UNIVERSITAT DE BARCELONA

Bachelor Thesis

COMPUTER SCIENCE DEGREE

Faculty of Mathematics and Informatics
Universitat de Barcelona

Analysis of hate speech detection in social media

Ferran Sanchez Llado

Director: Dra. Maria Salamó Llorente
Written in: Faculty of
Mathematics and Informatics

Barcelona, July 20, 2021

Abstract

The presence of social networks has increased in our daily lives and have become platforms for sharing information. But, it also can be used for sending hate messages or for propagating false news. Users can take advantage of their anonymity to provide these toxic interactions. Furthermore, some groups of people (minorities) get disproportionately more targeted than the rest. This raises the problem of how to detect if a message contains hate speech. A solution could be the use of machine learning models that would be in charge of this decision. In addition, it could handle the enormous amount of texts interchanged daily. However, there are many approaches to tackle the problem, which are divided mainly into two groups. The first one is through the use of classical algorithms to extract information from the text. The other one is through the use of deep learning models that can understand some context that allows for better predictions.

The main objectives of the project are the exploration and comparison of different types of models and techniques. The diverse models are trained with three distinct toxicity datasets, of two natural language processing competitions. Generally, the best performing model is BERT or SBERT, both models based on the deep learning approach, with metric scores much higher than any model based on the traditional methods. The results show the vast potential of Natural Language Processing for the detection of hate speech. Although the best models did not have a very high perplexity, a more reliable model could be trained with more training data or new architectures. Even at the current state, the models could be used as an external font for helping humans in the decision-making process. Moreover, these models could filter the most confident predictions while leaving the rest for the reviewer team.

Resumen

Las redes sociales cada vez tienen más presencia en nuestra vida y se han transformado en plataformas donde se comparte información, pero también puede utilizarse para mandar mensajes de odio o contaminar con noticias falsas. Los usuarios pueden aprovechar el anonimato para realizar interacciones tóxicas. Además algunos grupos de personas (minorías) reciben desproporcionadamente más odio que el resto. Esto nos lleva al problema de cómo detectar si un mensaje contiene odio o no. Una posible solución es a través del uso de modelos de aprendizaje automático, los cuales serían capaces de encargarse del proceso de decisión. También, tendrían la capacidad para poder procesar las grandes cantidades de datos intercambiados diariamente. Hay muchas maneras de abordar este problema, pero se pueden dividir principalmente en dos grupos. El primero es a través del uso de algoritmos clásicos capaces de extraer información básica del texto. El otro está basado en modelos de aprendizaje profundo (deep learning), los cuales son capaces de entender algo del contexto para realizar mejores predicciones.

Los principales objetivos del proyecto son la exploración y comparación de diferentes tipos de modelos y algoritmos. Todos los modelos han sido entrenados utilizando tres conjuntos de datos sobre toxicidad, basados en dos competiciones de procesamiento del lenguaje natural. Generalmente los modelos que han obtenido mejores resultados han sido BERT y SBERT, los dos basados en el enfoque del aprendizaje profundo (deep learning), con unos resultados mucho más altos que los modelos basados en el enfoque tradicional. Los resultados muestran el gran potencial del procesamiento del lenguaje natural para la detección de odio en los mensajes. Aunque los mejores modelos no tenían una perplejidad muy alta, un modelo más eficaz podría haber sido entrenado con mayores datos o una nueva arquitectura. Incluso en el estado actual, los modelos podrían ser utilizados como fuente externa para ayudar a humanos durante la toma de decisiones. Además, los modelos podrían servir como filtro para las predicciones más seguras, dejando el resto a un equipo de revisores.

Resum

Les xarxes socials tenen cada vegada més presència en les nostres vides i s'han transformat en plataformes on es comparteix informació, però també poden utilitzar-se per a enviar missatges d'odi o contaminar amb notícies falses. Els usuaris aprofiten l'anonimat per realitzar interaccions tòxiques. A més, alguns grups de persones (minories) reben desproporcionadament més odi que la resta. Això ens planteja el problema de com detectar si un missatge conté odi. Una possible solució és l'ús de models d'aprenentatge automàtic, els quals es poden encarregar del procés de prendre decisions. També, seria capaços de processar les grans quantitats de missatges que són intercanviats diàriament. Hi ha moltes maneres d'atacar aquest problema, però es poden principalment dividir en dos grups. El primer és a través de l'ús d'algoritmes clàssics capaços d'extreure informació del text. L'altre grup està basat en l'ús de models d'aprenentatge profund (deep learning), els quals són capaços d'entendre un poc el context per fer millors prediccions.

Els principals objectius del projecte són l'exploració i comparació de diferents tipus de models i algorismes. Tots els models has sigut entrenats utilitzant tres conjunts de dades sobre toxicitat, basats en dues competicions de processament del llenguatge natural. Generalment els models que han tingut millors resultats has sigut BERT i SBERT, els dos basats en l'enfocament de l'aprenentatge profund (deep learning), amb puntuacions molt més altes que els models basats en l'enfocament tradicional. Els resultats mostren el gran potencial del processament del llenguatge natural per la detecció d'odi. Encara que els millors models no tenen una perplexitat molt alta, un millor model podria ser entrenat amb major quantitat de dades o amb una nova arquitectura. Inclús en l'estat actual, els models poden ser utilitzats com una font externa als humans la presa de decisions. A més, els models poden ser utilitzats per filtrar les prediccions més fiables, deixant el resta per un equip de revisors.

Index

1. Introduction

This chapter describes the problem this project will focus on and the motivation for choosing this specific project. Also, it explains the principal and secondary objectives that are considered to be accomplished. Finally, there is a brief explanation of the focus of each chapter.

1.1. Problem framework

In the recent decade, there has been an increased presence of technology in our daily lives, to the point that we are constantly surrounded by it, from mobile phones to smartwatches. Therefore also allowed accessible communication between people across the globe through the World Wide Web. Furthermore, social media applications allow you to express your thoughts anonymously, such as Facebook or Twitter.

In fact, this social relationship on the web brought the problem where people can behave negatively without or with almost no repercussions (e.g. close account). Furthermore, people tend to have high difficulties changing their beliefs even if it is factually incorrect. Also, research (Lenhart et al., 2016) has shown that some groups of people are disproportionately targeted with abuse online. Including women, people of colour, lesbian, gay, bisexual, transgender, queer, intersex, asexual individuals, marginalized and historically underrepresented communities.

The above-mentioned fact brings us to the current problem of detecting if a message contains hate speech or not. First of all, a reporting system can be implemented, where the users of social media can choose to report a message if they consider it inappropriate, reducing by a large margin the number of texts that would be needed to be analyzed. Then, several teams of people could be dedicated to just reviewing reported messages and deciding the verdict. However, this is not feasible, at least for popular social media companies, due to the enormous number of texts sent per second (e.g. 6,000 for Twitter).

The solution to the aforementioned problems is the usage of machine learning classifiers for identifying it. However, this brings a new set of questions ranging: from how to train a model for Natural Language Processing to which model is the most optimal for accomplishing this task. Exploring these questions will be the main focus of this paper.

1.2. Motivation

Since I started the computer science degree, artificial intelligence, especially machine learning, has been one of the disciplines in which I was most interested. I am fascinated with the topic of self-learning algorithms. Also in how a machine learning model can be used across all disciplines of science. For example, in psychology, the use of a model to interpret thoughts through an MRI scanner.

The recent publication of the model of GPT-3 (B. Brown et al., 2020) of OpenAI, caught my interest in the Natural Language Processing discipline. The model was very capable of

accomplishing different types of tasks without being previously trained. Even the model was able to create articles that were not differentiable from a human-written one. Furthermore, the model learned how to solve simple mathematical operations. Even though it has never been trained specifically for this task. At the time, I thought it was the nearest humans have been to creating a general artificial intelligence and that it had an immense amount of potential.

Moreover, during the time I was selecting the topic of this bachelor thesis, I did not have any practical knowledge of machine learning. So, I thought it would be an excellent opportunity to introduce myself to machine learning through natural language processing. While also learning the different technologies used in this discipline.

1.3. Project objectives

The main objective of this paper is the investigation, implementation, recreation, competition and comparison of different Natural Language Processing models through the use of datasets of two competitions (Task 5 of SemEval-2019[1] and DETOXIS[2]), that are centred on the detection of hate speech on messages. In particular, the specific objectives of this final grade project are:

The first objective was a basic study of the most relevant papers published on natural language processing with an emphasis on models and architectures based on neural networks.

The second objective was to implement a basic statistical analysis of the datasets. Where statistics about the features to predict and the input text are displayed. Facilitating decisions down the road, such as the techniques of preprocessing to be applied.

The third objective was to implement a preprocessing pipeline where the text of the dataset is filtered and transformed into tokens. The process can range from lowercasing all text to replacing contractions with the original word.

The fourth objective was to encode the preprocessed text of the datasets through the use of Term Frequency-Inverse Document frequency. Then, the encoded data is used for training several basic machine learning algorithms.

The fifth objective was the optimization of the hyperparameters of all models through the use of grid search cross-validation. Once they are found, models are re-trained with the best parameters.

The sixth objective was creating the framework to train and evaluate specialized neural networks. After training and evaluating the datasets with the different versions of BERT (Devlin et al., 2018) and GPT-2 (Radford et al., 2019).

The seventh objective was to recreate the model that the team Atalaya created for the competition of SemEval 2019. Once trained, the results their team got are compared against the recreation.

---

[1] "SemEval-2019 Task 5: Multilingual Detection of Hate Speech ...." https://www.aclweb.org/anthology/S19-2007.
[2] "Welcome | DETOXIS-IberLEF 2021 - Wix.com." https://detoxisiberlef.wixsite.com/website.

The eighth objective was to encode the preprocessed text by using SBERT (Reimers & Gurevych, 2019). It is similar to Universal Sentence Encoder (Cer et al., 2018), allowing comparison with the best participants of the English subtask 1 of the SemEval competition.

The ninth objective is the creation of a flexible framework to store and retrieve predictions from the disk. Afterwards, create a new notebook to join all predictions while calculating the scores.

The tenth objective is the selection, training and submission of the 5 best models for the DETOXIS competition to be submitted.

1.4. Project organization

The paper is divided into six chapters.

The first chapter (the current one) is a general introduction to the project, where the problems, motivations, objectives, and organization are described.

The second chapter contains an explanation of some of the most popular models used in natural language processing. Dividing into two subsections, one with the classical approach, while the other explains some of the current state-of-the-art models.

The third chapter is an analysis of the environment, the datasets, the preprocessing and the implementation, which will explain how the project was developed.

In the fourth chapter, the results of all models are compared across the datasets. Also, the model's results and methodology will be explained. Moreover, the final results of the DETOXIS competition are also shown.

The fifth chapter is an economic analysis of the entire project and a breakdown of the work done.

The sixth chapter contains the conclusions of the project, where a personal evaluation of the accomplishments is made, while expressing some facts learned during the entire duration of the project.

2. State of Art

This chapter explains two different approaches for tackling the problem of natural language processing, while also analyzing and comparing some of the best models used across the field in deep learning. However, firstly an explanation on how most tasks were tackled and the inner workings of some classification models is provided. Furthermore, almost all mentioned models that are going to be described are used throughout the project.

2.1. Traditional

The traditional approach tries to tackle the problems of Natural Language Processing (Basile et al., 2019) by using classical algorithms. The main difference with the deep learning approach is that it can only extract static information without context. Whereas the deep learning approach may make more complex predictions through the interactions of words. However, most success has been brought in Natural Language Processing tasks through the use of deep learning models (B. Brown et al., 2020; Cer et al., 2018; Devlin et al., 2018; Liu et al., 2019; Mikolov et al., 2013; Peters et al., 2018; Radford et al., 2019; Reimers & Gurevych, 2019; Vaswani et al., 2017). One of the main differences is that the model can use the context from the text, whereas traditional approaches can not. Some of the most commonly used techniques are:

Bag Of Words (Zhang et al., 2010) is a statistical measure that counts the number of times words appear in a document. This technique is one of the most simple techniques in information retrieval that does not consider the term ordering nor rareness of the term.

Term Frequency-Inverse Document Frequency (Robertson, 2004) is a statistical measure that evaluates the relevance of words in a collection of documents. The information retrieval technique is an improvement in comparison to Bag Of Words. Instead, the values are a combination of the term frequency with the inverse document frequency. Similarly to the Bag Of Words, Term Frequency-Inverse Document Frequency does not account for the term order.

Variants of term frequency (tf) weight

weighting scheme	*tf weight*
binary	$0, 1$
raw count	$f_{t,d}$
term frequency	$f_{t,d} \Big/ \sum_{t' \in d} f_{t',d}$
log normalization	$\log(1 + f_{t,d})$
double normalization 0.5	$0.5 + 0.5 \cdot \frac{f_{t,d}}{\max_{\{t' \in d\}} f_{t',d}}$
double normalization K	$K + (1 - K)\frac{f_{t,d}}{\max_{\{t' \in d\}} f_{t',d}}$

Table 1: Variants of term frequency weight.

The term frequency(tf) measures how frequent is a term within a document. There are several variants (see Table 1) for the term frequency tf(t,d), where t is a term and d is a document, the most commonly used is term frequency:

- *Binary*: 1 if a term t appears at least once on document d, 0 otherwise.
- *Raw count*: Number of times the term t appears on document d. (ft, d)
- *Term frequency*: The raw count scaled with the document length.
- *Log normalization*: The raw count adjusted logarithmically.
- *Double normalization K*: To prevent a bias towards longer documents, the raw count is divided by the most occurring term in the document

Variants of inverse document frequency (idf) weight

weighting scheme	*idf weight* ($n_t = \|\{d \in D : t \in d\}\|$)
unary	1
inverse document frequency	$\log \frac{N}{n_t} = -\log \frac{n_t}{N}$
inverse document frequency smooth	$\log\left(\frac{N}{1+n_t}\right)+1$
inverse document frequency max	$\log\left(\frac{\max_{\{t' \in d\}} n_{t'}}{1+n_t}\right)$
probabilistic inverse document frequency	$\log \frac{N-n_t}{n_t}$

Table 2: Variants of Inverse Document Frequency weight.

The inverse document frequency (idf) measures how much information the word provides by how common the term across the documents is. Also, there are several variants (described in Table 2) for the inverse document frequency idf(t,d), where t is a term and d is a document, the most commonly used is the inverse document frequency:

- *Unary*: Always 1.
- *Inverse document frequency*: Number of documents divided by the number of times a term appears at least once on the document adjusted logarithmically.
- *Inverse document frequency smooth*: Similar formula to the previous but smoothed by adding two additions.
- *Inverse document frequency max*: Instead of the number of documents uses the maximum amount of times a term appears across all documents.
- *Probabilistic inverse document frequency*: Similar to base formula but subtracting to the number of documents, the number of times a term appears across all documents.

The combination of techniques allows transforming a list of input texts into a floating matrix. Then, the matrix can be used for training machine learning models to predict the objective task. However, to further increase the performance, a preprocessing of the text is needed. The preprocessing of data consists of transforming the inputs to improve the quality of the data.

Some of the most common preprocessing techniques consist in the removal or modification of some words that do not bring extra information (e.g. links, numbers, emojis). Another is the standardization of words to avoid splitting similar terms so that the term value is higher (e.g. lowercasing, lemmatization, spelling errors). Also, the removal of the most commonly used terms (e.g. stopwords).

Machine learning models are a family of algorithms that, after being trained with a labelled dataset, can predict the labels from never-seen data. The dataset consists of a numerical matrix of features (columns), with the number of inputs (rows). Labels are the features that the models need to learn to predict. An example of a dataset is the Term Frequency-Inverse Document Frequency (TF-IDF) matrix, where the columns are the words and the rows are the entries of the original text.

Training a model using the traditional approach has two stages. First, transforming the text data using one of the previously mentioned techniques. Second, training a machine learning model to predict an output using the transformed text. Similarly, in the prediction phase, the new text is also converted using the same transformation. Finally, the model predicts using the new matrix. There are two types of supervised model predictions: regression, where the model tries to predict a number, and classification, where the model tries to predict a class.

Some of the most well-known classifiers used to predict the classes in NLP are:

- *Naive Bayes classifier* (Hand & Yu, 2001): a family of probabilistic classifiers, based on Bayes' theorem. The model tries to minimize the probability of misclassification, by assuming strong independence between features. Also, it can be trained in linear time by evaluating a closed-form expression. The kernel used can be changed to any distribution. The ones used in the project are the multinomial and Bernoulli naive Bayes classifiers.
- *Ridge classifier* (Singh et al., 2016): a method of classification based on linear regression adding an extra parameter for regularization of features and dividing the classes to predict in two (-1 and 1). The model can be trained in linear time by the use of ordinary least squares.
- *Random Forest* (Breiman, 2001): an ensemble learning method that operates by constructing a multitude of decision trees and selecting the class through an average of all individual trees. A decision tree is based on a tree-like structure where the leaves are the predictions and the rest of the nodes are questions to be asked to the data.
- *Adaboost* (Freund & Schapire, 1996): another ensemble learning method similar to Random Forest, but instead of the input being the same for all the decision trees, the inputs are processed sequentially through the decision trees called "weak learners". The standard weak learner is a Decision Tree Classifier with a max depth of one.
- *Support Vector Machine* (Hearst et al., 1998): a family of supervised learning methods that tries to find the best hyperplanes that divide distinctively the data points. The used kernel can be modified, which allows mapping any linear function.

2.2. Deep Learning

The deep learning approach uses complex artificial neural networks to solve Natural Language Processing problems. As in other subfields of computer science (e.g. computer vision), it seems to bring the most success in almost all types of task objectives. Furthermore, nearly all the investigation in Natural Language Processing is through the use of deep learning models.

There are currently three techniques to solve a problem using neural networks. The first option, create a task-specific architecture from the ground up. The second option, create a fixed model that is pre-trained with a vast dataset of unlabeled data (e.g. BERT (Devlin et al., 2018) and RoBERTa (Liu et al., 2019)). After that, the model is fine-tuned by training it with the labelled data from the specific task. This option allows the transfer of knowledge gained during the first phase to the second. Third, is creating a general model also pre-trained with unlabeled data, but without the fine-tuning step (e.g. GPT-2 (Radford et al., 2019) and GPT-3 (B. Brown et al., 2020)). These types of models try to solve multiple Natural Language Processing tasks. Moreover, a context window can be used to allow some examples before generation.

Artificial Neural Networks are inspired by the biological neural networks that can be found in our brains. The concept is based on a collection of connected nodes that represent the neurons, connected through connections, like synapses in the biological brain. The input from the network is converted into a signal that propagates to all nodes until the output. Normally, a neural network is divided into a sequence of layers, composed from an input layer, hidden layers and an output layer. Each layer connection has a weight and bias. The bias is unique to the neuron and the weight to the connection between two neurons. That is each layer has a matrix of weights and a vector of biases. Also, the node has a function that maps from the nodes of the previous layer through weights and biases.

Word embedding is the technique to transform a word into a vector with a fixed dimensionality. The most basic strategy is to use one-hot-encoding with the size of a vocabulary. This approach consists of mapping a word to a fixed position on the vector. However, in this option, the words with similar meanings have the same distance as words without any relationship. A solution is to create another neural network model to generate a fixed size embedding, which can learn complex relationships between terms (e.g. word2vec (Mikolov et al., 2013) or SBERT (Reimers & Gurevych, 2019)).

2.2.1. Transformer

Transformer (Vaswani et al., 2017) is an architecture based on encoder-decoder (seq2seq) that tries to solve the constraints of sequential computation by allowing bidirectionality. The model relies entirely on the self-attention mechanism without using any sequential recurrent or convolution layers. Self-attention is a mechanism for mapping the importance of input in

comparison to the rest. The model is auto-regressive because it consumes previously generated symbols as additional input when generating the next.

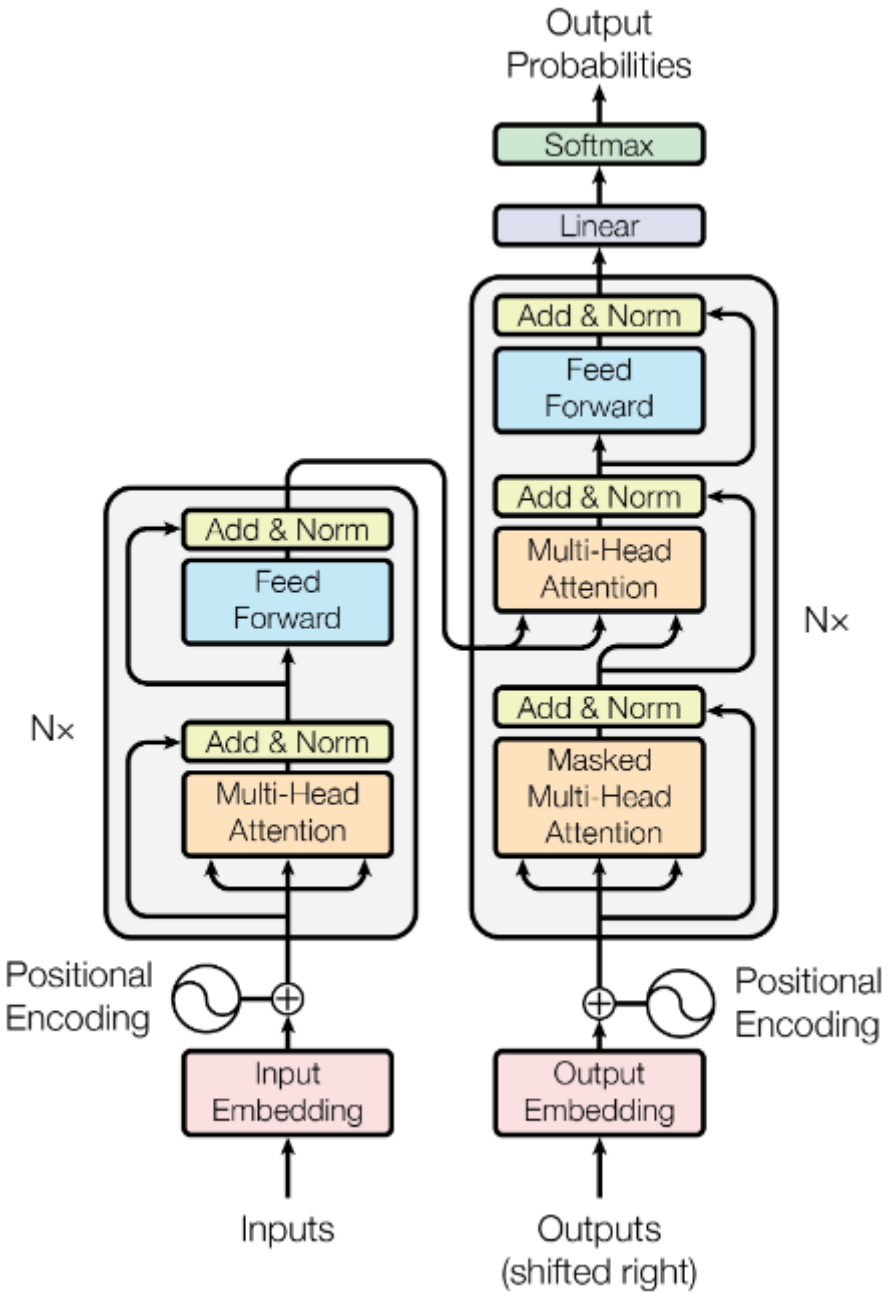

Figure 1: The model architecture of the Transformer. The left half is the encoder, and the right half is the decoder.

Both the encoder and the decoder use six layers but changing the number of sublayers (see Figure 1). The encoder has a multi-head attention layer and a feed-forward layer, followed by normalization. Meanwhile, the decoder has a masked multi-head attention layer, a multi-head attention layer with the results of the encoder and a feed-forward layer. Furthermore, a regularization is added with a dropout layer in each sublayer and also label smoothing.

Multi-head attention consists of several attention layers running in parallel. Meanwhile, the basic attention layer consists of scaled dot-product attention. The queries come from the previous decoder layer, while the keys and values come from the encoder, allowing the model to attend to all positions in the input text. The masked version weights the attention to solve the problem of allowing the model to interact from distant positions. The feed-forward layers consist of two linear transformations with a ReLU in between.

As previously mentioned, due to the model not having an order, positional information is needed to be added to the input of both the encoder and decoder. The original paper opts to use sine and cosine functions to add this. Furthermore, the model has an increase in performance due to the parallelisms and reduced complexity per layer in comparison to recurrent or convolutional neural networks.

2.2.2 Bert

Bidirectional Encoder Representations from Transformer (Devlin et al., 2018) is a language model based on multi-layer bidirectional encoder transformer architecture (Vaswani et al., 2017), with improvements on the constraints of unidirectionality by using a technique called Masked Language Model (MLM). The model achieved state-of-the-art results during its publication in eleven Natural Language Processing tasks, beating even task-specific models. Moreover, the team trained different sizes of the model and confirmed the well-known hypothesis that increasing the model size will lead to continual improvements in downstream tasks. The largest model trained was BERT large with a parameter size of 340 million.

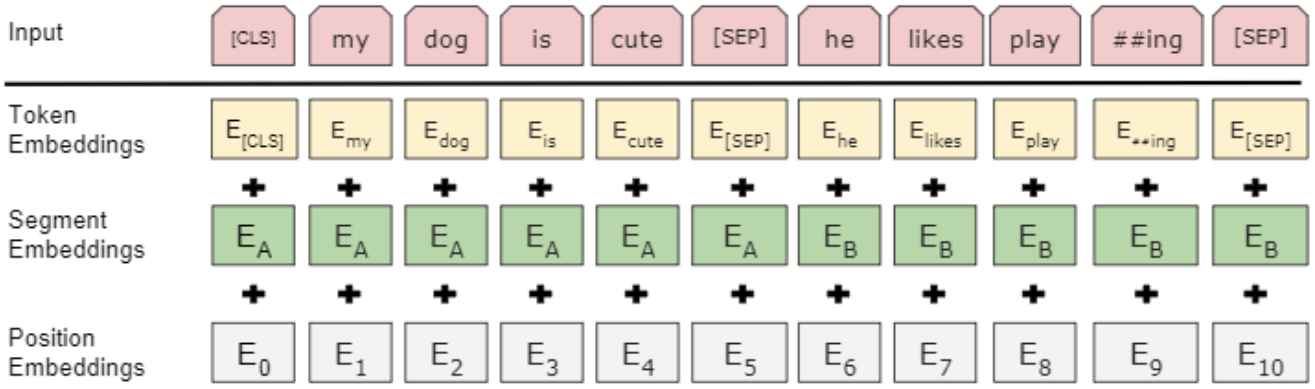

Figure 2: BERT input representation. The input embeddings are the sum of the token embeddings, the segmentation embeddings and the position embeddings.

The technique MLM randomly masks tokens from input with a special token, a random token or leave it unchanged. Also, this technique allows the model during pretraining to learn to predict the original text using only the context of both directions. BERT has two steps in its framework: First, the pretraining phase where the model is trained to predict from the input the output with a large unlabeled dataset (a cloze task) and with a very simple binary Next Sentence Prediction task. Second, the fine-tuning phase in which the model is trained with

the task objective using a labelled dataset. The first step can be reused for any time a specific task is needed.

The word embedding is WordPiece with a 30,000 vocabulary size. The text is tokenized using the embedding, while also adding some special tokens. Finally, the token embedding can be passed to the model along with segment embedding (indicating to which pair the text corresponds) and a position embedding (indicating the position of the token, similar to the original transformer). Which are combined to generate the input for the model (see Figure 2).

2.2.3. RoBERTa

A Robustly Optimized BERT Pretraining Approach (Liu et al., 2019) presents some modifications to the original BERT (Devlin et al., 2018) model, starting with increasing the amount of data and time the model is pre-trained using longer sentences. Moreover, the model is not pre-trained with the Next Sentence Prediction task. Also, the technique “Masked Language Model” is modified, to change dynamically the pattern that is applied to the input, instead of being fixed. Furthermore, they tried changing the text encoder to a Byte-Pair Encoding (BPE) with a size of 50,000 tokens.

First of all, the authors considered that BERT was significantly under fitted and therefore, trained with larger batches improving perplexity[3] for the Masked Language Model (MLM) objective, as well as end-task accuracy. Second, they found that removing Next Sentence Prediction (NSP) objective matches or slightly improves downstream task performance. Third, modifying MLM to be dynamic allows avoiding the duplication of data to generate different masking patterns. Finally, the change in text encoder resulted in slightly worse performance, but the authors believe that the universal encoding outweighs it.

2.2.4. GPT-2

Generative Pre-trained Transformer 2 (Radford et al., 2019) is based on the decoder transformers architecture (Vaswani et al., 2017), which is very similar to GPT with a few modifications. The largest model has a 1.5 billion parameters size and was trained without any task-specific dataset. The authors demonstrate that a general model can learn to perform tasks without the need to create training datasets, nor change the parameters, nor the model. The model achieved state-of-art performance in 7 out of 8 tasks when it was published.

The authors predict that the model will need to learn to use the unsupervised data to achieve specific tasks, resulting in better predictions. In a way, the model manages to perform unsupervised multitask learning. The dataset for training had 40 GB of size, where overlaps with the task datasets were checked, to avoid the possibility of the model scoring highly through memorization. Overlaps were checked through bloom filters and n-gram overlaps.

[3] Measurement of how well a probability model predicts a sample.

The average overlap was only 6% and considered that it provides small but consistent benefits to the results. Moreover, it is shown that the models were underfitted, due to the constant increase of perplexity as the model size increases. When a large language model is trained on a sufficiently large and diverse dataset it can perform well across many domains and datasets.

The word embedding used had a size of 50,000 tokens and is a mix between Byte-Pair Encoding (BPE) and word-level using Unicode strings. Also, they use a greedy approach to avoid merging character categories with byte sequence (e.g. only “dog” instead of “dog.”)

2.2.5. GPT-3

Generative Pre-trained Transformer 3 (B. Brown et al., 2020) is based on the same architecture as GPT-2 (Radford et al., 2019). The largest model is two orders of magnitude larger than the previous biggest model with a 175 billion parameters size. This time, the authors focused on few-shot performance, where a context of some examples is inputted, before trying to make the prediction. The process does not update any weight as it passed as input. This methodology sometimes even reaches competitiveness with prior state-of-the-art fine-tuning approaches.

The objective of creating a general model is the same as in GPT-2. The first reason is to reduce the need to build large datasets that could be complex to collect depending on the task. The second one is to avoid the model learning potential exploits in the training data, due to having similar expressiveness as the testing. The final reason is that humans do not require an enormous supervised dataset to learn most language tasks.

GPT-3 was evaluated in 3 conditions: zero-shot (no examples are provided before making the prediction), one-shot (one example is provided) and few-shot (10-100 examples are provided depending on the task). As expected, the more examples are provided the better the perplexity is. The examples are passed through a context window of 2048 tokens. Furthermore, 8 different models with different sizes were trained. The model performance followed a power-law with the size.

A dataset of fewer than 500 billion tokens was used for the training phase, which is also two orders of magnitude bigger than the one for GPT-2. Furthermore, they observed that the largest model did not overfit due to the immense amount of data. Overlaps with the tasks dataset were filtered, but during the late stages of training the model, they found a bug on the detection algorithm, which only detected partially. Even though the contamination was likely to be frequent, the authors reasoned that its effect may not be as large as feared. A study on the data contamination concluded that the problem caused a minimal effect on the performance. They came up with this conclusion by comparing the results of a new clean dataset for all benchmarks with 13-gram overlaps removed. However, a possible problem is that these new clean datasets were not drawn from the same distribution.

The authors observed some limitations the model had. First of all, the text synthesis generated through GPT-3 had an overall high quality. However, sometimes, the model would repeat itself semantically, make contradictions and write some sentences or paragraphs that do not make sense (non-sequitur). Secondly, the model in discrete language tasks lacked common sense with basic physics. Thirdly, due to model structure, it had some limitations in some tasks better fitted for bidirectionality, such as cloze tasks. Fourthly, scaling the model could eventually run into the limits of the pre-training objectives. Fifthly, the model had a poor sample efficiency because it required too much text to train in comparison to a human. Sixthly, it is unknown if the model learned from scratch with few-shots or simply recognized and identified the task that it had learned during training. Seventhly, distillation could have been explored, which consists of training a smaller model through the predictions of a larger one as a ground truth. Lastly, as with all neural network models, it is difficult to interpret the decisions the model makes. Furthermore, the model was shown to have some biases as in gender, race and religion.

3. Development

This section explains the development process of the implementation and explains the reasons for the different decisions taken. First, starting, with an explanation of the environment, continuing with a description and analysis of the datasets. Also, explaining the diverse preprocessing techniques applied depending on the model. Finishing with a general explanation of all the implementation.

3.1. Environment

The chosen language for the implementation of the project was Python[4], for several reasons, starting with the great popularity it has. Also, it provides an extensive collection of libraries and packages, for all types of machine learning applications with a multitude of functions for visualization, processing and implementation. Another reason is code readability since Python is an interpreted language, which allows creating concise and readable code that accomplishes while still being clear. Also, for similar reasons, Python allows to easily create flexible code that can be modified for any situation.

The integrated development environment (IDE) used is Jupyter Notebook[5]. Allowing to run the project through Google Colab. This IDE is especially useful for data science because it allows splitting the code across different cells, where each cell returns a different result. Furthermore, these results are also saved along with the code, allowing us to visualize the results of previous executions, avoiding the need to rerun everything. Moreover, markdown cells can also be used to write text, providing the tools to further increase the readability of the project. For example, by dividing different parts of the code into sections.

Another feature that Jupyter brings is the ability to create different environments for the execution of a project with a specific configuration of package versions. This project ended up having to use two different environments because of the need to use two versions of TensorFlow. One for almost all code and another for running a specific model. This situation was caused during the recreation of the model Atalaya (Pérez & Luque, 2019), resulting in the need for using the ELMO model (Peters et al., 2018), which can only be executed on a different version of the rest of the project. Moreover, to ease the creation process of the environments, two shell scripts were created on the folder of requirements.

All the project was executed using an i7-7700k CPU (5 GHz), 16 GB of RAM (DDR4) and NVIDIA 1080 (8 GB RAM). Which brought several limitations, especially for deep learning where some models could not be stored. For example, BERT large (Devlin et al., 2018) or RoBERTa large (Liu et al., 2019), which had better performance than the smaller versions and could have resulted in better results. Also, some models were able to be stored, but not the data, which resulted in a need for reduction in the batch size, which could have decreased the performance. This was a problem especially for datasets with larger maximum character length as Detoxis, which resulted in not being able to train even with a batch size of 1, while HatEval did not have any problem.

[4] "Python.org." https://www.python.org/.
[5] "Jupyter Notebook." https://jupyter.org/.

3.1.1 Modules

As previously mentioned, Python allowed the use of a diversity of packages and modules for the execution of the project. Which has some libraries already built-in for a diversity of tasks. For example, the module “re” provides regular expression matching operations or “random” that implements pseudo-random number generators for various distributions. The main libraries used during the project are the following:

Numpy is one of the most well-known libraries in Python for scientific use, that provides efficient support for large, multi-dimensional arrays and matrices, through the use of high-level mathematical functions. Which can bring a performance increase of x50, in comparison to traditional Python lists, through the use of optimized memory location. The library works as a base for other packages that will later be explained. The library is used across the project, whatever there is the need to interact with arrays, such as the features of the text.

SciPy is a library that expands the previously mentioned module with the addition of more optimized and new functions that are frequently used in Data Science. The SciPy ecosystem includes general and specialised tools for data management and computation, productive experimentation, and high-performance computing. However, this module is mainly employed for managing sparse arrays. Which is a type of data that is more suited for storing large quantities of data with the value zero. Such as the Term Frequency-Inverse Document Frequency matrix, reducing the amount of memory needed for storing, in comparison to a dense array.

Pandas is a package that provides fast, flexible, and expressive data structures for managing big data, through the use of functions for analyzing, cleaning, exploring, and manipulating data. The library is crucial in data science because it facilitates the process of managing and evaluating data. Pandas, in the project, is used across as the central data structure for storing the datasets. Even all the predictions of all models are stored and loaded in disk through this module.

Matplotlib is a low-level graph plotting library for creating static, animated, and interactive visualizations in Python. The module is utilized across all the notebooks, for generating all the graphs. Such as the ones in the basic statistical analysis or the results of training.

Seaborn is another library for making high-level statistical graphics in Python, that is built on top of matplotlib and is closely integrated with pandas data structures. This module is sometimes used instead of the previous one because it generally tends to generate better-looking graphs. However, depending on the task, sometimes it is not possible to substitute with matplotlib.

Scikit-learn (or *sklearn*) is a machine learning library for Python that features a multitude of basic models. Such as classification models, regression models, clustering models, dimensionality reduction techniques, model selection and even preprocessing. The module implements almost all machine learning models, excluding deep learning neural networks. The package is mainly used, in the project, for the implementation of Term Frequency-Inverse Document Frequency and SBERT (Reimers & Gurevych, 2019).

Tensorflow is another machine learning that allows the creation of flexible neural network architectures that can be trained across a variety of platforms (CPUs, GPUs, TPUs), originally developed by Google. The library is mainly used as a base package for other libraries, excluding the Atalaya model, which is designed entirely on TensorFlow. Furthermore, this was the module that caused the need to create two Jupyter environments, due to the incompatibilities between versions 1.x and 2.x.

TensorFlow Hub is a module related to TensorFlow, that allows downloading trained machine learning models that are ready for fine-tuning. The package is only used for downloading the ELMO model (Peters et al., 2018) for the recreation of the architecture the team Atalya (Pérez & Luque, 2019) used in the SemEval-2019 competition (Basile et al., 2019).

PyTorch is a machine learning framework similar to TensorFlow, that is used in applications such as computer vision and natural language processing, developed by Facebook. It also allows the modules to be trained through a variety of platforms (CPUs, GPUs, TPUs). Similarly, the package is mainly used as a base for other libraries. Also, to modify small parts of the BERT (Devlin et al., 2018) model and training and validating models.

Transformers is a machine learning library that provides general-purpose architectures for Natural Language Understanding and Natural Language Generation with pre-trained models and deep interoperability between PyTorch and TensorFlow. This is the main module used for getting the different pre-trained deep learning architectures that are fine-tuned for the competition.

SentenceTransformers is a Python package for state-of-the-art sentence, text and image embeddings. The concept is based on the paper of SBERT (Reimers & Gurevych, 2019) which generates fixed size embeddings through the use of BERT, similar to Universal Sentence Encoder (Cer et al., 201). In the project, it is used for generating the embeddings of the text inputs of the datasets.

Optuna is a python hyperparameter optimization module for deep learning, which uses state-of-the-art algorithms for efficient search space while pruning unpromising results. The library is used for hyperparameter search for the deep learning model of BERT.

Pure Python Spell Checking is a library for simple spell checking, that uses Levenshtein Distance algorithm to find the best permutations that would result in a word of a frequency list. Resulting in the word that is more likely to be the correct spelling. The library is used on the project during the preprocessing.

Pycontractions is a python library for expanding and creating common English contractions in the text through the use of deep learning models. By using the context and simple replacement rules for English contractions. Another library is used for the preprocessing of the project. The project uses the word2vec model of Google (Mikolov et al., 2013).

SMOTE-variants is a package that implements 85 variants of the Synthetic Minority Oversampling Technique (Chawla et al., 2002) for Python. That allows it to generate new entries from the data to get the same ratio between negatives and positives, reducing the unbalancedness of the feature. The project uses this module on SBERT embeddings to try to train a better model.

Pickle 5 is a Python module that backports all features and APIs added in the pickle module in Python 3.8.3. This package only is used to allow compatibility between the two

environment versions of Jupyter, allowing the Atalaya environment to store and read results as the main environment.

3.2. Analysis of the dataset

This subsection explains a basic analysis of both datasets. Such as the targeted features and information related to the input text. Additionally, it mentions basic information on how the dataset was constructed and other relevant information.

The main framework for the different models first started with only the English dataset of SemEvel-2019 Task 5 (Basile et al., 2019) for accomplishing subtask 1. After the addition of some more models to the framework, it was modified for the use of both English and Spanish datasets, at the same time. Finally, nearly at the end of the project, the framework was modified for the use of the DETOXIS[6] dataset. However, due to the lack of time (only two weeks instead of several months the competition had), the priority ended up being trying new technologies for better scores, at the expense of not analysing the dataset during the time. Which ended up being completed after the competition finished. However, fortunately, the analysis did not bring any major changes that would result in any performance increase.

Fortunately, both datasets have similar targets, which allowed the reuse of almost all functions, with little modification. As previously mentioned, due to the lack of time, some extra features of DETOXIS could not be taken advantage of during the competition. Such as the extra binary features for training and information related to the comment thread.

3.2.1 HatEval

There are two datasets used on task 5 of the SemEval-2019 competition (Basile et al., 2019), one in English and another in Spanish. The dataset is a compilation of messages extracted from Twitter where the task is divided into two classification subtasks: a main for detecting the presence of hate speech (evaluated with macro-averaged F1-score) and a finer-grained one devoted to identifying hateful content if there is an aggressive attitude, and if the target being harassed is a specific individual or a generic group (also evaluated with macro-averaged F1-score).

The dataset was collected from July to September 2018, with targeted categories being immigrants and women. This last category was collected from previous misogyny identification challenges. The tweets were collected from potential victims of hate accounts, history of identified haters and filtering through streams of keywords (i.e. hashtags, keywords).

The entire dataset is composed of 19600 tweets, 13000 for English and 6600 for Spanish. Also, the targets are distributed across 9091 immigrants and 10509 women. Furthermore, the provided datasets are split into three parts for both languages: train, development and test. The first one will be used for training all models, while the second one will be used to evaluate the trained models for the user and the last one will be evaluated for the competition. This division of datasets allows for a comparison of the different models before

[6] "Welcome | DETOXIS-IberLEF 2021 - Wix.com." https://detoxisiberlef.wixsite.com/website.

submitting them to the competition. However, as the competition has ended, it will allow for more data points of comparisons between the scores of the recreation and the original team.

The datasets are split into:

- *Train*: 9000 entries for English and 4500 entries for Spanish.
- *Dev*: 1000 entries for English and 500 entries for Spanish.
- *Test*: 3000 entries for English and 1600 entries for Spanish.

In the combined English dataset of all subsets, there is not any case where a Target Range (TR) or Aggressiveness (AG) feature is true while Hate Speech (HS) is false. This indicates that the second task can be solved as a multiclassification problem of five groups. Furthermore, the distribution of features does not seem to be very unbalanced with only a 38% on Target Range (as seen in Figure 3). As the distribution of other features is dependent on Hate speech, the percentage is calculated using hate speech as the divider. Indicating that positive Hate speech and the rest of the features negative is the most common for the second task.

```
Distribution of features:
HS: 42.07692307692308%
HS -> TR: 38.19012797074954%
HS -> AG: 43.089579524680076%
HS -> TR & AG: 13.875685557586838%
!HS & (TR | AG): 0.0%
```

Figure 3: English HatEval feature distribution.

Some hashtags and user mentions seem to have some correlation with the features. Especially with target range, where most mention of a top hashtag or top user results in a negative target range (as seen in Figure 4 and 5). Another observation is that users and hashtags are relatively common in the texts. This would indicate that these special words could be used on the Term Frequency-Inverse Document Frequency. Oppositely, the same links do not seem to appear enough, with the most common only appearing 24 times (as seen in Figure 6). Also, there is some usage of Unicode characters, where we can especially observe the presence of emojis and non-standardized punctuation (as seen in Figure 7). That should be treated appropriately on the pretraining.

```
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
```

Figure 4: Information related to mentions of Twitter users on the English HatEval 2019 dataset.

```
Hashtags:
Number of mentions per tweet (MIN, MEAN, MAX): (0, 1.0751538461538461, 27)
Number of tweets that contain a mention: 0.31361538461538463
Most Common mentions:
                Count        HS  HS -> TR  HS -> AG
Mention
#buildthatwall    871  0.594719  0.040541  0.666023
#maga             630  0.549206  0.057803  0.627168
#buildthewall     476  0.733193  0.057307  0.656160
#nodaca           386  0.577720  0.035874  0.573991
#trump            281  0.572954  0.043478  0.552795
#deportthemall    265  0.883019  0.055556  0.705128
#sendthemback     252  0.730159  0.016304  0.538043
#illegalaliens    246  0.369919  0.021978  0.637363
#noamnesty        218  0.738532  0.068323  0.689441
#immigration      201  0.587065  0.042373  0.567797
```

Figure 5: Information related to the mentions of Twitter hashtags on the English HatEval 2019 dataset.

```
Links:
Number of mentions per tweet (MIN, MEAN, MAX): (0, 0.42438461538461536, 4)
Number of tweets that contain a mention: 0.37884615384615383
Most Common mentions:
                        Count        HS  HS -> TR  HS -> AG
Mention
https://t.co/qzfjsqwwj8    24  0.000000       0.0  0.000000
http://t.co/6n4v9kbtya     19  0.263158       0.0  0.000000
https://t.co/hccqmsq9vl    10  0.400000       0.0  0.250000
https://t.co/91x12t4ljg     8  0.000000       0.0  0.000000
https://t.co/sn5zuuu6nv     7  0.571429       0.0  0.500000
https://t.co/ersjopu1hw     4  0.750000       0.0  0.333333
https://t.co/c0ghvd6imb     4  0.750000       0.0  0.333333
https://t.co/t1t62atnwf     3  0.000000       0.0  0.000000
https://                    3  0.333333       0.0  1.000000
https://t.co/7fgo8dfr2e     3  0.000000       0.0  0.000000
```

Figure 6: Information related to mentions of links on the English HatEval 2019 dataset.

```
Non-ascii:
Number of mentions per tweet (MIN, MEAN, MAX): (0, 0.47584615384615386, 48)
Number of tweets that contain a mention: 0.18715384615384614
Most Common mentions:
         Count        HS  HS -> TR  HS -> AG
Mention
’         1049  0.354623  0.443548  0.298387
😂          504  0.521825  0.764259  0.216730
…          470  0.291489  0.642336  0.299270
“          340  0.332353  0.309735  0.318584
”          320  0.334375  0.317757  0.336449
ÿ          203  0.458128  0.021505  0.365591
ð          202  0.460396  0.021505  0.365591
‘          195  0.241026  0.148936  0.382979
â          169  0.165680  0.035714  0.464286
           116  0.474138  0.672727  0.272727
```

Figure 7: Information related to mentions of non-ASCII characters on the English HatEval 2019 dataset.

The tweets have a mean of 141 characters, a minimum of 10 and a maximum of 851, as depicted in Figure 8. We observe that it surpasses the limitation of 240 characters that Twitter has, but that is due to how Twitter counts complex Unicode characters such as emoji, which can be composed of several characters but are visually displayed as one. Also, the tweets contain an average of 21 words with an average length of 6, so there should not be any problem for a basic TF-IDF model to find some significantly important features on the tweet.

```
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, 5.836345554093726, 51.666666666666664)
```

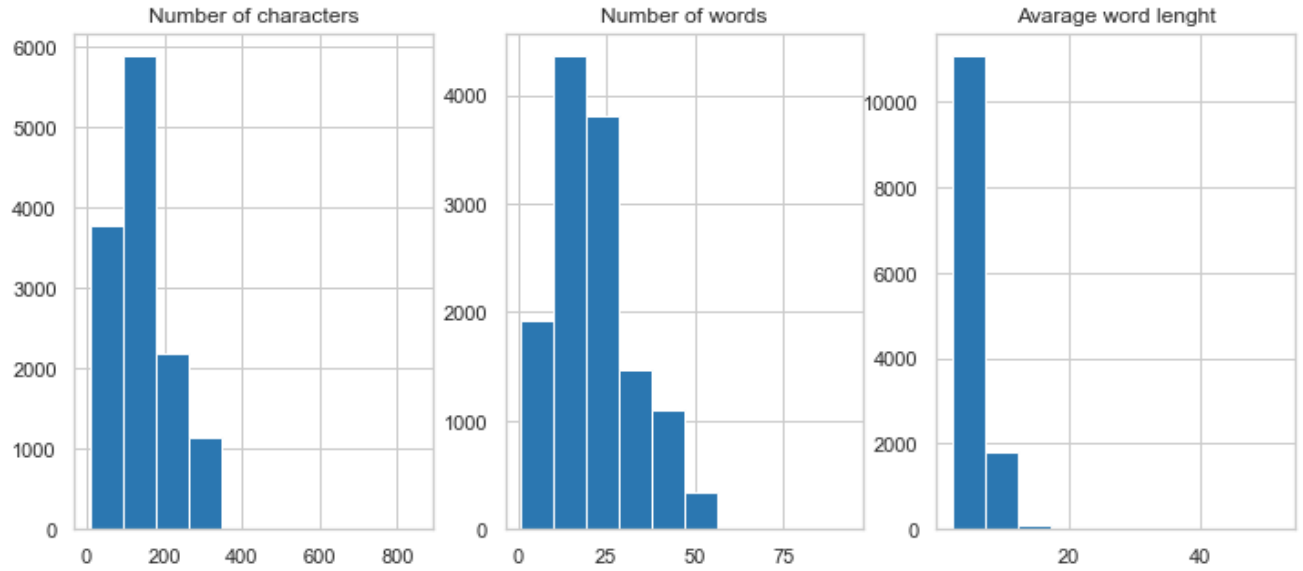

Figure 8: Distribution of words and characters on the English HatEval 2019 dataset.

The ten most common words excluding stopwords are: *bitch*, *like*, *women*, *illegal*, *get*, *#build, refugees*, *immigrant*, -, *hoe*. We can observe clearly that the dataset was specially selected for the targets of women and immigrants (as seen in Figure 9). Furthermore, it can also be observed that the presence of a hashtag could be useful for identifying hate. The most common bigrams are links (https and co) and a combination of contractions, as mentioned links should be removed. Similar to the most common words, we can observe that the dataset is chosen with specific targets.

All English sub-datasets more or less have the same distribution of features, with a deviation of less than 1% in the Hate Speech feature and 15% in the other two. Furthermore, the distribution of the most common users and hashtags change a bit between sub-datasets but is more or less consistent across all the sub-datasets. However, an interesting observation is that the test sub-dataset uses more emojis than the rest.

Similar to feature distribution, the character count, word count and average word length is consistent across sub-datasets with differences of less than 1. While the top most common words change between sub-datasets, it is also consistent across. Although, the usage of the top most common word *bitch* is more than three times used than the second *#buildthatwall*. This could indicate that the test sub-dataset contains a larger distribution of women targets than immigrants. Also, decreasing the performance of the models due to observing fewer examples than expected during training. This observation correlates with the score decrease across all teams during the competition.

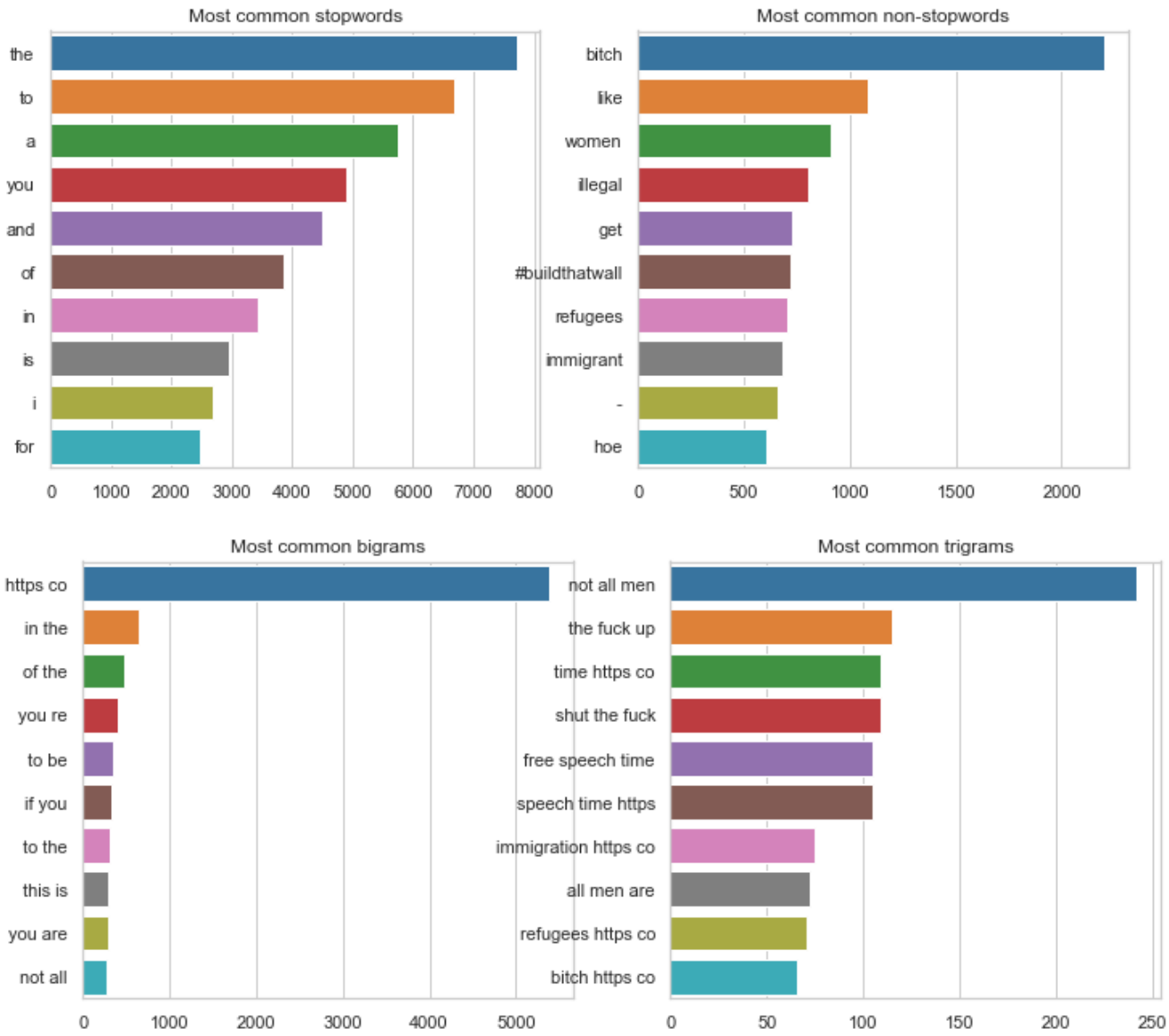

Figure 9: The ten most common words on the English HatEval dataset.

While the Spanish dataset contains a similar ratio of hate speech (41.5%) as the English dataset (42%), the other two features have a much higher percentage. Where if hate speech is positive there is a 60.8% that target range is also true. Similarly, there is a 80.9% for Aggressiveness (as seen in Figure 10). Almost double the English dataset for both features. Moreover, nearly half of the data has all three features positive at the same time. This could indicate that the data is very unbalanced and oversampling should bring benefits for the second task.

```
Distribution of features:
HS: 41.266666666666666%
HS -> TR: 60.796984383414106%
HS -> AG: 80.88314485729671%
HS -> TR & AG: 56.70436187399031%
!HS & (TR | AG): 0.0%
```

Figure 10: Spanish HatEval feature distribution.

Similar to the English dataset there are 46.4% of tweets that mention at least a user and could be used to find a correlation with the features. For example, the tweets that mention the most mentioned user (as seen in Figure 11) have only 3.8% hate speech. However, hashtags are not as mentioned as the English dataset with just 10.3% (as seen in Figure 12). Although they probably are as relevant as in English, some hashtags can contain relevant information such as #ceuta, #otgala7, #inmigrantes, for identifying the target range. Furthermore, similar to the English dataset, the links are very distinctive, with the most common only used twice (as seen in Figure 13). Also, the Spanish dataset contains emojis (as seen in Figure 14).

The tweets have a mean length of 130 characters with a minimum of 6 and a maximum of 846, as depicted in Figure 15. Which could mean that the text was extracted from a direct message, where there is not any word limit or that Twitter counts words differently. The tweets have an average length of 21.3 words with a length of 5.4 words. So there should not be any problems with creating a vocabulary using Term Frequency-Inverse Document Frequency.

```
Users:
Number of mentions per tweet (MIN, MEAN, MAX): (0, 0.7271212121212122, 50)
Number of tweets that contain a mention: 0.4637878787878788
Most Common mentions:
                 Count        HS  HS -> TR  HS -> AG
Mention
@barbijaputa       104  0.038462  0.750000  0.750000
@sanchezcastejon    65  0.523077  0.088235  0.823529
@guardiacivil       39  0.358974  0.142857  0.500000
@elmundoes          28  0.428571  0.000000  0.583333
@youtube            25  0.120000  1.000000  1.000000
@anaisbernal        22  0.000000  0.000000  0.000000
@zurine3            19  0.000000  0.000000  0.000000
@policia            17  0.470588  0.250000  0.875000
@interiorgob        17  0.470588  0.125000  0.500000
@relatofeminista    17  0.176471  0.333333  0.333333
```

Figure 11: Information related to mentions of Twitter users on the Spanish HatEval 2019 dataset.

```
Hashtags:
Number of mentions per tweet (MIN, MEAN, MAX): (0, 0.19484848484848485, 22)
Number of tweets that contain a mention: 0.10257575757575757
Most Common mentions:
                Count        HS  HS -> TR  HS -> AG
Mention
#ceuta             40  0.150000       0.0  0.333333
#otgala7           27  0.148148       1.0  0.750000
#inmigrantes       19  0.105263       0.0  0.500000
#inmigración       17  0.529412       0.0  1.000000
#inmigracion       12  0.083333       0.0  0.000000
#españa            12  0.500000       0.0  0.666667
#eleccionesya      10  0.700000       0.0  0.285714
#refugiados        10  0.200000       0.0  0.000000
#indocumentados     9  0.000000       0.0  0.000000
#venezuela          7  0.571429       0.5  1.000000
```

Figure 12: Information related to the mentions of Twitter hashtags on the Spanish HatEval 2019 dataset.

```
Links:
Number of mentions per tweet (MIN, MEAN, MAX): (0, 0.3362121212121212, 4)
Number of tweets that contain a mention: 0.31545454545454543
Most Common mentions:
                        Count   HS  HS -> TR  HS -> AG
Mention
https://t.co/cpexso5ai1     2  1.0       0.0       1.0
https://t.co/s7p45pnkxk     2  1.0       0.0       1.0
https://t.co/s6ursjjugy     2  0.5       0.0       0.0
https://t.co/eap7pkrma3     2  1.0       0.0       1.0
https://t.co/myqvt7kt8m     2  0.5       0.0       1.0
https://t.co/2pr0yptnca     2  1.0       0.0       1.0
https://t.co/47cdr1xuoy     2  0.0       0.0       0.0
https://t.co/wm7yeqgvxd     2  1.0       0.5       1.0
https://t.co/4qgbnyklpo     2  0.0       0.0       0.0
https://t.co/vexrwwos7k     1  1.0       1.0       1.0
```

Figure 13: Information related to mentions of links on the Spanish HatEval 2019 dataset.

```
Non-ascii:
Number of mentions per tweet (MIN, MEAN, MAX): (0, 2.101060606060606, 60)
Number of tweets that contain a mention: 0.7410606060606061
Most Common mentions:
         Count        HS  HS -> TR  HS -> AG
Mention
á         2866  0.411026  0.515280  0.790323
í         2269  0.362715  0.373026  0.715674
ó         1826  0.319825  0.333904  0.654110
ñ         1467  0.398091  0.380137  0.720890
é         1226  0.340946  0.497608  0.729665
ú          746  0.414209  0.686084  0.799353
¿          355  0.447887  0.283019  0.402516
😂          265  0.430189  0.885965  0.780702
ü          242  0.272727  0.484848  0.696970
¡          198  0.444444  0.318182  0.727273
```

Figure 14: Information related to mentions of non-ASCII characters on the Spanish HatEval 2019 dataset.

The most common words (excluding stopwords) indicate that the dataset was extracted from a toxic environment of the targeted categories, with words such as *puta*, *callate*, *perra*, *inmigrantes*, *valla* (as seen in Figure 16). Even the most common bigrams and trigrams are mainly negative words, excluding the bigram for links (*https co*). The increase in the presence of these words could be attributed to the large increase in the features of target range and aggressiveness. Also, almost half of the top 10 bigrams are contained in the top 10 trigrams.

The distribution of features of the Spanish dataset is almost the same across all the sub-datasets. With a deviation of less than 3% for hate speech, target range and the combination of target range and aggressiveness, and less than 7% for aggressiveness. Similarly, the most common mentions are almost the same, with some small changes in order. Same for the most common words, bigrams and trigrams. Also, the average number of characters, number of words and word length is more or less the same with similar differences.

```
Tweets: (MIN, MEAN, MAX)
Number of characters: (6, 130.32954545454547, 846)
Mean of Number of words: (1, 21.29909090909091, 88)
Mean of Avarage word lenght: (1.9090909090909092, 5.406306823839929, 18.0)
```

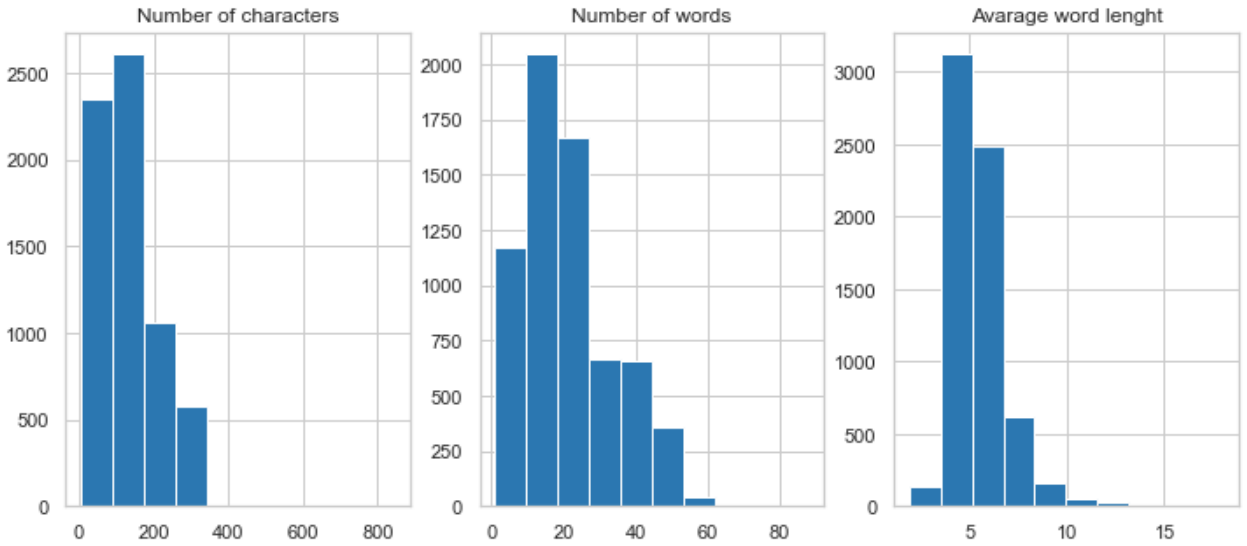

Figure 15: Distribution of words and characters on the Spanish HatEval 2019 dataset.

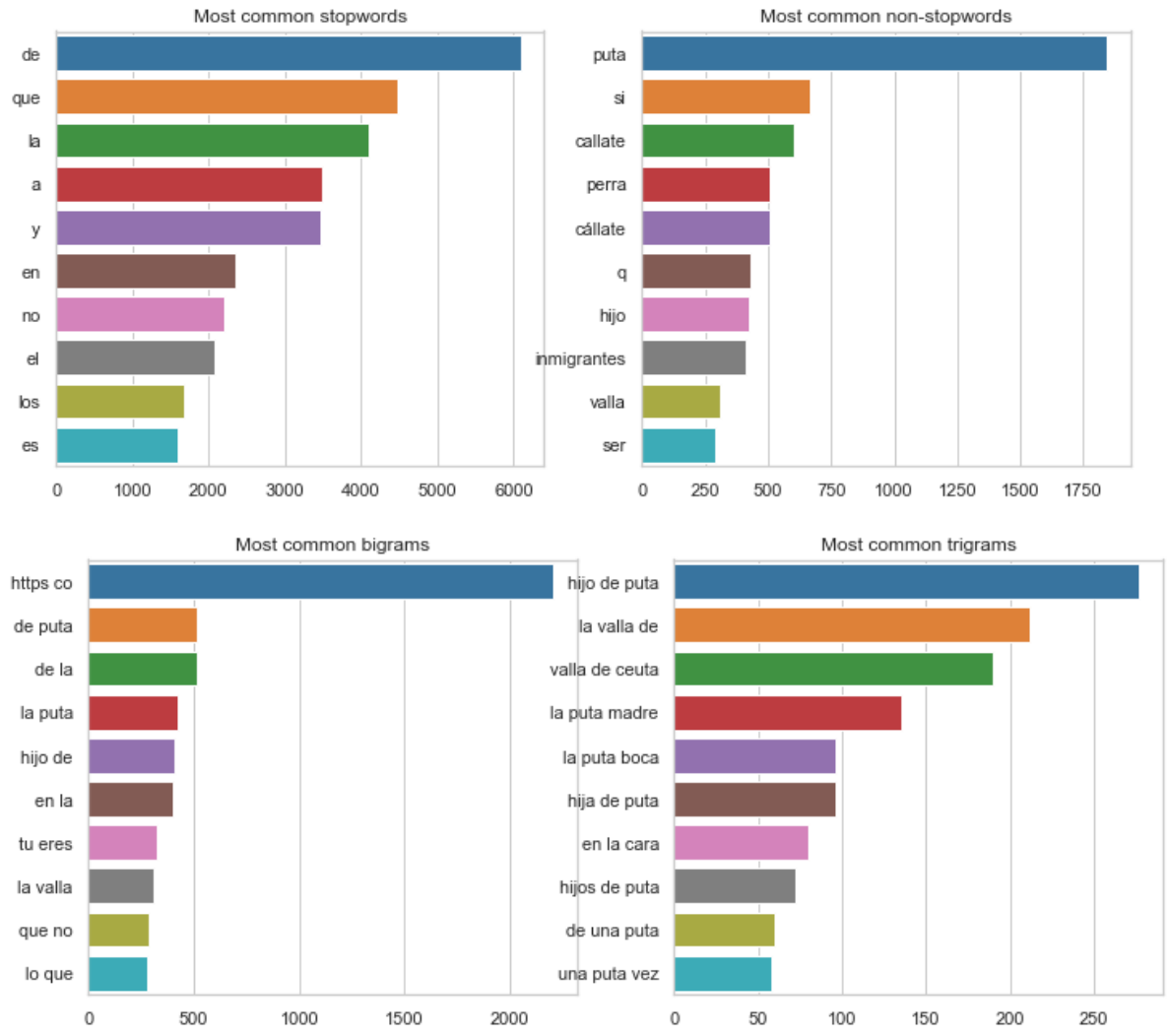

Figure 16: The ten most common words on the Spanish HatEval dataset.

3.2.2 DETOXIS

The DETOXIS dataset[7] is a Spanish dataset based on NewsCom-TOX corpus, which consists of 4354 comments posted in the response section of different Spanish online newspapers (ABC, elDiario.es, El Mundo, NIUS, etc.) and discussion forums (such as Menéame) from August 2017 to July 2020. The articles were chosen through keyword-based search with topics related to immigration. The articles then were filtered by how controversial the subject was, the potential toxicity and the number of comments posted (with a minimum of 50). The number of comments selected ranges from 65 to 359 per article, with toxicity of approximately 30%.

The dataset contains two objective features, toxicity and toxicty_level, which correspond to different tasks. The first one, is a binary classification (scored using the standard F1 score), while the second, is a multitask classification (scored using Closeness Evaluation Metric). Which is composed of 4 levels:

- toxicity_level_0: not toxic
- toxicity_level_1: mildly toxic
- toxicity_level_2: toxic
- toxicity_level_3: very toxic

Furthermore, the dataset contains extra binary features that could be useful for training: argumentation, constructiveness, stance, target, stereotype, sarcasm, mockery, insult, improper language, aggressiveness and intolerance. Also, includes information related to the conversation, such as the topic of discussion, the thread_id, comment_id, reply_to.

Each entry was annotated with at least 3 annotators from two expert linguists and two trained annotators who are students. In case of disagreements, the annotations are discussed until an agreement is reached.

The dataset is composed of 4354 comments which are divided into 80% (3463) for training and 20% (891) for the test. However, I split 20% (693) of the training dataset for validation, leaving just 2770 for training. Allowing the use of the same framework constructed for the HatEval datasets.

The combined dataset has only toxicity of 31.8%, which indicates that the dataset is unbalanced and using oversampling could result in a performance increase (as seen in Figure 17). Also, the dataset seems to contain some errors with the labels, because the percentage between the cases of toxicity level 0 and toxicity negative is 100.03%. Similarly, the sum of percentages of all levels (excluding 0) between the number of cases of the toxicity level and positive toxicity is only 99.93 %. Moreover, the toxicity level is unbalanced, because level 1 has three fourths of the total.

[7] "Welcome | DETOXIS-IberLEF 2021 - Wix.com." https://detoxisiberlef.wixsite.com/website.

```
Distribution of features:
Toxicity: 31.832797427652732%
!Toxicity -> Toxicity Level (0): 100.03369272237197%
Toxicity -> Toxicity Level (1): 71.86147186147186%
Toxicity -> Toxicity Level (3): 5.6998556998557%
Toxicity -> Toxicity Level (2): 22.366522366522368%
```

Figure 17: DETOXIS feature distribution.

Although in the HatEval dataset the texts were extracted from Twitter, which has special words such as user and hashtags, in DETOXIS there no special patterns. So, the only pattern searched was Unicode characters, as depicted in Figure 18. There are some words with accents and the character *ñ.* These words could be standardized during the preprocessing for the Term Frequency-Inverse Document Frequency. Furthermore, it seems that the texts do not contain emojis.

```
Non-ascii:
Number of mentions per comment (MIN, MEAN, MAX): (0, 3.153881488286633, 53)
Number of comments that contain a mention: 0.7429949471750115
Most Common mentions:
         Count  toxicity    0         1          3          2
Mention
í         3880  0.324179  0.0  3.084713   9.254139   6.169426
á         2838  0.321649  0.0  3.108981   9.326944   6.217963
ó         2373  0.297060  0.0  3.366323  10.098969   6.732646
ñ         1603  0.338410  0.0  2.954995   8.864986   5.909991
é         1405  0.316911  0.0  3.155458   9.466373   6.310915
ú          665  0.331610  0.0  3.015590   9.046771   6.031180
¿          588  0.308562  0.0  3.240838   9.722513   6.481675
¡           77  0.530055  0.0  1.886598   5.659794   3.773196
”           63  0.356589  0.0  2.804348   8.413043   5.608696
“           52  0.284314  0.0  3.517241  10.551724   7.034483
€           51  0.352273  0.0  2.838710   8.516129   5.677419
ü           38  0.435294  0.0  2.297297   6.891892   4.594595
»           22  0.476190  0.0  2.100000   6.300000   4.200000
«           20  0.475000  0.0  2.105263   6.315789   4.210526
à            9  0.428571  0.0  2.333333        NaN   4.666667
…            8  0.277778  0.0  3.600000        NaN   7.200000
’            7  0.200000  0.0       NaN        NaN  10.000000
•            5  0.333333  0.0       NaN   9.000000        NaN
–            5  0.181818  0.0  5.500000        NaN        NaN
´            4  0.875000  0.0  1.142857        NaN        NaN
```

Figure 18: Information related to mentions of non-ASCII characters on the DETOXIS dataset.

The comments have a mean of 210.4 characters, with a minimum of 1 and a maximum of 3270 (as seen in Figure 19). The average number of words is 37.3, with a mean word length of 4.8. The maximum of words (556) is too big and could bring problems to a neural network model with a size limit (i.e. BERT). Also, the bigger the model, the more limitations on data due to the amount of GPU RAM I have available (8 GB). Although, the Term Frequency-Inverse Document Frequency should not have problems extracting relevant words.

```
Tweets: (MIN, MEAN, MAX)
Number of characters: (1, 210.44533762057878, 3270)
Mean of Number of words: (1, 37.315571887919155, 556)
Mean of Avarage word lenght: (1.0, 4.826279506661524, 105.0)
```

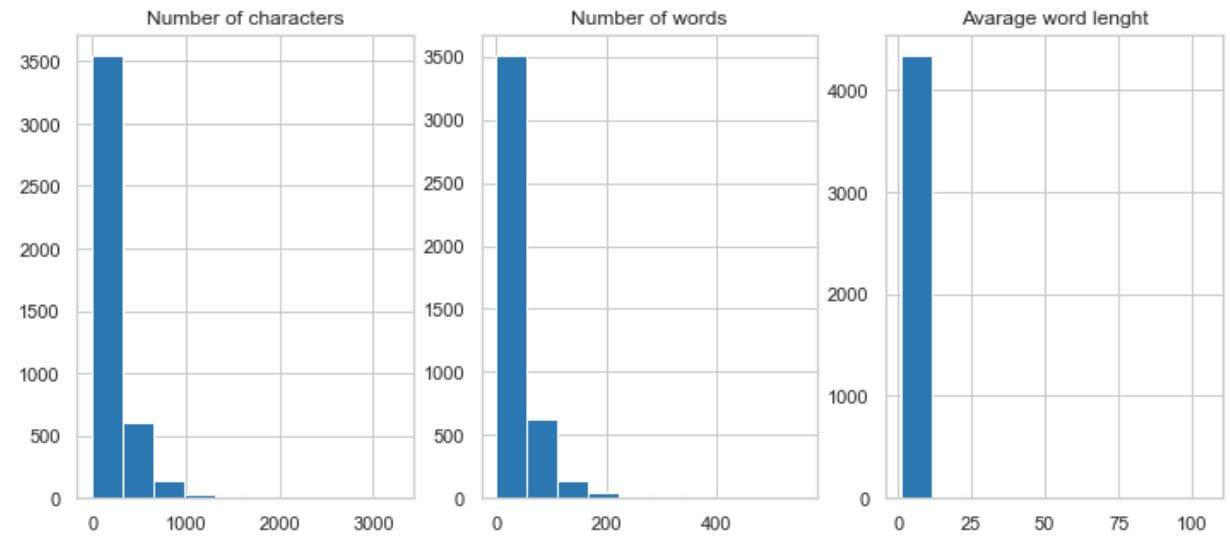

Figure 19: Distribution of words and characters on the DETOXIS dataset.

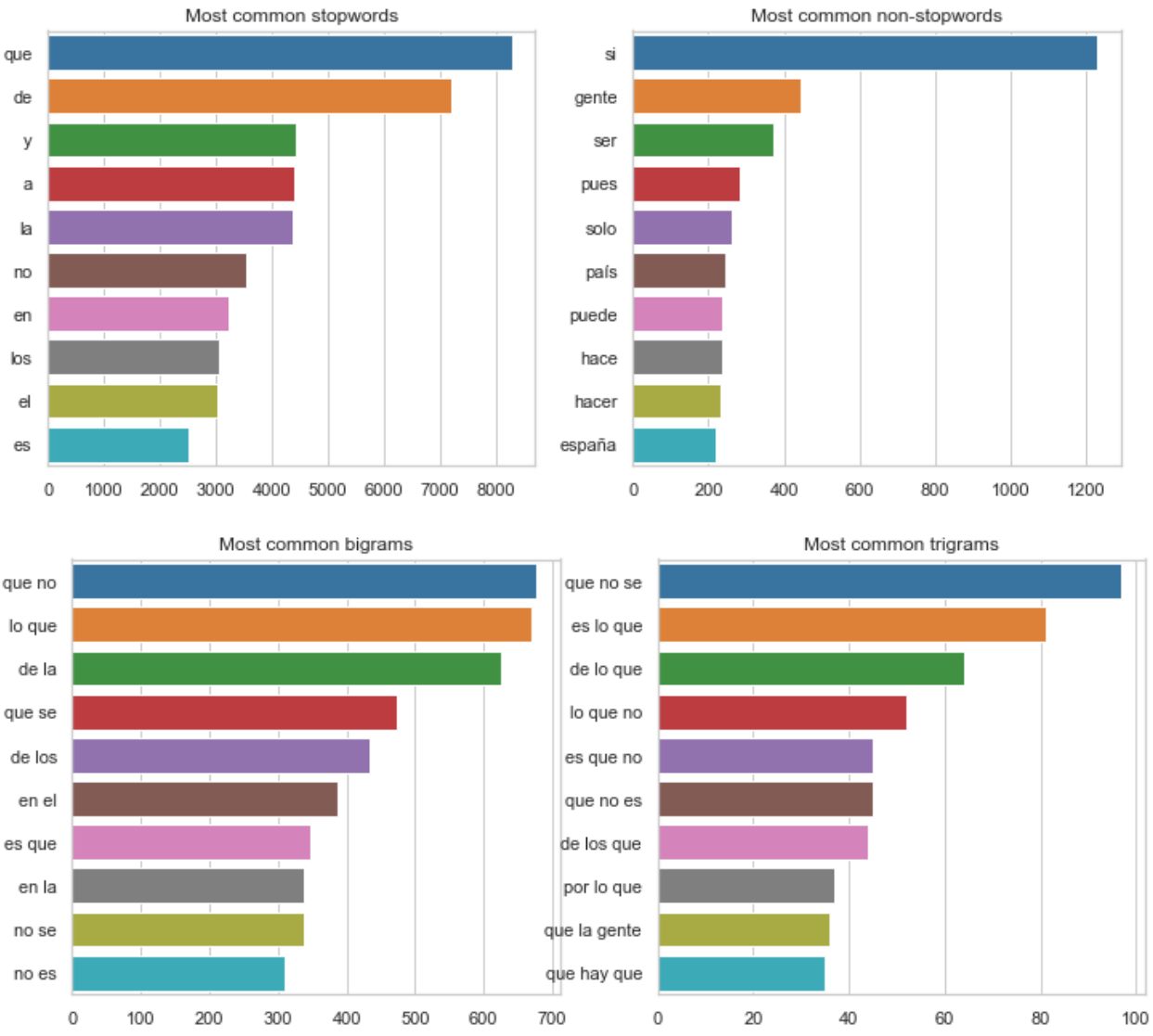

Figure 20: The ten most common words on the DETOXIS dataset.

The most common words (excluding stopwords) are commonly used and do not seem to allow identifying the dataset targets (as seen in Figure 20). This could be a consequence of the lower percentage of the dataset targets, the opposite of the HatEval dataset. Similarly, bigrams and trigrams do not reveal much information about the dataset and are mainly composed of combinations of prepositions.

The training and development dataset contains a higher toxicity level than the test dataset with 31.8%, 32.5% and 26.8% respectively. This is a relevant difference taking into account how unbalanced the data is already. Furthermore, the test dataset does not seem to contain incorrect labels problem, having the same number of entries for where toxicity is negative and where the toxicity level is 0. However, the toxicity levels are more or less the same across the dataset for training and development, whereas the test has bigger differences. Finally, the most common words, bigrams and trigrams are more or less similar, mainly composed of prepositions.

3.3. Data Preprocessing

Data preprocessing is the step in which data gets transformed to facilitate the process of interpreting the features. In our case, the features are the input text of the datasets, where we will apply some transformations (replacement and removal) to facilitate the different tokenizations.

Three different types of preprocessing are applied to the datasets depending on the model type. Starting with the one for Term Frequency–Inverse Document Frequency which uses all the techniques. Another for generally all neural networks that apply only some techniques (i.e. the removal of users, links and hashtags). The last one, only for GPT-2 (Radford et al., 2019), where some word groups are replaced instead of removed. The main cause is that GPT-2 due to its architectural design can not predict with an empty input string. Some of the following techniques are applied depending on the preprocessing.

First, all characters have been lowercased, to avoid during the use of Term Frequency–Inverse Document Frequency, the same word, but with different combinations of lowercase and uppercase, being counted as distinct words. As the latter case does not usually bring extra meaning, it should increase downstream task performance. Also, some models such as BERT (Devlin et al., 2018) has a version for just lowercase.

Second, as mentioned in the dataset analysis, links seem to be very different from each other, decreasing their meaning. So for all models, excluding GPT-2, all links are removed. Instead, in GPT-2, links are replaced with the word “link”. The main reason is that this model does not accept an empty input after the filter.

Thirdly, some emojis are replaced with the meaning that it represents, using two dictionaries[8]: *simple_emojis.json* and *complex_emojis.json*. In case the emojis are not found in the dictionaries, they are removed from the text, allowing all models to gain from this step. Furthermore, neural networks could benefit more, being able to use the description of the emojis for extracting more context.

Fourthly, the words that are contained inside the stopword list of the respective language are removed. Stopwords are usually the most common words in a language. Normally these words are so common in a text that the feature correlation with the number of times the word

[8] "Full Emoji List, v13.1 - Unicode." https://unicode.org/emoji/charts/full-emoji-list.html.

appears is very small. However, later this probably became not as relevant in Term Frequency–Inverse Document Frequency, due to a filter of words that appear more than 65%. Furthermore, it is worth mentioning that neural networks use context information even from stopwords. Although, a study (Qiao et al., 2019) has shown that even though the stopwords receive as much attention as non-stopwords, the performance is similar. However, just in case, the stopwords were not removed so the neural networks could get all the context.

Fifthly, correcting small spelling mistakes of words through the usage of Levenshtein distance. The grammatically correct word and the misspelt word would end up in the same group. Contributing to performance increases in general into the traditional approach. Just in case the correction is not applied on the deep learning preprocessing to avoid possible errors of wrong spell correction. While on the traditional approach, even in the worst-case scenario, all wrongly corrected words will end up going in the same group. This should result in a marginal decrease in downstream task performance that it is easily compensated with good corrections.

Sixthly, removing numbers from the text because semantically, they do not bring extra meaning. All models can benefit from this, except GPT-2. Similarly to the second, instead of removing the number, it is replaced with the word “number”.

Seventy, replace “&” with “and” and standardize contractions and other special characters, mainly punctuation. The first replacement is mainly useful for Term Frequency–Inverse Document Frequency due to joining groups of words with the same meaning. While the rest of replacements are not as useful, since it only processes words, users and hashtags. On the contrary, the deep network models can benefit more from the standardization of punctuation, because they can use it for extracting some extra context. For example, Unicode characters such as “…” can be replaced with dot characters instead. This is mainly done because of the limitation of the number of characters of the deep learning tokenizers. The cause of the characters is that some Twitter users use these special Unicode characters to be able to write more.

Eighthly, all English contractions are expanded through the use of Google word2vec (Mikolov et al., 2013). The model uses the context and some of the basic rules to predict what the original abbreviated word was. Similar to the previous preprocessing, allow grouping for both cases where the word was contracted and when it was not. Furthermore, avoids the word which received the contraction to not be counted as the same group as the word without it. The change is not as significant for the deep neural approach because they are already trained for it. So it is not used for the preprocessing of the deep learning models.

Ninthly, a lemmatization is applied to all words to allow the grouping of words from the same family. Similar to previously mentioned, the Term Frequency–Inverse Document Frequency can benefit from grouping words with very similar meaning, allowing downstream task classifiers to find correlations with the features more easily. However, neural networks will not use it as it removes some context that could be key.

Tenthly, some abbreviations are replaced with the original through a self-made dictionary (the entire dictionary can be found in Appendix A), mainly designed from common abbreviations used on Twitter. All models can benefit from understanding the meaning. This technique is used for similar reasons to the emojis (the third technique)

Eleventhly, for the Spanish datasets, all accents are removed. Facilitating the aggrupation of words with and without accents. This is more targeted to the HatEval dataset, due to being extracted from Twitter where people usually do not write using accents. While DETOXIS is probably the opposite because the comments are extracted from online newspapers.

3.4. Implementation

The implementation of the code has been written through the use of Jupyter Notebook. Also, it has been divided into two different files. The main notebook (“Main”), in which all datasets are investigated and the different models are trained. The other (“Results”), where the results of all models can be easily seen, facilitating comparisons. Moreover, the main notebook is divided into 15 sections. However, before that, some imports are made to modify some settings and make the results prettier.

The general pipeline of the project can be divided into three branches (as seen in Figure 21) depending on the model/technique used. The blue and green rectangles represent the input (a dataset) and the output (a prediction), respectively. The yellow boxes represent transformations to the data, where the encoder in the implementation can be Term Frequency-Inverse Document Frequency (Robertson, 2004) or SBERT (Reimers & Gurevych, 2019). Furthermore, the preprocessing varies depending on the model used, as described in Section 3.3. The red rectangles are the different models used for classification, where the deep learning models can be any version of BERT (Devlin et al., 2018) or GPT-2 (Radford et al., 2019). Also, the machine learning classifier refers to traditional machine learning models. Finally, the orange rectangle refers to the technique used for searching the best hyperparameters for a model. Although this technique also uses the encoded data and the model, the figure was simplified to facilitate the comprehension of the pipeline.

First, all paths are preprocessed depending on the type of model used, as described in Section 3.3. The leftmost route encodes the input text into a fixed-size representation through a transformation that can be based on classical methods (i.e. TF-IDF or BOW (Zhang et al., 2010)) or deep learning encoders (i.e. SBERT or USE (Cer et al., 2018)). Then the encoded data can be passed directly to the classifier or be oversampled using SMOTE (Chawla et al., 2002). After the machine learning model is trained with the training sub-dataset, the models can make predictions for all the sub-datasets. Moreover, these models can use grid search cross-validation for finding the best hyperparameters. As mentioned, this method uses both the encoded data and the model.

The middle path is for the recreation of the model of the Atalaya team (Pérez & Luque, 2019), which accepts two inputs, one directly from the preprocessed data and the other after being encoded through Term Frequency-Inverse Document Frequency. Then the model combines both inputs to generate a prediction through a neural network. Similarly, the model can make predictions of the different sub-datasets only after being trained.

The rightmost path is for the deep learning models that have the capacity for a sequence to classification. Furthermore, they have been pre-trained with large amounts of data, and some extra layers are added to allow the classification of data. However, the models do not directly use the preprocessed data. Instead, they use a tokenizer, which is a transformation of the data. After the model is trained, it makes predictions through the neural network without updating the weights and biases of the different layers.

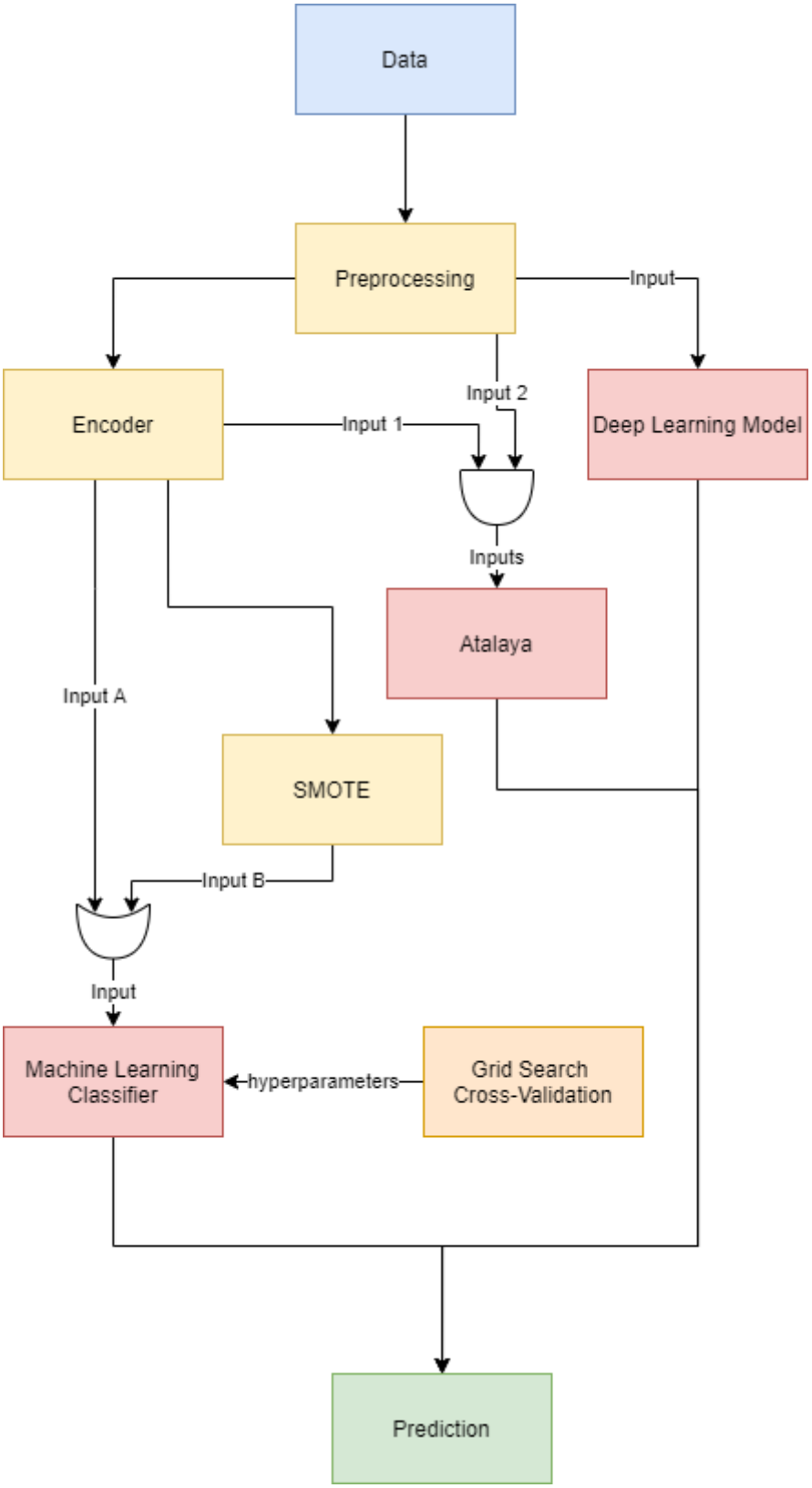

Figure 21: General pipeline of the different types of models implemented.

3.4.1 Load datasets

The first section is where the three datasets are loaded (two of HatEval and one of DETOXIS). All datasets are stored using comma-separated values (CSV) files, in which each feature of an entry is separated by commas. Also, each entry is in a different line. Also, all datasets are separated into different sub-datasets.

The HatEval dataset is composed of three parts (train, development and test) and two languages (English and Spanish). The train part, as its name suggests, is the data that will be used for training the different models. The development part can be considered to form part of the training dataset for the final training of models. However, it is used as a way to check the performance of the models (a validation) with data that has never been seen. Finally, there is the test section, where normally during the competition, the provided dataset does not contain the features that we are trying to predict ("HS", "TR", and "AG"). Then the participants can be evaluated through the prediction of this unbiased data. The ranking is made through how better the predictions were to the rest of the competitors using a metric (F1 macro). However, because the competition has ended, the organizers already provided the dataset with the labels (features trying to predict).

The DETOXIS dataset is currently composed of three parts (train, test and test with labels). Differently from the previous one, the train data was not already split into two. So, once loaded the data is separated into two different datasets: one for training and another for development. The new training dataset contains only 80 % of the original data, while the development dataset contains 20%. Furthermore, the datasets were chosen to be split randomly without any strategy because it seemed that the test dataset did not have a similar distribution as the train, which ended up being true. Similar to HatEval, during the competition, the only other dataset available was the test, which did not contain any features that had to be predicted or the extra features. Once the competition ended, the organizers released the test dataset with labels.

During the competition, the scores obtained on the development dataset did not influence any decision of parameters. To find the best parameters any model was made through the use of cross-validation on the training dataset. The only exception was at the time of choosing the models for the competition. Mainly to avoid adding biases on the validity of the scores obtained of development.

All parameters that use the datasets will contain the words: English, Spanish or DETOXIS. Even though the DETOXIS dataset is also Spanish, the first two would be from HatEval datasets of the corresponding language. The main reason was that the implementation of the code was made using only the HatEval datasets.

3.4.2 Basic statistical analysis

In the second section, a small statistical analysis of the data is performed on all datasets. The analysis will be performed on the combination of three parts (train, development and test) of each dataset. Also, the process will be performed on each sub-dataset separately. Also, for the HatEval datasets, an extra analysis was made to check if there are differences between data with only positive hate speech.

The analysis applies a small preprocessing of lowercasing all characters, before starting printing the statistics. Also, in some sections of the analysis, changes a bit depending on the dataset.

Firstly, information printed is the feature distribution in which the distribution of the features predicted is shown. It is useful for getting an understanding of how unbalanced the labels are. Also, because in both competitions the second task (multi-class classification) features are dependent on the first one (toxicity classification) feature, the second tasks features are displayed as the percentage of the total positive of the first feature. This is because the rest of the features can not be positive if the first feature is not.

Secondly, prints information of a specified pattern, which contains the number of mentions per entry (min, max and average), the number of entries that at least mention once and a table containing the most common mentions of the pattern. The table has the columns: the total quantity of time the word appears and the correlation it has with the different features of the dataset. Allowing to find correlations between a pattern and a specific feature. The common pattern searched across all datasets are non-ASCII characters.

Thirdly, shows basic information related to the text feature. Starting with printing the minimum, maximum and mean of the number of characters, several words and the average word length of a comment. Also, it displays a histogram of the previously mentioned statistics.

Fourthly, displays four sorted horizontal bar graphs that are grouped in a grid of 2x2. The graphs contain the most common stopwords, the most common non-stopwords, the most common bigrams and the most common trigrams.

The main difference between the two datasets (DETOXIS and HatEval) is that the features are different. The second-task HatEval features are the combination of the three binaries. While DETOXIS are stored using a single multivalue feature. Another change is the patterns searched, whereas the HatEval text is extracted from Twitter where there are special words, such as users (words starting with a @) and hashtags (words starting with #), while the DETOXIS dataset does not contain any special words.

3.4.3 Term Frequency–Inverse Document Frequency

This section contains the methods for preprocessing the data, the pipeline for transforming the data and a method to manage to store the first time Term Frequency–Inverse Document Frequency matrix and loading the rest of the times. Also, contains the different methods to preprocess the text (as explained in Section 3.3).

Two auxiliary methods were constructed to track the progress of any type of recurring call that has a fixed amount. These methods can be considered a generic version of the package tqdm[9], which tracks the number of executed iterations, the remaining number of iterations, the estimated time remaining and the time that has transcurred from execution. The methods were created to keep track of the progress of the fitting and transforming of the pipelines.

In addition, there is a pipeline for each dataset that applies the different preprocessing techniques and finishes with the Term Frequency–Inverse Document Frequency vectorizer.

[9] "tqdm documentation." https://tqdm.github.io/.

Additionally, the pipeline is a bit different if the dataset is Spanish or English because the Spanish contain some extra arguments specifying which is the language that is passed as arguments to the preprocessing functions.

The vectorizer needs to first be fitted before being able to transform the data to generate the matrix. The vectorizer will keep track of the 3500 words that appear the most during training. While also excluding the words that appear in more than 65 % of the documents. The reason is that these are the words that have the highest significance (on term frequency) for the training model. Also, the reduction of dimensionality can facilitate the learning process of the models, which should result in a downstream task performance increase. Similarly, the most common words are removed because of the low weight (high inverse document frequency), providing less information to the classification model. Also, the special words of hashtags and users will be treated as other words, from the reasons explained in the analysis of the HatEval dataset, in Section 3.2.2.

The process of fitting and transforming the data can take up to 10h for all the datasets. To avoid the need to retransform the data into the same output each execution. A method to manage the pipeline was implemented. The method is designed to be the most optimal in time usage while also being robust enough, such as when the training data is loaded from disk but the development is not. Also, if the pipeline has already been fitted, it will only transform it, reducing the total time. Finally, once the text data is transformed, it will be stored on the disk. Allowing for the next execution to just load the data from the disk.

Some extra information that could have been useful was also stored on the disk. The data consisted of: the most common words of the vocabulary of the vectorizer and the names of the columns of the matrix. Similarly, this information can only be obtained from the vectorizer once fitted. The solution was that this information is stored on the disk, so that it can be loaded in posterior runs.

A method for adding extra features to the matrix was implemented but not used, such as the number of uppercase words, which could correlate with toxicity. The main reason was that it did not bring any relevant performance increase to the models. This could be caused due to the already high dimensionality of the Term Frequency–Inverse Document Frequency matrix. The code showing the relevance of these extra features is written as a comment before Section 5, which mainly consisted of checking the importance of these features on two different models: ridge classifier and random forest tree.

3.4.4 Save results

This section defines two auxiliary methods for managing predictions of all models. The method allows the storage of a prediction and some extra information at a disk through a metadata file.

The first extra information that can be stored, is the name of the model. The second is which language the model uses. The third is the task name. The fourth is the dataset type which can have three values: train, development and test. The fourth is the group, an optional parameter to indicate the type of algorithm it uses. The fifth is a description, another optional parameter explaining extra information, such as the parameter values used on the model.

The sixth is the ground truth, a binary parameter indicating if the list of values is the gold standard or a prediction for the dataset. There should be a maximum of one entry with a positive ground truth of the group of dataset type, task and language.

The method will store the results following a tree structure. Through the use of some of the specified parameters. All predictions and metadata will be stored in the folder of results. This method will allow in the notebook of Results to generate the scores of all the predictions.

There is another method for removing an entry from both the disk and the metadata. It also allows you to remove all files that are not in the metadata. This facilitates the removal of data by first removing the data of the metadata data frame and then calling the function.

Finally, this section also stores the ground truth of all datasets (HatEval English, HatEval Spanish and DETOXIS) and sub-datasets (train, development and test). The only exception is the dataset of DETOXIS that checks if the labels are included. This was because during the implementation of the code for this dataset the labels of the test were not released.

These methods were implemented to facilitate the comparison between a large number of models for each dataset. Allowing the creation of another notebook to unify all results, which will be explained in Section 3.4.16.

3.4.5 Classification models

This section tests some base models that use the Term Frequency–Inverse Document Frequency matrix as input. Furthermore, some methods to print scores are defined.

The defined functions include one for scoring a prediction, which returns the accuracy score, the F1 score and a classification report. The F1 score is different depending on the dataset, for HatEval uses F1 macro, while DETOXIS uses F1 binary. The classification report is a string table containing all relevant metrics from precision, recall, F1-score and accuracy across all labels. There is another function for printing these scores.

There are three other auxiliary functions for fitting a dictionary of items with the same data. The models are stored in a dictionary with pairs of the model and the model name. There is another method for predicting the same data and returning the scores and the predictions. Finally, there is a third for joining two dictionaries of models intercalated, which will be used in later sections.

The used models are a Dummy Classifier, Multinomial Naive Bayes classifier, Bernoulli Naive Bayes classifier, Ridge Classifier, Random Forest classifier, Support Vector Classification, AdaBoost classifier, and Multi-layer Perceptron classifier. All models are created with the default parameters, excluding the random state, which will be set to seed value of the notebook, to make it consistent across runs. Also, the Multi-layer Perceptron classifier model has early stopping enabled to avoid extreme overfitting, it has otherwise.

The same models are fitted each time for each of the different datasets. The models are not duplicated for each dataset as fitting resets the model. Once the models are fitted, the scores are printed for training and development (just to check for possible errors). Also, a compact version of the scores is printed to facilitate the comparison of results.

3.4.6 Searching the best hyperparameters

This section is devoted to finding the best parameters for all the previously mentioned models and datasets through grid search cross-validation.

The grid search cross-validation is an exhaustive search over all parameters specified of a parameter grid. Which will search for all the combinations of parameters several times. The method works by splitting the data into different training and validation sets several times.
The data is split into the number of times (folds) in each iteration, where the validation will be composed by one of the folds, while the training dataset will contain all remaining folds. Once finished, the search will allow you to rank a parameter combination through the mean on the validation.

Two common methods were refactored, due to the high similarity between searches. Furthermore, the values were ranked differently depending on the competition. The HatEval datasets used F1 macro, while for DETOXIS dataset used F1 binary.

The search only uses the data of the training dataset. Allowing later, to validate the scores through the development dataset. Also, depending on the estimated time that the grid search cross-validation will take, the number of folds used is different (5 or 10). The higher the folds the better the precision on ranking results, but it will take much longer. Furthermore, parallelism on the CPU is enabled to accelerate the total search time.

The section defines two general methods for visualizing the results of the grid search cross-validation. In case the parameter grid only contains one parameter, a two-dimensional graph displaying the parameter values on the X-axis and scores on the Y-axis. The plotted data is the mean scores (both test and validation) as a line and the standard deviation of validation as an orange area. Also, returns a table sorted by average validation score that allows getting all the relevant data of the ten best scores. Which contains the parameter values, the mean scores and the standard deviation of both train and validation.

Finally, all models have a different grid of parameters to be searched for all the datasets. Sometimes, to accelerate the search process, the grid search process is split into two. The first performs a partial search of some of the parameters, whereas the second uses the best parameters of the first to search for the remaining. Furthermore, the final results of the Multi-layer Perceptron classifier were stored on the disk because it was the model that took the longest time. This was done to ensure that if there was a power outage, once the search was completed, the results could be recovered even if the notebook was not saved properly.

3.4.7 Best Models

This section is where all models are built with the best parameters found previously through grid search cross-validation. The new dictionary of models is combined with the base models. Then the combined models are fitted. Finally, the models are asked to make predictions for all sub-datasets (train, development and test). The process is repeated for each of the datasets.

The values of the chosen parameters are generally a rounded value of the best or second result obtained in the previous section. The parameters were manually written, instead of

dynamically passed through a parameter value. This allows the execution of the section without the need to run the previous one.

The combination of models is intercalated between the base model and the model with the best parameters found. This allows for the base models to be near the optimized ones. Once combined all models are fitted with the training dataset, then predictions for all the sub-dataset are made (train, development and test). Next, the main scores are printed for all dataset types and models. Finally, all predictions are stored on results in the two different groups. The first one for the base models, called “traditional”. While the other stores the models with the best parameters called “best traditional”. Moreover, the name of the model will use the other pair of the dictionary. The difference between the base models names and the optimized models names will be that the second will add an extra “ (best)” to the name.

The process is repeated for all three datasets (HatEval English, HatEval Spanish and DETOXIS), which end up storing the predictions of all models that used Term Frequency-Inverse Document Frequency matrices.

3.4.8 Neural Networks

This section defines a lot of methods that will be used for neural network models. Including the preprocessing data for neural networks, training and evaluation for neural network models based on the library of transformers. Also, will define modified BERT classes, to try experimental modifications on models. Furthermore, will end up generating the base datasets for neural networks, excluding the one for GPT-2 (Radford et al., 2019).

The first two methods define the preprocessing for both all neural network models and for GPT-2, which will be mainly the removal of hashtags and users for neural networks. While for GPT-2 will be replaced with words and also replace the links with a word. Further explained in Section 3.3.

There is also the definition of two modifications to the original BERT architecture. One of the modified models uses the mean of the last two layers, instead of the first token. The concept is based on the paper (Li et al., 2020) of the model called BERT “last2avg”. Furthermore, several versions to calculate the average differently were implemented. Another experiment was the use of SMOTE (Chawla et al., 2002) oversampling during the training, by oversampling the results of first position (corresponding to the special token [CLS]). The results of these models are further discussed in Section 4.3.

Next, there are several refactored functions for training and validating data of neural network models. The first method is for printing an example of tokenization to check that everything is working properly. The second method is used to find the max length of tokens after tokenization of the model. This will determine the horizontal size of the models. The third method generates the datasets containing all the information needed for the training and predicting. Also, there are methods for training the model and validating the predictions. There is also a method for visualizing the training progression tracked on training. Finally, there is an implementation of cross-validation, which uses different training and prediction methods. Finally, there is a method to create a table with the results of the cross-validation.

The generation of the dataset is designed to add the special tokens if needed (e.g. BERT), add padding to have consistent inputs and truncate the text if it occupies longer than the specified max length. The max length was calculated as the minimum between the maximum

size the model can have and how long the largest input text is. Using the function the tokenizer should not truncate the text if the maximum token size is not larger than the model limit.

The training method starts with setting all the seeds of the random generators to the same value, to get consistent results across runs. Every 40 batches, displays the elapsed time from the beginning of the epoch. Also, at the end of every epoch, a validation will be run by printing some information. There is also an extra function that creates a graph of the validation loss and test loss across epochs. Also returns the statistics in a table format for easier visualization.

The evaluation works very similarly, however, the processed data does not train the model and is only repeated one time. The method will construct the prediction and truth of the model, which will be returned.

The cross-validation method performs stratified cross-validation on a specified dataset. Where the data is divided evenly into a specified number of folds, then one fold is used for validation and the rest for training. Later, the model is trained and both datasets are validated to obtain the predictions the model does. Next, both predictions are scored against the truth and are stored in a list. The process is repeated the number of folds and returns the scores of training and validation of each fold. As mentioned there is a function to combine the data of the list of both scores. The list of scores is transformed into the mean and standard deviation.

The cross-validation instead of accepting as parameters a model and an optimizer uses a generator function that should create a new copy of both each time it is called. The reason is that in each iteration the cross-validation needs a brand new copy of the model. Otherwise, the model would end up memorizing the dataset. Moreover, the optimizer also needs to be reconstructed, because the learning rate will be different after training.

3.4.9 BERT

This section loads the pre-trained BERT model and tokenizer from a checkpoint, then fine-tunes the loaded model by training with the data, evaluates the trained model to get the predictions and stores the results on disk. The process is repeated for each dataset. The checkpoint is different for the English version and Spanish since there are two versions one for English and another for multilingual which includes Spanish.

The chosen model can be the original or one of the modified versions that have been implemented. Also, the tokenizer should be loaded using the same checkpoint as the model.

Then an example is printed allowing us to compare the tokenization of a specific entry on the preprocessed and original text. This is a sanity check to ensure the preprocessing was properly applied. Next, the max length is searched through the preprocessed text inputs to decide the input and output model size. The DETOXIS dataset will return the value of the maximum length that BERT can have, which will result in the inputs longer than this size being truncated, leaving only the beginning of the text. Then the sub-datasets (train, development and test) for the model are created. Finally, the optimizer is built using, generally, the recommended parameters for pre-training. Also, the parameters batch size, epochs and device are defined.

Afterward, the model is trained with all the specified parameters using the training and validation datasets. Once finished, the scores are printed and the predictions for all the sub-datasets are made. Finally, the predictions are saved on disk. A description is also added to understand the parameters used to achieve the results. Each time the model is changed, the parameters on save should also be modified. All results are stored in the group “deep_learning”.

Finally, the process is repeated for three datasets (HatEval English, HatEval Spanish and DETOXIS). There is an extension on DETOXIS where cross-validation was made for choosing the best parameters for the competition.

The subsection starts by creating a combined set of the training and validation datasets, to check if the performance of the model increases with more data. Then the learning rate and epsilon of the optimizer are saved as a global. The model and optimizer are removed and the cache of Cuda is cleaned to ensure the creation of a model fits in the memory. Then a method to generate the model and the optimizer, which will use the saved values, is constructed.

Different tests are conducted from changing the number of epochs to comparing results with or without all training data, testing other models and testing the other models with an optimizer that has two learning rates. The best performing one seemed to be two epochs and the default optimizer parameters.

Another subsection is implemented for searching the best parameters for learning rate, epsilon on the optimizer and also the number of epochs. For more reliable results, the search should be done using cross-validation of the training data instead of the score of the development sub-dataset. However, due to the lack of time for the competition, instead it uses the results of the validation. Even though it results in adding a certain bias, the results were similar to the cross-validation. Furthermore, the best parameters end up being very similar to the recommended values by the authors of BERT (Devlin et al., 2018).

3.4.10 GPT-2

This subsection is very similar to the previous one because all methods have been refactored. When the model is first loaded, the next different variables are created. Then, the model is trained and evaluated, finishing with saving the results. The process is repeated for all datasets (excluding DETOXIS). However, it uses a different preprocessing technique in comparison to the rest of the neural network models. Also, the model is loaded a bit differently, where some settings are needed to be changed.

As mentioned, the GPT-2 tokenizer has problems tokenizing empty text, which results in errors. To prevent this, instead of removing words, they are replaced. The error only seemed to happen on the HatEval dataset, because some tweets only contained links, hashtags and users. The solution that was implemented was to replace the keywords for users, hashtags and links with a fixed word.

The process of initialization of the model is different. Starting with the need to define some settings to the tokenizer such as the padding side of the token used for padding. Similarly, the model needs to set the padding token id. This is because GPT-2 is not designed for direct model classification and instead for the next sentence prediction. Also, the official model was

only pre-trained for English. So for the Spanish datasets the pre-trained model used was *datificate/gpt2-small-spanish*[10].

Once the model is loaded, the rest of the execution is the same as in BERT. However, differently to BERT, the model for GPT-2 does not have a limit for the number of tokens. Allowing to train the model with all the data. Also, the base model generally needs more memory than BERT. Forcing the need to reduce the batch size for all datasets. This resulted in the DETOXIS dataset being unable to be trained, even with a batch size of 2. Even though a fixed token limit could have been manually set, it was discarded as it had a worse performance than BERT. All results are stored in the group “deep_learning” and the name "GPT2 base".

3.4.11 Atalaya

This subsection tries to recreate the model that the team of Atalya (Pérez & Luque, 2019) used in the 2019 SemEval competition. As mentioned in Section 3.1, to be able to run the code, you need to use the other environment (tf-gpu-1.15) because of incompatibilities with TensorFlow.

The module of pickle5 allows storing the predictions in the same format between environments. This is due to the modification of the pickle protocol between the two versions of Python. Also, some warnings for TensorFlow are removed, because this section is designed to run for only 1.X versions. Then the ELMO model (Peters et al., 2018) is downloaded from the internet or loaded from disk. The model is converted into a layer function that transforms the input text into an embedding.

Furthermore, there are definitions of some auxiliary functions that will be used for all datasets. The first one is a function that creates the model from zero, following the architecture described in the Atalaya paper. Also, the same optimizer parameters are used. Then there is a function for training the model. The training process uses early stopping with patience of 5. Once the model is trained, a prediction for all sub-datasets (train, development and test) is made. The last function creates a graph plotting both the loss and the accuracy of the model across the epochs.

The main process for each dataset consists of: first creating the model, then training and getting the predictions, next displaying the training history through a graph to check if everything is working properly. Concluding with storing the predictions on disk. The description model is the same for all datasets because there are no changes between models. The process is repeated for all datasets. All results are stored in the group “deep_learning” with the name of “Atalaya”.

3.4.12 SBERT

This section uses the SBERT (Reimers & Gurevych, 2019) model to generate embeddings for all the datasets. These embeddings can be used as inputs for the machine learning models. Similarly to Section 3.4.5, some base models are trained to ensure that everything is properly working.

[10] "datificate/gpt2-small-spanish · Hugging Face." https://huggingface.co/datificate/gpt2-small-spanish.

First of all, the dataset indexes needed to be reset, to avoid a problem with the sentence transformers package, where it gets confused if indexes do not describe the position on the dataframe. Furthermore, two different checkpoints for the encoder are used. One that is trained only for English and another that is multilingual, which includes Spanish. Once the models are loaded, all the datasets are encoded into a fixed size embedding.

Then the base models are loaded, which consist of Ridge Classifier, Random Forest classifier, Support Vector Classification, AdaBoost classifier and Multi-layer Perceptron classifier. Similarly to Section 3.4.5, the only parameters changed are the random state, to get consistent results and enabling early stopping on the Multi-layer Perceptron classifier to avoid overfitting. The missing models are due to the incompatibility with the data, e.g. the Bayes Classifiers can not work with negative numbers.

All models are trained and evaluated with training and development datasets, to test if there are any problems. Also allowing to test how much better the performance is compared to using Term Frequency-Inverse Document Frequency. All score information is printed during the scoring process. However, to facilitate the comparison, two tables were constructed. Then the process is repeated for each dataset.

3.4.13 Searching the best hyperparameters (SBERT)

This section tries to find the best parameters for the previously mentioned models through grid search cross-validation. This is practically the same as Section 3.4.6, except the data used is from the encoded data through SBERT. The other change is that some parameter grids have been extended or modified. Moreover, some models are missing due to incompatibilities of data. This section uses the same methods like the ones defined at the beginning of Section 3.4.6. For example, the models with more parameters are split into two cross-validations. Also as in all sections, the process is repeated for all datasets.

3.4.14 Best Models (SBERT)

This subsection creates the models with the best parameters found in the previous section. This section is also very similar to 3.4.7, where the only notable difference being the model parameters values, and that two models are missing. The models are stored as the new group called “sbert“ and “sbert_best” with the same naming strategy..

3.4.15 Oversampling

This section tries to use different Synthetic Minority Oversampling Technique (Chawla et al., 2002) over the SBERT embedding. The common objective is to generate new data to balance a feature. As observed in Section 3.2, both datasets are unbalanced and using SMOTE could result in an improvement.

The main method of this section trains a list of models with a list of oversampling techniques. Each oversampler, if possible, is created with a fixed random state to get consistent results across runs. Also, parallelism is enabled to fasten the oversampling process. Once the input data is expanded, the models are cloned (which allows them to maintain the parameters) and fitted with this oversampled data. Returning a dictionary of new fitted models, which are

combinations of the original list of models and the oversampled data of different smote techniques.

The new models are also a dictionary composed of pairs of model and name, can use previously implemented methods for scoring the predictions. Similarly, once the models have printed all the information related to scores, a more compact table is also printed. However, due to the high quantity of models (twenty-seven), not all scores are printed. However, the predictions can be stored properly on disk, allowing comparison on the other notebook ("Results"). The predictions are stored under the group of “sbert_oversampling”. Also, the name is composed of the model name and the oversampler name.

All models and oversamples that were used were using default parameters. Excluding the random state for consistent results and the number of jobs for faster processing. This could have been further explored through grid search cross-validation of both the model and the oversampler.

3.4.16 Results

This section describes the "Results" notebook, which allows the comparison of all the stored predictions. The notebook displays several options for comparing the different models through the use of tables. The metrics that will be used are F1 score, accuracy, precision and recall. The F1 score will change depending on the task, where it will be F1 macro for task 1 of SemEval 2019 and F1 binary for task 1 of DETOXIS. Further explained in Section 4.2.

The notebook starts by importing some files and loading the auxiliary functions explained in Section 3.4.4. There are two modifications to these methods. First, the addition of a method to load all metadata and results from the disk. Second, the dataset types enum has been modified, to allow comparison between elements. Also, has a new method to shorten the names for the table names.

There are also three more methods for scoring depending on the task. The main method, which scores the prediction using the previously mentioned metrics, only if a ground truth exists, otherwise returns null. The other two are auxiliary methods for each dataset task, both of them call the main scoring method.

The results are loaded from the disk and split between predictions and ground truth. Then all predictions are scored with their respective truth depending on the task, only if it exists. Once the prediction has been scored, the extra data from the metadata file is added. Also, the metadata is formatted to be more readable. Finally, once all predictions have been scored, they are grouped on a base table containing all the information. Finally, different formats for displaying the results are printed.

The first option for displaying data is one where all metadata is used as the index, while the scores are the columns. The data is sorted using the F1 score. However, this option is the one that is the least compact. Furthermore, with the huge quantity of tested models and datasets, it is very hard to compare results.

The second option aggregates the data, so the score and the dataset type are displayed as multiple levels on the columns, where the score is at the top and the dataset types (train, development and test) are below. This view allows for a more easy comparison between dataset types. Furthermore, the data is grouped by the task, language and group, which

allows for easy comparison between models of the same group. Also, the generated table is more compact. Moreover, the data in each group are sorted by the F1 test score or F1 development score, in case the first does not exist.

The third option is based on the previous one, but without grouping by the column of the group. Allowing to visualize the best models for the specific task and language. Especially useful for comparing results between all models.

The fourth option is the same as the second one but limiting the number of models to the top of each group. These options can facilitate the comparison process between groups, allowing to find which group is the best.

Similarly, the fifth option is the same as the third one but limiting the number of models to the top of each group. This table allows for easier comparison between the best models.

4. Evaluation

This section describes how the results of each model were obtained and how they were evaluated. In addition, it shows some of the model results of each dataset to allow for comparison. For the visualization of all the results, see the “Results” notebook.

Model/Technique Name	Implemented/Modified?	Reference
TF-IDF	Yes	Robertson, 2004
Dummy Classifier	No	
Multinomial Naive Bayes Classifier	No	Hand & Yu, 2001
Bernoulli Naive Bayes Classifier	No	Hand & Yu, 2001
Ridge Classifier	No	Singh et al., 2016
Random Forest Classifier	No	Breiman, 2001
Support Vector Classification	No	Hearst et al., 1998
AdaBoost classifier	No	Freund & Schapire, 1996
Multi-layer Perceptron classifier	No	Marius et al., 2009
BERT	No	Devlin et al., 2018
BERT Average	Yes	Li et al., 2020
BERT SMOTE	Yes	
GPT-2	No	Radford et al., 2019
Atalaya	Yes	Pérez & Luque, 2019
SBERT	No	Reimers & Gurevych, 2019
SMOTE	No	Chawla et al., 2002

Table 3: Models and Techniques used in the project.

The result analysis is one of the most important parts of the project problem. Consisting on finding which models are better suited for the detection of hate speech because it allows understanding how capable the models are, while also allowing a way to compare between them, especially through the test dataset, which should be data that the model has not seen. Providing a way to evaluate without bias a model. This is the same principle applied to the validation dataset. All results that will be shown have been trained using only the training sub-dataset.

Nearly 60 models were trained across the three different datasets (HatEval Spanish, HatEval English and DETOXIS) and have predicted all the sub-datasets (train, development and test). These models the majority were combinations between different techniques and models (as seen in Table 3). This extensive quantity of models will be evaluated and compared later on through the specified metrics.

4.1 Methodology

The methodology implemented consisted of using the different datasets (HatEval and DETOXIS) to allow all models to be fitted with the training data of a specific dataset. Then, the models were asked to make predictions for all the sub-datasets (train, development and test). All the predictions were then stored on a disk, along with the ground truth. The process was repeated for almost all models, where they were trained three times for each dataset (HatEval English, HatEval Spanish and DETOXIS). This process is further explained in Section 3.4.

The predictions then were compared with the ground truth, allowing us to score the predictions through the different metrics. Next all scores were combined in a table where they were grouped according to the task and language. Finally, the table was reordered and reorganized in different ways to facilitate different investigations. The most relevant (options 2 and 3) allowed for easy comparisons between models of the same group, while the other allowed for easy comparisons between models of all groups. The main difference was that the first grouped models with the same group name, while the other did not. However, due to the abundant number of models, more compact tables (as seen in Table 4, 5 and 7) were created (options 4 and 5).

4.2 Metrics

The first task of both competitions was a binary classification, reducing the scope of possible metrics. The principal metric, which both datasets used for ranking participants, was the F1 score. However, the specific formula changes between the two datasets. Whereas SemEval 2019 (Basile et al., 2019) uses F1 macro (average F0 score and F1 score), DETOXIS uses F1 binary (standard F1 score). Other metrics for extra information were used during the analysis and competition, such as accuracy, precision and recall.

All the metrics were calculated through the following formulas. “tp” stands for true positive, which means the number of outcomes where the model predicted correctly positive class. “fp” stands for true negative, which represents the number of times where the model predicted incorrectly the positive class. Oppositely, “tn” (true negative) and “fn” (false negative) are the same as true positives and false positives but instead targeting the negative class. All these values are computed through the comparison of the prediction and the truth.

$$F_1\ binary = \frac{tp}{tp + \frac{1}{2}(fp + fn)}$$

$$F_1\ macro = \frac{1}{2}\left(\frac{tp}{tp + \frac{1}{2}(fp + fn)} + \frac{tn}{tn + \frac{1}{2}(fp + fn)}\right)$$

$$Precision = \frac{tp}{tp+fp}$$

$$Recall = \frac{tp}{tp+fn}$$

$$Accuracy = \frac{tp+tn}{tp+fp+tn+fn}$$

4.3 Discussion

This section analyzes some of the results of the models using the explained methodology and metrics. While also explaining some observations of the results of the models in all the datasets. There are some general traits across the results of the three different datasets.

First, in all datasets, the models that use deep learning have higher performance, by a large margin, against almost all the models that used the Term Frequency-Inverse Document Frequency. This could be attributed to the fact that deep learning models can understand the complex relationships between words.

Second, generally, a version of the BERT model was always the best across the group of deep learning. Probably due to the fact that the GPT-2 model has not been designed for direct sequence classification. Whereas the Atalaya model, excluding ELMO, was too small for learning complex patterns between embeddings. Another possibility is due to the ELMO architecture, which does not use Transformers.

Third, the different implementations of the modification of the model BERT called BERT Average did not seem to bring any relevant increase or decrease to the original version. The model was based on the model called BERT last2avg (Li et al., 2020). Different ways to calculate the average were tested, which resulted in similar results. The tested average methods were: mean of the two first tokens (the position of the special token [CLS]) of the last two layers, the average of the last tokens (last word token and special token [SEP]) of the last layer and the average of the entire two last layers. This last one was the one that the BERT Average ended up representing because it seemed to be the most similar to the one described on the paper.

Fourthly, the F1 scores were very similar between BERT and SBERT across all datasets. Whereas BERT seemed to be the best model for the Spanish datasets (HatEval and DETOXIS), SBERT was the best model for the English dataset (HatEval). However, most SBERT models beat BERT on the English dataset with a larger margin than the Spanish counterpart. A possible explanation for this difference is that there is a simpler correlation between features of the embedding on the English model and hate speech. Another possible cause is that the SBERT works better if trained with a single language because a multilingual model has to be able to interpret texts of multiple languages with the same embedding size.

4.3.1 HatEval

The best models for the English test dataset were models that use SBERT embeddings, the random classifier and support vector classification with f1 scores of nearly 0.65 (as seen in Table 4). Furthermore, it seems that the oversampling did not end up increasing the performance enough, with only an f1 score of 0.63. Finally, the traditional models were the worst-performing, with the best model only scoring 0.51.

However, if the models are ranked by the development score instead of the test score. Instead, the best models are the different versions of the BERT average with scores of 0.75. Furthermore, the 4th best result in development is based on the Term Frequency-Inverse Document Frequency using the model of Random Forest Classifier, with a score of 0.75. Strangely this last model decreases its performance by a lot on the test sub-dataset. This probably caused by some big difference in the data between train and development in comparison to the test. A possibility that the distribution of data is different between datasets. For example, the first two dataset types have more data that targets immigration, while the test contains more data that targets misogyny.

The best models on the first task of the SemEval 2019 competition scored 0.65 for the English dataset. The best model used a Support Vector Classifier with sentence embeddings from Google's Universal Sentence Encoder, which was presented by the Fermi team (Indurthi et al., 2019). This strategy is very similar to the usage of SBERT on the project. Even both implementations got nearly the same f1 macro score.

The recreation of the model the team Atalaya (Pérez & Luque, 2019) created, which was composed of what they called LSTM-ELMo+BoW, had very similar results to the ones they reported. The F1 macro scores they obtained were 0.743 for validation and 0.461 for the test, whereas our implementation resulted in 0.740 for and 0.470, respectively. Similarly, the other metric they reported was accuracy. They reported 0.738 for validation and 0.502 for the test, whereas the recreation got 0.743 for and 0.508, respectively. The small differences can be attributed to the random sampling during training. Also, it could be attributed to the number of epochs and batch size, which they did not specify.

The Spanish dataset got the best results with BERT, specifically the base model, which ended up scoring a bit higher than the rest with f1 scores of 0.76 (as seen in Table 5). The second model was BERT oversampled which was unexpected, considering it was an invention. All three models are then followed by different versions of SBERT with the best one scoring 0.72. Similarly, the models based on the traditional approach are generally the ones that scored the worst. However, the difference is much lower than in the English dataset, with the best model scoring 0.7. A possible cause could be that the development sub-dataset was more similar to the test. This hypothesis would also explain why the results between development and test are more or less similar. Furthermore, the difference is constant across both sub-datasets, where the ranking has almost the same order.

Furthermore, the best models in the Spanish competition were a tie with an f1 score of 0.73. Our current best model, BERT, performs a bit better. Both teams used Support Vector Machines but focused on different traditional techniques. Indicating that deep learning models have more potential for hate speech detection.

		F1			Accuracy			Precision			Recall		
		Train	Dev	Test	Train	Dev	Test	Train	Dev	Test	Train	Dev	Test
Group	Name												
Bert Avarage	CLS layers	0.93160	0.75035	0.58977	0.93311	0.753	0.60267	0.91003	0.69149	0.51496	0.93312	0.76112	0.92857
	Entire layer	0.94064	0.75725	0.58663	0.94200	0.759	0.59933	0.92274	0.69057	0.51280	0.94079	0.78923	0.92222
	Tokens	0.93611	0.74822	0.58653	0.93744	0.750	0.59900	0.91092	0.68098	0.51261	0.94343	0.77986	0.91984
Deep Learning	Bert base	0.92999	0.74690	0.59415	0.93156	0.749	0.60533	0.90886	0.68257	0.51698	0.93048	0.77049	0.91825
	Bert Avarage	0.93938	0.75085	0.59399	0.94078	0.753	0.60600	0.92186	0.68750	0.51729	0.93867	0.77283	0.92619
	Bert SMOTE_TomekLinks	0.69742	0.65283	0.54307	0.70033	0.654	0.55467	0.62529	0.57864	0.48287	0.71636	0.69789	0.85000
Sbert	Random Forest classifier	0.99897	0.70037	0.64926	0.99900	0.715	0.64933	0.99974	0.70170	0.56125	0.99789	0.57845	0.75635
	Support Vector Classification	0.89996	0.73415	0.63010	0.90267	0.739	0.63267	0.88895	0.68950	0.53970	0.87814	0.70726	0.85238
	AdaBoost classifier	0.75236	0.67376	0.62681	0.76056	0.681	0.62733	0.72737	0.62736	0.53834	0.68834	0.62295	0.79127
Sbert Best	Random Forest classifier (best)	0.99897	0.70943	0.64897	0.99900	0.722	0.64900	1.00000	0.70411	0.55857	0.99762	0.60187	0.78333
	Support Vector Classification (best)	0.99897	0.57058	0.63429	0.99900	0.656	0.66533	1.00000	0.82677	0.64781	0.99762	0.24590	0.44524
	AdaBoost classifier (best)	0.80969	0.70296	0.63062	0.81622	0.710	0.63133	0.79994	0.66348	0.54113	0.75046	0.65105	0.80397
Sbert Oversampling	SVC polynom_fit_SMOTE	0.91803	0.73415	0.63163	0.92011	0.739	0.63433	0.90486	0.68950	0.54081	0.90510	0.70726	0.85714
	SVC CCR	0.89754	0.73241	0.63006	0.90022	0.737	0.63267	0.88354	0.68552	0.53966	0.87840	0.70960	0.85317
	SVC LVQ_SMOTE	0.90041	0.73321	0.62953	0.90300	0.738	0.63233	0.88635	0.68793	0.53923	0.88237	0.70726	0.85635
Traditional	Multi-layer Perceptron classifier	0.78679	0.71006	0.51049	0.79989	0.725	0.52600	0.83188	0.71965	0.46438	0.65662	0.58314	0.83810
	Ridge Classifier	0.87636	0.71097	0.48698	0.88067	0.718	0.51333	0.88295	0.67386	0.45868	0.82554	0.65808	0.88095
	Multinomial Naive Bayes classifier	0.81284	0.73487	0.47697	0.82067	0.742	0.50400	0.82114	0.70660	0.45293	0.73302	0.67681	0.87063
Traditional Best	Multi-layer Perceptron classifier (best)	0.81200	0.72553	0.50020	0.82011	0.733	0.52267	0.82318	0.69608	0.46380	0.72852	0.66511	0.87460
	Multinomial Naive Bayes classifier (best)	0.81622	0.73487	0.47901	0.82344	0.742	0.50633	0.81964	0.70660	0.45447	0.74359	0.67681	0.87540
	Ridge Classifier (best)	0.83368	0.73129	0.47851	0.84156	0.740	0.50633	0.86161	0.71247	0.45458	0.74227	0.65574	0.87778

Table 4: Evaluation of the 3 best models of each group on the English HatEval Task 1. F1 in the table stands for the F1 macro score.

The worst-performing deep learning model on the Spanish dataset was GPT-2, which had a similar difference as the English version. However, differently from the English checkpoint, which was an official, the Spanish was made by a user[11]. The results show similar performance, pointing that the pre-training was of the same quality as the original.

Differently from the English dataset, the results the team Atalaya reported are not the same as the results of our recreation. However, this only happens on the development sub-dataset, where they got an f1 macro score of 0.821 and an accuracy score of 0.824, while our model only got 0.775 and 0.778, respectively. Unexpectedly, on the testing data, the scores were very similar. While they got 0.712 for f1 and 0.719 for accuracy, our model got 0.713 for f1 and 0.715 for accuracy. This situation is strange, the only reason for this discrepancy is that the development sub-dataset they used is different from ours. Otherwise, the testing results should not be so similar on both metrics. For example, the entire dataset was mixed.

4.3.2 DETOXIS

The best models for the DETOXIS end up being the different versions of BERT, with the best model scoring 0.56 on the F1 score (as seen in Table 7). These models are closely followed by different models that used SBERT and oversampling, with the best one scoring 0.53. Differently from the other datasets, several parameter configurations were stored to allow comparison between them. Furthermore, similar to the rest of the datasets, the models based on Term Frequency-Inverse Document Frequency perform generally much worse than the rest, with the best one only scoring 0.42.

Also, similarly to the HatEval Spanish dataset, the ranking based on the f1 score is more or less the same between the development and test dataset. Indicating that the decision for splitting the training data was useful for the decision making process during the competition. Furthermore, the oversampling on the models that SBERT used, brought a general performance increase across all of them, with at least a boost of 0.05 for the best model.

All models that used Term Frequency-Inverse Document Frequency benefited from the parameter optimization done through grid search cross-validation. Increasing the performances of some models up to 0.3 in f1 scores, such as the AdaBoost classifier model.

The competition allowed the submission of 5 models for each subtask. The models were chosen by their performance on the development set and also that they were different architectures. Increasing the chance that one of those models performed much better than the rest. Furthermore, the models were trained with the combined dataset of train and development to further increase the performance with more data. Also, the chosen SBERT models had the optimized hyperparameters found through grid search cross-validation. The chosen models were:

1. Bert base (2 epochs)
2. Bert average
3. Sbert Ridge SMOBD
4. Sbert MLP SMOTE_TomekLinks
5. Ensemble (bagging) of all previous methods

[11] "datificate/gpt2-small-spanish · Hugging Face." https://huggingface.co/datificate/gpt2-small-spanish.

		F1			Accuracy			Precision			Recall		
		Train	Dev	Test	Train	Dev	Test	Train	Dev	Test	Train	Dev	Test
Group	Name												
Deep Learning	Bert base	0.95680	0.85247	0.76070	0.95800	0.854	0.76313	0.94034	0.82819	0.68036	0.95907	0.84685	0.80303
	Bert SMOTE_TomekLinks	0.95645	0.84942	0.75699	0.95756	0.850	0.75875	0.93250	0.79757	0.67040	0.96715	0.88739	0.81667
	Bert Avarage	0.95685	0.82873	0.74829	0.95800	0.830	0.74938	0.93711	0.79149	0.65509	0.96284	0.83784	0.82879
Sbert	Support Vector Classification	0.91727	0.77670	0.72430	0.92044	0.782	0.73250	0.92561	0.78109	0.67470	0.87776	0.70721	0.67879
	Ridge Classifier	0.78926	0.73404	0.70186	0.79822	0.738	0.71125	0.77668	0.70968	0.65091	0.71729	0.69369	0.64697
	Multi-layer Perceptron classifier	0.81160	0.75904	0.70135	0.82178	0.766	0.71375	0.83031	0.77202	0.66450	0.71405	0.67117	0.61818
Sbert Best	Support Vector Classification (best)	0.94555	0.76527	0.72469	0.94733	0.770	0.73062	0.94311	0.75845	0.66241	0.92838	0.70721	0.70758
	Ridge Classifier (best)	0.79798	0.73693	0.70433	0.80578	0.740	0.71250	0.77942	0.70536	0.64837	0.73829	0.71171	0.66212
	AdaBoost classifier (best)	0.90909	0.72099	0.69009	0.91244	0.726	0.70063	0.91119	0.70142	0.64031	0.87291	0.66667	0.62576
Sbert Oversampling	SVC SMOTE_TomekLinks	0.93050	0.76414	0.73667	0.93244	0.768	0.74250	0.91063	0.74766	0.67664	0.92730	0.72072	0.71970
	SVC SMOTE_IPF	0.93141	0.76867	0.73559	0.93333	0.772	0.74125	0.91212	0.74771	0.67422	0.92784	0.73423	0.72121
	SVC Assembled_SMOTE	0.93205	0.75816	0.73343	0.93400	0.762	0.73875	0.91489	0.73953	0.66947	0.92623	0.71622	0.72424
Traditional	Ridge Classifier	0.94179	0.74015	0.69866	0.94378	0.746	0.70375	0.94260	0.73399	0.62705	0.91976	0.67117	0.69545
	Support Vector Classification	0.97568	0.75159	0.69784	0.97644	0.760	0.70875	0.97401	0.77419	0.65252	0.96877	0.64865	0.62879
	Multi-layer Perceptron classifier	0.90000	0.74864	0.69751	0.90400	0.754	0.70625	0.90878	0.74146	0.64222	0.85299	0.68468	0.65000
Traditional Best	Multi-layer Perceptron classifier (best)	0.92631	0.73197	0.70015	0.92867	0.738	0.70813	0.91785	0.72414	0.64212	0.90845	0.66216	0.66061
	Ridge Classifier (best)	0.90766	0.74703	0.69866	0.91156	0.754	0.70625	0.92437	0.75385	0.63848	0.85568	0.66216	0.66364
	Support Vector Classification (best)	0.91948	0.74733	0.69406	0.92222	0.752	0.69875	0.91561	0.73333	0.61995	0.89391	0.69369	0.69697

Table 5: Evaluation of the three best models of each group on the Spanish HatEval Task 1. F1 in the table stands for the F1 macro score.

RESULTS FOR SUBTASK 1:

team_name	run_number	f1_toxic
Gold_Standard		1.000000
maia	5	0.556962
maia	1	0.543967
maia	2	0.522936
maia	3	0.503623
maia	4	0.470588
RandomClassifier		0.376087
ChainBOW		0.374670
BOWClassifier		0.183746

Table 6: Results of submitted models for DETOXIS Task 1

The best performing model was the one that used bagging, where the prediction was made using the other four models. The process consisted of finding which label had more votes, and in case of a tie, the positive would win. The tie was solved using positive in consideration of the formula used for ranking the competition, where predicting negative does not change the score (if right). Whereas in the case of predicting the positive the score will increase (if guessed correctly). Also, the penalty for guessing wrong is the same for both cases.

This combination of models with a score of 0.56 ended up 8th on the ranking of 31 participants of the DETOXIS competition[12] (as seen in Table 6). The score obtained could be considered to be the bottom of the top-ranking teams. This is because there is less distance between the 2nd team (0.60) and our model than our model and the 9th team (0.49). Furthermore, the 6th and 7th teams (0.56 and 0.57) only have a difference of less than 0.1. Indicating that a better exploration of parameters could result in a better ranking position.

[12] "evaluation measures - DETOXIS-IberLEF 2021 - Wix.com." https://detoxisiberlef.wixsite.com/website/evaluation-results.

		F1			Accuracy			Precision			Recall		
		Train	Dev	Test	Train	Dev	Test	Train	Dev	Test	Train	Dev	Test
Group	Name												
Deep Learning	Bert base (2 epochs)	0.83636	0.66368	0.55882	0.89928	0.78355	0.76431	0.88682	0.74000	0.56118	0.79134	0.60163	0.55649
	Bert base (3 epochs)	0.85899	0.65471	0.54286	0.91336	0.77778	0.74860	0.91261	0.73000	0.52988	0.81132	0.59350	0.55649
	Bert Avarage	0.98041	0.64819	0.53975	0.98736	0.76190	0.75309	0.98871	0.68161	0.53975	0.97225	0.61789	0.53975
Sbert	Multi-layer Perceptron classifier	0.65392	0.61072	0.46860	0.79747	0.75902	0.75309	0.73611	0.71585	0.55429	0.58824	0.53252	0.40586
	Ridge Classifier	0.62939	0.57985	0.44776	0.79422	0.75325	0.75084	0.75981	0.73292	0.55215	0.53718	0.47967	0.37657
	Support Vector Classification	0.82315	0.56000	0.40563	0.90072	0.76190	0.76319	0.97859	0.81395	0.62069	0.71032	0.42683	0.30126
Sbert Best	Ridge Classifier (best)	0.65409	0.57957	0.44917	0.80144	0.74459	0.73850	0.75472	0.69714	0.51630	0.57714	0.49593	0.39749
	Support Vector Classification (best)	0.99778	0.59722	0.44545	0.99856	0.74892	0.72615	1.00000	0.69355	0.48756	0.99556	0.52439	0.41004
	Multi-layer Perceptron classifier (best)	0.54848	0.55208	0.42857	0.76462	0.75180	0.76655	0.72928	0.76812	0.62400	0.43951	0.43089	0.32636
Sbert Oversampling	MLPClassifier Assembled_SMOTE	0.72934	0.63315	0.53333	0.80253	0.71573	0.69360	0.65804	0.58419	0.45087	0.81798	0.69106	0.65272
	RidgeClassifier G_SMOTE	0.69433	0.65672	0.53039	0.78195	0.73449	0.71380	0.63814	0.60690	0.47368	0.76138	0.71545	0.60251
	RidgeClassifier SMOTE_TomekLinks	0.68725	0.65291	0.52817	0.77690	0.73304	0.69921	0.63163	0.60627	0.45593	0.75361	0.70732	0.62762
Traditional	Ridge Classifier	0.82495	0.26039	0.19385	0.90072	0.61472	0.61728	0.96716	0.40870	0.22283	0.71920	0.19106	0.17155
	Bernoulli Naive Bayes classifier	0.66345	0.21605	0.18889	0.82347	0.63348	0.67228	0.87319	0.44872	0.28099	0.53496	0.14228	0.14226
	Random Forest classifier	0.97798	0.15385	0.12389	0.98592	0.61905	0.66667	0.99540	0.36364	0.21000	0.96115	0.09756	0.08787
Traditional Best	AdaBoost classifier (best)	0.49262	0.52081	0.42413	0.32996	0.35209	0.27160	0.32680	0.35311	0.26914	1.00000	0.99187	1.00000
	Dummy Classifier (best)	0.49087	0.52396	0.42301	0.32527	0.35498	0.26824	0.32527	0.35498	0.26824	1.00000	1.00000	1.00000
	Random Forest classifier (best)	0.72641	0.37168	0.29328	0.83357	0.59019	0.61055	0.78061	0.40777	0.28571	0.67925	0.34146	0.30126

Table 7: Evaluation of the three best models of each group on the Detoxis Task 1. F1 in the table stands for the F1 binary score.

5. Economic analysis

This chapter describes how the time was spent for the realization of the project and an estimation of the total economic cost of the entire project. Also, explains a breakdown of the individual task for accomplishing the current project.

5.1 Time

An estimated 450 hours (700 hours counting training) were spent to accomplish this project, between the 1st of February of 2021 to the 20th of June of 2021. The project was divided into different primary tasks:

1. *Research*: Study of the different state-of-the-art models in Natural Language Processing and other alternatives.
2. *Analysis*: The process of extracting meaningful information from the data of the project.
3. *Implementation*: Creation of a framework to allow to run different types of models and also be able to compare them.
4. *Documentation*: Writing of the actual report for the bachelor's thesis.
5. *Training*: Time spent training the different models or running other code. The Gantt diagram implementation tasks also count this task.

First of all, the documentation was mostly written during the interval outside of the implementation, with some exceptions. The task that most time was inverted was the implementation. A big cause was the time it took to run some code. This resulted in the implementation taking up to 75% more. Especially in the case where bugs were found, forcing the need to rerun the affected section. Furthermore, the implementation of most models brought the complications of learning new technologies from scratch, which is time-consuming. Also, the analysis was made across long periods with lower work density, to allow to come up with new ideas.

As seen in Table 8 and Figure 22, the project was mainly centred around the implementation, which was the task that took the most time (250h). Followed by the training (150h) as a consequence of the long time it took to run some sections, especially the cross-validation ones. The next one was the documentation (80h) of the current paper. The last two had the same amount of time (60h), which were the analysis and research tasks.

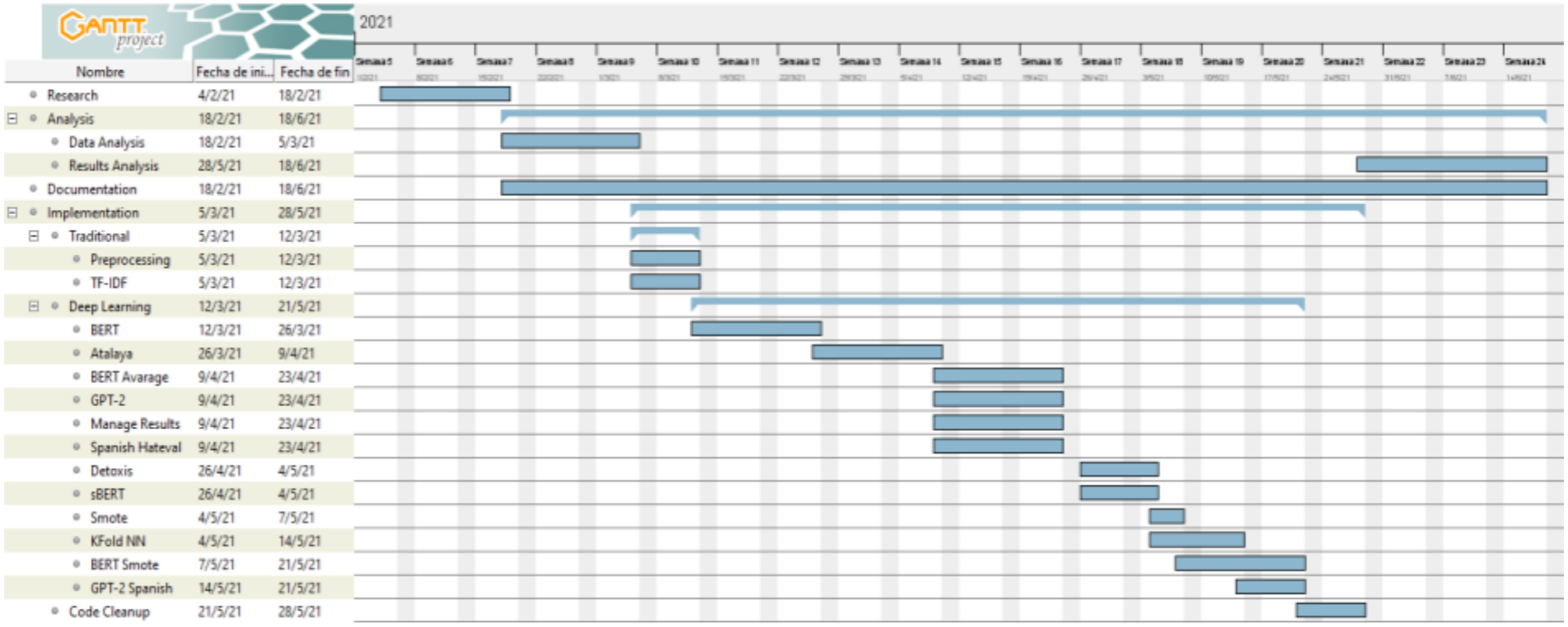

Figure 22: Gantt Diagram of work interval of the different tasks and subtasks of the project.

5.2 Cost

The project did not need to buy any product, material or licenses because all applications and modules used were open-sourced. Almost all costs were related to manual labour, excluding the cost of running the code (training task), which were electrical costs. All values used are approximations of the estimated prices for the services.

Task	Time	Price	Cost
Research	60 h	10 €/h	**600 €**
Analysis	60 h	15 €/h	**900 €**
Implementation	250 h	20 €/h	**5.000 €**
Documentation	80 h	15 €/h	**1.200 €**
Training	150 h	0,25 €/h	**38 €**
		Total Cost	**7.738 €**

Table 8: Approximated costs of each task for the realization of the project.

The total cost of the project is 7.738€. As expected, the main cost was the implementation which correlates with being the most important part of the project. It ends up costing two-thirds of the total costs. Almost the rest is from the other services, excluding the implementation. The physical costs are nearly negligible because of how small is the setup used for executing the code. However, this cost could end up being higher, if instead of using the current one, the hardware was rented.

6. Conclusions

The project studied different Natural Language Processing models and techniques for hate speech detection on messages, which can be divided into two categories: the traditional approach and the deep learning approach. The former uses algorithms to extract static information from the texts. While the latter group are models that have been trained with huge amounts of data. These models can extract more complex information through the context for better predictions. Furthermore, all the implemented models have been tested on three different datasets with two distinct languages.

All objectives of the project were accomplished, starting first through research on the current state of the natural language processing discipline. Proceeding with the implementation of a framework that would allow running all types of models, while also having a more compacted representation of all the results. Finishing with analysis and documentation of the current report.

The main focus of the project was the implementation of different models. Which allowed learning new technologies from scratch. However, the toughest part was the research of the current state-of-art for Natural Language Processing (NLP). This was caused by how difficult it was to understand the new concepts and lexicon. Also by how highly interconnected the different research papers were.

Some deep learning model versions could not be tested due to hardware limitations. Generally, with the same architectures but with more layers, the performance should have increased in downstream tasks. An example of this could be the version BERT large or RoBERTa large, which occupied more than the 8 GB of RAM available on the GPU. Furthermore, it could have been interesting to see the results that GPT-3 could obtain, the most recent state-of-the-art model. Additionally, the framework could have been expanded to also test the different models for the second task of all datasets, which instead of being binary classification is multiclass. Moreover, a combination of the Spanish datasets for training could have resulted in performance increases on all tasks. Furthermore, the best parameters for both the models and the oversamplers that used SBERT embeddings could have been explored, which should have resulted in a further performance increase. Also, the oversampling could have been tested on the Term Frequency-Inverse Document Frequency matrix, which could have increased the perplexity on unbalanced datasets. Moreover, a deep learning model pre-trained purely on the language, could have better performance than a multilingual version (i.e. BETO). Additionally, an investigation could be conducted to analyze the performance between the different multilingual checkpoints. As future work, we plan to analyze all the above-mentioned issues that remained open in the current work.

In conclusion, the current state of the different tested machine learning models has immense potential for hate speech detection, especially for deep learning models. Furthermore, they can process the large amounts of data interchanged in social media. Although the best models did not have a very high perplexity, a more reliable model could be trained with more training data or new architectures. Even at the current state, the models could be used as an external font for helping humans in the decision-making process. Moreover, these models could filter the most confident predictions, while leaving the rest for the reviewer team.

References

Basile, V., Bosco, C., Fersini, E., Nozza, D., Patti, V., Rangel Pardo, F. M., Rosso, P., & Sanguinetti, M. (2019). SemEval-2019 Task 5: Multilingual Detection of Hate Speech Against Immigrants and Women in Twitter. In *Proceedings of the 13th International Workshop on Semantic Evaluation* (pp. 54-63). Association for Computational Linguistics. 10.18653/v1/S19-2007

B. Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., M. Ziegler, D., Wu, J., Winter, C., … Amodei, D. (2020). Language Models are Few-Shot Learners. arXiv preprint arXiv:2005.14165

Breiman, L. (2001). Random Forests. *Machine Learning - ML*, *45*, 5-32. 10.1007/978-1-4419-9326-7_5

Cer, D., Yang, Y., Kong, S., Hua, N., Limtiaco, N., John, R. S., Constant, N., Guajardo-Cespedes, M., Yuan, S., Tar, C., Sung, Y.-H., Strope, B., & Kurzweil, R. (2018). Universal sentence encoder. arXiv preprint arXiv:1803.11175

Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: Synthetic Minority Over-sampling Technique. *Journal of artificial intelligence research*, *16*, 321-357.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv preprint arXiv:1810.04805

Freund, Y., & Schapire, R. E. (1996). Experiments with a new boosting algorithm. In *icml* (Vol. 96, pp. 148-156).

Hand, D. J., & Yu, K. (2001). Idiot's Bayes: Not So Stupid after All? *International Statistical Review*, *69*(3), 385-398. 10.1111/j.1751-5823.2001.tb00465.x

Hearst, M. A., Dumais, S. T., Osuna, E., Platt, J., & Scholkopf, B. (1998). Support vector machines. *IEEE Intelligent Systems and their Applications*, *13*(4), 18-28. 10.1109/5254.708428

Indurthi, V., Syed, B., Shrivastava, M., Chakravartula, N., Gupta, M., & Varma, V. (2019). FERMI at SemEval-2019 Task 5: Using Sentence embeddings to Identify Hate Speech Against Immigrants and Women in Twitter. In *Proceedings of the 13th International Workshop on Semantic Evaluation* (pp. 70-74). Association for Computational Linguistics. 10.18653/v1/S19-2009

Lenhart, A., Ybarra, M., Zickuhr, K., & Price-Feeney, M. (2016). *Online Harassment, Digital Abuse, and Cyberstalking in America*.

Li, B., Zhou, H., He, J., Wang, M., Yang, Y., & Li, L. (2020). On the sentence embeddings from pre-trained language models. arXiv preprint arXiv:2011.05864

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., & Stoyanov, V. (2019). RoBERTa: A Robustly Optimized BERT Pretraining Approach. arXiv preprint arXiv:1907.11692

Marius, P., Balas, V. E., Liliana, P.-P., & Mastorakis, N. (2009). Multilayer perceptron and neural networks. *WSEAS Transactions on Circuits and Systems*, *8*.

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781

Pérez, J. M., & Luque, F. M. (2019). Atalaya at SemEval 2019 Task 5: Robust Embeddings for Tweet Classification. In *Proceedings of the 13th International Workshop on Semantic Evaluation* (pp. 64-69). Association for Computational Linguistics. 10.18653/v1/S19-2008

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., & Zettlemoyer, L. (2018). Deep contextualized word representations. arXiv preprint arXiv:1802.05365

Qiao, Y., Xiong, C., Liu, Z., & Liu, Z. (2019). Understanding the Behaviors of BERT in Ranking. arXiv preprint arXiv:1904.07531

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). *Language Models are Unsupervised Multitask Learners*. https://openai.com/blog/better-language-models/

Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. arXiv preprint arXiv:1908.10084

Robertson, S. E. (2004, 10). Understanding Inverse Document Frequency: On Theoretical Arguments for IDF. *Journal of Documentation - J DOC*, *60*, 503-520. 10.1108/00220410410560582

Singh, A., Prakash, B. S., & Chandrasekaran, K. (2016). A comparison of linear discriminant analysis and ridge classifier on Twitter data. In *2016 International Conference on Computing, Communication and Automation (ICCCA)* (pp. 133-138). IEEE.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention is all you need. arXiv preprint arXiv:1706.03762

Zhang, Y., Jin, R., & hi-Hua, Z. (2010, 12). Understanding bag-of-words model: a statistical framework. *International Journal of Machine Learning and Cybernetics*, *1*, 43-52. 10.1007/s13042-010-0001-0

Appendix

A Abbreviations Dictionary

The abbreviations dictionary was constructed from the most common abbreviations on Twitter. The abbreviations that could have several meanings were generally not included, excluding cases where there was a very clear winner. Also, most abbreviations were not translated directly to Spanish because of the difference in languages (i.e. ayfkmwts). The dictionary is a JSON with two levels to get the meaning: the first is the abbreviation and the second is the language. The dictionary created is the following:

{"cc": {"en": "carbon-copy", "es": "con copia"},
"cx": {"en": "correction", "ex": "correcion"},
"ct": {"en": "cuttweet"},
"dm": {"en": "direct message", "es": "mensaje directo"},
"mt": {"en": "partial retweet"},
"prt": {"en": "please retweet"},
"rt": {"en": "retweet", "es": "retweet"},
"em": {"en": "email marketing"},
"fb": {"en": "facebook", "es": "facebook"},
"li": {"en": "linkedin"},
"seo": {"en": "search engine optimization"},
"sm": {"en": "social media"},
"smm": {"en": "social media marketing"},
"smo": {"en": "social media optimization"},
"sn": {"en": "social network"},
"ugc": {"en": "user generated content"},
"yt": {"en": "youtube", "es": "youtube"},
"ab": {"en": "about"},
"abt": {"en": "about"},
"afk": {"es": "ausente"},
"afaik": {"en": "as far as i know"},
"ayfkmwts": {"en": "are you fucking kidding me with this shit?"},
"b4": {"en": "before"},
"bfn": {"en": "bye for now"},
"bgd": {"en": "background"},
"bh": {"en": "blockhead"},
"br": {"en": "best regards"},
"brb": {"es": "ahora vuelvo"},
"btw": {"en": "by the way"},
"cd9": {"en": "code 9, parents are around"},
"chk": {"en": "check"},
"cul8r": {"en": "see you later"},
"dam": {"en": "don't annoy me"},
"dd": {"en": "dear daughter"},
"df": {"en": "dear fiancé"},
"dp": {"en": "used to mean 'profile pic'"},
"ds": {"en": "dear son"},

"dyk": {"en": "did you know, do you know"},
"em": {"en": "email"},
"eml": {"en": "email"},
"ema": {"en": "email address"},
"f2f": {"en": "face to face"},
"ftf": {"en": "face to face"},
"ff": {"en": "follow friday"},
"ffs": {"en": "for fuck's sake"},
"fml": {"en": "fuck my life"},
"fotd": {"en": "find of the day"},
"ftw": {"en": "for the win"},
"fubar": {"en": "fucked up beyond all repair"},
"fwiw": {"en": "for what it's worth"},
"gmafb": {"en": "give me a fucking break"},
"gtfooh": {"en": "get the fuck out of here"},
"gts": {"en": "guess the song"},
"hagn": {"en": "have a good night"},
"hand": {"en": "have a nice day"},
"hotd": {"en": "headline of the day"},
"ht": {"en": "heard through"},
"hth": {"en": "hope that helps", "es": "espero que sirva de ayuda"},
"ic": {"en": "i see"},
"icymi": {"en": "in case you missed it"},
"idk": {"en": "i don't know"},
"iirc": {"en": "if i remember correctly"},
"imho": {"en": "in my humble opinion"},
"irl": {"en": "in real life", "es": "en la vida real"},
"iwsn": {"en": "i want sex now"},
"jk": {"en": "just kidding", "es": "es broma"},
"jsyk": {"en": "just so you know"},
"jv": {"en": "joint venture"},
"kk": {"en": "okey"},
"kyso": {"en": "knock your socks off"},
"lhh": {"en": "laugh hella hard"},
"lmao": {"en": "laughing my ass off", "es": "me parto el culo de risa"},
"lmk": {"en": "let me know"},
"lo": {"en": "little one"},
"lol": {"en": "laugh out loud", "es": "reirse a carcajadas"},
"mm": {"en": "music monday"},
"mirl": {"en": "meet in real life"},
"mrjn": {"en": "marijuana"},
"nbd": {"en": "no big deal"},
"nct": {"en": "nobody cares, though"},
"nfw": {"en": "no fucking way"},
"njoy": {"en": "enjoy"},
"nsfw": {"en": "not safe for work", "es": "no es seguro para el trabajo"},
"nts": {"en": "note to self"},
"oh": {"en": "overheard"},
"omg": {"en": "oh my god", "es": "oh dios mio"},
"omfg": {"en": "oh my fucking god"},
"oomf": {"en": "one of my friends"},
"orly": {"en": "oh, really?"},

```
"plmk": {"en": "please let me know"},
"pnp": {"en": "party and play"},
"qotd": {"en": "quote of the day"},
"re": {"en": "in reply to"},
"rlrt": {"en": "real-life re-tweet"},
"rtfm": {"en": "read the fucking manual"},
"rtq": {"en": "read the question"},
"sfw": {"en": "safe for work"},
"smdh": {"en": "shaking my damn head"},
"smh": {"en": "shaking my head"},
"snafu": {"en": "situation normal, all fucked up"},
"so": {"en": "significant other"},
"sob": {"en": "son of a bitch"},
"srs": {"en": "serious"},
"stfu": {"en": "shut the fuck up"},
"stfw": {"en": "search the fucking web"},
"tftf": {"en": "thanks for the follow"},
"tftt": {"en": "thanks for this tweet"},
"tj": {"en": "tweetjack"},
"tl": {"en": "timeline", "es": "linea temporal"},
"tldr": {"en": "too long, didn't read"},
"tl;dr": {"en": "too long, didn't read"},
"tmb": {"en": "tweet me back", "es": "contestame"},
"tt": {"en": "trending topic", "es": "marcan tendencia"},
"ty": {"en": "thank you"},
"tyia": {"en": "thank you in advance"},
"tyt": {"en": "take your time"},
"tyvw": {"en": "thank you very much"},
"w": {"en": "with"},
"w/": {"en": "with"},
"w/e": {"en": "whatever"},
"wtv": {"en": "whatever"},
"ygtr": {"en": "you got that right"},
"ykwim": {"en": "you know what i mean"},
"ykyat": {"en": "you know you're addicted to"},
"ymmv": {"en": "your mileage may vary"},
"yolo": {"en": "you only live once"},
"yoyo": {"en": "you're on your own"},
"yw": {"en": "you're welcome"},
"zomg": {"en": "oh my god to the max"},
"gtg": {"en": "got to go", "es": "me tengo que ir"},
"imo": {"en": "in my opinion", "es": "en mi opinion"},
"omw": {"en": "on my way", "es": "estoy de camino"},
"pls": {"en": "please", "es": "por favor"},
"plz": {"en": "please", "es": "por favor"},
"rofl": {"en": "rolling on floor laughing", "es": "revolcarse por el suelo de risa"},
"rpg": {"en": "role playing game", "es": "juego de rol"},
"thx": {"en": "thanks", "es": "gracias"},
"wtf": {"en": "what the fuck", "es": "¿que demonios es esto?"},
"xd": {"en": "happy face", "es": "cara sonriente"}
}
```