BiLSTM for bipolarity sentiment analysis.

A Gated Recurrent Neural Network for Supervised Text Classification: detecting hate speech from different online textual genres.

Diletta Goglia[1]

[1]MSc student in Computer Science, Artificial Intelligence curriculum.
Computer Science Department, University of Pisa, Largo B. Pontecorvo 3, 56127, Pisa, Italy.
d.goglia@studenti.unipi.it

This project was developed for the
"Human Language Technologies" course
of Professor Giuseppe Attardi.

UNIVERSITÀ DI PISA

Contents

1 Introduction and motivation

Sentiment analysis is a very relevant task in Natural Language Processing (NLP) which has become very popular and extensively used in the most recent years given the increasing amount of available text online, as well as the power that this technique can offer. Identifying, understanding and forecasting in-depth knowledge of people's emotions has become crucial in many fields and companies. From a NLP perspective, much attention has been paid to the automatic detection of Hate Speech (HS) and related phenomena (e.g., offensive or abusive language among others) and behaviours (e.g., harassment and cyberbullying). This has led to the recent proliferation of contributions on this topic ([15], [21], [7]). Although there is no universal definition of hate speech, we refer to the one provided by Nockleby [20]: "any communication that disparages a target group of people based on some characteristic such as race, colour, ethnicity, gender, sexual orientation, nationality, religion, or other characteristics".

1.1 Project aim and novelties

In this project we will focus on Emotion-based Sentiment Analysis[10] with Bi-Polarity Classification[1] and prediction, working with unstructured and implicit sentiments retrieved from already labelled corpora. The goal of the project is to present a Deep Learning model implementation able to recognise hateful and discriminatory online text, together with its evaluation, prediction and further reflections which this work is aiming to describe. The ultimate purpose of this model is to assign a binary label to a text sequence in order to classify it as containing or not hate speech.

The first novelty we introduce in this work, with respect to existing sentiment analysis projects, is the multi-source nature of the corpus built. In order to have a more comprehensive overview of the hate speech phenomenon on Internet we did not limit to a unique source of data: we built an integrated dataset for hate speech analysis of text retrieved from three different online sources: Twitter, Wikipedia and Stormfront, a discussion forum. The corpus we worked on is the result of this integration of different corpora coming from three different textual domains: tweets, comments and post. The combination and integration of these new textual genres is made possible by the syntactic similarity of their structure (typically short, quick and non-revised), as well as of the their linguistic variety in terms of register and style used in the written text. Given these premises about the properties of e-language, extensively described and contextualised in [3, 6, 8, 14], we justify our choice of building this new kind of integrated corpus. The resulting embed remains homogeneous and coherent within their lexical characteristics. A subsequently advantage of this approach is the substantial amount of data collected: since we are interested in implementing a Deep Neural network, a considerable volume of data is definitely a fundamental pre-requisite on order to have a better learning model.

Although what is presented in this project refers to already existing architectures and techniques, for the aim of this work we decided to opt for a trade-off between traditional and original regarding the structure of our model. We investigated many pre-existing works and projects finding out two common approaches in NLP that seem to be always used in opposition: TF-IDF and GloVe Embeddings, being, respectively, non-semantic (count based) and semantic (non context-based) vector space models. A simple Google search and just a superficial exploration of the state of the art could convince the reader of the existing polarity in application between these two techniques. We decided to implement a model that could be able to take both methods into consideration and learn from a meaningful combination of the two, though not reusing already existing experiments in this regard (i.e., multiplication [17, 11]).

After a long experimental phase, the final model we propose consists in a Bidirectional Long Short-Term Memory (BiLSTM) Artificial Neural Network, a sequence processing model that consists of two LSTMs, one taking the input in a forward direction, and the other in a backwards one. Thanks to these two layers in opposite directions, this model is able to capture context information of a text sequence: BiLSTMs effectively increase the amount of information available to the network, improving the context available to the algorithm[2].

[1] https://www.sciencedirect.com/topics/computer-science/polarity-classification

[2] In the LSTM model, the hidden-layer state of each location encodes only the context information in the forward direction and ignores the backward context information. The BiLSTM model utilises both the positive and reverse orders of the sequence information by concatenating the hidden-layer outputs of each model.

Table 1: Data split for each source.

Datasets not splitted from source	
Source	**TR/VAL/TS split**
Twitter_1	23,544 TR records (95%) of which 90% TR and 10% VAL 1,239 TS records (5%) Total: 24,783 records
Forum posts	10,397 TR records (95%) of which 90% TR and 10% VAL 547 TS records (5%) Total: 10,944 records
Datasets with TR/TS split already provided by source	
Source	**TR/VAL/TS split**
Twitter_2	31,962 TR records (54%) of which 90% TR and 10% VAL 17,197 TS records (46%) Total: 49,159 records
Wikipedia comments	159,571 TR records (51%) of which 90% TR and 10% VAL 153,164 TS records(49%) Total: 312,735 records

Indeed, for sequences other than time series (e.g., text), it is often the case that a Recurrent Neural Network models (RNN) can perform better if it not only processes sequence from start to end, but also backwards, since in sentiment analysis, as well as in similar tasks, it is often useful to have the context around the word available rather than information related only to adjacent words.

The BiLSTM model has been widely used with good performance in many NLP tasks, besides representing a substantial improvement of LSTM that effectively solves the problem of gradient disappearance or explosion in simple RNN. For example, BiLSTM with an attention mechanism has widely been proved to be an effective model for sentiment analysis[24, 19, 22]. In this work we implement a BiLSTM for supervised text classification, which topology is able to concatenate and learn complex relationships from both pre-trained embeddings and count-based language representation inputs.

2 Data

2.1 Data availability

The corpus (embedded dataset) included in this work, as well as the whole project implementation is openly available on GitHub at the following link: github.com/dilettagoglia/BiBiNET. All the sources with which the dataset has been built are freely and openly available at the links provide in the following section.

2.2 Sources description

In this section a description of the corpus, as well as of each single data source, is provided. In order to develop a fairly intuitive, though not exhaustive, analysis on the hate speech detection, different types of datasets already labelled have been exploited and combined, resulting in a text integration from different online sources as described in Section 1. Table 1 summarise all the details related to each single data source.

2.2.1 Twitter data

For collecting and embedding tweets in our corpus we exploited two different sources. The first Twitter dataset "Hate Speech and Offensive Language"[4] comes from `data.world` and is openly available (under CC-BY license) at the following link https://data.world/thomasrdavidson/hate-speech-and-offensive-language. It contains content that is racist, sexist, homophobic, and offensive in many other ways, so that we considered it a good starting point for our hate speech detection project. The CSV file includes 24,783 tweets, formerly labelled by human agents in three different categories: *hate speech*, *offensive language*, or *neither*.

Second dataset "Twitter hate speech" from Kaggle https://www.kaggle.com/datasets/vkrahul/twitter-hate-speech contains a total of 49,159 tweets, already splitted in two CSV files, one for training (31,962 records, already labelled in two categories - 1 and 0, identifying, respectively, tweets containing or not hate speech -) and one for test (17,197 not labelled records).

2.2.2 Stormfront forum posts

As representative of the second kind of textual genre we are including in our analysis, i.e., forum posts, the source chosen is Stormfront, an internet discussion forum characterised by giving expression to positions of white nationalism, white supremacy, anti- Semitism and neo-Nazism, also referred to as the largest hate speech site on the Internet [23]. We therefore consider it unnecessary to specify the reason for which it was selected as a data source, given our goals.

The dataset "Hate speech dataset from a white supremacist forum"[5] https://github.com/Vicomtech/hate-speech-dataset contains a random set of 10,944 forums posts sampled from several Stormfront subforums which have been manually annotated according to specific annotation guidelines https://arxiv.org/pdf/1809.04444.pdf: `hate` and `noHate` are categories indicate, respectively, the presence or absence of hate speech in a sentence; the label `relation` is used for specific cases where the sentences in a post do not contain hate speech on their own, but the combination of several sentences does, while the label `skip` refers to sentences that are not written in English or that do not contain information as to be classified into `hate` or `noHate` [5]. For the purpose of our analysis we consider `relation` and `skip` categories as not containing hate speech, so that this multi-class structure is reduced in a binary label which perfectly fit our purposes.

2.2.3 Wikipedia comments

The last textual genre we included in our work is the online comment, for which the referring selected source is Wikipedia. The "Toxic Comment Classification Challenge" is part of a competition aimed at identify and classify toxic online comments provided by the Conversation AI team, a research initiative founded by Jigsaw and Google. It is available on Kaggle at the following link: https://www.kaggle.com/competitions/jigsaw-toxic-comment-classification-challenge/overview and it assesses the threat of abuse and harassment on a large number of Wikipedia comments which have been categorised by human raters for toxic behaviour. Six types of toxic comments are labelled (`toxic, severe_toxic, obscene, threat insult, identity_hate`) but, for the purpose of this work, we considered just the binarization of them in one variable, which describes the presence or absence of abuse comments. This source represent most substantial one we gathered, with its 312,735 records (comments) already splitted (about halfway) in separated CSV files for training and test.

2.3 Obtained corpus

Our corpus include 225,474 TR records of which 90% TR (subsequently upsampled) and 10% VAL (22,320 records). All the text entries collected are in English.

3 Software specifications

Before approaching the methodology, which goes in detail of the substantial contributions of this project, we provide a brief summary of relevant computational details.

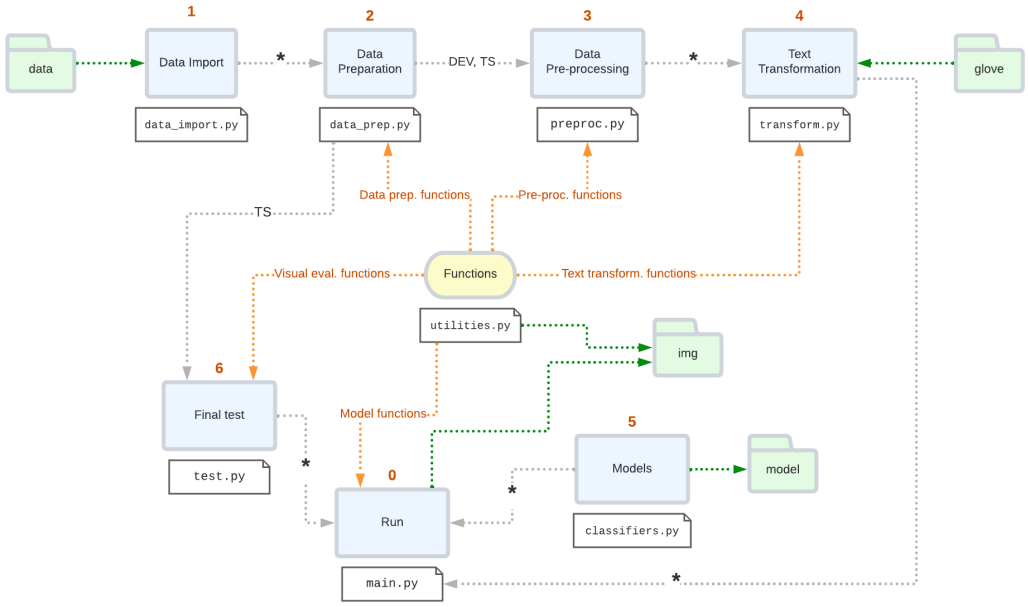

Figure 1: Illustration of the software structure.

The hardware used consists in a 4-Core Intel i7 1,8 GHz. The code of the entire software is written in Python 3.9, with the aid of the following modules, tools and libraries: `ipython`, `keras`, `Keras_Preprocessing`, `matplotlib`, `mlxtend`, `nltk`,`numpy`, `pandas`, `scikit_learn`, `scikit_plot`, `seaborn`, `spellchecker`, `tensorflow`, `tensorflow_gpu`, `wordcloud`. The code is commented following the `reStructuredText` format[3], the official Python documentation standard. The software structure is summarised in Figure 1 in an intuitive visual way: if the reader would need further specification about, all the directory structures can be accessed at the following link: github.com/dilettagoglia/BiBiNET/blob/main/README.md. Each module fulfils a specific phase or sub-task and its implementation coincides with the respective Python file. Many auxiliary functions were necessary to implement: for greater clarity and cleanliness of the software structure, they are kept separate from the main steps and collected in the `utilities` file. The content and the order in the global flow of each single file resemble the conceptual structure illustrated in Figure 2: the fact that file names, modules and order of iterations follow the same common thread of more abstract elements (ideas, phases, development design) is thought to make it easier for the reader how the concrete implementation follows the theoretical workflow.

Further technicalities will be specified afterwards since they may loose meaning if took out from their context.

4 Methodology

In this section we provide a detailed description of the procedure developed for building the entire project from scratch. We can summarise the multiplicity of steps in four main phases of the process:

- **Data preparation** phase, which includes data extraction and acquisition from the previously describes sources, followed by some preliminary analysis and all the preprocessing procedures needed to prepare the data before the actual NLP analysis could start.
- A **transformation** phase, needed to vectorize the text with different methodologies in order to prepare it to be fed in the learning models.

[3]https://www.sphinx-doc.org/en/master/usage/restructuredtext/index.html

- **Model building** and **evaluation** phases, in which many experiments have been conducted investigating performance outcomes based on task and data. Different classifiers have been built and evaluated, trying different structures, parameter tuning and degrees of complexity. This phase also includes the selection of the best model, the final test performed on it and some methodologies to analyse its explainability.

The following subsections are spent to provide a more detailed explanation of all the steps implemented for each phase, so that to present a clear and exhaustive overview of how the project was built and enable a straightforward possibility of repeatability to the reader. For further clarity, Figure 2 offers a simplified scheme of the whole process, including also a brief mention of discarded paths and failed experiments.

4.1 Pre-processing

After having gathered, collected and imported data[4], in order to obtain a feasible corpus to work with, we integrated all the different datasets[5] taking into consideration whether they were already separated into Development (already labelled text) and Test sets (text without labels). For those not already divided in this sense, we decided to opt for a random selection of 5% of records as Test set to be kept separated for the final evaluation of the model, dropping the associated label. All the different corpora have been then merged obtaining two different sets, a Development one (DEV) and a Test one (TS)[6], which have the structure of a Pandas dataframe containing each text entry (comment, tweet or post) and, only for the DEV set, the associated binary label `hate` that can assume 1 or 0 value whether the record contains or not hate speech.

For both corpora, the pre-processing phase[7] consisted in text cleaning (e.g., dealing with links, numbers, special characters contained in text), and in a **preliminary text analysis**. During this phase an Exploratory Data Analysis (EDA) was performed to investigate some useful insights regarding the text. We analysed the distribution of length in records (samples of text) looking both at character and at token level.

We also explored the **class distribution** in the training set discovering a substantial unbalance between the two (the occurrence of the 0 class is very high compared to the 1 class, i.e. there is a bias towards the majority class present in the target) as it can be noticed in Figure 3. The fact that the two classes are not represented equally in the dataset needed to be handled before starting the learning phase since it could lead to poor performance results. Indeed the difference in class frequencies affects the overall predictability of the model since, due to the discrepancy in class distribution, the algorithms could tend to get biased towards the majority values present and may not perform well on the minority values [2]. We describe more in depth how we addressed that issue in the following subsection.

We performed Tokenization, Stemming and Lemmatization on both DEV and TS corpora by applying the respective implemented functions for each row (text entry) in the dataset. We tokenized the text using the NLTK `TweetTokenizer`[8] that we believed appropriate for our entire integrated corpus given the similarity of the belonging textual genres described before. Stemming and lemmatization tools (`SnowballStemmer`[9] and `WordNetLemmatizer`[10], respectively) also belong to NLTK module.

4.2 Splitting the dataset and dealing with unbalanced data

Once the corpora were ready, before starting the core implementation of the project one last step was missing: we needed a validation set to evaluate the performance of our models, as well as a way to fix the unbalance found between the two classes. For this reason we firstly randomly selected 10% of records from the DEV set to be used for the validation phase. Since at this point the split was completed and the training (TR), validation (VAL) and test (TS) set were finally disjoint, we explored and tried several **sampling techniques for balancing the class distribution** on the TR set only (e.g. grid searching proper values to

[4] see box number 1 in Figure 1
[5] see box number 2 in Figure 1
[6] this latter have been maintained independent and disjoint for the rest of the work
[7] see box number 3 in Figure 1
[8] https://www.nltk.org/api/nltk.tokenize.casual.html
[9] https://www.nltk.org/api/nltk.stem.snowball.html
[10] https://www.nltk.org/_modules/nltk/stem/wordnet.html

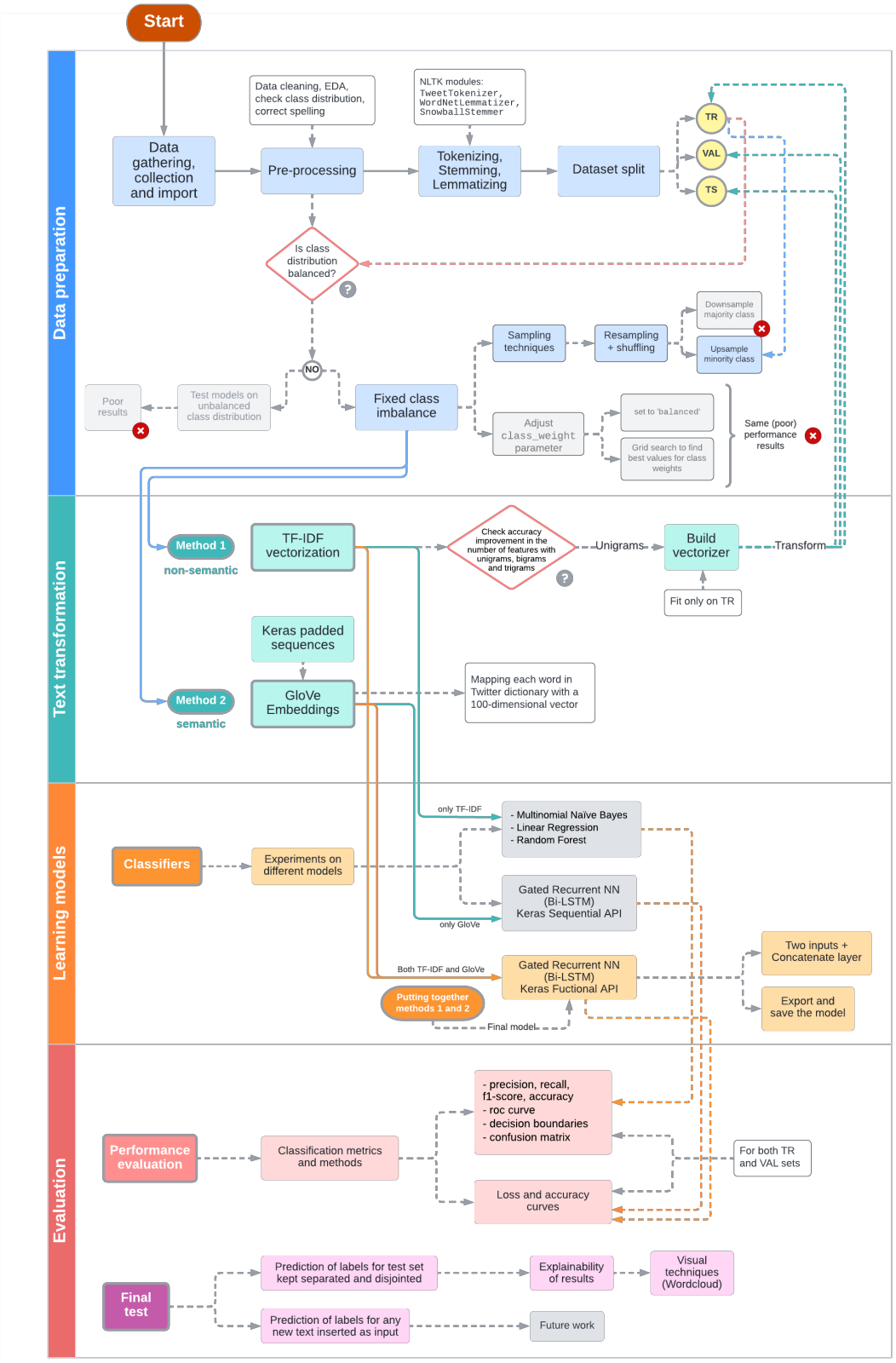

Figure 2: Illustration of the project workflow. The flow chart summarizes the main steps, choices and processes needed for each of the four main phases.

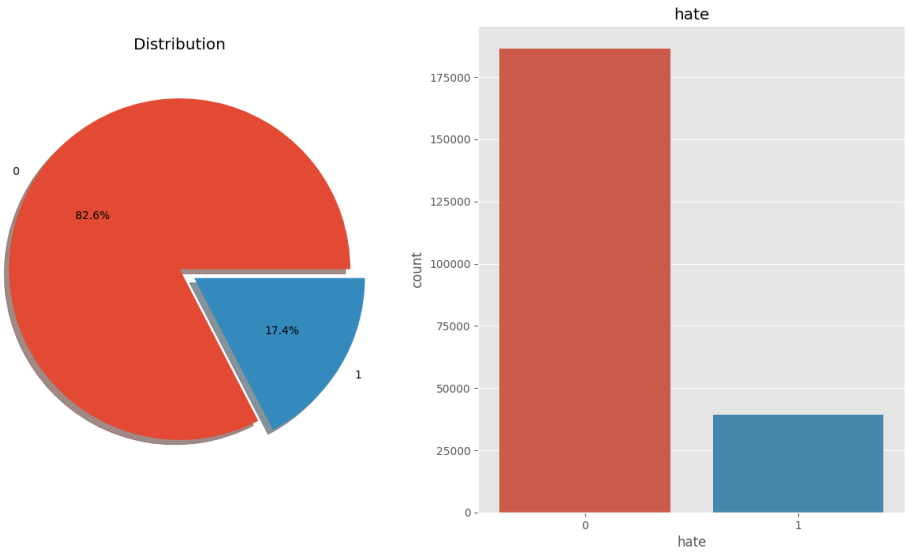

Figure 3: Original class distribution in the DEV corpus, expressed as percentage (left) and total number of records (right). A bias towards the 0 class (i.e., non-hateful) is evident.

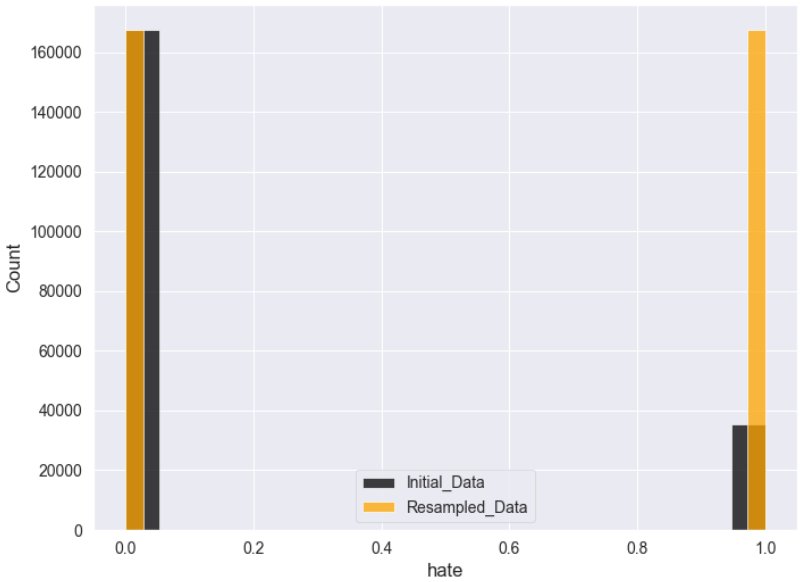

Figure 4: Class distribution of TR set: before (black) and after (yellow) resampling.

assign as input weight to each class in order to penalise the misclassifications). Performing resampling also on data destined to be used in validation phase is usually considered a bad practice in Machine Learning field since validation set may not replicate the distribution of real-world data. Moreover, this could expose the model to data leakage [18]. In order to decide which sampling method was the best for our data we tried fitting a simple but enough powerful model like Logistic Regression with three different corpora, upsampled, downsampled and the original one, so that to observe how different sampling techniques affect the learning of a classifier. The final chosen technique (the one that returned the best performance in terms of precision and recall metrics[11]) consists in a random up-sampling of the minority class on TR set (see Figure 4) by using an implemented function that exploits the `resample` function provided by `utils` module of Scikit Learn[12] and also performs random permutations (**shuffles** the order of records) in the dataset, since we do not want any information associated with the ordering of samples to affect and influence the learning process. In literature, this sampling method is confirmed to be one of the most robust existing [13]. The whole process of investigation in this context, including discarded choices, is summarised in Figure 2, under the "data preparation" phase.

4.3 TF-IDF

Once the data was finally cleaned, pre-processed and prepared for the analysis. the transformation phase started since text needs to be converted in a machine-readable way to let our models learn from it.

Term Frequency–Inverse Document Frequency (TF-IDF) represents one of the most popular term-weighting schemes, particularly used for text classification tasks, to convert textual data to numeric form in order for it to be fed into a neural model[13]. It consists in a statistical measure used to evaluate how relevant a word is to a document (in our case, a text entry) in a corpus. Instead of only counting the occurrence of a word in a single document it also does so in relation to the entire corpus. It works by increasing proportionally to the number of times a word appears in a document but is offset by the number of documents that contain the word, which helps to adjust for the fact that some words appear more frequently in general. In this way, common words that occur in similar frequencies in all documents (i.e., words that are not particularly unique to the text samples in the dataset) are penalised, so that the limitations of other commonly used vectorization methods for n-grams (e.g., one-hot encoding, count encoding) are solved. The drawback of this method, besides occupying more memory and requiring more computing time) is that it does not hold the semantic meaning of the words: we will address this point more in detail in the next section.

In order to figure out the exact number of features needed with respect to the kind of n-grams extracted (unigrams, bigrams or trigrams) we performed the following preliminary analysis: we built a pipeline (exploiting the Scikit Learn `Pipeline` class [14]) made by the vectorizer and a simple logistic regression model to evaluate and print out accuracy scores associated with the number of features. For time and memory constraints we opt for the smaller number of features to extract (corresponding to the lower bound of our analysis, i.e., 100 features), also taking into consideration the fact that the improvement in terms of accuracy towards the upper bound is not so high to let our choice penalise our work. As reported by the plot in Figure 5, the best score associated with the selected number of features to extract is related to unigrams and so we set the `ngram_range` parameter of the TF-IDF vectorizer to $(1, 1)$ range.

Since at this point we needed to convert the text into numerical vectors in order for it to be processed by the machine, we applied TF-IDF encoding on all the corpora (TR, VAL, TS sets), extracting and selecting the top 100 features from the vector of tokens previously obtained by discarding tokens that appear fewer than 2 times. The vocabulary is learned from training corpus an then the three corpora are vectorized[15] using the Scikit Learn

[11] Using the accuracy score may not be accurate when working with unstable datasets: even when the algorithm gives all the predictions 0 (no hate speech), a very high score could be obtained and this is the reason why it does not show sensitivity to detect 1 (hate speech) label.

[12] https://scikit-learn.org/stable/modules/generated/sklearn.utils.resample.html

[13] https://scikit-learn.org/stable/modules/feature_extraction.html#text-feature-extraction

[14] https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html

[15] Vectorization results in a sparse matrix of type `numpy.float64` but since not all classification models support SciPy sparse matrices, we convert it to dense when needed.

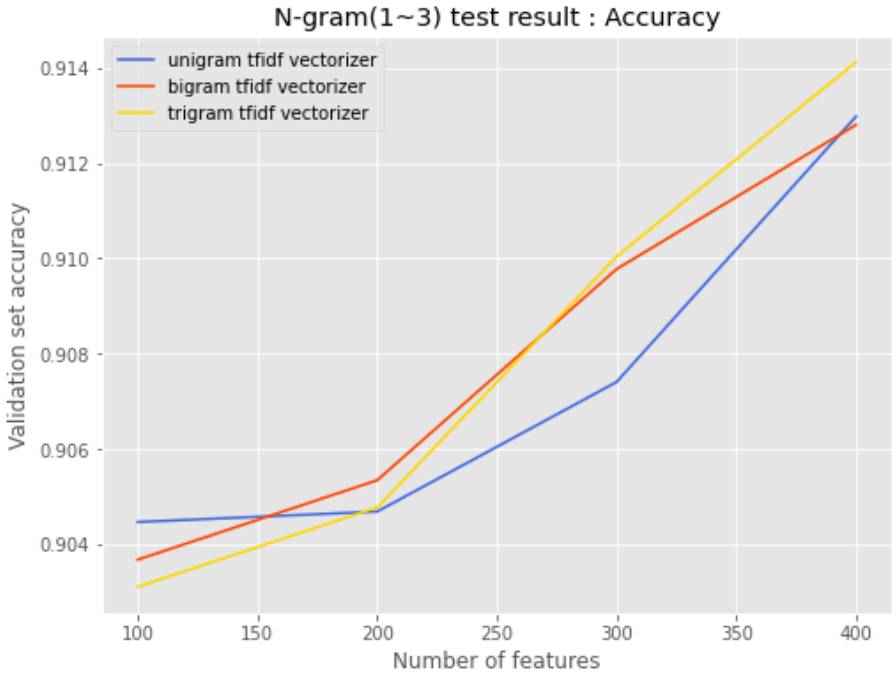

Figure 5: Accuracy check for feature extraction. Three vectorizers, extracting three different kind of n-grams, are put into pipelines together with logistic regression models which evaluate the related performance in terms of accuracy scores w.r.t. the range of extracted features.

`TfidfVectorizer`[16].

With uni-gram vector representation, we discard a lot of information about word order and grammar. This representation is commonly used in conjunction with models that do not take ordering into account. We fed the result of TF-IDF vectorization into three simple classifiers in order to perform a first experiment and evaluate the utility and suitability of this encoding approach for our data and task: a Naïve Bayes model, a Logistic Regression and a Random Forest. The results reported in Table 2 in the next section suggest that considering the relevance of certain words in each text is useful for classifying it as hateful or non-hateful, but still not enough. What is missing at this stage is an emphasis on the importance of word-word co-occurrences to extract meaning from text, which is a goal not achievable with bag-of-words approaches alone. In order to include this further element into our analysis we used neural embeddings, in particular GloVe embeddings, described in the subsection below.

4.4 Neural Embeddings

Word embeddings allows for the representation of words in a way that captures their meanings, semantic relationships and the contexts they are used in. Word embeddings are basically a form of word representation that bridges the human understanding of language to that of a machine. They have learned representations of text in an n-dimensional space where words that have the same meaning have a similar representation. Two similar words are represented by almost similar vectors that are very closely placed in a vector space. Thus when using word embeddings, all individual words are represented as real-valued vectors in a predefined vector space. Each word is mapped to one vector and the vector values are learned in a way that resembles a neural network. Word Embeddings are an improvement over traditional sparse representations used in simpler bag-of-word model encoding schemes where large sparse vectors were used to represent each word or to score each word within a vector to represent an entire vocabulary. These representations were sparse because the vocabularies were vast and a given word or document would be represented by a large vector comprised mostly of zero values. Instead, in an embedding, words are represented by dense

[16] https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

vectors where a vector represents the projection of the word into a continuous vector space. The position of a word within the vector space is learned from text and is based on the words that surround the word when it is used. The position of a word in the learned vector space is referred to as its embedding.

Word Embeddings overcome the limitations of TF-IDF approach, capturing and deriving semantic similarity between words from the co-occurrence matrix. To obtain a vector representation for words we used an unsupervised learning algorithm called GloVe (Global Vectors for Word Representation)[16]. GloVe method provide a suite of pre-trained word embeddings on their website released under a public domain license. Its embeddings relate to the probabilities that two words appear together. The package we used in this project is the Twitter pre-trained 100-dimensional word vector: it was trained on a dataset of 2 billion tweets (27 billion tokens) with a vocabulary of 1.2 million words.

In order to include the embeddings in our learning model, we exploit the `Embedding` layer offered by Keras[17]. It requires that the input data be integer encoded: this data preparation step was performed using the `Tokenizer API` also provided with Keras[18]. This class was fit on the training data by using the `fit_on_text()` method and then was used to convert text to sequences consistently by calling the `texts_to_sequences()` method. The class also provides access to the dictionary mapping of words to integers in a `word_index` attribute. After these steps, sequences were padded to be the same length[19] using the Keras `pad_sequences()` function, since the inputs for our neural model must have the same size.

We loaded the entire GloVe word embedding file into memory as a dictionary of word to embedding array, and we created a matrix of one embedding for each word in the training dataset. We did that by enumerating all unique words in the `Tokenizer.word_index` and locating the embedding weight vector from the loaded GloVe embedding. The result is a matrix of weights only for words seen during training.

4.5 Model building and evaluation

Once the setting of both semantic and non semantic text transformation approaches was concluded, we started experimenting on different classifiers and configurations to try to learn from them (and from a combination of the two). Since the task of this work is a supervised binary classification to determine whether a text entry contains or not hate speech, models and metrics have been chosen accordingly. In this subsection we describe the implementation detail, as well as the evaluation process, of each single models, analysing the preliminary results and evidence obtained.

Although during the evaluation phase we do not proper refer to *intrinsic* and *extrinsic* metrics and approaches [1, 9], we distinguish between:

- An evaluation approach that focuses on the performance of an NLP component on a defined subtask (e.g., prediction on TR and VAL sets of models that belongs to the experimental phase): at this stage we refer to the metrics listed below.
- The evaluation of the overall performance of the final objective (i.e., the complete application). Beside referring to learning curves produced by the released model, in section 5 we propose a visual evaluation tool as well as reflections on some of the results obtained.

The selected metrics for evaluating these classifiers are the most commonly used in NLP:

- Precision, which valuates the percentage of true positives identified given all positive cases. This formulation is particularly helpful when identifying positives is more important than the overall accuracy, e.g., if you want to be safe before categorising a text sample as containing hate speech. This formulation perfectly fits an hypothetical application in which the purpose of accurately detecting hate speech is more important than generating false alarms. The downside of this approach is that some label 1 can be missed.

$$Precision = \frac{TP}{TP + FP} \tag{1}$$

- Recall (or Sensitivity), indicating the percentage of true positives versus combined true and false positives. In an hypothetical applicative domain in which sharing text

[17] https://keras.io/layers/embeddings/#embedding
[18] https://keras.io/preprocessing/text/#tokenizer
[19] https://keras.io/preprocessing/sequence/#pad_sequences

containing hate speech is considered unacceptable by the regulations, you may want to automatically delete the account of the user who shared it: in this case, create a sensitivity for text containing hate speech using the Recall score could be useful. Rather than ignoring a text sample containing hate speech, we may have the option of suspending the accounts for which a wrong guess could have been made.

$$Recall = \frac{TP}{TP + FN} \tag{2}$$

- F1-score: used since both precision and recall are of interest to us then we use this score to combine them into a single metric (computing the harmonic mean of the two).

$$F1score = \frac{2TP}{2TP + FP + FN} \tag{3}$$

- Accuracy, which aim is to measure the proportion of correct predictions (both true positives and true negatives) among the total number of cases, and which is therefore typically used in instances where the output variable is categorical or discrete (e.g., classification task). We were able to safely include and use this score since we are working on a stable configuration of data (after having fixed the class unbalance).

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{4}$$

- Visual tools, such as confusion matrix[20], decision boundary plot[21] (using PCA[22]) and Area Under the Curve (AUC) with ROC curve plot[23] (since observations are balanced between each class).

A textual report including precision, recall, F1-score and accuracy has been automatically generated by feeding true and predicted labels for each model to an implemented function that exploits the Scikit Learn `classification_report`[24].

4.5.1 Experiments with BOW method

As anticipated before, we investigated the possibility of building a model that could learn and predict the presence of hate speech only with count-based vectorization. We used the following Scikit Learn models: a Naive Bayes classifier for multinomial models[25], a Logistic Regression classifier[26] and a Random Forest classifier[27].

Each model can efficiently handle sparse matrices, so is trained on the TR set (vectorized with TF-IDF approach): the performance of prediction on both TR and VAL sets are reported in Table 2.

It is easy to notice, observing the obtained scores, that even these simple models which use bag-of-words vectors are capable of discriminating the classes reasonably well[28]. Nevertheless, as previously anticipated, further observations about the limitations of this approaches can be extracted from the numerical results. Values obtained for precision, recall and F1 scores related to the `hate` category predicted on VAL set show that all classifiers are overfitting to class 0 to minimize the error: this is due to class imbalance in the data (few samples of class 1 in the set). Again for the `hate` label, but predicted on the TR set, all the models have good values for precision (more than 90% of of positive identifications were

[20] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html, https://scikit-learn.org/stable/modules/generated/sklearn.metrics.plot_confusion_matrix.html

[21] http://rasbt.github.io/mlxtend/api_subpackages/mlxtend.plotting/#plot_decision_regions

[22] https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

[23] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.plot_roc_curve.html

[24] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html

[25] https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html

[26] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

[27] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[28] as also noticed by Gibert et al. in [5]

Table 2: Performance comparison of three different classifiers on training and validation sets, based on TF-IDF approach.

Model	**Pred. on**	**Precision**		**Recall**		**F1-score**		**Acc**
		no_hate	*hate*	*no_hate*	*hate*	*no_hate*	*hate*	
Naive Bayes	TR	0.75	0.91	0.93	0.68	0.83	0.78	0.81
	VAL	0.93	0.66	0.93	0.67	0.93	0.67	0.88
Logistic Regression	TR	0.76	0.92	0.94	0.70	0.84	0.79	0.82
	VAL	0.93	0.71	0.94	0.68	0.94	0.70	0.90
Random Forest	TR	0.85	0.95	0.96	0.83	0.90	0.89	0.89
	VAL	0.93	0.74	0.95	0.66	0.94	0.70	0.90

actually correct) but not for recall: for this latter only the Random Forest classifier passes a satisfying 80% of actual positives identified correctly. The integration of these numerical results with the visual aids provided by Figures 10, 11 and 12 in Appendix A allows to have a more comprehensive overview for evaluating classifiers. Both confusion matrices and ROC curves are plotted w.r.t. VAL set predicted values and probabilities[29]. All the three can be referred to as "models with good skill" since they have high AUC; however, the amount of wrong predictions suggests that some improvements are needed. Due again to class imbalance issues, accuracy should not be taken so much in consideration for VAL set predictions, while for TR set there is still a good margin of improvement.

It was exactly this preliminary result obtained that led us not to completely discard the TF-IDF text transformation but to integrate it with more sophisticated techniques (semantic approaches, i.e., embeddings), and then fed it into a more complex learning model architecture.

4.5.2 Experiments with Embeddings

After having reserved an entire experimental phase to classification task models based only on BOW methods, as well as analysing their results and limitations, we performed an analogous investigation phase on models using only word embeddings representations. As emerged from the previous exploration, the performance of non-complex and non-order-based classifiers can be improved: we believe that, for the purposes of our final task, other types of learning models are more appropriate (i.e., deep learning paradigms that employs neural networks to automatically learn multi-layers of abstract features from raw data). The most popular network architectures used in this context are Convolutional Neural Network (CNN) and Recurrent Neural Networks (RNN). In the literature, CNN is well known as an effective network to act as 'feature extractors', whereas RNN is good for modelling orderly sequence learning problems[25].

Since we want to include the capability to learn long-distance dependencies within textual sequences, we opted for a LSTM which consist in a Gated Recurrent Neural Network, i.e. a more advanced version of RNNs that can preserve important information from earlier parts of the sequence and carry it forward. This feature allows the architecture to achieve excellent performance as it ensures the learning of linked but distant information within given sequential data (i.e., text). This feature is doubly exploited in this work since our model consists of a BiLSTM (Bidirectional LSTM) that therefore allows us to take advantage from it in both directions. Keras provides the following API for building bidirectional RNNs: the `keras.layers.Bidirectional`[30] wrapper. This was used together with the Keras LSTM layer[31] in a Sequential API setting[32] to build the network structure.

A simplified framework of this architecture is illustrated in Figure 6. We do not dwell too much on the description of this model (dropout layers, parameter tuning, and further practicalities) as it still represents a part of the experimental phase, and as its most important structural features and elements have been maintained in the final architecture and therefore will be described in Subsection 4.6.

[29] Scikit-Learn `predict()` and `predict_proba()`

[30] https://keras.io/api/layers/recurrent_layers/bidirectional/

[31] https://keras.io/api/layers/recurrent_layers/lstm/

[32] https://keras.io/api/models/sequential/

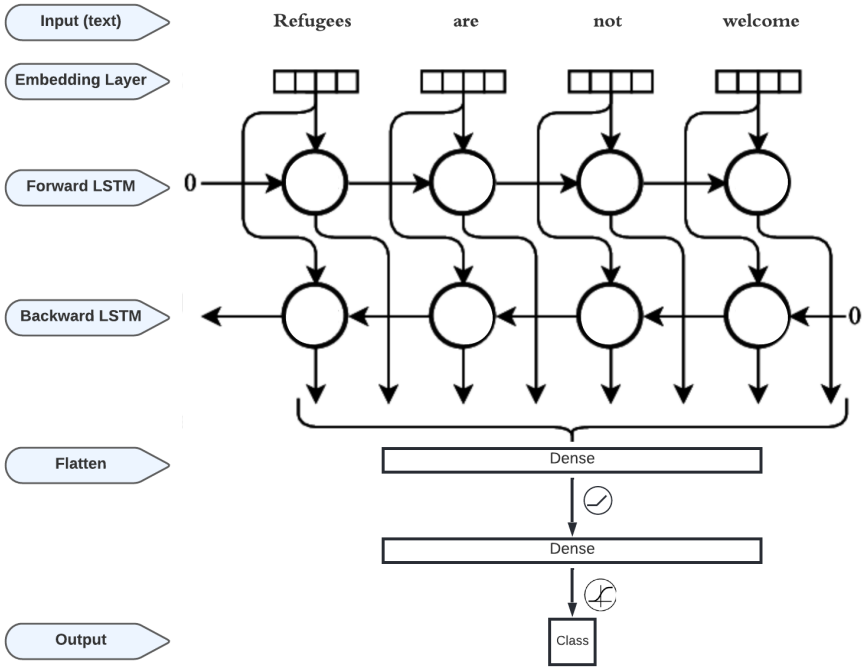

Figure 6: Architecture of a Bidirectional Long-Short Term Memory network for text classification with embeddings. The figure intends to represent just the 'skeleton' of the model, therefore it does not include all the layers and the structural components of the actually employed architecture.

In order to evaluate this network we plotted the accuracy and loss curves (see Figure 13 in Appendix B) for both training and validation in order to assess the learning capability outcome. The smooth behaviour and the numerical results obtained for accuracy were satisfying in terms of performance: however, the loss was still pretty high, maybe suggesting an underfitting behaviour of the learning model.

4.6 Final model: integration of both techniques

As a final solution to overcome the limitations encountered during both phases, but at the same time to maintain their strengths, we opted for a BiLSTM classification model that included a way of 'concatenating' an additional input provided by the BOW approach, and so that could include the features extracted with a count-based approach. Therefore, combining this latter with the ability of architecture to consider long-distance dependencies across sentences in the prediction process, the model includes all the elements to be sufficiently complex to overcome the underfitting limitations and to achieve excellent performance in the final task.

Figure 7 illustrates the structure of the final BiLSTM model, built with Keras Functional API[33].

The pre-trained embedding layer maps each text record (sequence) into a real vector domain, and exploits the knowledge obtained with GloVe as described in Subsection 4.4. It is obviously imposed to be non-trainable. Its output is fed to a Lambda layer, which averages the sentence's word vectors, and then TF-IDF vectors are included in the model with a Concatenate layer. The concatenated outputs are then reshaped into a 3-dimensional tensor (the Reshape layer converts inputs into the target shape) in order to be fed to a Bidirectional LSTM. Dense layers flatten the output space, which is passed to drop-out layers with a rate of 0.2, the purpose of which is to regularise learning to avoid overfitting by randomly shutting down 20% of the neurons. Intuitively, this can be thought of as randomly removing a word in sentences and forcing the classification not to rely on any individual words [11]. The rectified linear unit function is used for activation. Since we have only 2 classes (binary classification), our model should output a single probability score, and so, in the last layer, a sigmoid function is used to return as output the predicted class.

After having compiled and trained the model, it is saved into a h5 file, so that it can be conveniently load at a later time (including architecture, weights, training configuration and state of the optimizer).

We use the binary cross entropy loss function and the Adam optimiser to train the model. The first is perfectly suitable for our classification task, while the second is designed to improve the classic stochastic gradient descent (SGD) optimiser and in theory combines the advantages of two other common extensions of SGD (AdaGrad and RMSProp) [12].

In order to figure out the best configuration of values for hyperparameters, and so to find the best model in terms of complexity, flexibility and generalization capability, we adopted random search technique with 5-fold cross-validation. Our model have been tested on an internal validation set, while the previously defined VAL was used for the final evaluation of the chosen model.

[33] https://keras.io/guides/functional_api/

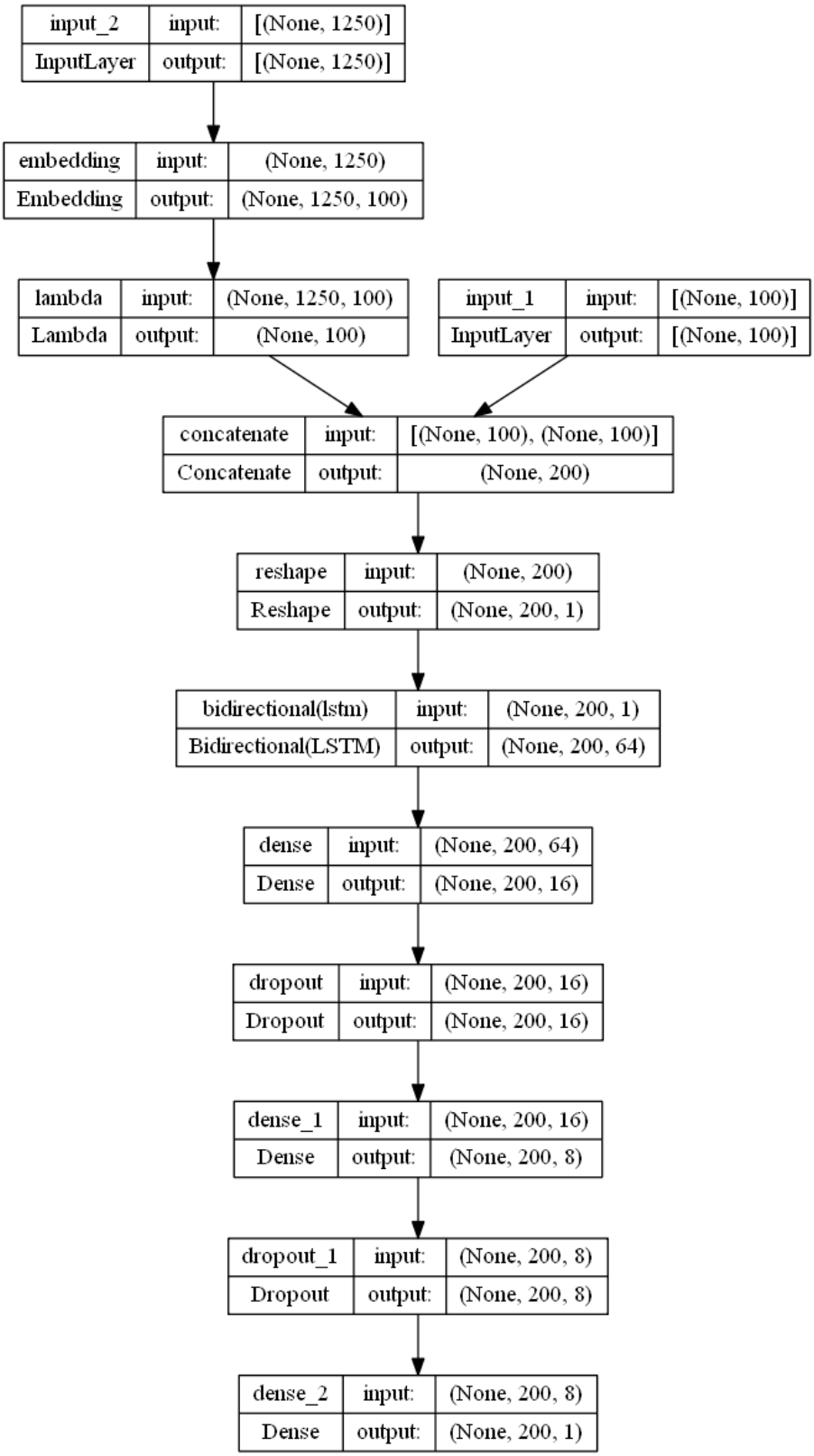

Figure 7: Topology of the final model. The illustration was automatically generated with TensorFlow `plot_model()`[35]

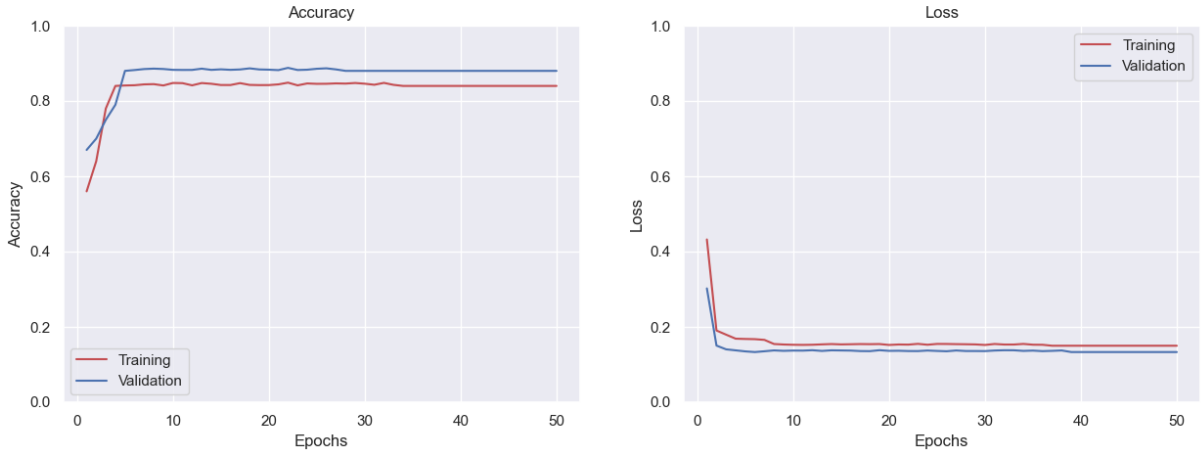

Figure 8: Learning curves of the final model.

5 Results and discussion

The results obtained by the final model both in terms of accuracy and error minimisation can be fully appreciated in the learning curves reported in Figure 8. The loss is significantly decreased and the respective curve does not suggest any overfitting or underfitting behaviour. The obtained scores of accuracy also contribute to believe that a total improvement has been obtained with this model w.r.t. the previous experimental phases.

As further check of the performance outcomes, predicted values on VAL set were used to automatically generate a `classification_report`[36] that returned precision and recall values respectively of 0.82 and 0.78 for the hate class.

5.1 Final test phase

The TS set created at the beginning of the project and kept independent and separate until this phase was exploited for an additional final prediction on new data. Since it is not easy to interpret the results, having no labels available to assess performance outcomes, we propose a visual tool for the evaluation, as well as a discussion on predicted classes.

5.2 Beyond the black box: explainable models.

Explainability and transparency have become, in recent years, more and more important for humans in order to understand decisions and predictions made by ML models. Our work wants to provide a straightforward, though not exhaustive, contribution in this sense. We implemented a function able to built a word cloud in order to visualise the most frequent words appearing in text for each of the two predicted class. A word cloud represents word usage in a document by resizing individual words proportionally to its frequency, and then presenting them in random arrangement.

From Figure 9 the prevalence of positive (or neutral) and negative keywords used, respectively, in non-offensive and offensive texts is evident. A clear semantic distinction emerges from the two plots: on the left we can find tokens (words) that are representative of non-hateful text, while on the right the most frequent terms appearing in hate speech. This very simple tool can be useful also for non-experts and for readers having no computational skills to overcome the conception of deep learning as a black-box, since it offers an intuitive "explanation" of how tweets, comments and forum posts have been classified.

However, assigning the label on the basis of offensive words alone would not be enough for such a complex model. Other types of semantic relationships have indeed been identified by our model which help classify text containing hate speech as such, despite not containing

[36] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html

Figure 9: Wordcloud for representative tokens in non-hateful (left) and hateful (right) text.

specific discriminatory terms. This probes the power and reliability of our model for the prediction on the final set test. We include here below a **random sample of original texts classified as hateful**: although there is no assigned label with which we can make a numerical comparison on the correctness of the classifications, we believe that in this sample all the texts have been correctly classified as containing hate speech, although some of them do not contain cursing or vulgar language. Hence, the deep learning model was able to understand the semantics of the sentence.

'Sitting at the bar at work watching Walking Dead and all these fucking wetbacks decide to come in turn on their spic music. Jesus Christ'

'I find it FUNNY ASF how people will try and tell you how you should live YOUR life. Like bitch! Stay in my shadows thank you lol 128514;128514;'

'This nigga on the train treating his bitch over the phone lmao'

'Check your slutty fat girl friends instagram pics.See if she don't have five niggaz in every pic thirsting for fat girl pussy."

'every time i try to quit smoking, some dumb bitch always gotta be fucking annoying and try my patience lmfao'

'religion. Most moaists are homosexual. so is k klos'

'shut down the mexican border withought looking bad.'

'This idiot can't even use proper grammar when hijacking a page."

'..... added at by :Somebody using the same IP as you used to ask Who the hell is ""Ileana Ros-Lehtinen"" any way? added a stupid comment to her article. If youŕe not the author and donť want to be confused with the author, simply acquire a username and use it.'

'Bill clinton is al-qaeda, its all about getting power. serbs were right to kill the terrorists. serbs are not communists.'

'Obviously you didnť read the entire article or look at its links. Iḿ not going to continue this silly argument. Your reasoning is based entirely on ""what people look like"" and does not even qualify as original research, the only person you cite is yourself and ""yourself"" only knows out to make baseless and ignorant claims.'

'Perfect example of a psychopath'

'"To see victory only when it is within the ken of the common herd is not the acme of excellence."thatś brilliant. So being a loser actually proves that you are really the acme of brilliance? I am sure millions of losers the world over will be delighted at the news.'

For transparency and as a counter-proof, we also report some texts classified as not containing hate speech despite containing trigger terms, that could have easily resulted in false positives:

'I wanna apologize for being such a bitch the other day.. But the way my emotions

```
are set up..'

   '" Please dont call another editor ""abusive little shit"" it is not appropriate
and is a personal attack. Personal attacks work to the detriment of wikipedia
and only produce acrimony. Please refrain in future. That is still no excuse.
You should avoid such language. "'
```

6 Conclusion and future work

The propagation of hate speech on online sources has been increasing significantly in recent years. Riding the wave of Deep Learning contributions within this context, we presented a Gated Recurrent Neural Network (specifically, a Bidirectional LSTM) for Sentiment Analysis. The ultimate goal of the model was a bi-polarity classification of text into hateful and non-hateful categories. Sequences of text belongs to a corpus resulting from the integration of three different online sources and genres: tweets, forum posts and Wikipedia comments. The major novelty introduced in this work was the inclusion in the topology of both BOW and neural embedding approaches, resulting into a complex model able to learn abstract feature representations from input data through its multiple stacked layers for the classification of hate speech. We presented an exhaustive workflow with all the experiments made through the process that led us to move from good to optimal results in terms of all the metrics employed. All the various phases of our project, from pre-processing to evaluation, follow the best practices of Data Mining, NLP and Machine Learning domains. We also provided some reflections on obtained results, as well as intuitive visualisation tools in order to address the model explainability.

We will explore future work in the following directions: 1) investigate the possibility of including further modification to the topology (e.g., considering convolutional layers); 2) implementing a tool able to predict a label for a text given in input by the user.

References

[1] Y. T. Cao, Y. Pruksachatkun, K.-W. Chang, R. Gupta, V. Kumar, J. Dhamala, and A. Galstyan. On the intrinsic and extrinsic fairness evaluation metrics for contextualized language representations. In *ACL 2022*, 2022.

[2] N. Chawla, N. Japkowicz, and A. Kołcz. Editorial: Special issue on learning from imbalanced data sets. *SIGKDD Explorations*, 6:1–6, 06 2004.

[3] D. Crystal. *Internet Linguistics: A Student Guide*. Taylor & Francis, 2011.

[4] T. Davidson, D. Warmsley, M. Macy, and I. Weber. Automated hate speech detection and the problem of offensive language. In *Proceedings of the 11th International AAAI Conference on Weblogs and Social Media*, ICWSM '17, 2017.

[5] O. de Gibert, N. Perez, A. García-Pablos, and M. Cuadros. Hate Speech Dataset from a White Supremacy Forum. In *Proceedings of the 2nd Workshop on Abusive Language Online (ALW2)*, pages 11–20, Brussels, Belgium, Oct. 2018. Association for Computational Linguistics.

[6] A. Durant and M. Lambrou. *Language and Media: A Resource Book for Students*. Routledge English language introductions. Routledge, 2009.

[7] P. Fortuna, J. Rocha da Silva, J. Soler-Company, L. Wanner, and S. Nunes. A hierarchically-labeled Portuguese hate speech dataset. In *Proceedings of the Third Workshop on Abusive Language Online*, pages 94–104, Florence, Italy, Aug. 2019. Association for Computational Linguistics.

[8] A. Goddard and B. Geesin. *Language and Technology*. Intertext (London, England). Routledge, 2016.

[9] T. Hailu, J.-q. Yu, and T. Fantaye. Intrinsic and extrinsic automatic evaluation strategies for paraphrase generation systems. *Journal of Computer and Communications*, 08:1–16, 01 2020.

[10] D. M. E.-D. M. Hussein. A survey on sentiment analysis challenges. *Journal of King Saud University - Engineering Sciences*, 30(4):330–338, 2018.

[11] M. Kamyab, G. Liu, and M. Adjeisah. Attention-based cnn and bilstm model based on tf-idf and glove word embedding for sentiment analysis. *Applied Sciences*, 11(23), 2021.

[12] D. P. Kingma. &ba j.(2014). adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2015.

[13] C. X. Ling and C. Li. Data mining for direct marketing: Problems and solutions. In *KDD*, 1998.

[14] G. Myers. *The Discourse of Blogs and Wikis*. Bloomsbury Discourse. Bloomsbury Academic, 2010.

[15] C. Nobata, J. Tetreault, A. Thomas, Y. Mehdad, and Y. Chang. Abusive language detection in online user content. In *Proceedings of the 25th International Conference on World Wide Web*, WWW '16, page 145–153, Republic and Canton of Geneva, CHE, 2016. International World Wide Web Conferences Steering Committee.

[16] J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[17] R. Rani and D. K. Lobiyal. A weighted word embedding based approach for extractive text summarization. *Expert Systems with Applications*, 186:115867, 2021.

[18] M. Santos, J. Soares, P. Henriques Abreu, H. Araujo, and J. Santos. Cross-validation for imbalanced datasets: Avoiding overoptimistic and overfitting approaches. *IEEE Computational Intelligence Magazine*, 13:59–76, 10 2018.

[19] Y. Wang, M. Huang, X. Zhu, and L. Zhao. Attention-based LSTM for aspect-level sentiment classification. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 606–615, Austin, Texas, Nov. 2016. Association for Computational Linguistics.

[20] W. Warner and J. Hirschberg. Detecting hate speech on the world wide web. pages 19–26, 06 2012.

[21] Z. Waseem, T. Davidson, D. Warmsley, and I. Weber. Understanding abuse: A typology of abusive language detection subtasks. In *Proceedings of the First Workshop on Abusive Language Online*, pages 78–84, Vancouver, BC, Canada, Aug. 2017. Association for Computational Linguistics.

[22] J. Wei, J. Liao, Z. Yang, S. Wang, and Q. Zhao. Bilstm with multi-polarity orthogonal attention for implicit sentiment analysis. *Neurocomputing*, 383:165–173, 2020.

[23] Wikipedia. Storm front — wikipedia, the free encyclopedia. [Online; Jul 13, 2022].

[24] Z. Yang, D. Yang, C. Dyer, X. He, A. Smola, and E. Hovy. Hierarchical attention networks for document classification. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1480–1489, San Diego, California, June 2016. Association for Computational Linguistics.

[25] Z. Zhang, D. Robinson, and J. Tepper. Detecting hate speech on twitter using a convolution-gru based deep neural network. In A. Gangemi, R. Navigli, M.-E. Vidal, P. Hitzler, R. Troncy, L. Hollink, A. Tordai, and M. Alam, editors, *The Semantic Web*, pages 745–760, Cham, 2018. Springer International Publishing.

A Evaluation of models (BOW-based approach)

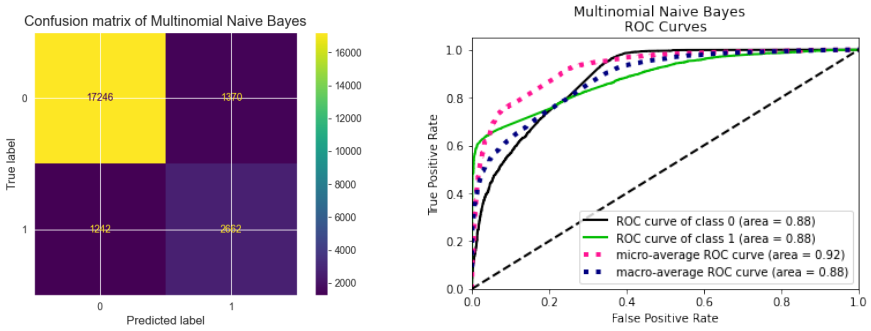

Figure 10: Naive Bayes classifier: confusion matrix and ROC curve.

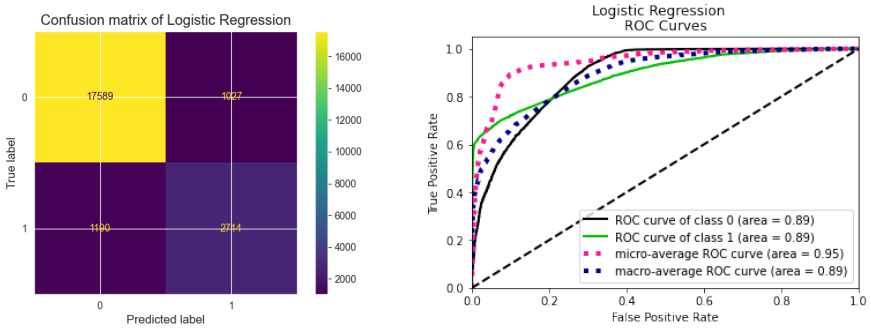

Figure 11: Logistic Regression classifier: confusion matrix and ROC curve.

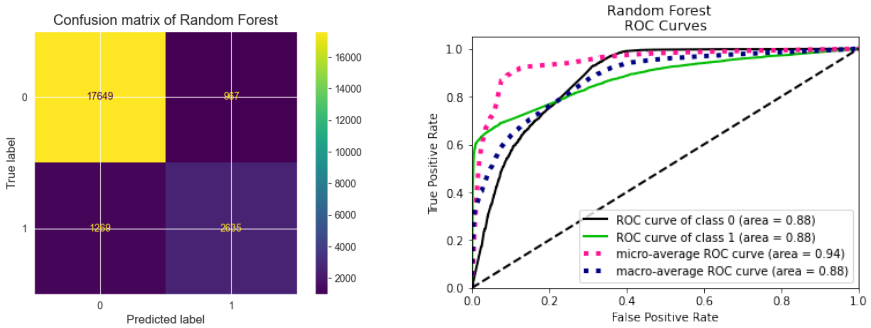

Figure 12: Random Forest classifier: confusion matrix and ROC curve.

B Experimental phase on BiLSTM (with word embeddings)

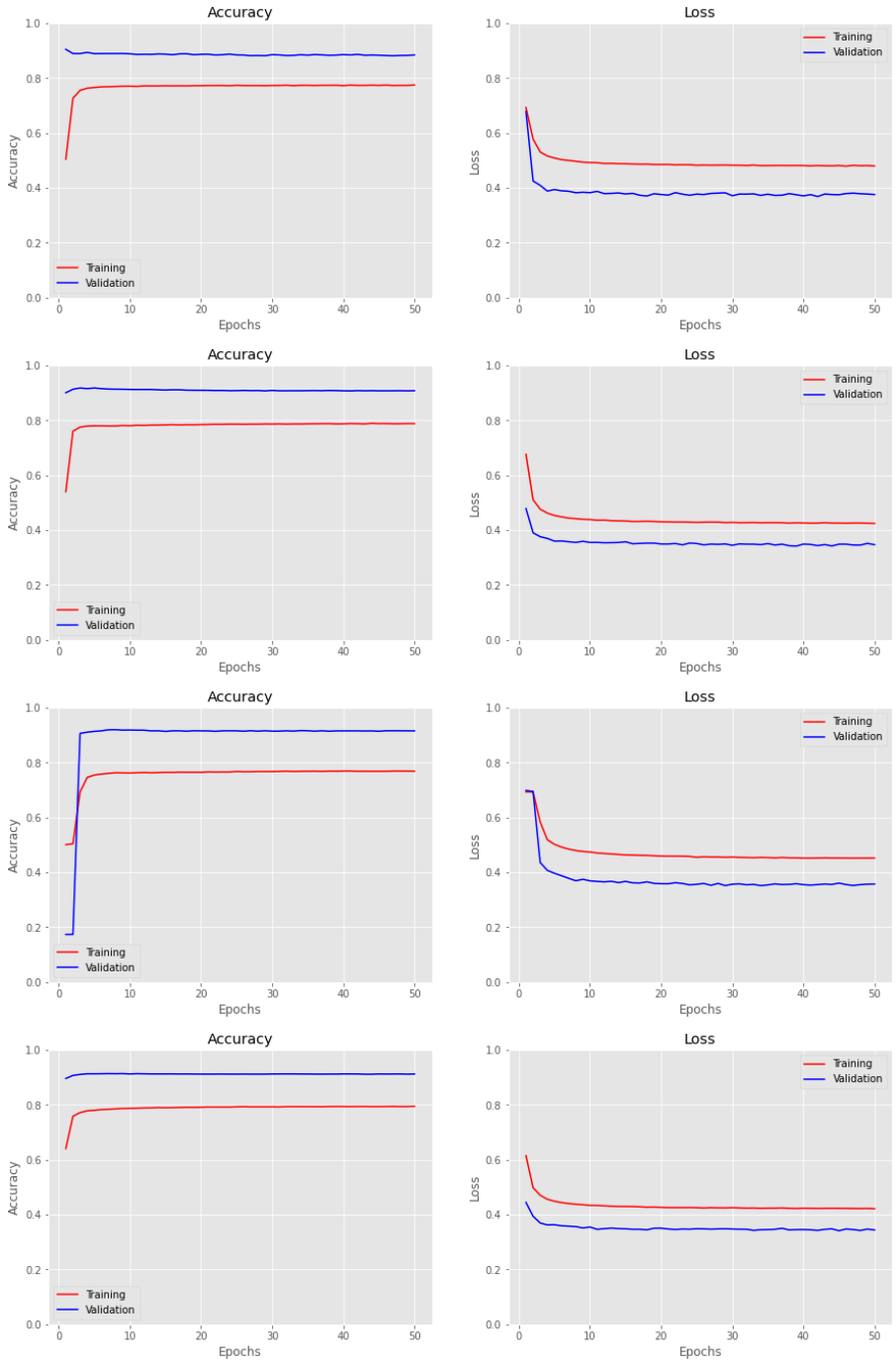

Figure 13: Loss and accuracy curves of BiLSTM (Keras Sequential API, without 'Concatenate' layer.