Detecting hate and offensive content on social media using Text Mining and Sentiment Analysis techniques

Silvia Fratarcangeli - 957178
Filippo Menegatti - 942288

Contents

1 Introduction

The aim of this work is to highlight and identify the key differences between hate and offensive speech lexicon in everyday languageˆ[The analysis is conducted referring to the English vocabulary].

In the last decades, the spread of social media has affected and increased our exposure to online "hateful" episodes, that most of the times target specific social groups or minorities.

Many recent works in sentiment analysis have addressed the issue of clarifying the difference between hate speech lexicon and the offensive one.

Despite the difficulties in providing two clear-cuts' definitions, hate speech can be defined as *a language that is used to express hatred towards a targeted group or is intended to be derogatory, to humiliate, or to insult the members of the group* [1].

Indeed, both lexicons are characterized by highly offensive and violent terms, usually inherited from pre-existing slangs born from the underground American Culture. However, the offensive lexicon could be seen as target-free whereas hateful web contents share common dynamics and are usually related to matters like race, color, ethnicity, gender, sexual orientation, nationality, religion, etc. [2].

2 Data

The data set used in this analysis can be found in the following github repository [0], together with a brief description of how the data have been retrieved and categorized in the 3 classes.
As for the hate speech lexicon, words and phrases were first taken from Hatebase.org – an online platform aimed at detecting and analyzing hate speech. Then, using the Twitter API to download tweets containing those terms, the final collection resulted in a sample of 25k tweets, manually assigned by CrowdFlower(CF) workers to one of the 3 categories:

- 0 - Hate Speech
- 1 - Offensive Speech
- 2 - Neutral

Note that, in order to proceed in our analysis, we subsampled the original data set in order to allow the computation and training of the models. More specifically, we implemented the *SMOTE algorithm* to downsize the tweets labelled as 1 - offensive speech from 14396 to 8638 (60% of the initial set) and use this sampling as input into the scikit-learn `train_test_split()` function.

2.1 Preprocessing and text cleaning

In order to correctly work with text data, a data preprocessing step is indeed needed. The methods we adopted to clean our corpus follow the most common practices in text mining and sentiment analysis:

- Lowercase and remove the retweet ("rt"), numbers and symbols from our initial corpora, which was subsequently tokenized[1];
- Application POS tagging to retrieve information about the syntactical meaning of each word. At this point:
  - Application of stemming (Porter Stemmer) and lemmatization (WordNet Lemmatizer) methods to preprocess data. The stemming has been finally choosen as suggested in [1], since lemmatization did not bring any improvement on the overall performance (table .. shows how results do not change

[1] the stopwords have not been removed in order to maintain the structure of the sentences

with respect to the methods applied);

  - Transformation of the data (tweets and pos tags) using the TFIDF function using unigrams, bigrams and trigrams;

- Creation of unigrams, bigrams and trigrams and multiply them by their weights of the TD.IDF matrix, as suggested in [1].
- Final concatenation of the two tfidf matrices to get our feature matrix.

3 Proposed approach and results

The models evaluated in this works are *Random Forest*, *Multinomial Naïve Bayes* and *XGBoost*. The tuning of the parameters for these models was performed using the `optuna` library.
Finally, we have also trained a *Sparse Tensor Classifier* algorithm and used its `.explain()` method to identify the most relevant words for each target category.

3.1 Classification Evaluation Metrics

The relevant metrics [3]that we considered in order to evaluate the performance of our models as a "good fit" were:

- **Precision** – it is the ratio between *True Positive* (TP) and *Total Positives* (TP + FP), where TP represents all the cases in which for instance an offensive tweet was correctly classified as offensive. When a tweet contains offensive language, our model will be correct result% of the times.

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

- **Recall** – it measures how much our model does correctly classify True Positives with respect to the cardinality of the class to predict. Recall also gives a measure of how accurately our model is able to identify the relevant data. We refer to it as Sensitivity or True Positive Rate.

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

- **Accuracy** – it is the ratio of the total number of correct predictions and the total number of predictions (cardinality of the test set).

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

Depending on which is the aim of our model, and how we would like to interpret our result, we could end up choosing models with high recall to ones with high precision, or vice versa.
If instead we would like to have a trade-off between the two, we should also consider other metrics:

- **F1-Score** – it is the harmonic mean between Precision and Recall. In general, a good score of F1 would imply a good Precision and a good Recall value as well.

$$F1Score = \frac{2 * (precision * recall)}{Precision + Recall}$$

Note that for each of the metrics presented in this section, the model was evaluated taking into account the weighted average of the single scores among the 3 classes.

3.2 Random Forest

The best performance in terms of accuracy, f1-score, recall and precision we got it from a random forest optimized using the following methods and parameters, highlighted in the table:

		Tuning Parameter				Evaluation Metrics			
Preprocessing	N-grams_range	Param1	Param2	Param3	Param4	Accuracy	Recall	Precision	F1-Score (weighted)
PorterStemmer	**(1,3)**	criterion: entropy	bootstrap: False	max_depth: 35	n_estimator: 300	**0.8584**	**0.86**	**0.88**	**0.865**
Lemmatization	**(1,3)**	criterion: entropy	bootstrap: False	max_depth: 22	n_estimator: 300	0.8602	0.86	0.88	0.8645

Figure 1: Random Forest Summary Performance

Although, we got **86% accuracy** and an **F1-Score of 0.865** which are nearly close to the one obtained in [2], by printing the confusion matrix we better get an idea of how the classification task was conducted:

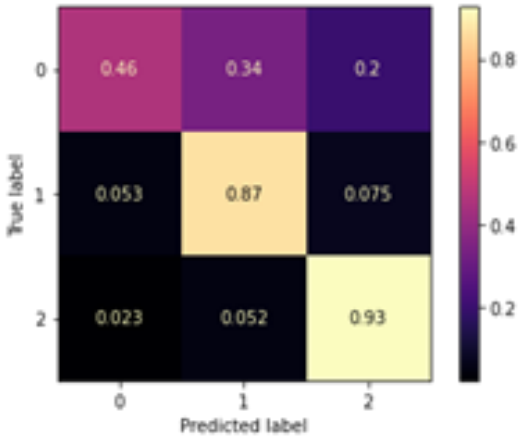

Figure 2: Confusion Matrix RF Best Model

While performing well in detecting offensive (1) and neutral speech (2), the major

misclassification occurred on the hate (0) class with only 46% of tweets correctly predicted, with the precision and recall values for this class being respectively **0.37** and **0.46**. This undoubtedly reflects the fact that human coders also found difficulties in correctly differentiating between hateful and offensive content, and that our model is biased towards classifying tweets as less hateful than human coders as also appeared in [1]. Further insights about this evidence will be given later in this paper.

3.3 Multinomial Naïve Bayes

The best performance we obtained from a Multinomial Naïve Bayes algorithm is summarized in the following table:

			Tuning Parameter				Evaluation Metrics			
Model	Preprocessing	N-grams_range	Param1	Param2	Param3	Param4	Accuracy	Recall	Precision	F1-Score (weighted)
Naïve Bayes	PorterStemmer	**(1,3)**	alpha: 0.01016	-	-	-	**0.8044**	**0.80**	**0.80**	**0.8042**
	Lemmatization	**(1,3)**	alpha:0.01051	-	-	-	0.8010	0.80	0.80	0.7987

Figure 3: Multinomial Naïve Bayes Summary Performance

The alpha parameter we set was tuned with the optuna library, and the scores we got for accuracy, F1, recall and precision are slightly below the ones of the previous model.

On the contrary, by printing the confusion matrix for this model we got something different with respect to what already analyzed previously:

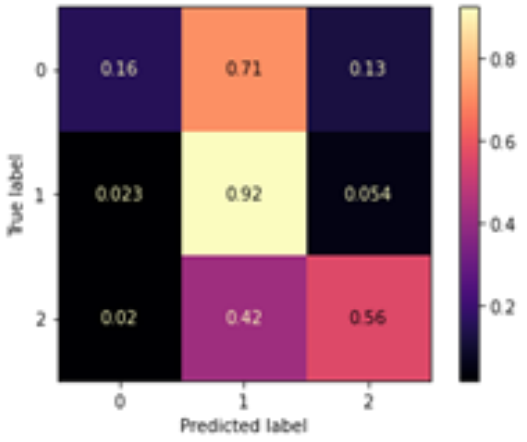

Figure 4: Confusion Matrix MNB Best Model

Not only the Multinomial Naïve Bayes does not seem to capture the difference between

hateful and offensive speech, with only 16% of the 0 class being correctly classified again reflecting how our model is biased towards classifying hateful speech as offensive (recall and precision are respectively 0.31 and 0.16). Moreover, it seems that some misclassification also occurred with neutral lexicon, with 42% of the points being misclassified as offensive content and reflecting how our model is classifying some originally neutral content as being offensive.

The explainability method that we called in the next section will give a more intuitive outlook of these results.

3.4 XGBoost

XGBoostis a decision-tree-based ensemble Machine Learning algorithm that uses a gradient boosting framework, where the gradient boosting employs gradient descent algorithms to minimize errors in sequential models [4].

Optimized Gradient Boosting algorithm through parallel processing, tree-pruning, handling missing values and regularization to avoid overfitting\bias.

The parameters we set in both models reported in the table below were set performing the hyperparameters optimization with the procedure we have seen in the previous analyses:

			Tuning Parameter				Evaluation Metrics			
Model	Preprocessing	N-grams_range	Param1	Param2	Param3	Param4	Accuracy	Recall	Precision	F1-Score (weighted)
XGBoost	PorterStemmer	**(1,3)**	learning rate: 0.03652	-	max_depth: 8	n_estimator: 100	**0.8863**	**0.89**	**0.89**	**0.8841**
	Lemmatization	**(1,3)**	learning rate: 0.24738	-	max_depth: 4	n_estimator: 100	0.8918	0.89	0.89	0.89

Figure 5: XGBoost Summary Performance

The scorings obtained are the highest, despite the impossibility to perform a deep tuning due to hardware limits.

The classification of the 1 and 2 classes returned a value of 0.92 and 0.96 respectively, while the hate class achieved only a 0.35. The 46% of the 0 class has been "misclassified" as offensive. The other parameters are satisfactory.

3.5 Sparse Tensor Classifier

In this section we report the main results we obtained by training a *Sparse Tensor Classifier* (STC) [5], a supervised classification algorithm for categorical data.

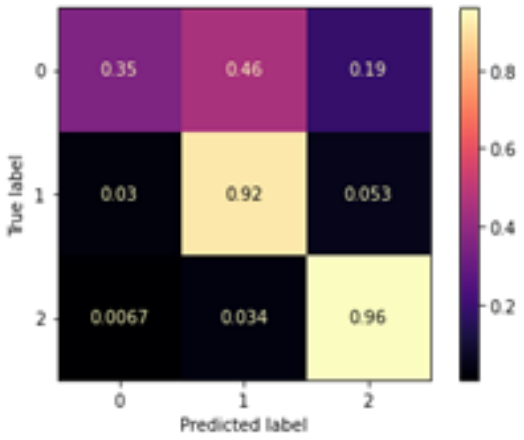

Figure 6: Confusion Matrix XGB Best Model

Unlike the other classifiers, the input data required by the STC must be of the form of a pandas.DataFrame or a JSON structured, and must be converted to string. We opted for the second option.

The main results we obtained are summarized in the following table:

			Tuning Parameter				Evaluation Metrics			
Model	Preprocessing	N-grams_range	Param1	Param2	Param3	Param4	Accuracy	Recall	Precision	F1-Score (weighted)
SparseTensorClassifier	PorterStemmer	(1,1)					0.8578	0.8578	0.8971	0.8727

Figure 7: Sparse Tensor Classifier Summary Performance

No tuning parameters was needed here, but the results obtained were satisfying and not too far away from what we have seen before. Better still, for precision and F1 we registered two of the highest scores, respectively of 0.90 and 0.87.

3.6 Explainability

Looking at the results we obtained, we got evidence of the fact that correctly classifying hate and offensive lexicon is not an easy task.

Using STC explain method, we deep-dive into our results to better understand which words are most significant for each class vocabulary. A score is assigned to each term to quantify its significance within the belonging class.

We sorted and visualized our result with wordcloud and try to draw some conclusion from it.

In the third column we also reported the classification provided by Hatebase.org in order to check its coherence.

Table 1: Top 10 words – **hate** wordcloud

Target	Features	Score	Hatebase.org
0	faggot	0.154332	extremely offensive
0	nigger	0.115234	extremely offensive
0	fag	0.104226	extremely offensive
0	bitch	0.082665	mildly offensive
0	nigga	0.056760	highly offensive
0	white	0.051456	extremely offensive
0	queer	0.047953	mildly offensive
0	wetback	0.043753	extremely offensive
0	fucking	0.043495	na
0	hoe	0.041326	highly offensive

Figure 8: Hate Wordcloud

Table 2: Top 10 words – **offensive** wordcloud

Target	Features	Score	Hatebase.org
1	bitch	0.160460	mildly offensive (sexist)
1	hoe	0.081475	highly offensive (sexist)
1	pussy	0.075720	moderately offensive (sexist)
1	nigga	0.042769	extremely offensive
1	faggot	0.039784	extremely offensive
1	fuck	0.035846	na
1	shit	0.034416	na
1	as	0.031395	na
1	fag	0.031275	extremely offensive
1	nigger	0.029754	extremely offensive

Figure 9: Offensive Wordcloud

By looking at those charts, there is clear evidence of the fact that it is also not easy for human coders to differentiate between hate and offensive speech. Many terms are shared by both classes but, despite that, it is still true that homophobic and racist slurs are correctly classified as hate speech – in accordance with the definition provided in the introduction.

By contrast, terms like b*tch, h*e and p*ssy which are extremely sexist and addressed to a specific group were nonetheless classified by human coders as offensive, thus being considered less hateful. Note that from Hatebase.org, the categorization goes from mildly to moderately offensive – clearly, there are some evident social biases which are reflected in the results we obtain as in [1].

Table 3: Top 10 words – **Neutral** wordcloud

Target	Features	Score	Hatebase.org
2	bird	0.105393	mildly offensive
2	yankee	0.087473	highly offensive
2	charlie	0.085108	extremely offensive
2	yellow	0.065782	extremely offensive
2	oreo	0.062158	extremely offensive
2	brownie	0.061109	extremely offensive
2	trash	0.046671	moderately offensive
2	mock	0.037417	highly offensive
2	sole	0.036769	extremely offensive
2	flappy	0.036643	na

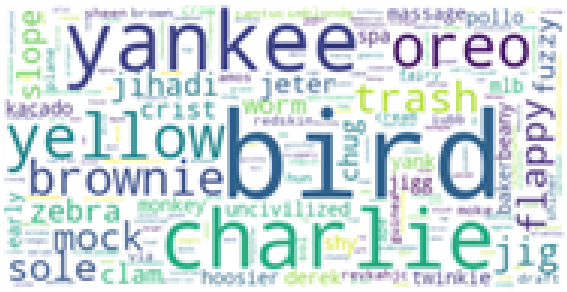

Figure 10: Neutral Wordcloud

On the other hand, the relevant terms identified for the neutral class were definitely misleading. One reason for that, could be that the context in which some terms are used is not always taken into account when carrying out these kinds of analysis.

4 Conclusion

In this paper we tried to build a model that could accurately detect hate and offensive speech on a labelled dataset of originally 25k tweets.

In the data cleaning and preprocessing phase, we applied the best-known methods in text mining and sentiment analysis, such as Porter Stemmer\WordNet Lemmatizer, tokenization, POS tagging, and form unigrams, bigrams and trigrams to fill our TD.IDF matrix.

The best performing model we obtained with a XGBoost algorithm, has an overall precision, recall, accuracy and F1-score of 0.89. The precision for the classification of hateful content is one of the highest obtained, with a score of 0.47, meaning that our model was more sensible to hateful content than human coders were. Indeed, the crucial role played by social biases is evidence of this misclassification as we also found out when analyzing the most relevant terms for each category.

Further improvements to this and related works could aim at including the context and circumstances in which hateful content arise in order to refine the overall classification.

5 References

[0] https://github.com/t-davidson/hate-speech-and-offensive-language

[1] *Automated Hate Speech Detection and the Problem of Offensive Language*, Thomas Davidson, Dana Warmsley, Michael Macy, Ingmar Weber

[2] *A Survey on Hate Speech Detection using Natural Language Processing*, Anna Schmidt, Michael Wiegand Spoken Language Systems Saarland University

[3] https://www.analyticsvidhya.com/blog/2020/09/precision-recall-machine-learning/

[4] https://towardsdatascience.com/https-medium-com-vishalmorde-xgboost-algorithm-long-she-may-rein-edd9f99be63d)

[5] <https://sparsetensorclassifier.org>