP field. Unlike other techniques, which looked at sequences either from left-to-right/right-to-left or with a combination of both single-direction language models [4], its strategy allows the model to achieve a better understanding of the language context and word surroundings. This major change with BERT can be seen in Figure 2.4 below:

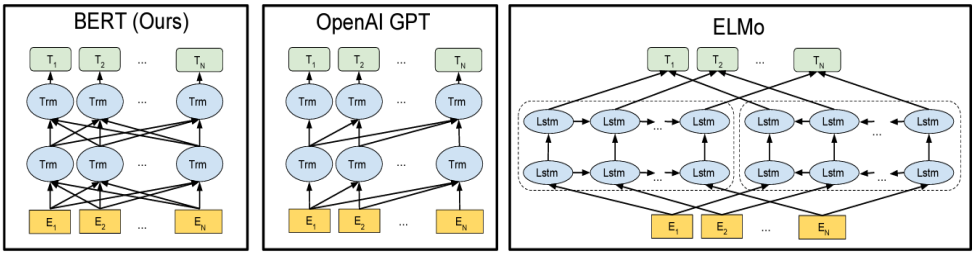


Figure 2.4: BERT bidirectional strategy compared to single-direction language models [24]

In terms of their main influences and underlying concepts, this model focuses on:

1. **Semi-supervised Sequence Learning:** its approaches are unsupervised feature-based methods, unsupervised fine-tuning procedures and Transfer Learning from Supervised Data
2. **Embeddings from Language Models (ELMo)**
3. **Universal Language Model Fine-tuning for Text Classification (ULM-FiT)**
4. **OpenAI Transformers:** they enable BERT to be the first NLP approach to rely purely on self-attention mechanisms.

Therefore, Masked Language Modeling (MLM) and Next Sentence Prediction [39, 33] are its NLP target tasks. To put it in another way, BERT is pre-trained to predict a hidden or masked token in a sentence based on the word's context and to understand what relationship ties two given sentences. Consequently, there are two steps in BERT's framework: **pre-training**, where unlabeled data is used for training the model across several tasks, and **fine-tuning**. In this last framework, the model is fine-tuned by first initializing it with the previous pre-trained parameters and, then, fine-tuning all of the variables using labeled data from the downstream tasks.

One of the many strengths of this model is its architecture: a multi-layer bidirectional Transformer encoder. In contrast to the GPT Transformer, BERT implements bidirectional self-attention mechanisms instead of constrained ones. So, basically, it is a trained encoder stack where, depending on the number of self-attention heads (A) and encoder layers (L), there will be different model sizes: $BERT_{BASE}$ (12 L and A) and $BERT_{LARGE}$ (24 L, 16 A).

2.4.3 Robustly optimized BERT approach (RoBERTa)

Although BERT became one of the SOTA language models of the moment, it had its perks [61]. For instance, it was remarkably under-trained (trained only with 16GB instead of 160GB as in RoBERTa [27]) and both framework steps (pre-training and fine-tuning) could be improved. One of BERT's major limitations is that the masking is done only once during pre-training. As a result, the model only had a single static mask which was used as input at every epoch for feeding it. Inevitably, Facebook took over Google's open source BERT to develop its optimization: **RoBERTa** [48, 20].

Even thought both BERT and RoBERTa have the same architecture, they diverge on other issues, such as Next Sentence Prediction. With RoBERTa, this task disappears in favor of *dynamic masking*, where the masked token changes over training epochs. Also, larger batch sizes were found more useful in training with larger datasets. As a consequence, training started to be done using more data for longer periods of time and with full input sequences to improve performance. This was caused because it was found that using single sequences (as in Next sentence prediction) hurt performance on downstream tasks.

Roberta ended up reaching SOTA levels in several important benchmarks, outperforming its predecessor. However, it is still a model based on RNNs, so it will also suffer from gradient vanishing and explosion, making it quite problematic to optimize NN parameters. In addition, Roberta and, subsequently, BERT, rely on masked input, neglecting dependency among tokens and, therefore, both tolerate pre-training and fine-tuning discrepancies. In other words, these models assume independence among masked and predicted tokens, oversimplifying language context.

2.4.4 XLNet: the best parts of Tranformer-XL and BERT

As stated above, RNNs and, especially, LSTM networks have serious difficulties optimizing parameters, so a new architecture arose to overcome these limitations and more: **Transformer-XL**, a deep self-attention transformer-based network that integrates the concept of recurrence. Transformer-XL [21] reuses the hidden states obtained in earlier segments rather than computing them from scratch. This auto-regressive language model outperforms several traditional models despite being heavily based on vanilla Transformers, as can be observed in Figure 2.5

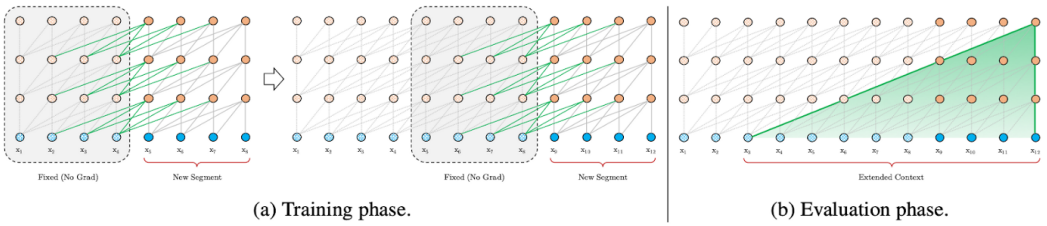


Figure 2.5: Transformer-XL's auto-regressive vanilla model with a segment of length 4 [22]

This is all thanks to the introduction of two innovative techniques which overcome auto-regressive language modeling limitations [34], such as bidirectional context training.

1. **Recurrence Mechanism**
   The hidden state sequence generated for the previous segment is fixed and cached during training so that the model can use it as an extended context for processing the following new segment. This supplementary input helps the network to use past knowledge, enabling it to model longer-term dependencies and prevent context fragmentation.

2. **Relative Positional Encoding**
   With the previous practice, it is essential to know, when the model is reusing hidden states, how positional information is kept consistent. Using a standard Transformer, each segment is handled separately, resulting in tokens from various segments having the same positional encoding. This technique's basic concept for solving this issue is to only encode relative positional information in the hidden states, so it provides a temporal guide to the model about how information should be obtained. In addition, each layer's attention score is improved by injecting the relative distance dynamically. Therefore, the model can easily recognize the several segments.

Based on Transformer-XL's architecture and SOTA performing techniques, **XLNet** [68] is an auto-regressive language model that outputs the joint probability of a sequence of tokens. Its target is to learn bidirectional context by maximizing the expected likelihood over all permutations of the input sequence factorization order. In other words, it combines the best parts of auto-regressive (Transformer-XL) and auto-encoding (BERT) modeling.

On the one hand, instead of applying a uni-directional likelihood factorization, regardless of whether it is a forward or backward product, XLNet introduces all possible permutations [30], enabling the context to be build using both left and right sub-contexts and becoming bidirectional. On the other hand, as the Transformer-XL architecture and techniques do not depend on masked input and independence of tokens (BERT's limitation), there are no discrepancies in either training or fine-tuning. All in all, XLNet has became the latest SOTA language model to reach top scores in many tasks, such as question answering, sentiment analysis or document ranking. Therefore, its auto-regressive approach, build on top of great models, gives significant results and it is worth of study.

2.5 Evaluations and metrics: testing for improving

Every pre-trained and fine-tuned model requires evaluating its performance using several metrics. It is critical to evaluate every model using a variety of evaluation metrics in order to ensure that your model is running correctly and optimally. This is due to the fact that a model may perform well when using one measurement but poorly another one. In addition, it provides us with a wide variety of metrics for model's comparisons in SOTA scale. The most important ones [37, 42], on which this project will focus, are:

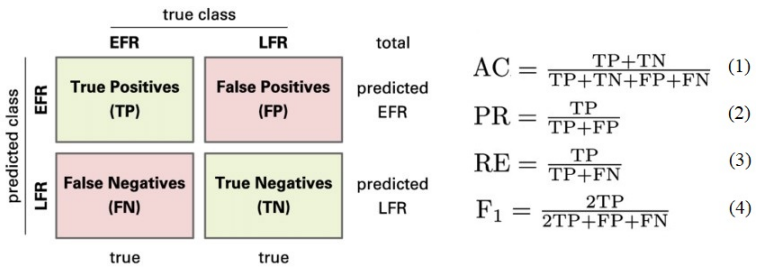


Figure 2.6: Precision (PR), Recall (RE), Accuracy (CA), and F1-score [11]

1. **Accuracy:** fraction of predictions or tokens our model got right. Hence, it is the sum of True Positives (TP) and True Negatives (TN) divided by the total population.
2. **Precision:** fraction of correct positive predictions within every positive prediction, or, in other words, TP divided by the total number of positive calls.
3. **Recall:** fraction of correct positives prediction within everything that actually is positive. So, it is number of TP divided by the sum of TP and False Negatives (FN).
4. **F1-score:** harmonic mean of precision and recall.

Chapter 3

Implementation

This chapter will focus on explaining in depth the implementation carried out in this project. It will begin by explaining the environment in which it has been developed, the tools used and the structure established to complete the results collection.

3.1 Implementation scenario: IDE, resources and tools

The world of Data Science and, above all, the NLP field, has quite clear preferences in terms of environment, programming language and other details. Either for its simplicity and usability or for being "trending topic among all programmers", Python as a language is the most widely used and the most popular. Besides, who better than the IDE Jupyter Notebook to support all the implementation when they are the perfect match.

Therefore, the language chosen was **Python** 3.6 together with Jupyter Notebook to support the code and be able to compile and run it. Although the memory usage is quite expensive, its versatility and simplicity while programming pre-processing, training and Deep-Learning optimizations makes up for it. Also, since all the pre-trained models we will use have been developed and stored in Python libraries, not choosing it was not an option.

If we focus on the Hardware that has been used, this Bachelor's thesis implementation has been carried out with an Asus VivoBook laptop with Intel Core i7-10510U processor, with 16GM of RAM, 512 GB SSD and a 2 GB NVIDIA GeForce MX250 graphics card. Since the graphics card is fairly standard for deep-learning and model training purposes, the idea of compiling and running the Jupyter Notebooks in client-server applications, such as Google Colab or Kaggle, came up. Here, instead of using your own local resources, the power of the application's servers with their GPUs and CPUs is being put to use. In this way, we increase the level of resources of the implementation, with the downside that the hours of activity are limited as they are a free and public community service.

As for the IDE, as mentioned, it will be **Jupyter Notebook**, but due to the already mentioned lack of resources it will be carried out together with two support systems: Colab Research Google and Physics Cluster. Firstly, **Google Colab's** client-server application, from the non-profit Jupyter Project, serves as a constant bridge between the code and the explanatory texts with a server that compiles and executes these portions of code. Jupyter Notebook allows software developers to create code along with rich text such as equations, images and text in one place. In addition, it also provides a way to view and share the results of the code (the output of each cell), as well as multimedia renderings embedded in the notebook. This is critical in order to be able to share and analyze these results a posteriori. This last feature will be one we will use the most throughout the whole process of project development and hate and sentiment analysis.

Secondly, as Goggle Colab's resources are quite limited in terms of time and GPU usage, as an exception, the code was derived to a **Cluster** at the Faculty of Physics. Through ssh bridges and several connections, it is possible to reach the directory that has been enabled for this Bachelor's thesis, in which experiments can be launched towards the cluster servers. Even though this cluster has three 11 GB NVIDIA GeForce RTX 2080 Ti graphic cards, as it can be seen in Figure 3.1, they are not completely available for usage as it is a researcher's service. Here, sharing resources and using Docker are the general rules for not overstepping other people's work. However, ultimately, a similar situation was encountered as described at the very beginning: there are not enough resources to launch heavy experiments. Despite the fact that this is a drawback, it will not prevent achieving the established goals.

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 418.87.00    Driver Version: 418.87.00    CUDA Version: 10.1     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  GeForce RTX 208...  Off  | 00000000:17:00.0 Off |                  N/A |
| 60%   87C    P2   138W / 250W |   4220MiB / 10989MiB |     82%      Default |
+-------------------------------+----------------------+----------------------+
|   1  GeForce RTX 208...  Off  | 00000000:65:00.0 Off |                  N/A |
| 62%   88C    P2   162W / 250W |   7263MiB / 10989MiB |     97%      Default |
+-------------------------------+----------------------+----------------------+
|   2  GeForce RTX 208...  Off  | 00000000:B6:00.0 Off |                  N/A |
| 55%   86C    P2   168W / 250W |   3061MiB / 10989MiB |     84%      Default |
+-------------------------------+----------------------+----------------------+
```

Figure 3.1: Table of CUDA and GPU's usage in Physics Cluster

3.2 Libraries

In order to accomplish our goals, many Python libraries are necessary. In this section we will explain the purpose of their use, to what we will apply their functions and curiosities of some of them. From libraries and packages for displaying graphs to understand the metrics resulting from training to the pre-trained models themselves, we will briefly review the most important ones:

LIBRARIES FOR GENERAL AND DATA SCIENCE PURPOSES

1. **os**: Python module which, as its name states, provides functions for interacting with the Operating System. here, its functionality relies on path names and the creation of directories to store the results

2. **re**: Python module which contains Regular Expression matching operations. As the pre-processing sections of the notebooks require the search for certain patterns and keywords (ex: hashtags, URLs, etc.), this module is often used for matching Regex patterns with the *corpus*.

3. **time**: Python module which provides time-related functions. Its goals in the notebooks is quite simple: measure the time spent during training and validation.

4. **datetime**: Python module which also provides time and date related functions for its manipulation. The purpose of its usage is mainly to transform time in seconds into a hours-minutes-seconds string format for facilitating printing the elapsed time during training.

5. **enum**: Python Module which defines enumeration classes used for defining the unique Dataset types while saving predictions and extra metadata during the pre-processing section.

6. **json**: Python module which provides an API for converting in-memory Python objects to a serialized representation called JavaScript Object Notation (JSON) and, vice-versa. Here, loading emojis and abbreviation dictionaries data corresponds to its main goal.

7. **pickle**: Python module which implements binary protocols for serializing and de-serializing data, like metadata files.

8. **collections**: Python module which implements specialized container data-types offering alternatives to Python's built-in data-types (dict, list, set, and tuple). The ones used are *namedtuple*, *defaultdict* and *Counter*.

9. **random**: Python module which generates pseudo-random numbers. It is mostly used for starting random numbers given an initial seed.

10. **logging**: Python API which defines functions and classes for implementing a flexible event logging system for applications and libraries. Here, its only purpose is disabling *smote_variants* logging so spam can be avoided during training.

11. **unicodedata**: Python module which provides access to the Unicode Character Database that defines character properties for all Unicode characters. As some of the languages studied in this project use accents, it is mandatory that these Unicode characters that contain them get the proper attention they deserve. Therefore, its treatment would be removing them and keeping the letter.

12. **SkLearn**: Scikit-Learn open source Machine Learning API which provides numerous efficient tools for statistical modeling and supervised-unsupervised learning, including classification, regression, clustering and dimensionality reduction.

(a) ***feature_extraction.text***: Scikit-Learn submodule for building feature vectors from text documents. Its main purpose is to provide vectorizers (TfidfVectorizer and CountVectorizer) needed for pre-processing the datasets.

(b) ***metrics***: Scikit-Learn submodule which includes score functions, performance metrics and pairwise metrics and distance computations, like accuracy scores and F1 metrics used in validation processes.

(c) ***model_selection***: MIRAR SI HACER CROSS_VALIDATION AYUDA

(d) ***pipeline***:Scikit-Learn submodule which helps to build the Pipeline used for generating the Tf-Idf matrix of one dataset.

(e) ***preprocessing***: Scikit-Learn submodule which includes scaling, centering, normalization, binarization methods, such as FunctionTransformer and OneHotEncoder. These are the functions used and needed during the generation of the Pipeline and training, respectively.

13. **Numpy**: Python library, which stands for Numerical Python, consists of multidimensional array objects and a collection of routines for processing these arrays. Its usage varies from numerical and statistical functions, like sum and mean, to stacking methods for arrays.

14. **Pandas**: open source Python library which provides multiple Machine Learning functions for dealing with DataFrames and Data Science tasks. Therefore, its functionality relies on loading datasets or files and concatenating and creating DataFrames from those loaded files.

15. **Scipy**: open-source Python library which is used to solve scientific and mathematical problems involving, typically, matrices. Here, it enables loading Sparse matrices from npz files and also saving them.

16. **Torch**: open-source Python Machine Learning library and a scientific framework which provides a wide range of algorithms and methods for Deep Learning.

    (a) ***nn***: Torch submodule which enables the creation and training of neural networks. Its main goal, apart from aiding the Neural Networks generation, is to provide Loss functions for their evaluation.

    (b) ***utils.data***:Torch submodule which provides numerous functions for loading, dealing and interacting with data. Several data samplers are extracted from this module to help specify the sequence of keys used in data loading, alongside with its DataLoader.

LIBRARIES FOR VISUALIZATION PURPOSES

1. **matplotlib**: open-source comprehensive library for creating static, animated and interactive visualizations in Python.

    (a) ***pyplot***: matplotlib submodule which provides a state-based interface and an implicit, MATLAB-like, way of plotting. Its main goal if to output figures for visualizing the pre-processing, training and validation results.

2. **Seaborn**: Python data visualization library based on matplotlib which provides a high-level interface for drawing attractive and informative statistical graphics. Therefore, this library works alongside matplotlib for a better understating of the training and evaluation results.

NLP-RELATED LIBRARIES

1. **Transformers** [67]: Python library which provides thousands of pretrained models to perform tasks on different modalities, such as sentiment analysis or text classification. Here, the pretrained models used where Bert, RoBERTa and XLNet with their respective learning rate scheduler from this library.
2. **Spellchecker**: Python module which allows setting a Levenshtein Distance algorithm to find permutations, within an edit distance of 2, from an original word. Afterwards, there is the possibility of checking for erroneous words and then return their right spelling, which is the principal goal of this module in the Bachelor's thesis code.
3. **pycontractions**: Python library which provides methods for expanding and creating common English contractions in text and helps with dimensionality reduction.
4. **nltk**: Python package which provides several methods for natural language processing, from dealing with stopwords and regex expressions to lemmatization.
   (a) ***corpus***: NLTK submodule which holds functions that can be used to read corpus files and, here, its goals relies on obtaining stopwords collections for some languages, such us Spanish and English.
   (b) ***stem.wordnet***: NLTK submodule from which it is retrieved the lemmatization principal function for the pre-processing section.
   (c) ***tokenize.regexp***: NLTK submodule from which it is retrieved the regex tokenizer for tokenizing corpus words without digits, while ignoring punctuation, except Users and Hashtags
5. **stopwordsiso**: Python package which provides a collection of stopwords for multiple languages, using ISO 639-1 language code. Its usage derives from the insufficiency of stopwords in certain languages in nltk, like Tagalo for the Universal Joy Dataset.

3.3 Structure of Every Jupyter Notebook

Once all the libraries, modules, APIs and packages that will be needed have been discussed, it is time to talk about how the project has been carried out. Starting from the idea that everything has been developed in Jupyter Notebooks, it is necessary to mention that there will be one notebook for each Dataset to be analyzed and an extra one for testing Transfer Learning between Datasets. Therefore, the project consists of five different notebook and this whole section 3.3 will be divided into the exact same parts as one of these notebooks. Among them, the structure is common. Nevertheless, all of them present some nuances in the code aspect: functions modified and adjusted to datasets, new methods implemented to overcome the tasks, etc.

3.3.1 Initial stage: Imports, Configurations and loading of Datasets

First of all, each Jupyter Notebook will perform all the necessary *imports* for the subsequent code along with, if working in a Google Colab environment, the Drive Mount and the relevant pip installs. All these imports of libraries, APIs and packages will be performed in a structured way. It will start with the General Purpose or Data Science libraries, and then move on to the more NLP-specific ones.

However, between them, a small configuration of the screen outputs of the results will be carried out with *matplotlib* and *seaborn* libraries. Its purpose is to establish a white aesthetic as background to make the statistical results stand out, together with their scaling.

Finally, the datasets contained in the inputs directory will be loaded. Among these files, two types of extensions prevail, *csv* and *tsv*, which will be read by Pandas library functions to end up saved in DataFrames with identifying names of their dataset and task.

3.3.2 Datasets used in the Implementation

In this subsection, even though its not a part of the structure of all Jupyter Notebooks, it is necessary to introduce which datasets could be loaded. Therefore, the central theme on which the explanation will orbit will be the Datasets selected for training and evaluating the models that will be presented will be presented later.

Even though all of them have in common the analysis of texts, in this case tweets or Facebook posts, for classifying the emotions they transmit and detecting if there is Hate Speech, each one has its own peculiarities. All of them will be subdivided into three distinct parts: training, development and test.

Despite the fact that the data they contain is a reduced percentage for testing the real content of each one, the train subdataset will invariably be the largest. As its name implies, its purpose is to help the models to train and classify the task they have to perform at that moment, whether or not there is Hate Speech or if it is offensive, for instance. Then, the developvent subdataset will serve for assessing the previous training and will contain a relatively minor portion of the data. Finally, the only remaining step would be to check the result of the whole process with the subdataset Test. Each dataset will have a similar structure and will be as follows:

1. **IDs column**: to list the different tweets or Facebook posts.
2. **Text column**: usually referred as "comment" or "text", it consists of the text previously cleaned from tweets or posts. It will be used to build the corpus of the project and will be rigorously analyzed and pre-processed.
3. **Columns about text information**: in them, depending on the Dataset and the task to be addressed, there will be the required information for preprocessing and training. Some examples would be the emotion that characterizes the text, the toxicity or level of Hate Speech presented, the language, etc.

Now, the specific Datasets will be explained and studied.

- **HatEval**

  Dataset used for the Task 5 of the SemEval 2019 competition [52] [36] [7] [25] [10] [62]. Its main task is to detect Hate Speech against immigrants and women in Spanish and English posts. Once its detected, it enables the detection of Aggressiveness in those posts and which minority is being targeted (women or immigrants). Therefore, this dataset will have an IDs column, a text column called "text" and three text information columns: HS (**Hate Speech**), AG (**Aggressiveness**) and TR (**Target Rate**). All in all, its task can be subdivided into finding whether there is HS or not and, if so, whether there is AG and who is the TR. Also, for accomplishing this subtasks, it will be necessary to apply binary classification (sentiment analysis [70]).

- **Detoxis**

  As its name states, this dataset main task is to DETect TOXIScity [18], among other issues. In this work, we will only focus on its main task, however it can also be used for detecting sarcasm, mockery, target types (person or minority group), intolerance and several others. Unlike other datasets, this one does not include a validation csv at first. Therefore, from the training subdataset, it will be established that 20% of it will become another subdataset for validations. As it should be, the initial index column will be reseted. Next we have a column with tweets called ”comment“ and, finally, the columns of text information that we shall use. As we want to study the **toxicity**, the chosen ones have been ”toxicity“ and ”toxicity_level“.

- **Emoevent**

  Similar to Detoxis, this dataset is an acronym of its full name: Multilingual Emotion Corpus based on different Events [54]. However, with Emoevent we move from detecting hate speech or toxicity, which are sentiment analysis tasks, to emotion classification. There are 7 labeled emotions (anger, disgust, joy, sadness, surprise, fear, others) and they imply a multi-class classification task that will not be performed. This is mainly due to the fact that our target will be to experiment with the other column of information: **offensiveness**. The aim is to map how these emotions affect offensiveness and vice versa, in addition to detecting it (sentiment analysis task).

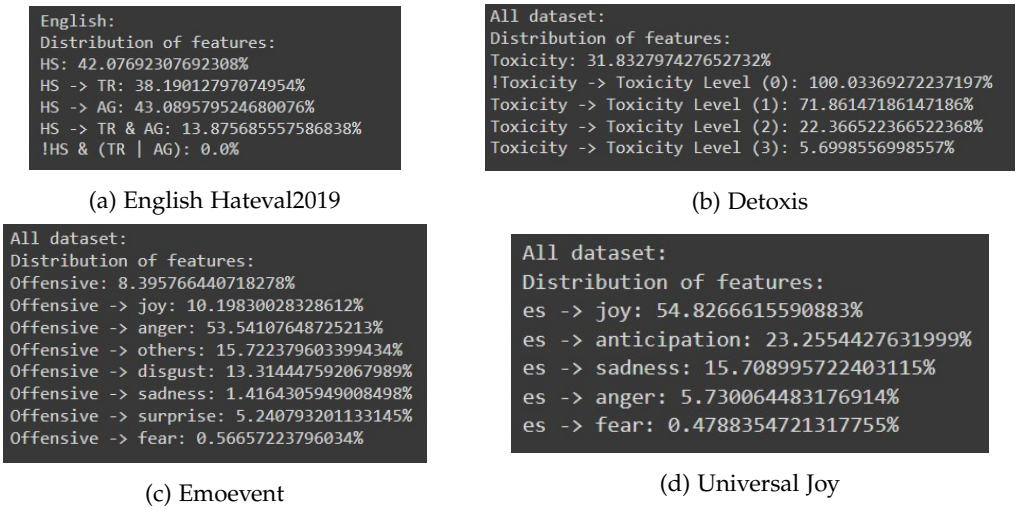
- **Universal Joy**

  Finally, Universal Joy is the most distant Dataset from the previous ones. Although it is subdivided into 3 well-structured subdatasets, its content differs from the previous tasks. It is a dataset for classifying emotions across languages [46]. In other words, although it has a text column called "text" like all the other datasets, none of its information columns is binary, but rather multi-class. First, a column of emotions ("emotion") will label each text with one of its five available emotions (joy, anticipation, sadness, anger, fear).Second, there will be a column with the language in which the Facebook post is written and it may be Spanish (es), English (en), Portuguese (pt), Tagalo (tl) or Chinese (zh). Therefore, what follows is to map the distribution of emotions by language and try to classify them.

3.3.3 Data and Statistical Analysis

The main idea in this subsection is to analyze the imported datasets in a customized approach to the task(s) performed. Hence, there will be a definition of functions to detail how the content of each dataset is, separately and together. This analysis is divided into:

1. **Distribution of features**:
   This first part depends heavily on the last columns of each Dataset and will be the most personalized code. It will try to map the main features to be studied in each Dataset. For HateEval2019 (*see Figure 3.2a*), it will focus on Hate Speech (HS) in the corpus and how its presence affects Aggressiveness (AG) and Target Rate (TR). Then, for Detoxis (*see Figure 3.2b*), our focus will be on whether there is Toxicity and what is its current level. However, from Emoevent onwards (*see Figure 3.2c*), HS is left behind to focus on Offensiveness. For Emoevent, a mapping of its emotions within offensive texts has been designed. Finally, Universal_joy (*see Figure 3.2d*) only tells us about emotions in its Facebook posts and the language in which they were written, so its distribution will be a ratio of posts per language that present one of its emotions.



(a) English Hateval2019

(b) Detoxis

(c) Emoevent

(d) Universal Joy

Figure 3.2: Feature Distributions through Datasets

2. **Patterns information of the corpus**:
   Here text patterns with the most frequent occurrences will be highlighted. Among them, there will be usernames, hashtags or urls. However, there is a major discrepancy between one half of the datasets and the other. On the one hand, HatEval2019 and Detoxis (*see Figure 3.3a*) will get an illustrative result as they show texts as they have been published on social media. However, Emoevent and Universal Joy (*see Figure 3.3b*) have been previously cleaned to anonymize sensitive data that posts may have, such as usernames or locations. All text parts with this description have been replaced with representative strings of their content (Users, Hashtags...). Despite the differences, all will show a count of their usage in the corpus, listing the most used ones for the first two Datasets and with a ratio according to emotions for the other two.

```
Users:
Number of mentions per tweet (MIN, MEAN, MAX): (0, 0.7923846153846154, 42)
Number of tweets that contain a mention: 0.4179230769230769
Most Common mentions:
                  Count        HS  HS -> TR  HS -> AG
Mention
@realdonaldtrump    549  0.553734  0.101974  0.608553
@potus              189  0.566138  0.093458  0.579439
@isupport_israel    105  0.580952  0.131148  0.442623
@anncoulter         102  0.627451  0.843750  0.250000
@youtube             71  0.309859  0.090909  0.454545
@housegop            66  0.878788  0.034483  0.672414
@foxnews             65  0.538462  0.200000  0.400000
@                    60  0.416667  0.480000  0.400000
@refugees            59  0.016949  0.000000  0.000000
@realjameswoods      58  0.293103  0.235294  0.529412
```

(a) English Hateval2019

```
Users:
Number of mentions per comment (MIN, MEAN, MAX): (0, 0.35699845403734093, 13)
Number of comments that contain a mention: 0.2159590914496373
Most Common mentions (%):
         Count  offensive      joy    anger    others   disgust  sadness  surprise  fear
Mention
user      3002   8.155212  8.266129  57.66129  11.895161  19.153226  0.201613  2.822581   0.0
```

(b) Emoevent

Figure 3.3: Types of Datasets and its resulting pattern's information

3. **Corpus characteristics using statistics**:
   In this penultimate subsection, a brief general review of the whole corpus will be carried out. This will consist of displaying general information of the text input. In other words, 3 generic facts of the dataset will be studied and they are: number of characters per tweet or post, number of words per tweet or post and average length of a word in a tweet or post. However, it will not be limited to these 3 values. For each fact studied, its minimum value, its maximum value and its average will be displayed. With this simple statistical analysis, the goal is to see the shape of a tweet or post in order to adjust, in the future, the size of each on of them to the maximum length of models.

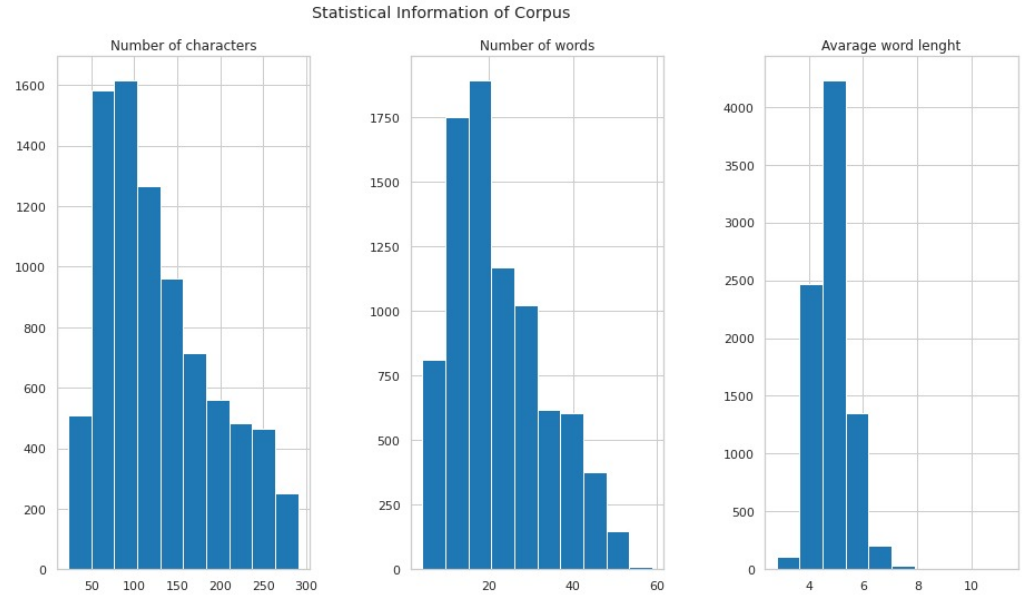
```
Tweets: (MIN, MEAN, MAX)
Number of characters: (10, 141.56215384615385, 851)
Mean of Number of words: (1, 21.619846153846154, 93)
Mean of Avarage word lenght: (2.5, 5.836345554093691, 51.666666666666664)
```

Figure 3.4: English Hateval2019 Corpus Characteristics

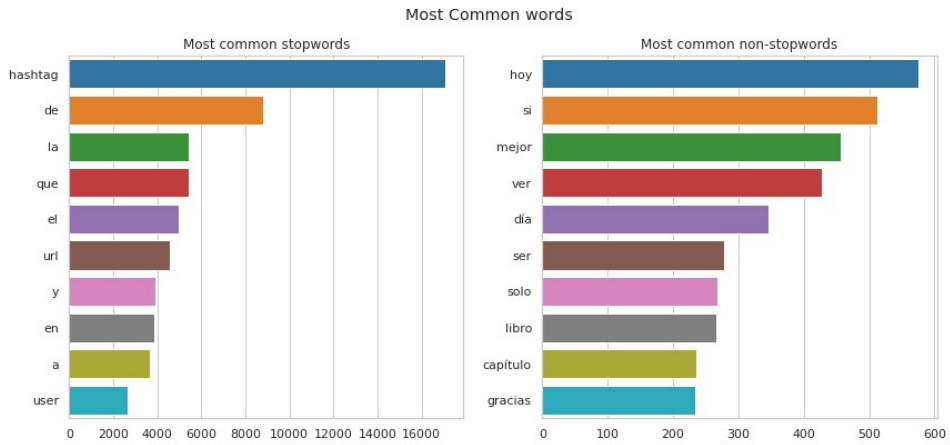
4. **Bar charts of statistics**:
   Continuing with the information gathered in previous subsections, in this last one a series of illustrative graphs will be plotted. Their main purpose is to show in a quick and intuitive way the calculated statistics. For this purpose, 3 types of bar charts will be plotted. First of all, 3 histograms will be plotted for the **corpus characteristics** (*see Figure 3.5a*): number of characters and words per post and average size of each post.

   With them, apart from seeing their peak, it will be possible to appreciate other nearby

measures. Afterwards, the focus will be on **stopwords of the corpus** and their weight in it (*see Figure 3.5b*). Two bar charts will be plotted: most common stopwords and words that are not stopwords (non-stopwords). Finally, the last two bar charts will make references to the **most common bigrams and trigrams** in the corpus, respectively (*see Figure 3.6*). These will highlight the most repeated combinations of words among posts, providing us with valuable information for sentiment analysis.



(a) Statistical Information of Corpus



(b) Most common words of corpus

Figure 3.5: Bar Charts of Emoevent Dataset

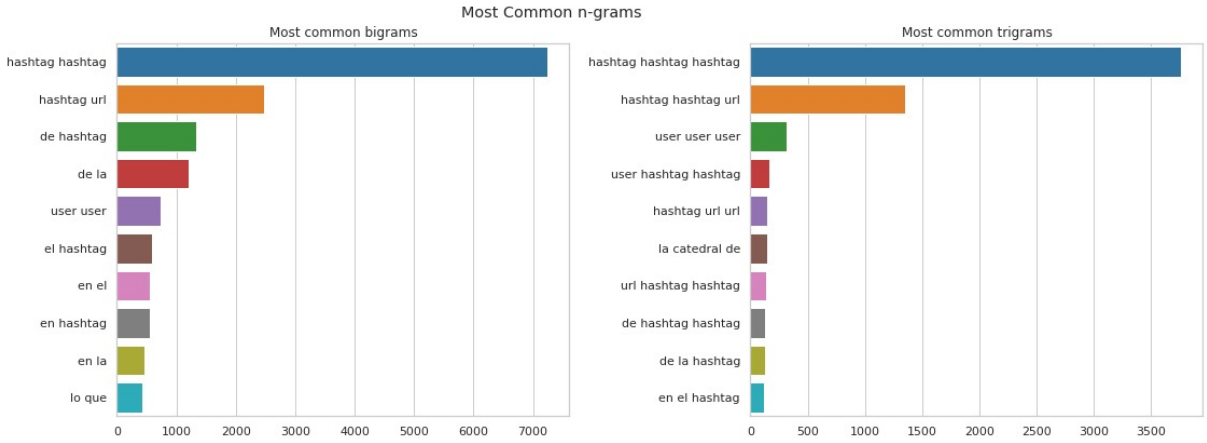


Figure 3.6: Most common n-grams of corpus

3.3.4 Datasets Pre-processing with TF-IDF

Once the initial phase of analyzing the datasets and showing their statistics is over, it is time to dive into the core issues of the notebooks: pre-processing and training. This section will deal with the first topic: the pre-processing of the corpus for its subsequent use in training. To achieve this we shall use a significant number of techniques and measures explained in previous Section 2.2 Pre-processing Techniques.

In a nutshell, the key concept on which the pre-processing will rotate and which will give it meaning is to convert the corpus from text to a Document-Term matrix populated with TF-IDF frequencies. This matrix will contain the frequencies of the words extracted from the corpus and to which, prior to anything else, a first phase of pre-processing will be applied. The *feature_extraction.text* module of the Sklearn library provides us with the *TfidfVectorizer* function to perform this conversion from almost raw tweets to a matrix of TF-IDF features. As for this first pre-processing, this will consist of 3 previously mentioned techniques: tokenization, text correction with Levenshtein distance and lemmatization.

Starting with **tokenization**, as already explained, it consists of turning the entire text of tweets or posts into text subunits called tokens (a single word). Numbers will not be taken into account to form tokens from them and accents and punctuation, at the moment, will be ignored for further treatment in the second phase of pre-processing. Following, using the python *Spellchecker* module, each token will undergo a **Levenshtein distance of 2** in order to detect and correct spelling errors in them. As usual, text in tweets or Facebook posts are written in colloquial speech and are highly susceptible to errors such as syllable shifts, mismatching letters or some missing letters. The purpose of this step is to correct these mistakes and to get the original tokens of all these misspelled words.

Finally, the last step will consist of the **lemmatization** of the tokens presented. As its name explains, this process will extract the lemma or root of each token taking into account the context of the word, avoiding simply cutting the word. For this purpose, the lemmatizer *WordNetLemmatizer* of the *NLTK* library will be used together with its lemmatize() function.

This process is performed with a very specific goal: to have a TF-IDF feature matrix as compact as possible. In other words, the lemma of a token can be the lemma of many others belonging to the same lexical family. So, by getting the common root, the dimensionality of the problem can be reduced and the result can be optimized. This whole process is being carried out by the defined function tokenization_text and the result of using *TfidfVectorizer* will be the vectorizer needed for the second phase of pre-processing.

Continuing with the pre-processing, the second phase is to create the **Pipeline** necessary to train models with the generated vectorizer. This Pipeline, also from Sklearn, will need mainly 2 parameters: the vectorizer already mentioned and a transform, each one in a tuple with its name tag. The transform will consist of the *FunctionTransformer* function of the Sklearn pre-processing module and will deliver the necessary interface for this pipeline (standard methods of other Sklearn estimators, such as fit or transform) without the need to overwrite these functions. This facilitates the execution of our pipeline, as you can simply feed the pre-processed data to the Pipeline's fit and transform methods instead of explicitly implementing each stage of the pipeline.

The function that we will use for the pipeline will be *direct_replacement* and in it, before anything else, another secondary function will be called to load the necessary dictionaries in JSONs for the following pre-processing (emojis and abbreviations). These dictionaries will be applied for replacing emojis, both simple and complex, and abbreviations within the tokens with meaningful words. Both parts of the text contain great value for the sentiment analysis of the corpus, since the use of certain emojis as abbreviations carries strong connotations of certain feelings: a heart is joy, a crying face is sadness, etc. Once they have been loaded, the appropriate replacements for the tokens will be made. The main ones to be implemented are the following:

1. **Lower-casing** every token for avoiding repeating tokens.
2. Standerizing **contractions** and other special characters, such us apostrophes, three dots, quote commas, etc.
3. Removing any remaining **digit**.
4. Replacing **emojis** for their meaning. In case there is no meaning found in both dictionaries, it will be removed from the tokens list.
5. Replacing **abbreviations** for their full meaningful version.
6. Removing **accents**.

Therefore, once the token replacement and their processing has been completed, the Pipeline required for the next section will have been achieved.

3.3.5 Fitting the new pipe and saving results

This new pipeline that has been developed needs to be fitted with data for it to work properly, i.e. it must go through its fit or *fit_transform* functions to learn how to process text, extract its most important features and model the result. To do this, for each subdataset, our auxiliary function *load_Tfidf_processing* will be called, which will start the fitting process of the pipe. Prior to starting, it will check whether this learning process, which is long and lengthy, has been done previously or not. If so, it will have been saved in a compressed npz file and can be retrieved.

3.3.6 Neural Network Pre-processing, Training and Evaluation Metrics

In this subsection we will deal with the next major issue: **training**. Once the pipe has been created and trained, we need to focus on how to train the different models in a general way and pre-process the subdatasets so that they are ready for the neural networks.

Beginning with the data, their **pre-processing** is very similar to that described for generating the vectorizer for the Pipeline and is by using the direct_replacement() method of TF-IDF pre-processing section. As a difference, in this case, the elimination of hashtags and usernames will be established to avoid difficulties when tokenizing them in the case of datasets that do not have these words in the text masked. To perform this step, for each subdataset the preprocessing_nn() function will be called, which will perform this generic pre-processing for neural networks. As an extra feature, an auxiliary function has been defined to sample the pre-processing performance of each subdataset. It shows the original text, followed by its tokenization and translation into IDs. Next, to emphasize the need for pre-processing, we will make a comparison with the pre-processed text as the original text, together with its tokenization and translation to IDS.

The following procedure is to start preparing the necessary parameters to generate the **final datasets** that will be introduced in the models so that they can be trained with them. These parameters are: the tokenizer, the pre-processed text that we have just prepared, the labels and the maximum length of the tokenized text. As for the tokenizer, we will talk about it in its specific section for each model since it is inherent to them. Therefore, we will discuss the other parameters. Starting with the **data** and **labels**, as mentioned above, the data entered are those texts that we have pre-processed for the neural network and their labels will be given by the column of the subdataset on which we want to perform the training. For example, for the Hateval subdatasets, the labels will be the hate speech column or, for Emoevent, it will be the offensiveness column. Finally, the last parameter is the **maximum length of the tokenized text** which, as its name indicates, consists of the maximum length in number of tokens for the inputs of the transformer model. It will be calculated by finding the minimum between the maximum length pre-established in the tokenizer configuration and the maximum length of tokens found in the text.

Once we have all these parameters ready, the final datasets will be generated with the function generate_dataset(), the device used with CUDA in the first GPU will be estab-

lished and the **optimizer** will be defined. This last one will be the AdamW optimizer of the *torch.optim* module and, with it, we will be able to set the parameters of the model we use, the *learning rate* we want and *epsilon* if we want. All that remains is to decide what *batch size* we wish and the number of *epochs* we want the training to perform. Like all training and evaluation functions, the **train function** will need the model to train, the previously pre-processed training and validation subdatasets, the optimizer, the batch size, the number of epochs and the device. This first method will serve as a prelude to the main training function, ***train_with_dataloader***. Its main goal is to create and set the training and validation DataLoaders and pass them to the main method, along with all the parameters that have been sent to it, except for the datasets that are changed for the DataLoaders.

Before starting the training, it is necessary to establish certain values needed in the process of the *epochs*:

1. Send the model to the selected **device**, which in this case will be GPU 0 with CUDA.

2. Set the **scheduler** for the *learning rate* with *get_linear_schedule_with_warmup()*. Basically, it creates a schedule with the chosen *learning rate* and decreases it linearly from the initial value to 0, after a warmup period during which it increases linearly from 0 to the initial value set in the *optimizer*.

3. Set the **random seed** all over the place to make the training reproducible (Torch, Numpy, etc.).

4. Initialize the **timer**.

Now, for each *epoch*, we will first set an extra timer to know how long each *epoch* lasts, reset the *epoch's* total training loss to 0 and put the model into **training mode**. Then, for each step of the DataLoader, we will extract its batch and subdivide it into its parts: first, the IDS of the tokens introduced as input; second, the attention masks set for the model; third, the labels. Once we have the three parts and before evaluating the model in this training batch, it is always necessary to clean any previously calculated gradient. To do this, we use the function *zero_grad()*, which clears old gradients from the last step. Otherwise the model would just accumulate them from all *loss.backward()* calls. Next we will perform the forward step, also called **forward propagation**. It constitutes the process of internal calculations which occur when passing the input data through all the neurons of the neural network of the model. Thus, the chosen model will decide what it believes to be the correct expected output.

Right after that, we accumulate the training loss over all batches in order to calculate the average loss at the end. We also perform the backward pass or **back propagation** to calculate the gradients and also control the exploding gradients by clipping the norm of the gradients to 1.0. Finally, we update the parameters of the *optimizer* and the *scheduler* steps and repeat the process until we finish the training with all the data of the *epoch*. As a note, every 40 steps, an update of the process is displayed on the screen. Once the *epoch* is finished, before proceeding to evaluate the result, we calculate the **average training loss**,

which is simply the total accumulated loss in the *epoch* steps divided by the training DataLoader size. Next, we stop the *epoch* timer since its function is already completed and change the model mode to validation.

The **validation mode** is pretty similar to the training mode. First, we set the loss accumulator to zero and, for each step of the validation DataLoader, we pick up its batch. This batch will be subdivided into input IDs, attention masks and labels. However, now there is no need to reset the gradients of the model, since the model will be evaluated without calculating them. For this purpose, **forward propagation** is performed in order to obtain the loss, which will be accumulated, and the logits, which are vectors of non-normalized predictions that the model generates.

Both these vectors and the labels taken from the batch will be blocked from back propagation and their data will be passed to the Numpy type of the CPU, instead of tensors in the GPU. Respectively, they will become the predictions and the real values to calculate the relevant metrics.

The **metrics** chosen to evaluate the model, of all those mentioned in Section 2.5 Evaluations and metrics: testing for improving, were *accuracy* and *F1-score*. To show in detail the calculations of both metrics, we will call the auxiliary function ***score***, which also provides a *classification report*. This will be used when we show the training stats right after the end of the training. Basically, it consists of plotting a graph of the training loss and validation loss across *epochs* and returning a Pandas DataFrame to visualize all training data and metrics. Here, Figure 3.7 depicts what has been explained about an epoch training and its validation:

```
======== Epoch 4 / 4 ========
Training...
  Batch    40  of    337.    Elapsed: 0:00:32.
  Batch    80  of    337.    Elapsed: 0:01:04.
  Batch   120  of    337.    Elapsed: 0:01:35.
  Batch   160  of    337.    Elapsed: 0:02:07.
  Batch   200  of    337.    Elapsed: 0:02:39.
  Batch   240  of    337.    Elapsed: 0:03:11.
  Batch   280  of    337.    Elapsed: 0:03:43.
  Batch   320  of    337.    Elapsed: 0:04:14.

  Average training loss: 0.22
  Training epoch took: 0:04:27

Running Validation...
  Accuracy: 0.91
  F1 macro: 0.75
  Validation Loss: 0.33
  Validation took: 0:00:22

Training complete!
Total training took 0:19:16 (h:mm:ss)
```

Figure 3.7: Final Epoch of Training and Validation for BERT model with Emoevent

3.4 Models implemented

For this concluding section of the Jupyter Notebooks structure, there will be a discussion of the pre-trained models selected from those discussed in Section 2.4 Deep Learning Approach with Transformers [40]. Since their technical explanation has already been provided above, our focus will be more on the specific models within each type, their tokenizers and whether they require complementary configurations.

3.4.1 BERT

The transformers library will provide us with all the models we need, along with their tokenizers and configurations to set up. In addition, each model tends to have two versions of it: a base version, much smaller and compact for quick and not so expensive training, and a long version, much heavier but at the same time much more complete.

If we focus on the BERT model, the BERT model's checkpoint we are interested in is *bert-base-uncased* [15]. Since resources are tight, we will not waste disk on the extended version of *bert-large* and we will stick to this model for 3 of the 4 datasets: HatEval, Detoxis and Emoevent. However, for Universal Joy, when moving to a multi-language task, this model does not work for us as it supports either English or Spanish and one language at a time. Therefore, the successor model will be *bert-case-multilingual-cased* [16] together with a sequence classification modified class to support multi-labels.

3.4.2 RoBERTa

As for RoBERTa, the situation is analogous to BERT's. For the first three datasets, there will be a pre-trained model common to all, which will be *roberta-base* [17]. Similarly to BERT, the predilection is for a reduced model in order to optimize resources and not saturate the disk. However, it does not support multiple languages at the same time and, therefore, it is necessary to switch to another pre-trained model that does contain a large number of languages: *xlm-roberta-base* [18]. It will be used to obtain the relevant tokenizer, together with the model and its configuration, which we will pass to a modification of the RoBERTa class for sequence classification. This modification, like BERT's one, consists of modifying its *forward* and *init* functions to support multi-label emotions.

3.4.3 XLNet

Finally, the XLNet model comes up with a slight inconvenience. This model is still quite young in the Natural Language Processing with Transformers field and its development is not as extensive as one would like it to be. At the moment, there are two separate pre-trained models developed for two languages: English and Chinese. In fact, the pre-trained model to be applied for the first 3 datasets will be *xlnet-base-cased* [19], of which there is also an extended version called *xlnet-large-cased*. However, for the Universal Joy dataset, no multilingual model is available, which makes it impossible to carry out this proposed task with XLNet given the resources available at the moment of writing this Bachelor's thesis.

Chapter 4

Results and Further Analysis

This chapter will illustrate in a straightforward, concise and intuitive manner the results obtained in the earlier chapter 3, the Implementation. This chapter will illustrate in a straightforward, concise and intuitive manner the results obtained in the earlier chapter.

4.1 Experiments Setup

In this section, the reader will find an overview of how the models and datasets have been configured to be able to launch the experiments. It should be noted that aspects related to the IDE, the language and the external setup of the Jupyter Notebooks, have been previously detailed in Section 3.1. Therefore, our focus will be on recreating the experiments detailed in the following sections. To do so, there are two concepts to understand: how the datasets are distributed and what parameters can be configured for the models.

Starting with the datasets, as we already know, these are subdivided into 3 parts, which are train, dev and test. Each one serves for the task that its name indicates, being training, validation and testing respectively. Although the rule is that the original dataset is already broken down into parts, it is necessary to highlight what are the percentages of each subdataset. First, the dev subdataset is a split of the train one, so first there will only be train-dev VS test. The test subdataset will represent 20% of the original dataset, compared to 80% of the training plus validation dataset. Thus, out from this 80%, the validation subdataset will be around 20 to 25% depending on the dataset. For instance, for Detoxis it has been defined to be 20%, whereas for Emoevent it is approximately 25%. This means that for testing we will get 20% of the data; for train, approximately 64%; and for dev, 16%.

Moving on to the model parameters, there are four parameters that can be scaled up or down as desired. These are the Learning rate, the Epsilon, the Batch size and the number of Epochs. For the experiments, both the Epsilon and the number of Epochs have been set to the same value across all notebooks and that is 4 epochs and $\epsilon = 1 \cdot 10^{-8}$. As for the Learning rate and Batch size, since they may vary considerably depending on the dataset and model to be trained, they can be consulted from the following table 4.1:

Parameters for each Model and Dataset													
Models	BERT		RoBERTa		XLNet		Models	BERT		RoBERTa		XLNet	
Datasets	Learning Rates	Batch Sizes	Learning Rates	Batch Sizes	Learning Rates	Batch Sizes	Datasets	Learning Rates	Batch Sizes	Learning Rates	Batch Sizes	Learning Rates	Batch Sizes
HatEval2019	$1 \cdot 10^{-4}$	16	$1 \cdot 10^{-5}$	16	$1 \cdot 10^{-4}$	8	Emoevent	$1 \cdot 10^{-4}$	8	$1 \cdot 10^{-5}$	4	$1 \cdot 10^{-4}$	2
	$3 \cdot 10^{-4}$	32	$2 \cdot 10^{-5}$	32	$2 \cdot 10^{-5}$	16		$3 \cdot 10^{-4}$	16	$2 \cdot 10^{-5}$	8	$2 \cdot 10^{-5}$	4
	$3 \cdot 10^{-5}$	64	$3 \cdot 10^{-5}$	64	$3 \cdot 10^{-5}$	32		$3 \cdot 10^{-5}$	32	$3 \cdot 10^{-5}$	16	$3 \cdot 10^{-5}$	8
	$5 \cdot 10^{-5}$		$5 \cdot 10^{-5}$		$5 \cdot 10^{-5}$			$5 \cdot 10^{-5}$		$5 \cdot 10^{-5}$		$5 \cdot 10^{-5}$	
Detoxis	$1 \cdot 10^{-4}$	4	$1 \cdot 10^{-5}$	4	$1 \cdot 10^{-4}$	1	Universal Joy	$1 \cdot 10^{-4}$	8	$1 \cdot 10^{-5}$	4	Not available	
	$3 \cdot 10^{-4}$	8	$2 \cdot 10^{-5}$	8	$2 \cdot 10^{-5}$	Not available		$2 \cdot 10^{-5}$	16	$2 \cdot 10^{-5}$	8		
	$3 \cdot 10^{-5}$	16	$3 \cdot 10^{-5}$	16	$3 \cdot 10^{-5}$			$3 \cdot 10^{-5}$	32	$3 \cdot 10^{-5}$	16		
	$5 \cdot 10^{-5}$		$5 \cdot 10^{-5}$		$5 \cdot 10^{-5}$			$5 \cdot 10^{-5}$		$5 \cdot 10^{-5}$			

Table 4.1: Summary Table for the Experiments

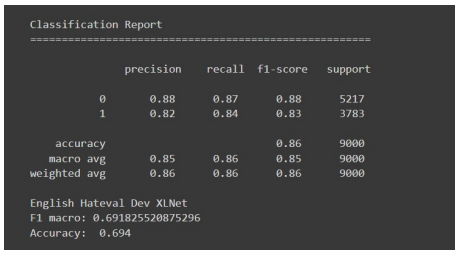
4.2 Early results of the model's training and evaluation

This section will discuss the overall performance of each task with the proposed models as well as a comparison among them. For each dataset, we will detail the best metrics achieved, problems that have come up as a consequence of their analysis and their causes.

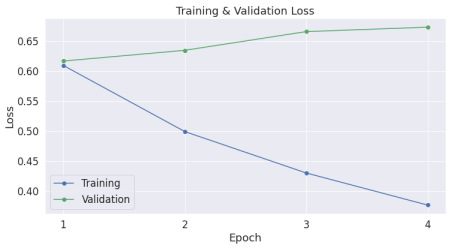
4.2.1 HatEval2019 Dataset

For HatEval2019, since both the English and Spanish versions are available, it has been decided that the English version should be used in order to be able to work properly with XLNet model. It should be pointed out that the task carried out with this dataset, which is to study and detect if there is hate speech in the tweets provided as text, has been the one that allowed the use of the **largest batches** in the whole training procedure (batches of 32 and even 64). Its simplicity and efficiency has prevented the three models from consuming excessively fast the RAM of the GPU, allowing a reasonable learning process.

If we focus on the **Dataset**, it is not the most imbalanced one to be encountered. In fact, it presents 58% of tweets with hatred compared to 42% without, as we can see in the classification report in Figure 4.1a. In this Figure, which shows the evaluation of the train dataset with XLNet (Learning Rate of $2 \cdot 10^{-5}$ and batch of 16), it can be seen that the learning has been satisfactory. However, it is not entirely balanced and this has generated a problem that will recur throughout the other tasks: overfitting (*seen in Figure 4.1b*).



(a) Classification Report with XLNet



(b) XLNet Loss Graph through epochs

Figure 4.1: HatEval2019 Dataset Status after training and validating

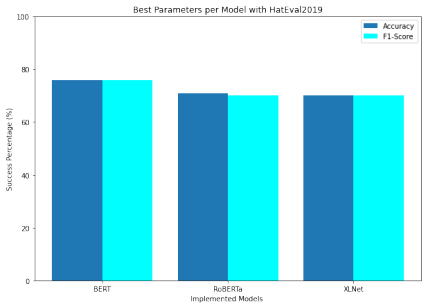
Overfitting is a modeling error in Statistics that occurs when a function fits too closely to a limited set of data items. As a result, the model is useful only in reference to its original dataset, for instance the training dataset that has already learned during training, and not to other datasets, like validation and test datasets. The clearest cases using Hateval, which are not too frequent, will be encountered using RoBERTa as a model. Taking a look at Figure 4.2, which can also be found with the rest of tables in Appendix A, smaller batches, such as batches of 16, and higher Learning Rates, such as $3 \cdot 10^{-5}$ and $5 \cdot 10^{-5}$, are more prone to this particular issue. The quick way to detect the lack of learning progression, as can be seen in the yellow trainings marked in Figure (*seen in Figure 4.2*), is by comparing the **evaluation metrics** of the training: Accuracy (Acc) and F1-Score.

RoBERTa with Hateval (4 epochs per training)													
Batch/LR		16				32				64			
Metrics		Acc	F1	Training Loss	Valid. Loss	Acc	F1	Training Loss	Valid. Loss	Acc	F1	Training Loss	Valid. Loss
$1 \cdot 10^{-5}$	1	0.65	0.58	0.67	0.64	0.70	0.69	0.49	0.59	0.68	0.68	0.42	0.67
	2	0.67	0.67	0.57	0.61	0.70	0.70	0.44	0.59	0.69	0.69	0.38	0.67
	3	0.64	0.64	0.51	0.66	0.70	0.70	0.42	0.59	0.67	0.66	0.35	0.72
	4	0.69	0.68	0.47	0.60	0.71	0.70	0.38	0.64	0.69	0.69	0.33	0.68
$2 \cdot 10^{-5}$	1	0.68	0.68	0.45	0.74	0.69	0.69	0.33	0.85	0.69	0.67	0.22	1.10
	2	0.70	0.67	0.36	0.88	0.69	0.68	0.24	0.77	0.68	0.68	0.17	0.99
	3	0.70	0.70	0.32	0.80	0.68	0.68	0.20	0.85	0.69	0.68	0.15	1.08
	4	0.70	0.70	0.25	0.92	0.69	0.68	0.16	1.03	0.67	0.67	0.10	1.22
$3 \cdot 10^{-5}$	1	0.57	0.36	0.69	0.68	0.57	0.36	0.70	0.69	0.57	0.36	0.69	0.68
	2	0.57	0.36	0.70	0.68	0.57	0.36	0.69	0.68	0.57	0.36	0.69	0.68
	3	0.57	0.36	0.69	0.68	0.57	0.36	0.69	0.68	0.57	0.36	0.69	0.68
	4	0.57	0.36	0.69	0.68	0.57	0.36	0.69	0.68	0.57	0.36	0.69	0.68
$5 \cdot 10^{-5}$	1	0.57	0.36	0.69	0.68	0.57	0.36	0.69	0.68	0.57	0.36	0.69	0.68
	2	0.57	0.36	0.69	0.68	0.57	0.36	0.69	0.68	0.57	0.36	0.69	0.68
	3	0.57	0.36	0.69	0.68	0.57	0.36	0.69	0.68	0.57	0.36	0.69	0.68
	4	0.57	0.36	0.69	0.68	0.57	0.36	0.69	0.68	0.57	0.36	0.69	0.68

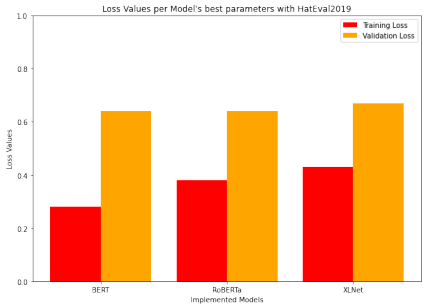
Figure 4.2: RoBERTa Overfitting during HatEval2019 Training (Batch size of 16, [I])

As it can be clearly noticed, the Accuracy and the F1-score remain unchanged during the four *epochs* of training, which is a great indicative. In addition, the **loss values**, both training and validation, are quite steady. Normally, if the model is undergoing a learning process, the training loss will tend to zero fairly quickly, while the validation loss decreases more gently. Here it can be seen that both remain stable, even increasing a slight percentage.

Focusing on the best parameters achieved by each model, it can be appreciated in the histogram 4.3a that all models have performed very similarly in the task of detecting hate speech. However, among them, BERT stands out a bit, with Accuracy and F1-score values of 0.76 both. As RoBERTa and XLNet are close behind, with both metrics around 0.70, this result is not meaningful as more epochs would allow them to catch up with BERT. In order to choose a model, one has to look at the losses plot (*seen in Figure 4.3b*) where, although the validation is pretty equal among them, the training loss is not.



(a) Metrics' Values Comparison



(b) Loss Values Comparison

Figure 4.3: Analysis of Models' Performance with HatEval2019 Task

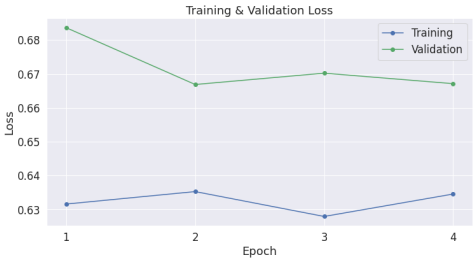
As a reminder, loss functions evaluate the deviation between the predictions realized by the NN and the true labels. The lower the result, the more efficiently the NN works. Therefore, it is desirable that both losses decrease as the model learns and they reach the lowest values. With that stated, it is evident that the best model for Hate Speech is **BERT**.

4.2.2 Detoxis Dataset

Moving on to Detoxis, it turns out that overfitting has already become a serious problem. As it can be seen from the *Appendix A.II Detoxis Tables*, the training strongly overfits for a simple reason: models learn that betting on a class will provide them with good scores. Even though this is not necessarily true once the results are observed, it is normal for the model. To understand this statement lets look at the ratios of classes in the datasets with their classification report in Figure 4.4a:

```
Classification Report
                    precision    recall  f1-score   support

                 0       0.67      1.00      0.81      1869
                 1       0.00      0.00      0.00       901

          accuracy                           0.67      2770
         macro avg       0.34      0.50      0.40      2770
      weighted avg       0.46      0.67      0.54      2770

BERT Dev Detoxis
F1 macro: 0.39210526315789473
Accuracy:  0.645021645021645
```

(a) Classification Report with BERT

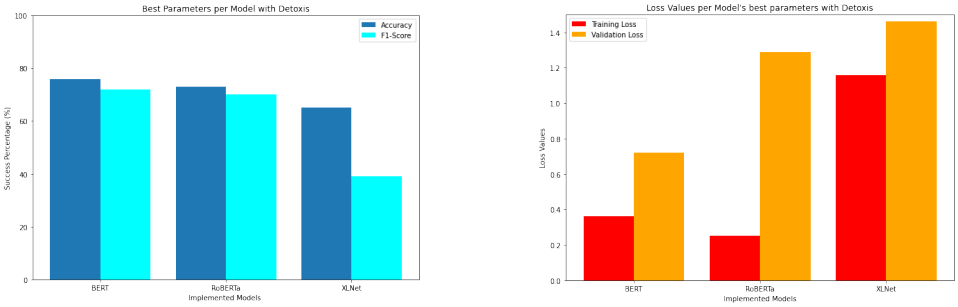


(b) BERT Loss Graph through epochs

Figure 4.4: Detoxis Dataset Status after training and validating

In this figure, the classification report of the BERT validation dataset with Detoxis shows that, for class 0 ("no toxicity in the text"), there are 1869 tweets classified as such. On the other hand, classified as "toxic" with class 1 there are just 901. In other words, there is a clear imbalance of 67.5% versus 32.5% between the classes, which means that almost two thirds of the train-dev-test datasets are made up of non-toxic tweets. Hence, it is normal for models to "bet" on saying that all input is non-toxic, since it is the majority.

Compared to HatEval2019, here overfitting is the default rule for all three models, except for some trainings where due to a low learning rate and a batch size of 8 (*seen in II Detoxis Tables*), which is relatively small, a gradual learning can be appreciated. It should be noted that this dataset with XLNet was too heavy for the GPU and the only batch size that was allowed with the available resources was 1. What is more, XLNet does not support other languages apart from English and Chinese and this dataset is made up with Spanish tweets. Therefore, it will not enter the discussion of the best model for Detoxis.



(a) Metrics' Values Comparison

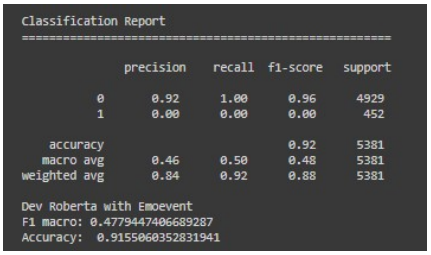
(b) Loss Values Comparison

Figure 4.5: Analysis of Models' Performance with Detoxis Task

Regarding BERT and RoBERTa, as can be seen in the graphs 4.5a and 4.5b, there is a dilemma. Both models in their few runs without overfitting show similar metrics: 73-76% for accuracy and 70-72% for F1-score. The highest metric values belong to BERT, but the best training loss belongs to RoBERTa, indicating that its learning is more efficient. Given that agile learning in few epochs is what matters and that, by adding a few more iterations of training, BERT values can be achieved, **RoBERTa** will be the best model for Detoxis.

4.2.3 Emoevent Dataset

Introducing datasets with information masking, here is Emoevent. As Detoxis, it exhibits overfitting. Nevertheless, the main difference is that here is extreme. To illustrate it, we shall study the classification report (*seen in Figure 4.6a*) for the validation dataset with RoBERTa. It is clearly observed in Figure 4.7b that learning is almost non-existent.



Classification Report

	precision	recall	f1-score	support
0	0.92	1.00	0.96	4929
1	0.00	0.00	0.00	452
accuracy			0.92	5381
macro avg	0.46	0.50	0.48	5381
weighted avg	0.84	0.92	0.88	5381

Dev Roberta with Emoevent
F1 macro: 0.4779447406689287
Accuracy: 0.9155060352831941

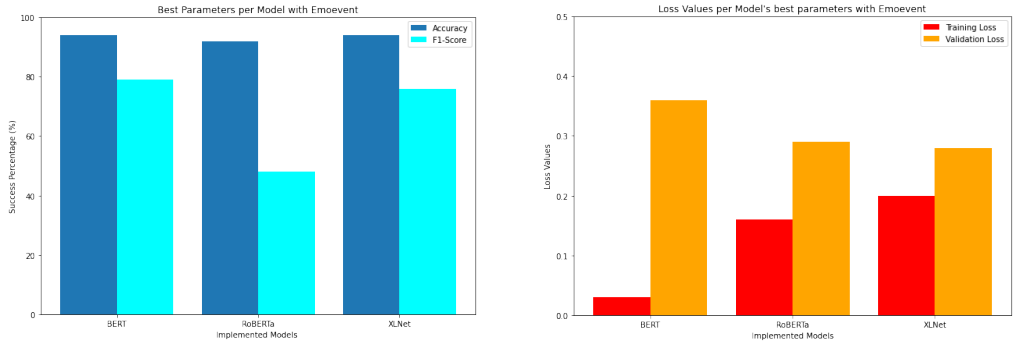
(a) Classification Report with RoBERTa

(b) RoBERTa Loss Graph through epochs

Figure 4.6: Emoevent Dataset Status after training and validating

Given that the proportion of non-offensive tweets is 92% vs. 8% of offensive ones, it is completely understandable that both losses tend to stay the same. What is more, all three models show the same values of accuracy, 92%, and F1-score, 48%, as can be observed in the last lines of Figure 4.6a. The only one with other overfitting values, surprisingly, is XLNet (*seen in Appendix A.III Emoevent Tables*). In spite of not having support for Spanish tweets, it is the model with the highest overfitting metrics: 93% accuracy and 68% F1-score.

For this dataset, there are only a couple of trainings among the three models that have surpassed the overfitting and they have been one with BERT and some more with XLNet. Therefore, if we compare them through Figure 4.7a, they have practically almost the same values in all metrics, which does not indicate anything remarkable. However, as always, loss values are revealing, as observed in Figure 4.7b. Regarding BERT, its training loss tends to zero rapidly, while XLNet has a more gradual learning process. On the other hand, during validation, BERT increases and reaches the highest validation losses, while XLNet remains in the average. This is caused by the fact that XLNet, not being prepared for Spanish datasets, encounters the same difficulty in training as in validating the dataset and makes the same mistakes.



(a) Metrics' Values Comparison

(b) Loss Values Comparison

Figure 4.7: Analysis of Models' Performance with Emoevent Task

However, BERT understands well its purpose in training by gradually studying errors and applying what has already learned in past iterations. Therefore, in validation at the end of each epoch, it performs poorly on new samples. This may indicate that the model is **underfitting**. Underfitting occurs when the model is unable to accurately model the data and, hence, generates large errors, as can be appreciated with such high validation loss in contrast of its extremely low train loss in Figure 4.7b.Its solution, as overfitting, will be tackled in Section 4.3.2 and it will consist of a new Sampler for Imbalanced Datasets, which takes into account both problems. Therefore, for now, **XLNet** will remain as the best model for Emoevent until this issues are resolved in section 4.3.

4.2.4 Universal Joy Dataset

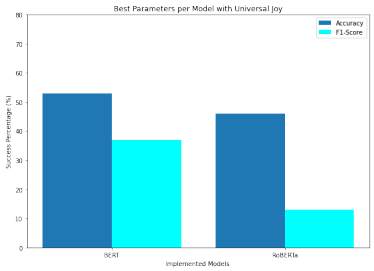
Finally, Universal Joy. This dataset has been taken as a challenge to study what can happen in case one moves from sentiment analysis to multi-class/lingual classification. Being the most different also entails having unique problems compared to its peers. First, Universal Joy presents Facebook posts in five languages, of which not all are contemplated in NLTK's **stopwords**. This drawback will be easily solved in Section 4.3.1 for better pre-processing. Another aspect not yet addressed is that **XLNet** is still under development and only features English and Chinese pre-trained versions. Therefore, other languages will be collaterally damaged as they are not supported. It is also not intended for multi-class classification. For this, XLNet will not form part of the benchmark.

Focusing on real problems, there are deficiencies. **Emotions** are not homogeneously distributed, regardless the language, and can overfit. Also, since it is necessary to analyze all languages, there is a need for larger batches to ensure they will appear. This can only be achieved with BERT's 32 batch size. Smaller ones usually make **Accuracy** drop to 0 (*seen in Figure 4.8*), because certain emotion have minimum representation, even less than 100 posts, and it is impossible to predict them. In the end, it goes into systematic failure although there are some good predictions that can be observed thanks to F1-scores.

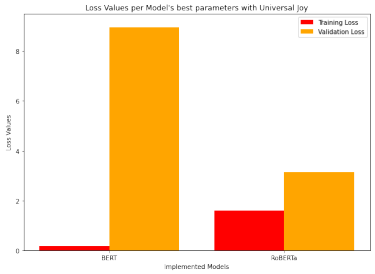
16				32			
Acc	F1 score	Training Loss	Valid. Loss	Acc	F1 score	Training Loss	Valid. Loss
0.28	0.09	1.49	3.20	0.22	0.37	0.85	3.16
0.00	0.19	1.50	2.75	0.33	0.35	0.71	3.45
0.00	0.27	1.49	2.93	0.35	0.36	0.48	4.27
0.00	0.21	1.48	2.96	0.38	0.34	0.29	4.86

Figure 4.8: Accuracy issue using BERT with 16-32 Batch Sizes and $1 \cdot 10^{-4}$ Learning Rate

Switching to the best achieved values, there is a conflict. The best metrics are undeniably BERT's, but its cost is expensive (*seen in Figure 4.9b*). RoBERTa, on the other hand, although it has high losses since there are some failures, they are not as big as BERT's validation. However, as seen in the RoBERTa table (IV), this model only allows small batches and the metrics collapse more often. Therefore, **BERT** will be the chosen one.



(a) Metrics' Values Comparison



(b) Loss Values Comparison

Figure 4.9: Analysis of Models' Performance with Universal Joy Task

4.3 Possible Solutions for the explained issues

In this section, all the problems mentioned in the previous section will be listed in an organized way as well as their possible solutions. In addition, those that have been carried out will be highlighted and the improvements they have brought about will be explained.

4.3.1 Hardware and Software problems

Hardware

As for the Hardware part, which is the problem that has led us to using Google Colab and the Physics Faculty cluster's services, it has a fairly obvious solution: a more powerful computer is required, in terms of GPU and RAM of the graphic card. Given that the available device has only 2GB of GRAM, the improvements are undeniable. However, as it is a regular laptop, the best would be a set-up with a tower where you can stack several graphic cards or install a powerful one.

Software

Certain models have been pre-trained in many tasks in a generic way within the NLP field (Text Classification, Next Sentence Prediction, etc.). However, if an excellent result is what one wants, perhaps it would be better to specialize the resolution of the problem. For instance, with Universal Joy, since several of its languages are not supported across the models, modifications would have to be made. If we focus on XLNet, the only languages it supports are English and Chinese, so the other texts would have to be translated into one of the enabled languages. However, this has not been carried out as it would disregard the initial objective of this dataset: check how language affects the classified emotions.

Following on the theme of poorly studied languages, the list of stopwords by language has been replaced by a more complete one: stopwordsiso. This collection presents all treated languages and improves the efficiency of the pre-processing respect to the NLTK list, which did not contain some Universal Joy's languages, such as Chinese or Tagalo.

4.3.2 Overfitting and Underfitting

The major concern of the whole Bachelor's thesis has been the overfitting of models by imbalanced datasets. In this section there will be a discussion of all the possible approaches that have been carried out in chronological order and why some have been rejected. For this study, since Emoevent is the dataset with the most pronounced imbalance, it has been chosen, alongside its notebook, as the one to perform the testing of the solutions proposed below.

Undersampling and Oversampling Datasets

Since all datasets, in a greater or lesser degree, are imbalanced, the first logical solution is to balance them. For this, techniques such as collecting more data from the original dataset can be applied, but this is not a viable option as these are datasets used for NLP competitions.

Therefore, only artificial balancing tactics are available, like oversampling and undersampling [45][59]. **Undersampling** is a strategy for balancing imbalanced datasets by maintaining all data in the minority class while reducing the size of the majority class. On the contrary, **oversampling** maintains all data in the majority class while increasing the size of the minority class. More specifically, a random oversampling/undersampling of the training, validation and test datasets will be performed using the *imblearn* library.

The main problem we encounter with a random technique is that the minority class is extremely small, specifically 8% of the entire dataset. For example, when we focus on oversampling, what is being performed is the creation of duplicates of the few tweets that are used as input for the "offensive" class. Therefore, once the model is trained with this balanced data, it will receive as offensive input the same 100-200 unique tweets that exist. The underlying problem is not as obvious as one might expect, since the results obtained are almost excellent, as can be seen in Figure 4.10. If we do a summary of what is happening, the original data has not changed, meaning that the same feature space with repeated samples is still there. In the end, overfitting is still present, but in a more subtle way.

BERT Emoevent Train
F1 macro: 0.9976668568471161
Accuracy: 0.9976668695475756

Classification Report

	precision	recall	f1-score	support
0	1.00	1.00	1.00	4929
1	1.00	1.00	1.00	4929
accuracy			1.00	9858
macro avg	1.00	1.00	1.00	9858
weighted avg	1.00	1.00	1.00	9858

BERT Emoevent Dev
F1 macro: 0.7675706401597966
Accuracy: 0.7765612327656123

Classification Report

	precision	recall	f1-score	support
0	0.70	0.97	0.81	1233
1	0.96	0.58	0.72	1233
accuracy			0.78	2466
macro avg	0.83	0.78	0.77	2466
weighted avg	0.83	0.78	0.77	2466

Figure 4.10: Emoevent oversampled, using BERT (16 Batch size and $3 \cdot 10^{-4}$ Learning rate)

On the other hand, the issue with random undersampling is clear. This technique randomly removes samples from the majority class to bring it to the level of the minority class. If the minority class only has around 100 tweets as input, the majority class will also have 100 and this is insufficient data for a proper training.

Synthetic Minority Oversampling Technique (SMOTE)

Since random oversampling-undersampling has not been an optimal solution, our next idea is to attempt a much more selective way of oversampling. For this purpose, SMOTE has been considered [66][43]: statistical approach for homogeneously balancing instances per class in a dataset. This tactic generates new instances based on the current minority class while majority class' examples do not change. However, unlike random oversampling that duplicates samples, this algorithm picks up existing samples of dataset's feature space that belong to the minority class and to its nearest neighbors [63]. Then, it generates new samples that blend features from the target class and its neighbors. This method expands the available features for the minority class while broadening the samples scope.

In order to be able to use it each few iterations, a modification has been made to the BertForSequenceClassification and RobertaForSequenceClassification classes of Transformers so that both can include a new pooler that incorporates the oversampling of SMOTE. Specifically, an instance of the new SmotePooler class, which itself is a modification of Transformers' Bert Pooler, will be added as the model pooler in its constructor. Furthermore, it will be used to calculate the logits, the dropout and the new oversampled labels of the forward function of the model.

After all, when it comes down to reality, since the smote oversample is done locally through iterations and there are very few offensive samples in the input, there are not always enough samples for it to work correctly. In other words, the imbalance is so severe that there are not always enough samples of the target class to be able to create new features with their neighbors. Therefore, the only thing this leads to is that the overfitting is still present at a higher cost in terms of losses, as can be seen in Figure 4.11. This last epoch of BERTForSequenceClassificationSmote's model shows that the training loss has been doubled up to 0.63, when overfitting training loss stays between 0.29 and 0.31. Also the model still bets all to the non-offensive class.

```
======== Epoch 4 / 4 ========
Training...
  Batch    40  of    337.    Elapsed: 0:00:34.
  Batch    80  of    337.    Elapsed: 0:01:09.
  Batch   120  of    337.    Elapsed: 0:01:42.
  Batch   160  of    337.    Elapsed: 0:02:15.
  Batch   200  of    337.    Elapsed: 0:02:48.
  Batch   240  of    337.    Elapsed: 0:03:23.
  Batch   280  of    337.    Elapsed: 0:03:53.
  Batch   320  of    337.    Elapsed: 0:04:28.

  Average training loss: 0.63
  Training epoch took: 0:04:42

Running Validation...
  Accuracy: 0.92
  F1 macro: 0.48
  Validation Loss: 0.29
  Validation took: 0:00:49
```

Figure 4.11: Emoevent locally oversampled with SMOTE-BERT'S

Imbalanced Dataset Sampler

The ultimate solution is a sampler for imbalanced datasets, developed by a group of researchers and uploaded to GitHub as open source [1]. Its main idea is to prevent models trained with imbalanced datasets from being biased towards the majority class just because it is the tendency when making predictions, which is our case. For solving this issue, researchers have developed a Dataloader sampler that performs **resampling**: strategy for balancing datasets by removing samples from the majority class (undersampling) and adding more examples from the minority class (oversampling).

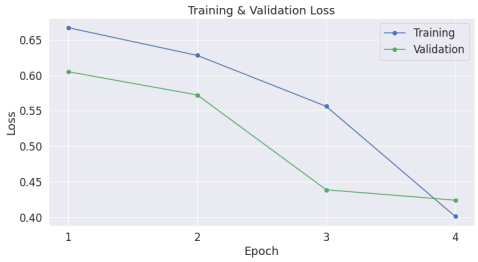
A part from rebalancing class distributions when sampling from our imbalanced dataset, the ***ImbalancedDatasetSampler*** also estimates the sampling weights automatically, avoids creating a brand new balanced dataset and mitigated overfitting when it is used in conjunction with data augmentation techniques. All in all, it will be places as the Train Dataloader's sampler and, for each epoch, it will sample the entire dataset and weigh it samples inversely to the class appearance probability.

Once the dynamics to be followed have been established, it is time for studying the results obtained from the training carried out. This training has been focused on the maximum possible batch size in BERT and XLNet models, since with RoBERTa the overfitting persists. This is due to the internal changes that the model has in comparison to BERT and its pre-training, making it too robust to be adapted to this third attempt for overcoming overfitting. Therefore, BERT allows a batch size of 16 and XLNet, of 8, as can be seen in its own table in Appendix B.

Starting with BERT, it initially shows excellent metrics, reaching 87% accuracy and 73% F1-score. It can be observed that it predicts correctly more than half of offensive samples (*seen in Figure 4.12a*), showing a substantial improvement. Furthermore, if we look at the loss progression graph (*seen in Figure 4.12b*), the training becomes a reality at last and both losses tend to zero quickly and efficiently. On the other hand, XLNet experiences the same change for the better but showing a more gradual learning. In addition, it has best predictions of the minority class: 69% (*seen in Figure 4.12c*). Therefore, **XLNet** would be the best model for the task.

```
BERT Emoevent Train
F1 macro: 0.7292752843169374
Accuracy:  0.872142724400669

Classification Report
=====================================================

              precision    recall  f1-score   support

           0       0.99      0.87      0.93      4929
           1       0.38      0.87      0.53       452

    accuracy                           0.87      5381
   macro avg       0.69      0.87      0.73      5381
weighted avg       0.94      0.87      0.89      5381
```

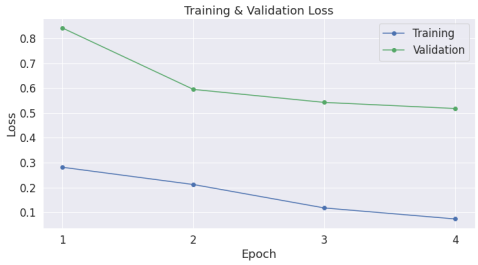
(a) Classification Report with BERT



(b) BERT Loss Graph through epochs

```
XLNet Emoevent Train
F1 macro: 0.8322570887246589
Accuracy:  0.9548411076008176

Classification Report
=====================================================

              precision    recall  f1-score   support

           0       0.96      0.99      0.98      4929
           1       0.82      0.60      0.69       452

    accuracy                           0.95      5381
   macro avg       0.89      0.79      0.83      5381
weighted avg       0.95      0.95      0.95      5381
```

(c) Classification Report with XLNet



(d) XLNet Loss Graph through epochs

Figure 4.12: Analysis of Model's Performance after solving the Overfitting issue

4.4 Transfer Learning

The end of this Bachelor's thesis is approaching and it is time to answer the last goal proposed at its beginning: what happens if we train models with one dataset and validate them with another one? After all, this would be transfer learning for investigating how the result is affected if the end of the model's training is not exactly with data from the same dataset already used. In more technical words, **transfer learning** is the application of knowledge gained from completing one task to help solve a different, but related, one.

As can be guessed, this technique has been constantly used in the entire project, since all models come from generic pre-trained ones. Now, it will be taken a step further and HatEval2019 and Detoxis will be used as our training-validation datasets, due to their high similarity. As for the code, it is almost the same as in previous notebooks, with minor modifications to handle the preprocessing of both together.

4.4.1 BERT using HatEval2019 for training and Detoxis for testing

Starting with BERT, as the title shows, HatEval2019 has been used as a training dataset on Hate Speech to then study its performance with Detoxis' Toxicity. It should be noted that, during training, BERT has obtained excellent results (*seen in Appendix B*) and with Detoxis it has achieved some of the best metrics in the study. However, this process has a high cost and, although learning is unquestionable, its validation loss is triggered.

Train Spanish Hateval and Detoxis with BERT
F1 macro: 0.9390312932606972
Accuracy: 0.9406666666666667

Classification Report

	precision	recall	f1-score	support
0	0.96	0.94	0.95	2643
1	0.92	0.94	0.93	1857
accuracy			0.94	4500
macro avg	0.94	0.94	0.94	4500
weighted avg	0.94	0.94	0.94	4500

(a) Classification Report for Training

Dev Spanish Hateval and Detoxis with BERT
F1 macro: 0.5447399420079739
Accuracy: 0.5815295815295816

Classification Report

	precision	recall	f1-score	support
0	0.68	0.67	0.67	447
1	0.41	0.42	0.42	246
accuracy			0.58	693
macro avg	0.54	0.54	0.54	693
weighted avg	0.58	0.58	0.58	693

(b) Classification Report for Evaluation

(c) BERT Loss Graph through epochs

Figure 4.13: Transfer learning Status after training and validating

This can be seen in Figure 4.13c, where it is observed that the training loss tends to zero quite efficiently while the validation loss even increases in epoch 3. To some degree this phenomenon is expected, since the model needs more than 4 epochs to understand that the dev subdataset does not perform exactly the same task as the train subdataset.

4.4.2 RoBERTa using Detoxis for training and HatEval2019 for testing

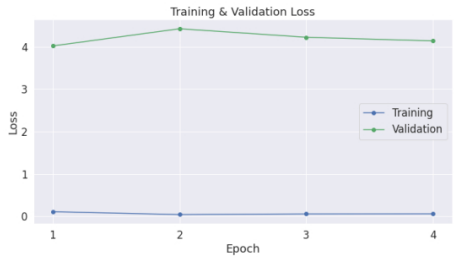
On the other hand, there is RoBERTa. With this model the reverse train-test process has been tested: Detoxis will be used as the training dataset and HatEval2019 will serve as the validation and testing dataset. Since Detoxis is much harder to pre-process and train, this means that the batch size will be affected. As seen in the RoBERTa table in Appendix B, the maximum size drops from 64, used with BERT in the previous section, to 16 due to GPU overload. As has been continuously proven throughout this Bachelor's thesis, the results improve as the input batches get larger and RoBERTa is a clear example.

```
Train Detoxis with RoBERTa
F1 macro: 0.9926147556766844
Accuracy:  0.9935018050541516

Classification Report
=====================================================

              precision    recall  f1-score   support

           0       1.00      0.99      1.00      1869
           1       0.99      0.99      0.99       901

    accuracy                           0.99      2770
   macro avg       0.99      0.99      0.99      2770
weighted avg       0.99      0.99      0.99      2770
```

(a) Classification Report for Training

```
Dev Spanish Hateval with RoBERTa
F1 macro: 0.5573608290371296
Accuracy:  0.558

Classification Report
=====================================================

              precision    recall  f1-score   support

           0       0.64      0.47      0.54       278
           1       0.50      0.67      0.57       222

    accuracy                           0.56       500
   macro avg       0.57      0.57      0.56       500
weighted avg       0.58      0.56      0.56       500
```

(b) Classification Report for Evaluation



(c) RoBERTa Loss Graph through epochs

Batch/LR		4			
Metrics		Acc	F1 score	Training Loss	Valid. Loss
$1 \cdot 10^{-5}$	1	0.54	0.53	0.10	4.16
	2	0.56	0.56	0.06	4.26
	3	0.55	0.55	0.08	4.33
	4	0.55	0.55	0.05	4.30
$2 \cdot 10^{-5}$	1	0.56	0.55	0.11	4.02
	2	0.57	0.57	0.04	4.42
	3	0.57	0.56	0.06	4.22
	4	0.56	0.56	0.06	4.14

(d) Example of overfitting with RoBERTa

Figure 4.14: Transfer learning Status after training and validating

Here the training results are perfect, but the validation drops considerably (*seen in Figure 4.14b*). Clearly, it plays an important role that there is little data available for testing, only about 220-270 tweets per class. However, the summary table shown in Figure 4.14d indicates a clear steadiness in learning, bordering on overfitting again. The problem is that, with the newly added data, the model does not manage to generalize well and it memorizes the patterns it finds. Hence, even if the Batch size or Learning rate are changed, the resulting metrics are quite alike and losses are very uneven, as it can be seen in Figure 4.14c for the losses and in Figure 4.14d for a general view.

4.4.3 XLNet using HatEval2019 for training and Detoxis for testing

Last but not least, XLNet did not shine much in Transfer Learning. It was by far the worst-performing model. In its case, training (*seen in Figure 4.15a*) went well enough without coming near to the results obtained by BERT or RoBERTa and, in terms of validation (*seen in Figure 4.15b*), its F1-score plummeted while the accuracy reached surprisingly good values. However, if one stops to analyze them, they are not a good indicator. In certain trainings, especially when the batch size is smaller (*seen in Figure 4.15d*), the overfitting is well appreciated as metrics that are obtained are twice the values normally achieved.

```
Train Spanish Hateval and Detoxis with XLNet
F1 macro: 0.7948798952962282
Accuracy:  0.798

Classification Report
======================================================

              precision    recall  f1-score   support

           0       0.86      0.78      0.82      2643
           1       0.73      0.82      0.77      1857

    accuracy                           0.80      4500
   macro avg       0.79      0.80      0.79      4500
weighted avg       0.80      0.80      0.80      4500
```

(a) Classification Report for Training

```
Dev Spanish Hateval and Detoxis with XLNet
F1 macro: 0.4670003232530402
Accuracy:  0.5021645021645021

Classification Report
======================================================

              precision    recall  f1-score   support

           0       0.62      0.59      0.60       447
           1       0.32      0.35      0.33       246

    accuracy                           0.50       693
   macro avg       0.47      0.47      0.47       693
weighted avg       0.51      0.50      0.51       693
```

(b) Classification Report for Evaluation

(c) XLNet Loss Graph through epochs

Batch/LR		8			
Metrics		Acc	F1 score	Training Loss	Valid. Loss
$1 \cdot 10^{-4}$	1	0.65	0.78	0.67	0.68
	2	0.38	0.32	0.67	0.72
	3	0.38	0.32	0.66	0.79
	4	0.38	0.32	0.66	0.72
$2 \cdot 10^{-5}$	1	0.38	0.30	0.65	0.73
	2	0.65	0.78	0.67	0.67
	3	0.38	0.30	0.67	0.71
	4	0.38	0.30	0.68	0.68

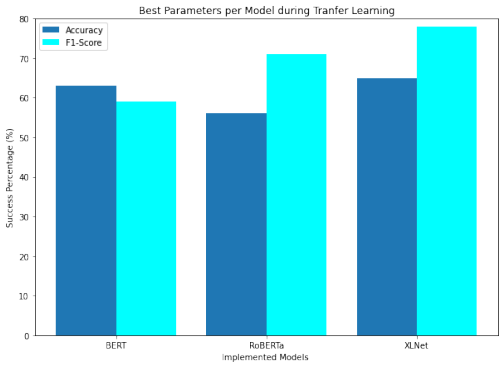
(d) Example of overfitting with XLNet

Figure 4.15: Transfer learning Status after training and validating

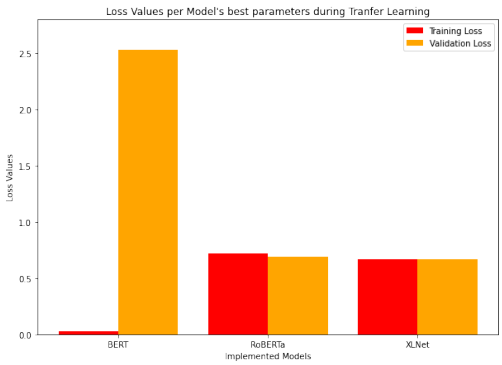
Moreover, like RoBERTa, XLNet does not generalize the new data accurately (*seen in Figure 4.15c*): the smaller the Batch and the higher the Learning rate, the worse the problem gets. Therefore, this model gives contradictory signals, since its average metric is much lower than its best values, obtained by pattern mechanization and overfitting.

4.4.4 Comparative among models

Once all three models and their difficulties have been seen, several points become clear. First, many more epochs with higher batch sizes are needed for observing real and consistent progress. However, for a quick study, 4 are quite revealing. Second, overfitting is a major issue to overcome and it tilts the balance towards one model: BERT, the only one that has shown good learning despite its high cost. Although it also indicates that this model suffers from some overfitting, it still learns to classify Detoxis toxicity relatively well. Applying the overfitting solutions described above, the validation loss will decrease.



(a) Metrics' Values Comparison



(b) Loss Values Comparison

Figure 4.16: Analysis of Models' Performance during Transfer Learning

Chapter 5

Conclusions

Having reached this point, a brief summary of what has been learned during this Bachelor's thesis will be made. Immediately afterwards, there will be detailed the future work that remains to be done and the objectives that will be established.
The take-home message from/of this research is that even small amounts of regular exercise make a huge difference to health.

What has been learned so far

This project is coming to an end and it has been a 6 month journey of **constant work**. Hours and hours invested in looking into the basic principles of NLP up to the state-of-the-art models used in the project. If we take a look back, the research of a large number of papers not only gave me back my fluency in reading technical information, but it also introduced me to a branch of Computer Science that had been overlooked during my 4 years of college.

As the fascination for the subject grew, the implementation came along and it began with the analysis of, at the time, only two datasets: HatEval2019 and Detoxis. As time went by and the project became more complex, the other major datasets were added to the mix and we completed the trio of models. In the end, dealing with so many factors helps to learn how each one performs. Hence the extensive study on overfitting, which has been crucial in the results of the project along with its analysis, closed the project cycle jointly with the Transfer Learning section. All in all, i can now affirm with full conviction and joy that all the objectives proposed in Section 1.3 of the Introduction have been **fulfilled**.

If I wonder what I would get out of this Bachelor's thesis, the **take-home message** would be that even the smallest details, like Learning Rates and Batch sizes, can make a huge difference on how the results come out and that Natural Language Processing is a deeply intriguing field that i am looking forward to getting lost in. Furthermore, as a **personal message** and as a reference to the very first section of this project (*see Section 1.1*), I can conclude that combining Reading and Computer Science is not a terrible idea. In fact, it is a wonderful and promising idea with a great horizon ahead. However, everything comes to an end, even if this Bachelor's thesis is not the permanent end of this project.

What the future holds for this project

After having looked in depth into sentiment analysis, with its ups and downs, there is no need to stay stuck on the same topic. Universal Joy has been, from the very beginning, the most mismatched dataset of all four and it was for a simple reason: it was multilingual and multi.class. Going forward, what will be studied and addressed is **multi-class analysis and classification**, along with better preprocessing of poorly supported languages in the state-of-the-art models. Given that social media are an infinite source of data and are becoming more and more crowded, input is completely secured. All that remains to be done is getting started.

Acknowledgments

To conclude this research as if I were a best-selling author, I would like to thank some of the people who have supported and helped me in this great journey. First of all, naturally, I would like to mention my mentor María Salamó and my co-mentor Alejandro Ariza for all the dedication and effort they have put into this project. For listening to each of my Friday presentations. For all the doubts that popped up and that they have always answered. For introducing me to this wonderful field that I will soon be working in.

Then, I would like to mention generally all the researchers who have carried out the numerous works, papers and articles mentioned above. Science is a precious resource for society and it should be promoted in each of its many fields and even more so with people who show such dedication and enthusiasm. In addition, I would like to thank my colleagues and friends who have lived through the whole process with me, from the early stress as the months went by to the joy of getting good results and closing chapters.

However, the person I truly would want to dedicate this Bachelor's thesis to is my father. Without him, the passion I have for reading and technology would not exist. For teaching me not to judge a book by its cover and that every story is a brand new adventure, I dedicate this to you. For being my reading partner until late at night and for being the first to tell me to pursue a technical career without being afraid of failure. From now on, a new chapter begins.

Appendix A

Initial Training-Evaluation Tables

In this first chapter of the Appendix A, one will find all the tables with the training outcomes mentioned above in chapters *3. Implementation* and *4. Results and Further Analysis*. It will be divided into four sections, one per dataset and, within them, there will be three tables corresponding to the training of the chosen models. Each one will be of four epochs per training and validation and will show:

1. **The Batch sizes**: it could be from 1 to 64, (1, 2, 4, 8, 16, 32, 64).
2. **The Learning Rate**: it can be found in the left side column.
3. **Metrics and Losses**: they can be found below the batch sizes ans they will be:
    - Accuracy (shown as *ACC*)
    - F1-score (shown as *F1*)
    - Training Loss
    - Validation Loss (shown as *Valid. Loss*)

,

It should be noted that every table will have two epochs in different colors. On the one hand, if the epoch is highlighted in electric blue, this will be the epoch with the best metric values in the whole experiment or table. On the other hand, if it is highlighted in a reddish-purple, this is an instance of overfitting to a greater or lesser degree. This last color may not appear if there is no overfitting.

I HatEval2019 Tables

BERT with Hateval (4 epochs per training)													
Batch/LR		16				32				64			
Metrics		Acc	F1	Training Loss	Valid. Loss	Acc	F1	Training Loss	Valid. Loss	Acc	F1	Training Loss	Valid. Loss
$1 \cdot 10^{-4}$	1	0.71	0.70	0.57	0.58	0.72	0.72	0.21	0.83	0.73	0.72	0.10	1.05
	2	0.71	0.71	0.41	0.63	0.72	0.71	0.14	1.00	0.71	0.71	0.06	1.23
	3	0.72	0.72	0.26	0.90	0.72	0.72	0.07	1.17	0.73	0.72	0.04	1.35
	4	0.73	0.73	0.15	1.06	0.73	0.72	0.05	1.23	0.73	0.73	0.03	1.32
$3 \cdot 10^{-4}$	1	**0.57**	**0.73**	**0.69**	**0.68**	0.57	0.73	0.70	0.68	0.57	0.73	0.65	0.69
	2	0.57	0.73	0.69	0.68	0.57	0.73	0.69	0.69	0.57	0.73	0.69	0.69
	3	0.57	0.73	0.69	0.68	0.57	0.73	0.69	0.68	0.57	0.73	0.68	0.68
	4	0.57	0.73	0.69	0.68	0.57	0.73	0.68	0.68	0.57	0.73	0.68	0.68
$3 \cdot 10^{-5}$	1	0.73	0.73	0.52	0.52	0.75	0.74	0.52	0.53	0.73	0.73	0.13	0.95
	2	0.75	0.75	0.33	0.59	0.75	0.75	0.35	0.54	0.74	0.74	0.07	1.12
	3	0.74	0.74	0.17	0.97	0.75	0.75	0.20	0.69	0.73	0.73	0.04	1.29
	4	0.76	0.75	0.09	1.19	0.75	0.75	0.10	0.87	0.74	0.74	0.02	1.38
$5 \cdot 10^{-5}$	1	0.74	0.73	0.49	0.54	0.74	0.74	0.50	0.52	0.74	0.73	0.07	1.22
	2	**0.76**	**0.76**	**0.28**	**0.64**	0.74	0.73	0.30	0.56	0.75	0.75	0.05	1.17
	3	0.75	0.75	0.11	1.05	0.74	0.74	0.14	0.87	0.75	0.74	0.03	1.29
	4	0.74	0.74	0.04	1.65	0.74	0.74	0.06	1.08	0.75	0.74	0.01	1.41

RoBERTa with Hateval (4 epochs per training)													
Batch/LR		16				32				64			
Metrics		Acc	F1	Training Loss	Valid. Loss	Acc	F1	Training Loss	Valid. Loss	Acc	F1	Training Loss	Valid. Loss
$1 \cdot 10^{-5}$	1	0.65	0.58	0.67	0.64	0.70	0.69	0.49	0.59	0.68	0.68	0.42	0.67
	2	0.67	0.67	0.57	0.61	0.70	0.70	0.44	0.59	0.69	0.69	0.38	0.67
	3	0.64	0.64	0.51	0.66	0.70	0.70	0.42	0.59	0.67	0.66	0.35	0.72
	4	0.69	0.68	0.47	0.60	**0.71**	**0.70**	**0.38**	**0.64**	0.69	0.69	0.33	0.68
$2 \cdot 10^{-5}$	1	0.68	0.68	0.45	0.74	0.69	0.69	0.33	0.85	0.69	0.67	0.22	1.10
	2	0.70	0.67	0.36	0.88	0.69	0.68	0.24	0.77	0.68	0.68	0.17	0.99
	3	0.70	0.70	0.32	0.80	0.68	0.68	0.20	0.85	0.69	0.68	0.15	1.08
	4	0.70	0.70	0.25	0.92	0.69	0.68	0.16	1.03	0.67	0.67	0.10	1.22
$3 \cdot 10^{-5}$	1	**0.57**	**0.36**	**0.69**	**0.68**	0.57	0.36	0.70	0.69	0.57	0.36	0.69	0.68
	2	0.57	0.36	0.70	0.68	0.57	0.36	0.69	0.68	0.57	0.36	0.69	0.68
	3	0.57	0.36	0.69	0.68	0.57	0.36	0.69	0.68	0.57	0.36	0.69	0.68
	4	0.57	0.36	0.69	0.68	0.57	0.36	0.69	0.68	0.57	0.36	0.69	0.68
$5 \cdot 10^{-5}$	1	0.57	0.36	0.69	0.68	0.57	0.36	0.69	0.68	0.57	0.36	0.69	0.68
	2	0.57	0.36	0.69	0.68	0.57	0.36	0.69	0.68	0.57	0.36	0.69	0.68
	3	0.57	0.36	0.69	0.68	0.57	0.36	0.69	0.68	0.57	0.36	0.69	0.68
	4	0.57	0.36	0.69	0.68	0.57	0.36	0.69	0.68	0.57	0.36	0.69	0.68

XLNet with Hateval (4 epochs per training)													
Batch/LR		8				16				32			
Metrics		Acc	F1 score	Training Loss	Valid. Loss	Acc	F1 score	Training Loss	Valid. Loss	Acc	F1 score	Training Loss	Valid. Loss
$1 \cdot 10^{-4}$	1	0.53	0.50	0.56	0.74	0.64	0.61	0.65	0.64	0.63	0.62	0.58	0.74
	2	0.62	0.62	0.61	0.65	0.63	0.62	0.64	0.64	0.65	0.65	0.58	0.65
	3	0.61	0.61	0.62	0.64	0.64	0.63	0.64	0.64	0.65	0.63	0.57	0.65
	4	0.66	0.63	0.63	0.64	0.64	0.63	0.63	0.64	0.65	0.65	0.54	0.65
$2 \cdot 10^{-5}$	1	0.57	0.73	0.67	0.69	0.68	0.67	0.61	0.62	0.61	0.54	0.64	0.68
	2	0.58	0.45	0.68	0.68	0.69	0.68	0.50	0.63	0.62	0.56	0.63	0.68
	3	0.63	0.60	0.66	0.66	**0.70**	**0.70**	**0.43**	**0.67**	0.62	0.57	0.62	0.68
	4	0.63	0.61	0.65	0.66	0.69	0.69	0.38	0.67	0.60	0.53	0.63	0.67
$3 \cdot 10^{-5}$	1	0.57	0.73	0.67	0.68	**0.69**	**0.67**	**0.61**	**0.61**	0.60	0.53	0.63	0.68
	2	0.60	0.60	0.66	0.72	0.69	0.69	0.49	0.67	0.61	0.51	0.64	0.68
	3	0.60	0.60	0.64	0.65	0.69	0.69	0.41	0.67	0.58	0.38	0.67	0.68
	4	0.60	0.56	0.64	0.66	0.69	0.69	0.35	0.70	0.61	0.51	0.64	0.68
$5 \cdot 10^{-5}$	1	0.59	0.52	0.61	0.65	0.66	0.63	0.64	0.64	0.63	0.62	0.63	0.69
	2	0.62	0.62	0.61	0.66	0.64	0.62	0.63	0.65	0.63	0.60	0.60	0.66
	3	0.61	0.61	0.60	0.68	0.63	0.62	0.63	0.65	0.63	0.61	0.58	0.66
	4	0.63	0.61	0.59	0.66	0.64	0.63	0.62	0.65	0.62	0.60	0.57	0.67

II Detoxis Tables

BERT with Detoxis (4 epochs per training)													
Batch/LR		4				8				16			
Metrics		Acc	F1 score	Training Loss	Valid. Loss	Acc	F1 score	Training Loss	Valid. Loss	Acc	F1 score	Training Loss	Valid. Loss
$1 \cdot 10^{-4}$	1	**0.65**	**0.39**	**0.65**	**0.70**	0.65	0.78	0.65	0.71	0.65	0.78	0.64	0.65
	2	0.65	0.39	0.64	0.67	0.65	0.78	0.65	0.65	0.65	0.78	0.63	0.66
	3	0.65	0.39	0.64	0.69	0.65	0.78	0.64	0.66	0.65	0.78	0.64	0.65
	4	0.65	0.39	0.64	0.67	0.65	0.78	0.64	0.65	0.65	0.78	0.63	0.66
$3 \cdot 10^{-4}$	1	0.65	0.39	0.65	0.68	0.65	0.78	0.67	0.76	0.65	0.39	0.65	0.70
	2	0.65	0.39	0.64	0.74	0.65	0.78	0.65	0.65	0.65	0.39	0.64	0.65
	3	0.65	0.39	0.64	0.69	0.65	0.78	0.65	0.67	0.65	0.39	0.64	0.65
	4	0.65	0.39	0.64	0.67	0.65	0.78	0.65	0.64	0.65	0.39	0.63	0.66
$3 \cdot 10^{-5}$	1	0.65	0.39	0.63	0.68	0.65	0.41	0.63	0.67	0.65	0.39	0.62	0.66
	2	0.65	0.39	0.64	0.67	0.72	0.65	0.60	0.56	0.65	0.39	0.62	0.66
	3	0.65	0.39	0.63	0.67	0.74	0.66	0.50	0.58	0.65	0.39	0.63	0.65
	4	0.65	0.39	0.63	0.67	0.76	0.72	0.36	0.72	0.65	0.39	0.63	0.66
$5 \cdot 10^{-5}$	1	0.65	0.39	0.63	0.69	0.72	0.63	0.48	0.63	0.65	0.39	0.62	0.66
	2	0.65	0.39	0.62	0.69	**0.76**	**0.73**	**0.35**	**1.09**	0.65	0.39	0.59	0.66
	3	0.65	0.39	0.63	0.67	0.74	0.72	0.32	1.05	0.65	0.39	0.61	0.65
	4	0.65	0.39	0.64	0.67	0.76	0.72	0.26	1.18	0.65	0.39	0.64	0.66

RoBERTa with Detoxis (4 epochs per training)													
Batch/LR		4				8				16			
Metrics		Acc	F1 score	Training Loss	Valid. Loss	Acc	F1 score	Training Loss	Valid. Loss	Acc	F1 score	Training Loss	Valid. Loss
$1 \cdot 10^{-5}$	1	0.72	0.67	0.52	1.03	0.65	0.39	0.65	0.72	0.72	0.68	0.31	0.80
	2	0.72	0.67	0.45	1.26	0.66	0.44	0.63	0.66	0.73	0.69	0.24	0.95
	3	0.72	0.66	0.38	1.28	0.68	0.51	0.58	0.68	0.73	0.68	0.25	0.95
	4	0.72	0.67	0.34	1.37	0.68	0.59	0.53	0.62	0.73	0.69	0.20	0.97
$2 \cdot 10^{-5}$	1	0.65	0.39	0.63	0.64	0.67	0.60	0.59	0.59	0.65	0.39	0.61	0.79
	2	0.68	0.48	0.62	0.81	0.70	0.55	0.53	0.66	0.68	0.54	0.55	0.61
	3	0.70	0.56	0.54	0.71	0.70	0.67	0.48	0.66	0.69	0.53	0.49	0.71
	4	0.71	0.65	0.55	0.95	0.71	0.68	0.41	0.70	0.72	0.66	0.46	0.62
$3 \cdot 10^{-5}$	1	0.70	0.66	0.64	0.76	0.70	0.57	0.62	0.83	0.66	0.43	0.56	0.61
	2	0.71	0.61	0.60	0.92	0.68	0.51	0.50	0.96	0.68	0.52	0.45	0.76
	3	0.73	0.69	0.53	1.10	0.71	0.63	0.47	0.95	0.70	0.59	0.40	0.68
	4	0.72	0.68	0.43	1.34	0.72	0.67	0.38	1.10	0.73	0.69	0.34	0.68
$5 \cdot 10^{-5}$	1	0.65	0.39	0.61	0.68	0.72	0.67	0.56	0.67	0.65	0.39	0.67	0.74
	2	**0.65**	**0.39**	**0.63**	**0.69**	0.73	0.66	0.45	0.68	0.65	0.39	0.63	0.65
	3	0.65	0.39	0.63	0.71	0.73	0.69	0.43	1.15	0.65	0.39	0.64	0.66
	4	0.65	0.39	0.64	0.66	**0.73**	**0.70**	**0.25**	**1.29**	0.65	0.39	0.64	0.66

XLNet with Detoxis (4 epochs per training)													
Batch/LR		1				2				4			
Metrics		Acc	F1 score	Training Loss	Valid. Loss	Acc	F1 score	Training Loss	Valid. Loss	Acc	F1 score	Training Loss	Valid. Loss
$1 \cdot 10^{-4}$	1	**0.65**	**0.39**	**1.36**	**1.55**	No disponible por falta de GPUs							
	2	0.65	0.39	1.38	1.70								
	3	0.65	0.39	1.39	1.63								
	4	0.65	0.39	1.39	1.54								
$2 \cdot 10^{-5}$	1	0.65	0.39	1.36	1.53								
	2	0.65	0.39	1.36	1.70								
	3	0.65	0.39	1.31	1.56								
	4	**0.65**	**0.39**	**1.16**	**1.46**								
$3 \cdot 10^{-5}$	1	0.65	0.39	1.36	1.54								
	2	0.65	0.39	1.36	1.68								
	3	0.65	0.39	1.36	1.61								
	4	0.65	0.39	1.35	1.53								
$5 \cdot 10^{-5}$	1	0.65	0.39	1.34	1.57								
	2	0.65	0.39	1.37	1.69								
	3	0.65	0.39	1.38	1.62								
	4	0.65	0.39	1.38	1.53								

III Emoevent Tables

BERT with Emoevent (4 epochs per training)													
Batch/LR		8				16				32			
Metrics		Acc	F1 score	Training Loss	Valid. Loss	Acc	F1 score	Training Loss	Valid. Loss	Acc	F1 score	Training Loss	Valid. Loss
$1 \cdot 10^{-4}$	1	**0.92**	**0.48**	**0.30**	**0.29**	0.92	0.48	0.30	0.29	0.92	0.48	0.30	0.29
	2	0.92	0.48	0.30	0.29	0.92	0.48	0.30	0.29	0.92	0.48	0.30	0.29
	3	0.92	0.48	0.28	0.29	0.92	0.48	0.29	0.29	0.92	0.48	0.29	0.29
	4	0.92	0.48	0.27	0.29	0.92	0.48	0.29	0.29	0.92	0.48	0.29	0.27
$3 \cdot 10^{-4}$	1	0.92	0.48	0.30	0.29	0.92	0.48	0.30	0.29	0.92	0.48	0.30	0.29
	2	0.92	0.48	0.30	0.29	0.92	0.48	0.29	0.29	0.92	0.48	0.30	0.29
	3	0.92	0.48	0.29	0.29	0.92	0.48	0.29	0.29	0.92	0.48	0.29	0.29
	4	0.92	0.48	0.29	0.29	0.92	0.48	0.29	0.29	0.92	0.48	0.29	0.29
$3 \cdot 10^{-5}$	1	0.92	0.48	0.31	0.28	0.94	0.80	0.12	0.27	0.92	0.48	0.30	0.26
	2	0.93	0.71	0.27	0.23	**0.94**	**0.80**	**0.06**	**0.25**	0.92	0.48	0.30	0.23
	3	0.94	0.78	0.21	0.25	0.94	0.79	0.04	0.34	0.92	0.48	0.29	0.21
	4	0.94	0.78	0.17	0.26	0.94	0.79	0.03	0.36	0.92	0.48	0.29	0.17
$5 \cdot 10^{-5}$	1	0.92	0.48	0.30	0.29	0.93	0.69	0.37	0.23	0.92	0.48	0.30	0.29
	2	0.92	0.48	0.30	0.29	0.94	0.74	0.20	0.23	0.92	0.48	0.30	0.27
	3	0.92	0.48	0.27	0.29	0.94	0.77	0.16	0.24	0.92	0.48	0.29	0.23
	4	0.92	0.48	0.21	0.29	0.93	0.78	0.10	0.23	0.92	0.48	0.29	0.21

RoBERTa with Emoevent (4 epochs per training)													
Batch/LR		4				8				16			
Metrics		Acc	F1 score	Training Loss	Valid. Loss	Acc	F1 score	Training Loss	Valid. Loss	Acc	F1 score	Training Loss	Valid. Loss
$1 \cdot 10^{-5}$	1	**0.92**	**0.48**	**0.32**	**0.29**	0.92	0.48	0.33	0.29	0.92	0.48	0.31	0.29
	2	0.92	0.48	0.32	0.31	0.92	0.48	0.32	0.31	0.92	0.48	0.29	0.30
	3	0.92	0.48	0.31	0.34	0.92	0.48	0.31	0.34	0.92	0.48	0.29	0.29
	4	0.92	0.48	0.32	0.32	0.92	0.48	0.33	0.32	0.92	0.48	0.29	0.29
$2 \cdot 10^{-5}$	1	0.92	0.48	0.32	0.30	0.92	0.48	0.31	0.31	0.92	0.48	0.30	0.29
	2	0.92	0.48	0.32	0.29	0.92	0.48	0.31	0.33	0.92	0.48	0.30	0.29
	3	0.92	0.48	0.33	0.29	0.92	0.48	0.29	0.30	0.92	0.48	0.29	0.29
	4	0.92	0.48	0.32	0.29	0.92	0.48	0.26	0.29	0.92	0.48	0.30	0.29
$3 \cdot 10^{-5}$	1	0.92	0.48	0.29	0.26	0.92	0.48	0.30	0.29	0.92	0.48	0.30	0.30
	2	0.92	0.48	0.27	0.25	0.92	0.48	0.29	0.29	0.92	0.48	0.30	0.32
	3	0.92	0.48	0.26	0.28	0.92	0.48	0.29	0.30	0.92	0.48	0.29	0.33
	4	0.92	0.48	0.26	0.28	0.92	0.48	0.27	0.31	0.92	0.48	0.29	0.31
$5 \cdot 10^{-5}$	1	0.92	0.48	0.27	0.30	0.92	0.48	0.26	0.29	0.92	0.48	0.27	0.29
	2	0.92	0.48	0.22	0.31	0.92	0.48	0.22	0.33	0.92	0.48	0.24	0.27
	3	0.92	0.48	0.17	0.30	0.92	0.48	0.19	0.31	0.92	0.48	0.21	0.29
	4	**0.92**	**0.48**	**0.16**	**0.29**	0.92	0.48	0.18	0.27	0.92	0.48	0.18	0.23

XLNet with Emoevent (4 epochs per training)													
Batch/LR		2				4				8			
Metrics		Acc	F1 score	Training Loss	Valid. Loss	Acc	F1 score	Training Loss	Valid. Loss	Acc	F1 score	Training Loss	Valid. Loss
$1 \cdot 10^{-4}$	1	**0.93**	**0.68**	**0.38**	**0.36**	0.92	0.48	0.38	0.37	0.92	0.48	0.33	0.29
	2	0.93	0.68	0.38	0.36	0.92	0.48	0.39	0.36	0.92	0.48	0.35	0.31
	3	0.93	0.68	0.38	0.38	0.92	0.48	0.38	0.40	0.92	0.48	0.31	0.34
	4	0.93	0.68	0.38	0.36	0.92	0.48	0.38	0.38	0.92	0.48	0.32	0.32
$2 \cdot 10^{-5}$	1	0.93	0.68	0.37	0.36	0.92	0.50	0.37	0.36	0.92	0.68	0.31	0.23
	2	0.93	0.68	0.37	0.35	0.93	0.74	0.30	0.28	0.94	0.75	0.35	0.20
	3	0.93	0.68	0.37	0.37	0.92	0.48	0.30	0.31	**0.94**	**0.76**	**0.20**	**0.28**
	4	0.93	0.68	0.38	0.37	0.92	0.48	0.29	0.30	0.93	0.75	0.19	0.26
$3 \cdot 10^{-5}$	1	0.93	0.68	0.37	0.35	0.92	0.48	0.38	0.37	0.92	0.48	0.23	0.25
	2	0.93	0.68	0.37	0.35	0.92	0.55	0.36	0.32	0.93	0.73	0.18	0.24
	3	0.93	0.68	0.37	0.38	0.93	0.74	0.28	0.31	0.94	0.73	0.15	0.33
	4	0.93	0.68	0.37	0.37	0.93	0.76	0.25	0.29	0.94	0.74	0.12	0.32
$5 \cdot 10^{-5}$	1	0.93	0.68	0.36	0.35	0.92	0.48	0.35	0.39	0.93	0.74	0.22	0.24
	2	0.93	0.68	0.36	0.36	0.93	0.70	0.34	0.30	0.90	0.72	0.21	0.29
	3	0.93	0.68	0.36	0.38	0.93	0.69	0.32	0.33	0.92	0.48	0.23	0.29
	4	0.93	0.68	0.38	0.37	0.93	0.69	0.32	0.32	0.92	0.48	0.29	0.30

IV Universal Joy Tables

BERT with Universal Joy (4 epochs per training)													
Batch / LR		8				16				32			
Metrics		Acc	F1 score	Training Loss	Valid. Loss	Acc	F1 score	Training Loss	Valid. Loss	Acc	F1 score	Training Loss	Valid. Loss
$1 \cdot 10^{-4}$	1	0.00	0.26	1.37	3.62	0.28	0.09	1.49	3.20	0.22	0.37	0.85	3.16
	2	0.00	0.19	1.59	3.19	0.00	0.19	1.50	2.75	0.33	0.35	0.71	3.45
	3	0.00	0.21	1.62	3.31	0.00	0.27	1.49	2.93	0.35	0.36	0.48	4.27
	4	0.00	0.21	1.61	3.35	0.00	0.21	1.48	2.96	0.38	0.34	0.29	4.86
$2 \cdot 10^{-5}$	1	0.46	0.33	0.23	9.34	**0.53**	**0.37**	**0.16**	**8.95**	0.23	0.36	0.79	2.74
	2	0.45	0.33	0.15	9.59	0.44	0.40	0.54	6.10	0.35	0.41	0.74	3.30
	3	0.47	0.34	0.10	9.54	0.50	0.41	0.22	6.92	0.35	0.41	0.54	3.80
	4	0.47	0.34	0.04	9.78	0.51	0.41	0.13	7.61	0.38	0.40	0.43	3.92
$3 \cdot 10^{-5}$	1	**0.00**	**0.21**	**1.60**	**3.37**	0.27	0.39	1.07	2.78	0.40	0.35	0.22	6.15
	2	0.00	0.19	1.59	3.08	0.33	0.41	0.85	2.01	0.43	0.34	0.14	6.31
	3	0.00	0.27	1.58	3.30	0.37	0.42	0.61	3.29	0.45	0.36	0.08	7.32
	4	0.00	0.21	1.60	3.32	0.42	0.40	0.43	3.79	0.44	0.36	0.07	7.25
$5 \cdot 10^{-5}$	1	0.00	0.21	1.60	3.37	0.26	0.15	1.48	3.15	0.42	0.36	0.11	7.76
	2	0.00	0.19	1.59	3.08	0.00	0.19	1.48	2.66	0.44	0.34	0.15	7.68
	3	0.00	0.27	1.58	3.30	0.00	0.15	1.46	2.86	0.49	0.36	0.08	8.69
	4	0.00	0.21	1.60	3.32	0.00	0.21	1.47	2.90	0.45	0.36	0.06	8.10

RoBERTa with Universal Joy (4 epochs per training)													
Batch / LR		4				8				16			
Metrics		Acc	F1 score	Training Loss	Valid. Loss	Acc	F1 score	Training Loss	Valid. Loss	Acc	F1 score	Training Loss	Valid. Loss
$1 \cdot 10^{-5}$	1	0.00	0.21	1.78	3.68	0.00	0.21	1.62	3.43	0.00	0.21	1.54	3.04
	2	**0.47**	**0.13**	**1.80**	**3.64**	0.00	0.19	1.62	3.25	0.45	0.12	1.52	2.86
	3	0.00	0.21	1.79	3.72	0.00	0.21	1.61	3.47	0.00	0.27	1.47	3.04
	4	0.00	0.21	1.80	3.71	0.00	0.21	1.62	3.40	0.00	0.21	1.49	3.03
$2 \cdot 10^{-5}$	1	0.00	0.21	1.81	3.78	0.00	0.21	1.60	3.30	0.00	0.21	1.49	3.02
	2	0.47	0.13	1.81	3.53	0.46	0.13	1.60	3.24	0.45	0.12	1.48	2.83
	3	0.00	0.21	1.79	3.76	0.00	0.27	1.60	3.39	0.00	0.27	1.47	3.03
	4	0.47	0.13	1.81	3.68	0.00	0.21	1.62	3.39	0.00	0.21	1.48	3.01
$3 \cdot 10^{-5}$	1	0.00	0.21	1.82	3.96	0.00	0.21	1.58	3.35	0.00	0.21	1.49	3.03
	2	**0.00**	**0.21**	**1.82**	**3.74**	0.46	0.13	1.60	3.18	0.45	0.12	1.48	2.82
	3	0.00	0.21	1.80	4.08	0.00	0.27	1.59	3.38	0.00	0.21	1.47	3.04
	4	0.00	0.21	1.82	3.93	0.00	0.21	1.61	3.35	0.00	0.21	1.48	3.01
$5 \cdot 10^{-5}$	1	0.00	0.21	1.77	3.80	0.00	0.21	1.62	3.41	0.00	0.21	1.50	3.08
	2	0.00	0.21	1.79	3.68	0.46	0.13	1.60	3.13	0.00	0.19	1.49	2.82
	3	0.00	0.21	1.77	4.01	0.00	0.27	1.61	3.36	0.00	0.21	1.48	3.03
	4	0.00	0.21	1.80	3.87	0.00	0.21	1.62	3.25	0.00	0.21	1.49	2.99

Appendix B

Overfitting-fixed Tables and Transfer Learning

This second chapter of the appendix will show the results achieved by correcting the imbalance of the Emoevent Dataset to prevent **overfitting**. Since this Dataset has been chosen due to its strong unbalancing between classes, it has been tested on the chosen models and will serve as an example for erasing this issue in the rest of the Datasets. A part from overfitting, here one can find all three tables gathered during the **Transfer Learning** Research with Detoxis and HatEval2019.

Emoevent (4 epochs per training)											
Model/LR		BERT (Batch 16)						XLNet (Batch 8)			
Metrics		Acc	F1 score	Training Loss	Valid. Loss	Metrics		Acc	F1 score	Training Loss	Valid. Loss
$1 \cdot 10^{-4}$	1	0.91	0.71	0.27	0.24	$1 \cdot 10^{-4}$	1	0.66	0.52	0.71	0.58
	2	0.93	0.75	0.20	0.20		2	0.72	0.57	0.66	0.86
	3	0.93	0.78	0.16	0.22		3	0.08	0.08	0.65	0.79
	4	0.93	0.79	0.10	0.24		4	0.87	0.63	0.66	0.63
$3 \cdot 10^{-4}$	1	0.92	0.78	0.06	0.46	$2 \cdot 10^{-5}$	1	0.76	0.60	0.60	0.58
	2	**0.94**	**0.80**	**0.04**	**0.44**		2	0.85	0.65	0.65	0.57
	3	0.93	0.78	0.03	0.49		3	0.75	0.59	0.55	0.95
	4	0.94	0.79	0.02	0.48		4	0.79	0.62	0.49	0.56
$3 \cdot 10^{-5}$	1	0.94	0.80	0.12	0.27	$3 \cdot 10^{-5}$	1	0.84	0.67	0.51	0.57
	2	0.94	0.80	0.06	0.25		2	0.87	0.70	0.30	0.57
	3	0.94	0.79	0.04	0.31		3	0.87	0.71	0.20	0.60
	4	0.94	0.79	0.03	0.36		4	0.90	0.74	0.13	0.52
$5 \cdot 10^{-5}$	1	0.93	0.69	0.27	0.23	$5 \cdot 10^{-5}$	1	0.80	0.64	0.28	0.84
	2	0.94	0.74	0.23	0.20		2	0.88	0.71	0.21	0.59
	3	0.94	0.77	0.16	0.24		3	0.91	0.74	0.12	0.54
	4	0.93	0.78	0.10	0.23		4	**0.91**	**0.74**	**0.07**	**0.52**

BERT Transfer Learning HatEval2019 to Detoxis (4 epochs per training)													
Batch/LR		16				32				64			
Metrics		Acc	F1 score	Training Loss	Valid. Loss	Acc	F1 score	Training Loss	Valid. Loss	Acc	F1 score	Training Loss	Valid. Loss
$1 \cdot 10^{-4}$	1	**0.35**	**0.52**	**0.46**	**0.71**	0.51	0.51	0.60	1.33	0.58	0.56	0.22	1.33
	2	0.35	0.52	0.70	0.71	0.58	0.56	0.48	0.73	0.62	0.54	0.16	1.14
	3	0.35	0.52	0.70	0.74	0.51	0.51	0.40	1.03	0.60	0.57	0.10	1.65
	4	0.35	0.52	0.70	0.69	0.57	0.56	0.36	0.90	0.60	0.57	0.09	1.42
$2 \cdot 10^{-5}$	1	0.60	0.56	0.56	0.70	0.56	0.55	0.26	1.27	0.61	0.58	0.03	2.11
	2	0.60	0.54	0.39	0.84	0.56	0.55	0.26	1.08	0.62	0.56	0.04	1.75
	3	0.58	0.56	0.29	0.97	0.55	0.55	0.21	1.27	0.63	0.57	0.02	1.89
	4	0.61	0.57	0.22	1.14	0.56	0.55	0.23	1.17	0.62	0.58	0.04	1.88
$3 \cdot 10^{-5}$	1	0.47	0.47	0.53	1.31	0.62	0.57	0.18	1.42	0.61	0.57	0.01	2.71
	2	0.56	0.54	0.31	1.30	0.58	0.55	0.19	1.35	0.61	0.59	0.02	2.53
	3	0.60	0.56	0.24	1.57	0.60	0.56	0.16	1.54	0.59	0.58	0.01	2.49
	4	0.58	0.54	0.23	1.48	0.57	0.55	0.18	1.45	**0.63**	**0.59**	**0.03**	**2.30**
$5 \cdot 10^{-5}$	1	0.59	0.55	0.56	0.93	0.55	0.55	0.13	2.03	0.54	0.53	0.03	3.02
	2	0.56	0.55	0.39	0.89	0.61	0.54	0.19	1.42	0.62	0.56	0.03	2.44
	3	0.60	0.58	0.27	1.25	0.56	0.55	0.15	1.54	0.62	0.58	0.02	2.55
	4	0.60	0.58	0.21	1.23	0.58	0.56	0.14	1.46	0.62	0.58	0.03	2.44

RoBERTa Transfer Learning Detoxis to HatEval2019 (4 epochs per training)													
Batch/LR		4				8				16			
Metrics		Acc	F1 score	Training Loss	Valid. Loss	Acc	F1 score	Training Loss	Valid. Loss	Acc	F1 score	Training Loss	Valid. Loss
$1 \cdot 10^{-5}$	1	0.54	0.53	0.10	4.16	0.56	0.56	0.14	3.18	0.52	0.52	0.45	1.26
	2	0.56	0.56	0.06	4.26	0.56	0.55	0.13	3.58	0.52	0.51	0.35	1.46
	3	0.55	0.55	0.08	4.33	0.55	0.55	0.13	3.36	0.56	0.56	0.37	1.18
	4	0.55	0.55	0.05	4.30	0.55	0.55	0.10	3.38	0.53	0.52	0.31	1.47
$2 \cdot 10^{-5}$	1	0.56	0.55	0.11	4.02	0.55	0.55	0.13	3.66	0.55	0.55	0.26	2.41
	2	**0.57**	**0.57**	**0.04**	**4.42**	0.53	0.52	0.09	4.33	0.53	0.52	0.23	2.97
	3	0.57	0.56	0.06	4.22	0.53	0.52	0.08	4.08	0.54	0.54	0.22	2.26
	4	0.56	0.56	0.06	4.14	0.55	0.54	0.08	3.72	0.56	0.56	0.18	2.52
$3 \cdot 10^{-5}$	1	0.54	0.54	0.19	3.69	0.53	0.51	0.17	4.17	0.53	0.53	0.25	3.45
	2	0.55	0.53	0.15	4.04	0.54	0.53	0.16	3.88	0.51	0.49	0.22	3.72
	3	0.57	0.56	0.07	4.37	0.54	0.54	0.11	3.89	0.55	0.55	0.21	2.41
	4	0.57	0.57	0.05	4.21	0.54	0.54	0.10	3.75	0.54	0.53	0.15	2.74
$5 \cdot 10^{-5}$	1	0.56	0.71	0.69	0.69	0.52	0.51	0.32	2.99	0.55	0.55	0.31	1.59
	2	**0.44**	**0.61**	**0.70**	**0.71**	0.53	0.51	0.24	3.13	0.55	0.54	0.25	1.97
	3	0.56	0.71	0.72	0.69	0.55	0.55	0.11	3.33	0.56	0.55	0.20	2.70
	4	0.44	0.61	0.70	0.70	0.56	0.56	0.09	3.35	0.55	0.55	0.16	2.71

XLNet Transfer Learning HatEval2019 to Detoxis (4 epochs per training)													
Batch/LR		8				16				32			
Metrics		Acc	F1 score	Training Loss	Valid. Loss	Acc	F1 score	Training Loss	Valid. Loss	Acc	F1 score	Training Loss	Valid. Loss
$1 \cdot 10^{-4}$	1	0.65	0.78	0.67	0.68	0.50	0.48	0.67	0.83	0.41	0.40	0.56	0.89
	2	**0.38**	**0.32**	**0.67**	**0.72**	0.53	0.52	0.52	0.85	0.56	0.48	0.52	0.79
	3	0.38	0.32	0.66	0.79	0.54	0.50	0.48	0.98	0.48	0.47	0.50	0.97
	4	0.38	0.32	0.66	0.72	0.53	0.50	0.46	0.94	0.50	0.47	0.48	0.92
$2 \cdot 10^{-5}$	1	0.38	0.30	0.65	0.73	0.51	0.49	0.41	1.09	0.50	0.48	0.42	0.97
	2	0.65	0.78	0.67	0.67	0.55	0.50	0.41	1.00	0.51	0.47	0.45	0.90
	3	0.38	0.30	0.67	0.71	0.54	0.50	0.40	1.08	0.49	0.48	0.44	0.95
	4	0.38	0.30	0.68	0.68	0.53	0.50	0.41	1.00	0.51	0.48	0.46	0.92
$3 \cdot 10^{-5}$	1	0.37	0.29	0.66	0.72	0.51	0.48	0.39	1.17	0.46	0.46	0.42	0.93
	2	0.37	0.31	0.65	0.73	0.53	0.48	0.41	1.07	0.47	0.46	0.54	0.91
	3	0.42	0.41	0.60	0.83	0.52	0.50	0.38	1.15	0.46	0.46	0.50	1.00
	4	0.39	0.36	0.64	0.78	0.54	0.49	0.40	1.06	0.47	0.47	0.52	0.91
$5 \cdot 10^{-5}$	1	0.52	0.48	0.58	0.75	0.56	0.53	0.36	1.09	0.43	0.43	0.50	1.03
	2	0.47	0.46	0.48	1.23	0.51	0.49	0.38	1.10	0.45	0.44	0.49	0.98
	3	0.50	0.49	0.48	1.20	**0.57**	**0.52**	**0.37**	**1.06**	0.44	0.44	0.49	1.07
	4	0.50	0.48	0.47	1.08	0.55	0.52	0.38	1.07	0.45	0.45	0.53	0.95

Bibliography

[1] Ming (ufoym). *Imbalanced Dataset Sampler*. June 2021. URL: https://github.com/ufoym/imbalanced-dataset-sampler.

[2] @shristikotaiah. *Word Embeddings in NLP*. Oct. 2020. URL: https://www.geeksforgeeks.org/word-embeddings-in-nlp/.

[3] Jay Alammar. *How GPT3 Works - Visualizations and Animations*. July 2020. URL: https://jalammar.github.io/how-gpt3-works-visualizations-animations/.

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

[5] Jay Alammar. *The Illustrated GPT-2 (Visualizing Transformer Language Models)*. Aug. 2019. URL: https://jalammar.github.io/illustrated-gpt2/.

[6] Baptiste Amato et al. *Learning about the Attention Mechanism and the Transformer Model*. Apr. 2019. URL: https://deepfrench.gitlab.io/deep-learning-project/.

[7] Luis Enrique Argota Vega et al. *MineriaUNAM at SemEval-2019 Task 5: Detecting Hate Speech in Twitter using Multiple Features in a Combinatorial Framework*. Minneapolis, Minnesota, USA, June 2019. DOI: 10.18653/v1/S19-2079. URL: https://aclanthology.org/S19-2079.

[8] *Bag-of-words model*. May 2022. URL: https://en.wikipedia.org/wiki/Bag-of-words_model.

[9] Nilesh Barla. *The Ultimate Guide to Word Embeddings*. Dec. 2021. URL: https://neptune.ai/blog/word-embeddings-guide.

[10] Valerio Basile et al. *SemEval-2019 Task 5: Multilingual Detection of Hate Speech Against Immigrants and Women in Twitter*. Minneapolis, Minnesota, USA, June 2019. DOI: 10.18653/v1/S19-2007. URL: https://aclanthology.org/S19-2007.

[11] Sebastian Bittrich et al. *Confusion matrix for Accuracy, Precision, Recall and F1-score*. Jan. 2019. URL: https://www.researchgate.net/publication/330174519_Application_of_an_interpretable_classification_model_on_Early_Folding_Residues_during_protein_folding.

[12] Tom B. Brown et al. *Language Models are Few-Shot Learners*. 2020. URL: https://arxiv.org/abs/2005.14165.

[13] Jason Brownlee. *A Gentle Introduction to the Bag-of-Words Model*. Oct. 2017. URL: https://machinelearningmastery.com/gentle-introduction-bag-words-model/.

[14] Juan Pedro Cabanilles Clara García and Benjamín Ramirez. *What is the difference between stemming and lemmatization?* July 2021. URL: https://blog.bitext.com/what-is-the-difference-between-stemming-and-lemmatization/.

[15] Hugging Face Community. *BERT*. URL: https://huggingface.co/docs/transformers/model_doc/bert.

[16] Hugging Face Community. *bert-base-multilingual-cased*. URL: https://huggingface.co/bert-base-multilingual-cased.

[17] Hugging Face Community. *RoBERTa*. URL: https://huggingface.co/docs/transformers/model_doc/roberta.

[18] Hugging Face Community. *XLM-RoBERTa*. URL: https://huggingface.co/xlm-roberta-base.

[19] Hugging Face Community. *XLNet*. URL: https://huggingface.co/docs/transformers/model_doc/xlnet.

[20] Alexis Conneau et al. *Unsupervised Cross-lingual Representation Learning at Scale*. 2019. arXiv: 1911.02116. URL: http://arxiv.org/abs/1911.02116.

[21] Zihang Dai et al. *Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context*. 2019. DOI: 10.48550/ARXIV.1901.02860. URL: https://arxiv.org/abs/1901.02860.

[22] Zihang Dai et al. *Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context*. 2019. DOI: 10.48550/ARXIV.1901.02860. URL: https://arxiv.org/abs/1901.02860.

[23] AYLIEN developers. *Word Embeddings and Their Challenges*. URL: https://aylien.com/blog/word-embeddings-and-their-challenges.

[24] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2018. arXiv: 1810.04805. URL: http://arxiv.org/abs/1810.04805.

[25] Yunxia Ding, Xiaobing Zhou, and Xuejie Zhang. *YNU_DYX at SemEval-2019 Task 5: A Stacked BiGRU Model Based on Capsule Network in Detection of Hate*. Minneapolis, Minnesota, USA, June 2019. DOI: 10.18653/v1/S19-2096. URL: https://aclanthology.org/S19-2096.

[26] Niklas Donges. *A Guide to RNN: Understanding Recurrent Neural Networks and LSTM Networks*. Aug. 2021. URL: https://builtin.com/data-science/recurrent-neural-networks-and-lstm.

[27] Preston Fan. *Tweet Sentiment Extraction: why RoBERTa and not BERT?* 2020. URL: https://www.kaggle.com/c/tweet-sentiment-extraction/discussion/157801.

[28] Fangyug. *NLP: Word Embedding Techniques for Text Analysis*. Feb. 2020. URL: https://medium.com/sfu-cspmp/nlp-word-embedding-techniques-for-text-analysis-ec4e91bb886f.

[29] Andrea Galassi, Marco Lippi, and Paolo Torroni. *Attention in Natural Language Processing*. Oct. 2021. DOI: 10.1109/tnnls.2020.3019893. URL: https://doi.org/10.1109%2Ftnnls.2020.3019893.

[30] Rupesh Gelal. *BERT vs. XLNet*. Aug. 2020. URL: https://rupeshgelal.com.np/model/bert-vs-xlnet/.

[31] Harshith. *Text Preprocessing in Natural Language Processing*. Nov. 2019. URL: https://towardsdatascience.com/text-preprocessing-in-natural-language-processing-using-python-6113ff5decd8.

[32] Cathal Horan. *Ten trends in Deep learning NLP*. Mar. 2019. URL: https://blog.floydhub.com/ten-trends-in-deep-learning-nlp/#:~:text=2.-,Recurrent%20Neural%20Networks%20(RNNs)%20are%20no%20longer%20an%20NLP%20standard,earlier%20innovations%20such%20as%20Word2Vec.

[33] Rani Horev. *BERT Explained: State of the art language model for NLP*. Nov. 2018. URL: https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270.

[34] Rani Horev. *Transformer-XL Explained: Combining Transformers and RNNs into a State-of-the-art Language Model*. Jan. 2019. URL: https://towardsdatascience.com/transformer-xl-explained-combining-transformers-and-rnns-into-a-state-of-the-art-language-model-c0cfe9e5a924.

[35] Jiawei Hu. *An Overview for Text Representations in NLP*. 2018. URL: https://towardsdatascience.com/an-overview-for-text-representations-in-nlp-311253730af1.

[36] Vijayasaradhi Indurthi et al. *FERMI at SemEval-2019 Task 5: Using Sentence embeddings to Identify Hate Speech Against Immigrants and Women in Twitter*. Minneapolis, Minnesota, USA, June 2019. DOI: 10.18653/v1/S19-2009. URL: https://aclanthology.org/S19-2009.

[37] Yashwant Jankay. *The 3 Pillars of Binary Classification: Accuracy, Precision and Recall*. Apr. 2018. URL: https://medium.com/@yashwant140393/the-3-pillars-of-binary-classification-accuracy-precision-recall-d2da3d09f664#:~:text=Accuracy%20shows%20us%20how%20comfortable,divided%20by%20the%20total%20population.

[38] Prateek Joshi. *How do Transformers Work in NLP? A Guide to the Latest State-of-the-Art Models*. June 2019. URL: https://www.analyticsvidhya.com/blog/2019/06/understanding-transformers-nlp-state-of-the-art-models/#:~:text=The%20Transformer%20in%20NLP%20is,Attention%20Is%20All%20You%20Need.

[39] Samia Khalid. *BERT Explained: A Complete Guide with Theory and Tutorial*. Nov. 2019. URL: https://medium.com/@samia.khalid/bert-explained-a-complete-guide-with-theory-and-tutorial-3ac9ebc8fa7c.

[40] Suleiman Khan. *BERT, RoBERTa, DistilBERT, XLNet — which one to use?* Sept. 2019. URL: https://towardsdatascience.com/bert-roberta-distilbert-xlnet-which-one-to-use-3d5ab82ba5f8.

[41] Renu Khandelwal. *Word Embeddings for NLP*. Dec. 2019. URL: https://towardsdatascience.com/word-embeddings-for-nlp-5b72991e01d4.

[42] Joos Korstanje. *The F1 score*. Aug. 2021. URL: https://towardsdatascience.com/the-f1-score-bec2bbc38aa6.

[43] Gyorgy Kovács and Saketh Bachu. *SMOTE-variants for imbalanced learning*. Oct. 2020. URL: https://github.com/analyticalmindsltd/smote_variants.

[44] Ria Kulshrestha. *NLP 101: Word2Vec: Skip-gram and CBOW*. Nov. 2019. URL: https://towardsdatascience.com/nlp-101-word2vec-skip-gram-and-cbow-93512ee24314#:~:text=Continuous%20Bag%20of%20Words%20Model,word%20by%20using%20neural%20networks.&text=In%20the%20CBOW%20model%2C%20the,the%20word%20in%20the%20middle%20.

[45] Benai Kumar. *10 Techniques to deal with Imbalanced Classes in Machine Learning*. July 2020. URL: https://www.analyticsvidhya.com/blog/2020/07/10-techniques-to-deal-with-class-imbalance-in-machine-learning/.

[46] Sotiris Lamprinidis et al. "Universal Joy A Data Set and Results for Classifying Emotions Across Languages". In: *Proceedings of the Eleventh Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis*. Online: Association for Computational Linguistics, Apr. 2021, pp. 62–75. URL: https://aclanthology.org/2021.wassa-1.7.

[47] Vijaysinh Lendave. *Guide To Word2vec Using Skip Gram Model*. June 2021. URL: https://analyticsindiamag.com/guide-to-word2vec-using-skip-gram-model/.

[48] Yinhan Liu et al. *RoBERTa: A Robustly Optimized BERT Pretraining Approach*. 2019. arXiv: 1907.11692. URL: http://arxiv.org/abs/1907.11692.

[49] Bhoomika Madhukar. *The Continuous Bag Of Words (CBOW) Model in NLP – Hands-On Implementation With Codes*. Sept. 2020. URL: https://analyticsindiamag.com/the-continuous-bag-of-words-cbow-model-in-nlp-hands-on-implementation-with-codes/.

[50] Eric Nguyen. *Term Frequency-Inverse Document Frequency*. 2014. URL: https://www.sciencedirect.com/topics/computer-science/inverse-document-frequency#:~:text=The%20inverse%20document%20frequency%20is,the%20term%20in%20the%20corpus.

[51] John Pavlopoulos et al. *ConvAI at SemEval-2019 Task 6: Offensive Language Identification and Categorization with Perspective and BERT*. Minneapolis, Minnesota, USA, June 2019. DOI: 10.18653/v1/S19-2102. URL: https://aclanthology.org/S19-2102.

[52] Juan Manuel Pérez and Franco M. Luque. *Atalaya at SemEval 2019 Task 5: Robust Embeddings for Tweet Classification*. Minneapolis, Minnesota, USA, June 2019. URL: https://aclanthology.org/S19-2008.

[53] Matthew E. Peters et al. *Deep contextualized word representations*. 2018. URL: https://arxiv.org/abs/1802.05365.

[54] Flor Miriam Plaza del Arco et al. "EmoEvent: A Multilingual Emotion Corpus based on different Events". English. In: *Proceedings of the 12th Language Resources and Evaluation Conference*. Marseille, France: European Language Resources Association, May 2020, pp. 1492–1498. ISBN: 979-10-95546-34-4. URL: https://aclanthology.org/2020.lrec-1.186.

[55] Alec Radford et al. *Language Models are Unsupervised Multitask Learners*. 2018. URL: https://d4mucfpksywv.cloudfront.net/better-language-models/language-models.pdf.

[56] Unnikrishnan C. S. *How sklearn's Tfidfvectorizer Calculates tf-idf Values*. Nov. 2021. URL: https://www.analyticsvidhya.com/blog/2021/11/how-sklearns-tfidfvectorizer-calculates-tf-idf-values/.

[57] Sheel Saket. *Count Vectorizer vs TFIDF Vectorizer | Natural Language Processing*. Jan. 2020. URL: https://www.linkedin.com/pulse/count-vectorizers-vs-tfidf-natural-language-processing-sheel-saket/.

[58] Prateek Sawhney. *Introduction to Stemming and Lemmatization (NLP)*. Sept. 2021. URL: https://medium.com/geekculture/introduction-to-stemming-and-lemmatization-nlp-3b7617d84e65.

[59] George Seif. *Handling Imbalanced Datasets in Deep Learning*. Nov. 2018. URL: https://towardsdatascience.com/handling-imbalanced-datasets-in-deep-learning-f48407a0e758.

[60] Priya Shree. *The Journey of Open AI GPT models*. Nov. 2020. URL: https://medium.com/walmartglobaltech/the-journey-of-open-ai-gpt-models-32d95b7b7fb2#:~:text=Model%20Architecture%20and%20Implementation%20Details:%20GPT-1%20used%2012-,the%20original%20work%20on%20transformers.

[61] Aastha Singh. *Evolving with BERT: Introduction to RoBERTa*. June 2021. URL: https://medium.com/analytics-vidhya/evolving-with-bert-introduction-to-roberta-5174ec0e7c82.

[62] Ashish Vaswani et al. *Attention Is All You Need*. 2017. DOI: 10.48550/ARXIV.1706.03762. URL: https://arxiv.org/abs/1706.03762.

[63] Raden Aurelius Andhika Viadinugroho. *Imbalanced Classification in Python: SMOTE-Tomek Links Method*. Apr. 2021. URL: https://towardsdatascience.com/imbalanced-classification-in-python-smote-tomek-links-method-6e48dfe69bbc.

[64] Jesse Vig. *GPT-2: Understanding Language Generation through Visualization*. Mar. 2019. URL: https://towardsdatascience.com/openai-gpt-2-understanding-language-generation-through-visualization-8252f683b2f8.

[65] Gonzalo Ruiz de Villa. *Introducción a Word2vec (skip gram model)*. May 2018. URL: https://gruizdevilla.medium.com/introducci%C3%B3n-a-word2vec-skip-gram-model-4800f72c871f.

[66] Cornellius Yudha Wijaya. *5 SMOTE Techniques for Oversampling your Imbalance Data*. Sept. 2020. URL: https://towardsdatascience.com/5-smote-techniques-for-oversampling-your-imbalance-data-b8155bdbe2b5.

[67] Thomas Wolf et al. *Transformers: State-of-the-Art Natural Language Processing*. Oct. 2020. DOI: 10.18653/v1/2020.emnlp-demos.6. URL: https://aclanthology.org/2020.emnlp-demos.6.

[68] Zhilin Yang et al. *XLNet: Generalized Autoregressive Pretraining for Language Understanding*. 2019. arXiv: 1906.08237. URL: http://arxiv.org/abs/1906.08237.

[69] Mariya Yao. *10 Leading Language Models For NLP In 2021*. May 2021. URL: https://www.topbots.com/leading-nlp-language-models-2020/.

[70] David Zimbra et al. *The State-of-the-Art in Twitter Sentiment Analysis: A Review and Benchmark Evaluation*. New York, NY, USA, Aug. 2018. DOI: 10.1145/3185045. URL: https://doi.org/10.1145/3185045.