Text Analytics Project

Hate Speech Tweet Detection

Francesco Daquino, 646706
Francesco Maggio, 605001
Ondrej Krasnansky, 665131
Federica Papa, 597084
Sandro Cantasano Martino, 660874

Contents

1 Introduction

The detection of hate speech has become crucial to promote an healthy communication in online discussion or to protect minority voices[1]. For this reason, considering that hate detection can be used to fight the proliferation of toxic language spreading on social media platforms, in this work we seek to expand the research in this field focusing on a dataset containing some religious and political tweets obtained from the the platform *X*, formerly known as *Twitter*.

In particular, we based our work on the *HaSpeeDe3*[1] shared task organized within Evalita 2023 with the same idea of putting importance to the identification of online hateful content in many fields. Previous editions of the task (*HaSpeeDe*[2] and *HaSpeeDe2*[3]) focused on hate speech against immigrants, Muslims and Roms, while this edition, as anticipated, investigates hate speech in political and religious topics.

It's important to note that we didn't follow the rules and objectives of the task, as we just used its datasets and its original motivation as a starting input, in order to then work for ourselves.

In this paper, we will document our work, starting from the background and motivation, explaining the methods that we used to reach our goals and discussing results and outcomes. In particular, in line with the subdivision of the sections of the report, it is possible to summarize the main tasks of the project in:

- exploratory analysis of tweets and user description using Topic Modeling technique;
- hate speech detection via binary classification;
- attempts to improve binary classification performance through target variable balancing and use of metadata.

2 Literature Review/Background

Hate speech is about the disparagement of a person or a group on the basis of some characteristics (e.g. race, colour, ethnicity...)[4] using discriminatory or pejorative language in order to attack the person or the group as a result of who they are, but it can be manifested in many forms; the most common one is *cyber-aggression*, a dangerously growing up phenomenon, especially in young people[5]. In particular, the term *cyber-aggression* includes both cyberbulling and cyberhate, two commonly used expressions to indicate denigratory behavior towards other people.

Given the progressive trend, the contributions on the matter of hate speech recognition increased in the last few years (some examples [6][7]), and Evalita followed the pattern with its own *HaSpeeDe* project, currently at its third edition (*HaSpeeDe3*).

In light of this interest, HaSpeeDe isn't the only project crafted around Hate Speech Detection: an example is the international evaluation campaign SemEval-2019 Task 5[8] from SemEval, which consists of a series of NLP seminars focused on hate speech against immigrants and is still based on a dataset collected from *X*.

Being such an hot topic, and due to the multiplication of social media platform in which hate speech is able to expand freely and promptly, Hate Speech Detection is also being discussed on journals and books, like Neurocomputing[9]. This high interest on the topic is a clear indicator of its importance and we carried out our work while mindful of this.

3 Methods

3.1 Data Understanding and Preprocessing

In this section, we outline the critical phases of data understanding and preprocessing, essential for providing clarity to the operational context and establishing the groundwork for the detailed analyses in the subsequent

[1] HaSpeeDe3, http://www.di.unito.it/ tutreeb/haspeede-evalita23/index.html

sections. The analysis is structured into four main sets of operations, the outcomes of which were elaborated on in section *4.1*.

The first phase involves the initial dataset description and preliminary data handling: we proceeded with merging textual and contextual data, handling duplicates, missing values, and useless features. Then, we performed the removal of unique values, the management of categorical values and the column renaming in order to facilitate data comprehension. The final result led to having and working with a single large dataset, which combines the two main political and religious domains and the two types of textual and contextual data, which we will refer to as ”metadata” in each subsequent section.

The second one is dedicated to Feature Engineering, which includes transformations and the creation of new features. These operations aim to enhance data representation and enrich the set of metadata used in the classification phase, as previously described.

The next step is about data exploration and visualizations, aimed at identifying relationships between variables and significant patterns. During this, we conducted an analysis of the distribution of the target variable and evaluated the temporal period of tweet creation. We then performed the visualization of tweets and hashtags extracted from the text using *Wordclouds* plots.

The final phase is dedicated to data preprocessing, where operations such as tweet cleaning, text tokenization, lemmatization, and stemming are performed. These activities prepare the data for subsequent project phases, ensuring optimal representation for further analyses.

3.2 Topic Modeling

In order to get some meaningful information and automatically extract hidden themes from documents without the need for predefined labels, we decided to perform some experiments using the **Topic Modelling** text analysis technique. As previously mentioned, since the domain of our dataset was already designed between politic and religious, we initially analysed the presence of possible interesting topics in the ”anonymized_description” feature, which can be seen as a ”status” on the main page of the user. Before this investigation, a small preprocessing of the text was performed, removing the ’Unknown’ token highly present in the tokenized text, so as to have a better rendering, the Italian and English stopwords and also some special words added manually.

The vectorized text (obtained using **tf_vectorizer**) was passed through **LatentDirichletAllocation** (LDA), a generative probabilistic model used in NLP setting seven components to be identified. With this initialized parameter, we set the first analysis on descriptions creating two different topic modelling tasks: the first one concerns only the division between hate and non-hate (using the target variable **is_hate_speech**), while the second considers the division between the domain **politic** or **religious**. We then decided to find the optimal number of topics that would maximise the coherence score[2] of the model training LDA with different numbers of topics and measuring the coherence of the generated models.

Lastly, we made the second analysis considering the text of the tweets instead of desciptions but evaluating the entire dataset without any division between domains.

3.3 Classification

3.3.1 Base Classifiers

In this section, we have tried to implement various classification algorithms such **SVM**, **Logistic Regression**, **Random Forest**, **XGBoost** and **Multinomial NB** and to optimize them to do the best prediction possible. In previous sections we have explained the preprocessing of the dataset, which was the crucial

[2] The coherence tries to understand the semantical correlation between topics, it is developed in a classical elbow-method fashion.

part for the classification part of the project. The main approach was to firstly test every algorithm on the column *clean_tweet*, which consisted of the clean text without punctation and noisy elements like hashtags or emojis. After the first try, the results of which were the worst, we tried *tokenized_text* and *lemmatized_text* as target variables in our pipeline. To improve our models, we implemented the pipeline for every algorithm in which we were tuning the hyperparameters using **RandomizedSearchCV** with 75 iterations in order to save time and not doing every possible combination as GridSearch does. As metric, we used **f1_macro** to find a best match and we also selected **F-score** as its more relevant scoring if the dataset is not balanced.

To make text classification possible, we always tested two possible vectorizer such as **CountVectorizer** and **TfidfVectorizer**, which are both from the sklearn library. As the third part of the pipeline, we introduced *SelectKBest* as a feature selection in order to reduce the space and time and possibly improve the performances, which resulted in a success. In this section, we have also tried to train models using the additional metadata described in the section 4.1.

Finally, when all results from pipelines were obtained, we tried to implement algorithms for imbalanced learning as **SMOTE**, **ADASYN** for oversampling and **Tomek Links** for undersampling (all three with default parameters). In the table *1* are presented the parameters tested inside the pipeline.

Parameter	**Values**
ngram_range	(1, 1), (1, 2), (1, 3)
max_df	0.75, 0.8, 0.85, 0.9
min_df	2, 5, 7, 10
SelectKBest (chi2)	50,100,200,500,1000,'all'

Table 1: Vectorizer parameters

The parameter **n_gram_range** concerns the type of ngram the vectorizer is using to create word features which are consecutively trained. For example, *uni-gram* (1,1) means that every word will be cosidered as a feature. **CountVectorizer**, as well as **bag of words**, can do a simple count of them in every tweet (as already its name indicates). The *n_gram* = (1,2) would be a mix of *uni-gram* and *bi-gram.* In **SelectKBest** the parameter *k* is just a desired number of features selected by the algorithm. All these parameters were used for every classifier described in this report.

3.3.2 Usage of metadata

As already mentioned, the dataset contains a lot of additional information (*features*) which were not primarily used. It was decided to use metadata once every classifier was already trained and they were tested only on the best model. To use metadata we were constraint to drop every not numerical column from the dataset. Next, we computed a simple correlation matrix where we were able to observe that there were a high correlation between *'tweet_len'* and *word_count* and, for this reason, we have dropped the first one mentioned. This dataframe contained just a numerical metadata and it was continuosly converted into numpy array. To create a full complete dataframe which contains the vectorized metadata (using CountVectorizer and TFdfVectorizer) and in order to be able to compute the classification task, we had to create one separate dataframe where we vectorized *tokenized_text* and than merged it with the metadata using the *hstack* function from sklearn library.

3.3.3 Hyper Parameter Tuning

As we have already described, the process for tuning all classifiers remains the same; we did two different pipelines for two different vectorizer where we were testing various paramaters mixed with hyper parameter tuning using *RandomizedSearchCV*. For **Support Vector Machine** in order to find the best possible parameters for the model, different values can be seen in table 2. As the most important parameter to test is obviously the *kernel* which permits the algorithm to manage a non separable cases if it's required and parameter C as a slack variable which permits missclassification. For this classifier, we won't specify further information about training the model using metadata as we have not obtained any satisfying results and all performances were significantly lower. Is also really important to mention that this classifier was the slowest among the others.

Considering that **Random Forest** has a high sensibility to overfitting, we have tried to check the *Learning Curve* with 5 cross-validation in order to prevent it. As a crucial variable to prevent overfitting, we introduced *max_samples*, which is the percentage of maximum of samples which the algorithm considers in every tree. To limit the *Max_depth* was tried too but the performance was significantly lower which was not happening including just max_samples. Random Forest classifier was also trained on the text with merged metadata using the parameters found by pipeline (all parameters metioned can be seen in table 4). As the performance was slightly worse or was not changing we have decided not to proceed with any further changes in training with metadata.

XGBoost classifier (table 5) is a very powerful classification algorithm and is one of the most used in the real enterprise world, famous for its high performances and relatively not expenses. For these reasons, we tried to include it in our study but, as for the previous random forest classifier, the metadata did not significantly improve performances so we did not proceed with a new pipeline.

Logistic Regression is considered as a good option for text classification, so we included its training in this project. Following the same procedure as for the previous ones, the most important parameter to find was *solver* (table 6). Then, we also included the parameter *class_weight* due to the dataset's imbalance.

Naive Bayes classifiers are famous for their high performances for these types of tasks, especially for bag of words, which can be confirmed also in this project where we used **Multinomial NB** (table 3). However, using metadata was not beneficial at all, also considering that the model stopped completely to work. For that reason, we haven't proceeded with a further training.

Parameter	**Values**
C	0.01, 0.1, 1, 10
gamma	auto, scale
kernel	linear, rbf, poly, sigmoid
class_weight	None, balanced

Table 2: SVM hyperparameters

Parameter	**Values**
fit_prior	True, False
alpha	0.0001, 0.001, 0.01, 0.1, 1, 10

Table 3: Multinomial NB Hyperparameters

Parameter	**Values**
n_estimators	100, 250, 500
max_samples	0.2, 0.3, 0.4
min_samples_leaf	2, 3
criterion	'gini', 'entropy'
max_features	'sqrt', 'log2'

Table 4: Random Forest Hyperparameters

Parameter	**Values**
learning_rate	0.01, 0.1, 0.2, 1
max_depth	3, 4, 5, 6, 7, 8, 9
gamma	0, 1, 2, 3
min_child_weight	1, 2, 3, 4
n_estimators	100, 200, 300, 400, 500

Table 5: XGBoost Hyperparameters

Parameter	Values
C	0.001, 0.01, 0.1, 1.0, 10
solver	'lbfgs', 'liblinear', 'sag', 'saga', 'newton-cg'
penalty	'l1', 'l2'
max_iter	500, 1000
class_weight	'balanced', None

Table 6: Logistic Regression Hyperparameters for RandomizedSearchCV

3.3.4 Long-short Term Memory

A further test was conducted with the **Long-Short Term Memory** (**LSTM**)[3] model, that is generally used for the preprocessing and classification in the field of Natural language processing. We tested it using *is_hate_speech* as the binary target variable and *clean_tweet* as a representative corpus. With the help of modules like *tensorflow* and *keras*, we deployed some sequential models in a different conformation.

The table 7 shows the implementation and composition of the models used.

Characteristics	**Model** 1	**Model** 2	**Model 3**
Number of Embedding Layer	1	1	1
Number of LSTM Layer	1	1	2
LSTM Neurons	16	32	16 — 32
Value of Recurrent Dropout	0.2	0.5	0.5
Value of L2 regularizer	0.25	0.25	0.25
Number of Dense Layer	1	1	1
Dense Layer Neurons	1	1	1
Activation Function	sigmoid	sigmoid	sigmoid
Loss	binary_crossentropy	binary_crossentropy	binary_crossentropy
Optimizer	Adam	Adam	Adam
Number of Drop Out Layer	Nan	2	2
Value of Drop Out Layer	Nan	0.2	0.2

Table 7: Models implementations and composition in LSTM

With the purpose of decreasing overfitting[4] we tested various conformations of the models, trying to model the number of neurons per layer, the dropout parameter presence or absence and then the presence of an entire Dropout layer. Moreover, we tried to train models using different kinds of *batch_size* [5] $(100, 128, 500, 1024)$ and considering both performance and computational cost. Observing the results, the best choice was the one with *batch_size* valued at 128.

3.3.5 Convolutional Neural Network

We also tested the model **Convolutional Neural Network** (**CNN**) using *is_hate_speech* as the binary target variable and *clean_tweet* as a representative corpus. With the help of modules like *tensorflow, Keras and transformers* we deployed some sequential models in different conformations and since the classification task is a binary one, we set the "binary cross entropy" as loss and "sigmoid" as the activation function in the output layer.

[3] LSTM is an optimized version of Recurrent Neural Network, used for the resolution of the problem connected to manage the long time dependence between sequences.

[4] Overfitting is a problem connected to every supervised learning tasks. It consists of the possibility of having a good performance on the training set (the portion of data in which we train the algorithm) and a bad performance on the test set (the portion of data in which we appreciate the results of the prediction).

[5] Number of training samples used in a single iteration during the training

In the table *8* we show the implementation and composition of the models used.composite of:

Characteristics	**Model 1**	**Model 2**	**Model 3**	**Model 4**
Number of Embedding Layer	1	1	1	1
Number of Conv1D Layer	1	1	1	1
Pooling	GlobalMax	GlobalAverage	GlobalMax	GlobalMax
Total number of Dense Layer	2	2	3	3
Value of L2 regularizer	0.01	0.01	0.01	0.01
Number of Vanilla Layer	2	1	2	2
Number of Drop Out Layer	2	Nan	2	5
Value of Drop Out Layer	Nan	Nan	0.2	0.2
Optimizer	Adam	Adam	RMSprop	SGD
Hidden Layer Activation Function	relu	relu	relu	relu

Table 8: Models implementations and composition in CNN

3.3.6 BERT

In this section we described the implementation of **BERT** (**Bidirection Encoder Representations from Transformers**) transformer, which revolutionized the field of natural language processing (NLP) with its ability to provide contextual and bidirectional language understanding. The objective is to verify whether the application of BERT to the classification of tweets as hate speech produces significantly better results than more traditional models. To do this, we used the HuggingFace's transformers library to fine-tune pretrained BERT Base model for our usual tweets binary classification task. In detail, the main steps performed are:

1. Loading the uncased BERT Pre-Trained Model using the Hugging Face Transformers library.
2. Data Preparation: tokenize training, validation and test data using the BERTtokenizer, where tokens are converted to vectors using an embedding operation; application and use of token padding, the process of adding special tokens so that all sequences have the same length, and attention mask, a binary sequence that has the same length as the input sequence and indicates which tokens should be considered (with a value of 1) and which should be ignored (with a value of 0) when processed by the model.
3. Creating the DataLoader to provide batches of data during training and validation and loading the pretrained BERT model with a single linear classification layer added on top.
4. Training the model using the training DataLoader and validating the validation one, defining the loss function and the optimizer.
5. Tuning and Optimization model hyperparameters, such as learning rate, number of epochs, batch size, etc., evaluation of the model and performance on test data.

4 Results

4.1 Data Understanding and Preprocessing

The results of the operations and analyses described in section *3.1* are listed below.

The two starting textual datasets called PolicyCorpusXL and ReligiousHate are composed of 7,000 tweets on political debates and 3,000 tweets in the religious domain, respectively. Contextual data are instead represented by means of the following features: 'anonymized_tweet_id', 'created_at', 'retweet_count', 'favorite_count', 'source', 'is_reply', 'is_retweet', 'is_quote', 'anonymized_user_id','user_created_at', 'statuses_count', 'followers_count', 'friends_count', 'anonymized_description', 'dataset'.

For simplicity of analysis it was decided to merge the textual and contextual data into a single dataset based on the tweet ID, from which 78 rows of missing values detected were deleted, as it was not possible to recover the metadata of the corresponding tweet.

In the figure 1 it is possible to notice that the target variable is quite unbalanced and this was taken into consideration for the subsequent supervised classification algorithms.

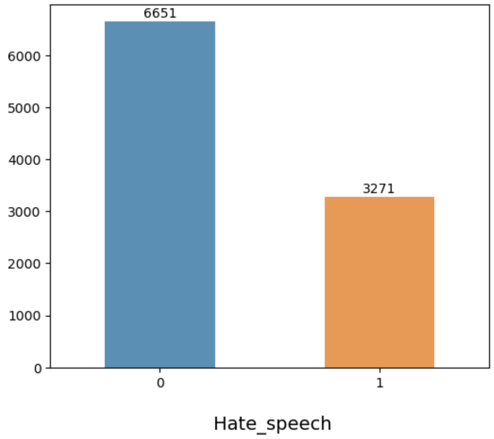

Figure 1: Target variable distribution

In relation to the Feature Engineering phase, new columns intended for different purposes were extracted: the hashtags present in the tweets were extracted for the creation of a Wordcloud plot, represented in figure *5*. This visualization is considered significant for better understanding the messages posted by users and this is in line with the main political and religious debates of the reference time period most used for the creation of tweets, which can be viewed in figure *2*. The 'Weighted_Engagement' feature was created using existing metadata and used to evaluate the content of tweets with a high level of engagement. The features of 'Tweet_length', 'word_count', 'device', and 'swear_word_bin' were defined and added to the set of metadata used later in the classification phase. In particular, 'swear_word_bin' was generated starting from a list of offensive Italian words obtained from *Free Web Headers*[6] , allowing the presence or absence of such words to be determined within each tweet.

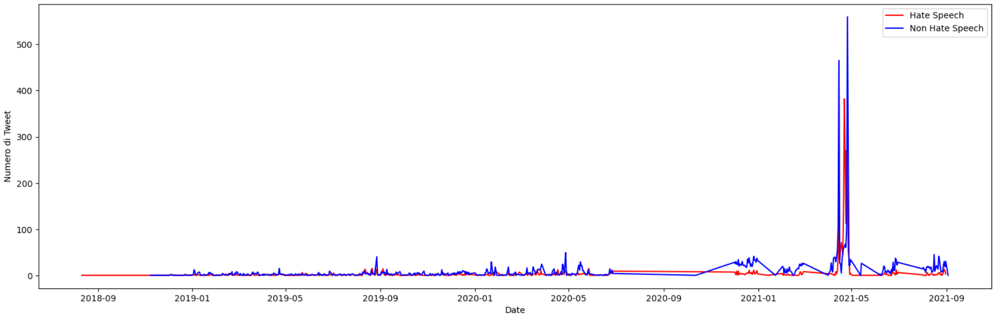

Figure 2: Political and religious tweets in the years

We conducted complete preprocessing on the tweets to ensure their cleanliness. This involved several steps, including the removal of emoticons, links, mentions (@username), retweet flags ("RT"), punctuation, multiple spaces, and all non-alphanumeric characters. Additionally, we employed the NLTK library to eliminate Italian and English stopwords, along with other frequently occurring words considered irrelevant. Short words were also filtered out during this process. After the initial cleaning, we used the NLTK library for tokenization, so that in our analysis each token corresponded to a word within the tweet. Finally, we applied the Spacy library to lemmatize the sentences, converting words to their base form taking into account contextual nuances. For Part-Of-Speech tagging, which is included in operations using Spacy functions, capitalizing words has been found to be useful for more accurate word recognition. Stemming transformation was tried but we considered it not effective for our goals. Upon completion of the entire preprocessing pipeline, of the renaming operations and of the elimination of usless features, the resulting dataset was composed by 9,922 rows and 25 columns. In figures *3* and *4* it is possible to observe Wordcloud plots of the most frequent words of the cleaned tweets divided by political and religious domains.

[6] Free Web Headers, https://www.freewebheaders.com/full-list-of-bad-words-banned-by-google/

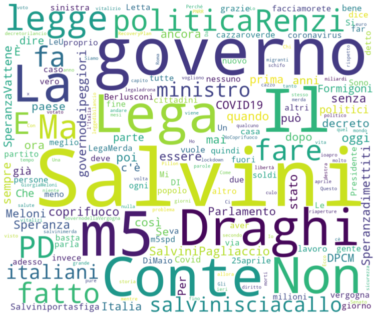

Figure 3: Wordcloud political domain

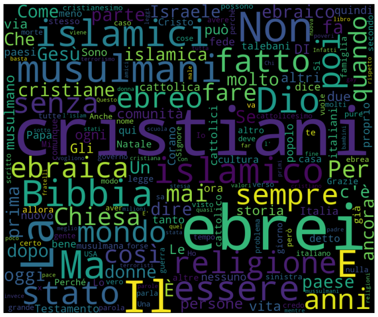

Figure 4: Wordcloud religious domain

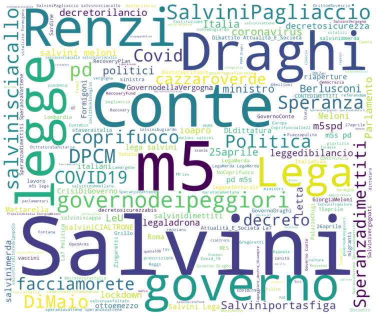

Figure 5: Wordcloud hashtags

4.2 Topic Modeling

As mentioned before The first experiment on Topic Modeling was a research on the description of the users. We implemented Topic Modeling based on descriptions for tweets with hate speech and for tweets without hate speech by arbitrarily varying the desired number of topics, but the results weren't much informative: some topics overlapped in both implementations, with the hate speech case being a little more dispersed.

Hoping to find some other and more interesting data, we implemented a Topic Modelling still based on descriptions but independent from the hate/non-hate speech for the religious and the political dataset. Observing the graphs in figures *6* and *7* obtained by training models on different numbers of topics, we considered the optimal number of topics to use equal to 8 and 10 respectively but, even in this case, the results weren't satisfactory enough to consider using them in our future implementations, due to high overlapping and non-informative nature of the topics.

Topics	**Top words**
Topic 1	salvini,salvinisciacallo,legaladrona
Topic 2	covid19,coronavirus,vaccini
Topic 3	draghi,speranzadimettiti, governodeipeggiori
Topic 4	salvini, m5s, governo, lega
Topic 5	cristiani, ebrei, islamico
Topic 6	conte, salvini, ancora
Topic 7	ebrei, formigoni, musulmani, islamici
Topic 8	salvini, politica, renzi, m5s, conte

Table 9: Top relevant words for topics in the whole dataset

Our last approach to Topic Modeling was an entirely different attempt, in which we considered the text into the tweets instead of the users' descriptions, and in this case we had some better separated topics, probably as a result of the mixed nature of the whole dataset tweets, containing both religious and politics ones; this can also be noticed by looking at the most relevant words for topics, as shown in Table 9.

Ultimately, we decided to abandon Topic Modeling and leave it only for informative purposes, as we didn't find a way to effectively employ this data to our research.

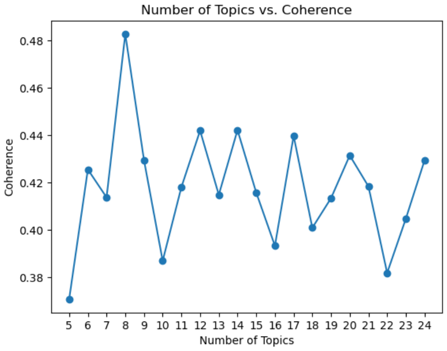

Figure 6: Trend of coherence variation based on the number of topics considered in the politics dataset

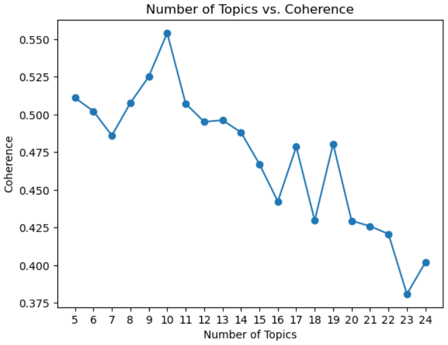

Figure 7: Trend of coherence variation based on the number of topics considered in the religious dataset

4.3 Classification

4.3.1 Base classifiers

In this section we will present the best performances for all trained classifiers also we will provide a demostration of unsuccessful and uselessness of metadata.

	CountVectorizer			TfidfVectorizer		
Model	**ngram_range**	**min_df**	**max_df**	**ngram_range**	**min_df**	**max_df**
SVM	(1, 2)	5	0.9	(1, 2)	5	0.9
XGB	(1, 3)	7	0.75	(1, 1)	5	0.75
LR	(1, 3)	2	0.8	(1, 2)	2	0.75
RF	(1, 3)	2	0.75	(1, 3)	2	0.7
MulNB	(1, 1)	2	0.75	(1, 1)	2	0.75

Table 10: Vectorizer Configurations for Different Models

Model	Accuracy	F1-score	Vectorizer	SelKBest	Configurations/Parameters
SVM(TomekLinks)	0.8344	0.7918	TFidf	100	linear, auto, balanced, 10
XGBoost(SMOTE)	0.8296	0.7909	CountVect	200	400, 1, 5, 1, 2
RF(TomekLinks)	0.8347	0.7881	CountVect	500	250, 2, 0.4, sqrt, gini
LR (SMOTE)	0.8371	0.8013	TFidf	200	liblinear, l2, 1000, balanced, 1.0
MultiNB(TomekLinks)	0.8290	0.7940	CountVect	200	False, 1

Table 11: Best Performances - SVM, XGBoost, RF, LR , MultiNB

In the table 10 are shown all configurations for vectorizers used in every pipeline. We can observe that for ”bag of words” vectorizer the combination of uni-grams, bi-grams and tri-grams (1,3) was preferred. It can also be observed that for 3 out of 5 trained models was preferred CountVectorizer but, on the other hand, the best F-score obtained was for Logistic Regression with TFidf. In the Figure 8 is displayed the confusion matrix obtained for the best trained model and in the 9 is shown the Precision-Recall curve for the same model.

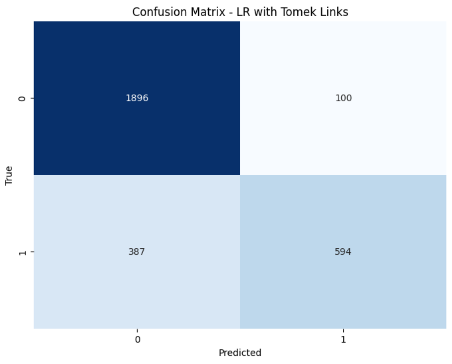

Figure 8: LR Confusion Matrix

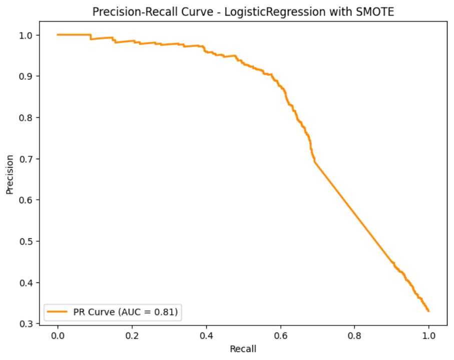

Figure 9: LR Precision-Recall Curve

None of these classifiers had better performances using metadata and that’s why we are not providing more information. However, it is important to mention that SVM, LR and Multinomial NB failed completely with metada, tree-based classifiers as Random Forest and XGBoost were able to do quite good prediction but not better (accuracy always around 80% and F-score around 75%).

4.3.2 LSTM

Initially, the models were implemented using the configurations explained in the previous section but, as optimizer, the Stochastic Gradient Descent (SGD). The results in terms of overfitting and classifications were not at all acceptable. Thus, various changes have been made: variations in the number of neurons per layer, activation function, the built-in dropout to the LSTM layer, the presence of additional Dropout Layers. The most important turning point comes with the introduction of the optimizer "Adam", that is the successor of SGD and is particularly useful for sparse gradient problems, just as in the case of data types such as textual ones. In fact, the overfitting problem turned out particularly filed with this operation and thus the classification improves (previously, the minority class was particularly misclassified). Therefore, we also tried to enlarge the embedding dimension from 10 to 100, integrating a greater number of batch_size in order to make the operation less computationally expensive. The model performed well, as shown in the figure 10.

From the figure 11, we could notice that the error dropped equally on both training and validation sets (unlike previous experiments, in which measures differed particularly). So, at this point, we tried to test a model with higher capacities, using an LSTM layer with 32 neurons and having the same characteristics of model 1. The results were still good, as we got good performance in terms of accuracy and F1-score, that can be seen in the table 12.

Observing the table 12, we could see that models showed a similar performance, although the second one seems to move better especially than the F1-score variable, suggesting a better model behavior on the minority class. In fact, even observing the F1-score of the minority class, the second model reaches a value of 0.71 while the first one only reaches 0.66.

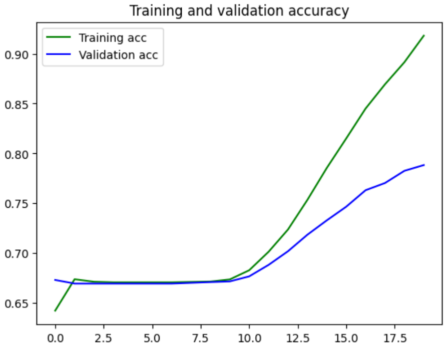

Figure 10: Model 1 - Projection of the accuracy across epochs

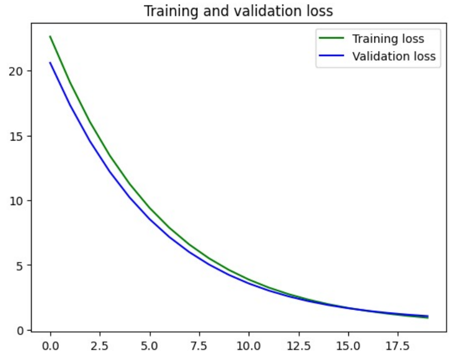

Figure 11: Model 1 - Projection of the Loss score across epochs

	Accuracy	**F1-score**	**Precision**	**Recall**
Model 1	0.82	0.77	0.83	0.75
Model 2	0.83	0.80	0.83	0.78

Table 12: Performances - LSTM best models

Given these good results, we decided to test a further model with more than one LSTM layer. The obtained results were different, even with the presence of L2 regularizer and dropout layer a slight overfitting is detected. Nevertheless, the model managed to perform quite well and to obtain good results in terms of performance (which we omit for a matter of synthesis).

As a future work, it would be interesting to test the models and also trying to implement a grid-search to understand what could be the optimal value of neurons or embedding size in the embedding layer. However, satisfactory experiments were found.

4.3.3 CNN

The analysis conducted on CNN concerned the impact that the presence or absence of fully connected layers, variations in pooling layers and different optimizers could report on the tested models.

We decided to test different conformation of the network making some difference in these layers; the models were all tested with 20 epochs; the first model we tested was formed by one Convolutional Layer with one dimension, a **Global maximal pooling** and two vanilla layers concatenated. The loss has become lower in a good way both in the train either in the validation but in the final epochs, while the loss on the validation restarted growing. For this motivation, we decided to introduce a L2 regularizer also in this algorithm for every model tested. This managed the problem.

So, we decided to test the **Global Average Pooling** in the second model. No satisfactory results were obtained, since the minority class (the one which is afferent to the hate tweet) was completely missed to the classification. We also tried to change the activation function in the hidden layer from ”tanh” to ”relu” and actually this change slightly improved the situation. However, there was no possibility to appreciate the distinction between the two analyzed Global Pooling Layer.

The model 3 and 4 were tested we tested these models with the optimizer ”RMSprop” and ”SGD”. Just for a comparison, in table *13* we deploy the results of model 1 and model 2:

	Accuracy	**F1-score**	**Precision**	**Recall**
Model 1	0.71	0.20	0.84	0.56
Model 2	0.67	0.41	0.84	0.50

Table 13: Performances - CNN best models

First of all, it’s important to bear in mind that the ”F1-score” is afferent to the minority class metric. As it is possible to notice, the models look quite the same; the identification of the minority class, our original goal to be achieved, is better gained in the second model. We believe that much work still needs to be done.

4.3.4 BERT

To implement BERT, all the steps described in section x were carried out; in particular, after various attempts, it was decided to use the pre-trained ”bert-base-multilingual-uncased” model, considered more effective for tokenizing the Italian tweets provided as input. For the fine-tuning task, we referred to the official BERT paper, which suggests tuning really few parameters with very few associated values. More precisely, we compared different performances varying the number of training epochs (taking care not to use a high number of epochs which can lead to overfitting), the learning rate and weight decay for the Adam optimizer and Batch-size, to define the number of training samples used in a single iteration during the training process. We used 10% of our training data as a validation set, we converted all data into ”torch.tensors”, and the ”BertForSequenceClassification”, also provided by the Hugging Face Transformers library, was used to use BERT in the classification task of text sequences.

Finally, the model was trained, and the final classification report and configuration parameters are reported in table *12* and *13*, while training loss curve and confusion matrix are showed in figure *14* and *15*.

As the epochs progress, we can observe a steady decay of loss. This suggests that the model is learning from the training data and improving its predictive capabilities, and the accuracy is among the highest achieved by classification models. For class 1, we notice a lower precision and a fair recall value, suggesting that the model manages to capture a good part of the instances of class 1 present in the test data, indicating a greater sensitivity of the model to this class.

	Precision	Recall	F1-Score	Support
0	0.88	0.91	0.89	1996
1	0.80	0.74	0.77	981
Accuracy				0.85

Figure 12: BERT classification report

Parameter	**Values**
Epochs	4
Learning rate	5e-5
Weight Decay	0.01
Batch-size	32

Figure 13: BERT best parameters

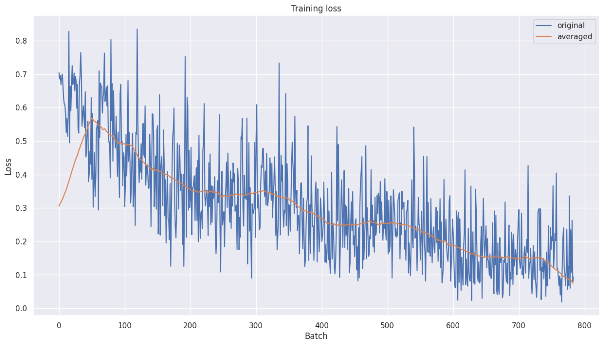

Figure 14: Training loss curve over all batches

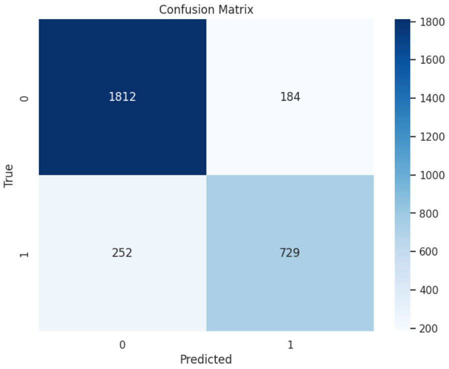

Figure 15: BERT confusion matrix

5 Discussion and conclusion

The purpose of this project was to compare the different classification's models in order to correctly detect hate speech on Twitter. Several algorithms were trained and we played a lot with pipelines to get the best version of the text, although the changes did not always lead to significant increases in results as hoped.

In light of the results obtained, it can be concluded that all models guaranteed a fair prediction capacity, managing to classify class 0 (hate speech) better than class 1 (no hate speech).

These results could be a consequence of several factors like data imbalance, as the model may learn to predict the majority class more accurately, but also discriminating features, considering that features present in political and religious tweets may be easier to distinguish when it comes to non-hate speech compared to hate speech.

Another factor could be the nature of the tweets used, which can significantly influence the performance of text classification models. In fact, brevity, slang, emoticons and informal grammar are distinctive features of tweets and can pose difficult challenges for classification models, but also for unsupervised learning models such as Topic Modeling, leading to poor and non-significant results.

Some classification approaches, such as recurrent neural networks (RNNs) or language transformation models (such as BERT), may be better suited to capturing linguistic nuances and addressing the brevity of tweets. In fact, BERT is confirmed to be the best classifier used.

In future works, it would be useful to consider a larger number of tweets and descriptions to better train the models considered and improve their performances. In addition, it would be interesting to test the models and try to implement a grid search on LSTM to understand what could be the optimal value of neurons or embedding size in the embedding layer.

References

[1] M. Duggan. *Online Harassment.* 2014.

[2] Cristina Bosco & Felice Dell'Orletta & Fabio Poletto & Manuela Sanguinetti & Maurizio Tesconi. *Overview of the EVALITA 2018 Hate Speech Detection Task*. Whitepaper. CEUR Workshop Proceedings, 2018.

[3] Manuela Sanguinetti & Gloria Comandini & Elisa di Nuovo & Simona Frenda & Marco Stranisci & Cristina Bosco & Tommaso Caselli & Viviana Patti & Irene Russo. *HaSpeeDe 2 @ EVALITA2020: Overview of the EVALITA 2020 Hate Speech Detection Task*. Whitepaper. CEUR Workshop Proceedings, 2020.

[4] Thomas Davidson et al. "Automated Hate Speech Detection and the Problem of Offensive Language". *Proceedings of the International AAAI Conference on Web and Social Media* 11 (2017). DOI: 10.1609/icwsm.v11i1.14955.

[5] Giovanni Fulantelli & Davide Taibi & Lidia Scifo1 Veronica Schwarze & Sabrina C. Eimler. "Cyberbullying and Cyberhate as Two Interlinked Instances of Cyber-Aggression in Adolescence: A Systematic Review". *Frontiers in Psychology* 13 (2022). DOI: https://doi.org/10.3389/fpsyg.2022.909299.

[6] Aditya Bohra & Deepanshu Vijay & Vinay Singh & Syed Sarfaraz Akhtar & Manish Shrivastava. "A Dataset of Hindi-English Code-Mixed Social Media Text for Hate Speech Detection". *Association for Computational Linguistics* (2018), pp. 36–41. DOI: 10.18653/v1/W18-1105.

[7] Tommaso Caselli & Valerio Basile & Jelena Mitrović & Inga Kartoziya & Michael Granitzer. "I Feel Offended, Don't Be Abusive! Implicit/Explicit Messages in Offensive and Abusive Language". *European Language Resources Association* (2020), pp. 6193–6202.

[8] Valerio Basile & Cristina Bosco & Elisabetta Fersini & Debora Nozza & Viviana Patti & Francisco Manuel Rangel Pardo & Paolo Rosso & Manuela Sanguinetti. "SemEval-2019 Task 5: Multilingual Detection of Hate Speech Against Immigrants and Women in Twitter". *Association for Computational Linguistics* (2019), pp. 54–63. DOI: 10.18653/v1/S19-2007.

[9] Md Saroar Jahan & Mourad Oussalah. "A systematic review of hate speech automatic detection using natural language processing". *Neurocomputing* 546 (2023). DOI: https://doi.org/10.1016/j.neucom.2023.126232.