

Offensive Tweets Detection using Machine Learning

Diogo Ferreira, Luís Leira

Abstract—In recent years, verbal aggression on the internet has grown exponentially, due to the higher reach of social media platforms and broad use of smartphones. Twitter has implemented a "safe search" feature to detect and remove offensive tweets and delete or suspend user accounts that successively post offensive tweets. In this report, we present an approach with document pre-processing and testing with classical machine learning classifiers to detect offensive tweets. It can be achieved an accuracy of 95.6% in our dataset. Our results outperform the current offensive tweet classifiers in the known papers.

I. INTRODUCTION

IN 2017, Twitter implemented "safe search" feature to detect and remove offensive tweets from searches with "safe search" enabled [1]. This feature is also useful when the Twitter widget is embedded in a reputed website and it is of great importance to show only adequate tweets. Currently, Twitter also uses this information to suspend or delete accounts that post successive aggressive tweets. In [2], the authors propose a similar approach using features generated by the unsupervised learning algorithm Doc2Vec, obtaining a final accuracy of 88.465%. In [3], the authors use a dataset with 1.6M tweets gathered over three months and can achieve an accuracy of around 91% in the classification of offensive tweets. In this report, it will be explained our approach to the problem of creating a classifier to differentiate between offensive and non-offensive tweets. We believe that it is possible to achieve better results with a more adequate pre-processing of the words in the tokens and classic machine learning classifiers well-tuned.

II. DATASET

The chosen dataset to make the analysis is part of the dataset used in the paper [2] and it can be found in Kaggle website¹. It contains 20001 tweets human labelled as offensive and non-offensive tweets. 7822 tweets were labelled as offensive, while 12179 tweets were labelled as non-offensive.

The charts in figure 1 and figure 2 show the most frequent tokens in the non-offensive and offensive tweets, respectively. It is possible to see that the most frequent tokens are the same in both datasets, which makes it hard for the model to classify tweets into the categories correctly. Word proximity must be taken into account when considering the features to feed the machine learning algorithm, to smooth the effect of the same frequent words being in both datasets and to try to higher the general accuracy of the model. The charts in figure 3 and figure 4 show the most frequent tokens grouped sequentially in pairs (bi-gram tokens). It can be seen the similarity between the most frequent bi-gram tokens in both charts.

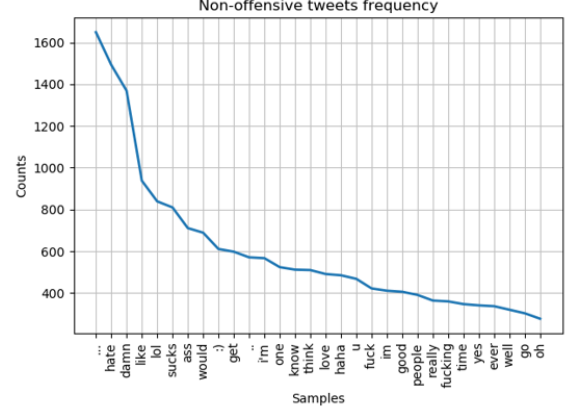


Fig. 1. Most frequent words in non-offensive tweets.

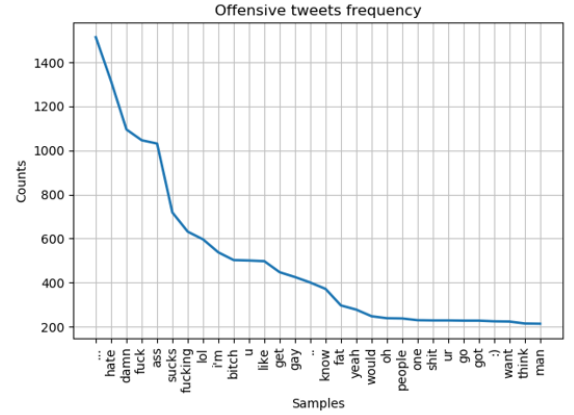


Fig. 2. Most frequent words in offensive tweets.

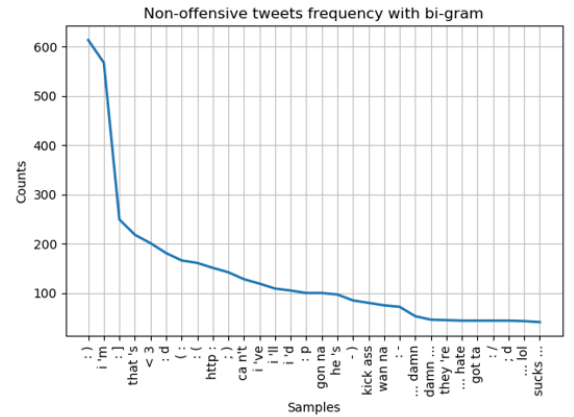


Fig. 3. Most frequent words in non-offensive tweets grouped in bi-gram.

¹<https://www.kaggle.com/daturks/dataset-for-detection-of-cybertrolls/home>

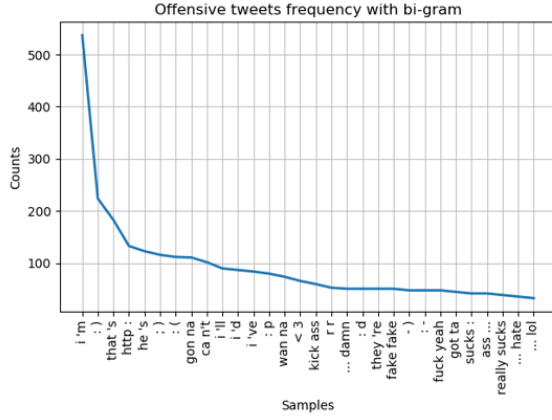


Fig. 4. Most frequent words in offensive tweets grouped in bi-gram.

In the dataset there are 17576 unique tokens. In the non-offensive tweets there are 14922 unique tokens, while in the offensive tweets there are 6486 unique tokens. If we group tokens sequentially in groups of two (bi-gram), there are 77973 unique tokens. In the non-offensive tweets there are 62609 unique tokens and in the offensive tweets there are 18458 unique tokens.

III. DATA PRE-PROCESSING

The pre-processing of words is done using a series of transformations applied to the words in the tweets. This step is fundamental to the normalization of words with similar meaning and removal of unnecessary words in the tweets. First, all characters in tweets are lowercased. To each tweet is applied a tokenizer to split the tweets in words and convert the words into tokens. In this case, it was applied the TweetTokenizer, present in the NLTK library ².

Afterwards, the punctuation is removed from the tokens, as well as the stop words ("the", "and", "of", ...). The stop words used are from the Stopwords Corpus in NLTK library. Finally, a lemmatization algorithm is applied to the tokens, to group the inflected forms of a word for better analysis of different words with the same meaning. The algorithm used was WordNet Lemmatizer, also included in the NLTK library ³. Instead of lemmatization, stemming could also be used, but lemmatization was chosen because of the use of a vocabulary dictionary and morphological analysis to get a better representation of the words.

IV. FEATURE EXTRACTION

To extract the features from the tokens in tweets two methods were used:

- N-gram counting;
- TF-IDF weighting.

The first method involves counting and selecting the top most used tokens in the tweets and use them as features. The related feature will be the number of times the tweet contains each

one of the tokens. The result will be a sparse matrix that is normalized after all the countings. To include token proximity, it is tested with 1-gram and 2-gram tokens and it will be evaluated if the general results of the model improve with 2-gram tokens. N-gram groups a contiguous sequence of N tokens. We have chosen to take a word as the basic unit. For example, the tweet with the tokens "we", "love" and "ai" can be transformed into "we love" and "love ai" 2-gram tokens. With that, the token locality can be better represented. This behavior was achieved with the use of CountVectorizer module in Scikit-learn library ⁴.

The second method involves calculating the TF-IDF (term frequency – inverse document frequency) and selecting the top most used tokens in the tweets, using the weight of each token in a tweet as feature. The TF-IDF is as term-weighting scheme used to represent the importance of a token in a tweet. The importance of a term increases logarithmically with its frequency in a tweet and the weight is normalized according to all other weights in the same tweet, being unnecessary further normalization. This behaviour was achieved with the use of TfidfVectorizer module in Scikit-learn library ⁵.

V. TRAINING & TESTING

A. Training approach

The dataset division for the training and testing phase was 80% of data for training and cross-validation (using five-fold cross-validation) and 20% for testing.

Different classifiers and parameters were tested.

The tested classifiers were:

- Logistic Regression;
- SGD Classifier;
- Linear SVC;
- NuSVC;
- SVC;
- Gaussian Naive Bayes;
- Decision Tree;
- Random Forest;
- AdaBoost;
- XGBoost;
- Multi-layer Perceptrons.

The different parameters to test were:

- Maximum number of most used tokens to create features;
- Different n-gram parameters for N-gram counting (1-gram, 2-gram and combination);
- N-gram counting and TF-IDF weighting combination;
- Classifiers hyperparameters.

Due to the many classifiers chosen to evaluate, the number of possible parameter combinations has an exponential growth, being virtually impossible to make an optimal evaluation.

To solve that problem, the evaluation phase was subdivided in a sub-optimal 3-step phase described below.

⁴https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html

⁵https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

²<https://www.nltk.org/api/nltk.tokenize.html>

³<https://wordnet.princeton.edu/>

B. First evaluation phase

On the first phase, the classifiers are tested using the default hyperparameters. The use of the N-gram counting and TF-IDF weighting approaches are tested separately and combined. The number of most used tokens vary between 100, 1000, 5000, 10000 and 25000. This number is used to give a maximum number of tokens used in the feature extraction. The tests are also done using 1-gram tokens, 2-gram tokens and both approaches combined (only applied to the N-gram counting). Finally, for each test, it is performed five-fold cross-validation using 80% of the dataset and is stored the confusion matrix, accuracy, precision, recall and f1-score. The most important metrics for the problem are accuracy and recall. A high recall is essential because if the classifier has many false positives, they can be detected by manual inspection. However, if the classifier produces many false negatives, they will not be analyzed and they will perturb readers.

On this phase, the best combination of parameters is chosen, along with the four classifiers with the best results.

C. Second evaluation phase

On the second phase, the four best classifiers from the first phase are tested with different hyperparameters. This phase is for selecting the best hyperparameters for each one of the classifiers based on the obtained results, also using five-fold cross-validation.

D. Third evaluation phase

On the last phase, the four best classifiers with the best hyperparameters and features will be tested with 20% of the dataset, the test data, to get the final results.

E. First testing phase

The extensive results of the first testing phase can be found in the Appendix A.

In this phase, some classifiers were only tested with the number of most used tokens below 5000, being removed from this analysis. NuSVC and SVC were eliminated due to the excessive memory used (due to the kernel transformations), causing a memory error (NuSVC and SVC are SVM's with polynomial kernel and Linear SVC was kept because it uses an SVM with a linear kernel). AdaBoost and XGBoost are two ensemble boosting methods that, due to the high number of features used, could not be trained in a reasonable time with a number of tokens higher than 5000.

In the figure 5 we can see the impact of choosing a different number of most used tokens to analyze. Better results are achieved when the number of tokens chosen to represent the features is higher.

However, other conclusions can be drawn from this figure. For a small number of tokens, the approaches with TF-IDF are the ones with the higher accuracy. While the performance of models that take into account bi-gram tokens improves the accuracy of the classifiers, the performance of the models that only take into account one-gram tokens seems to decrease after the 5000 most used tokens. This can happen because

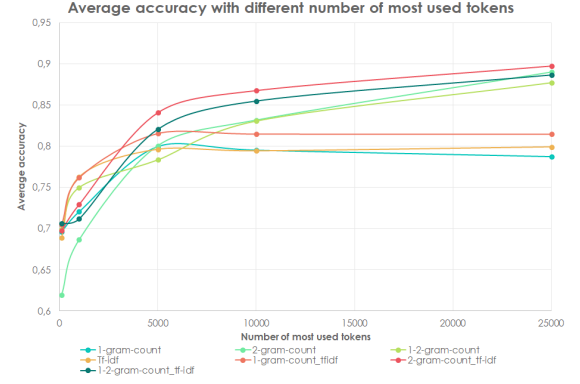


Fig. 5. Average accuracy with different number of features.

while the bi-gram tokens have 81075 unique tokens in both classes (62609 unique tokens in non-offensive tweets and 18458 unique tokens in offensive tweets), one-gram tokens only have 21408 unique tokens in both classes (14922 unique tokens in non-offensive tweets and 6486 unique tokens in offensive tweets). According to the Zipf's Law⁶, 80% of the words in a document appear only once. Which means that in this case, we are only interested in the most used 20% tokens, which are 16215 tokens using bi-gram tokens, and 4281 tokens using one-gram tokens. Increasing the number of one-gram tokens after 4281 tokens may only add noise to the model, adding unnecessary tokens to characterize the classes, only overfitting on the training data. On the other hand, the accuracy of the models using bi-gram tokens as features seems to increase logarithmically with the number of tokens used.

It can also be seen that the models with TF-IDF features improve approximately 0.01 when compared with the same model without TF-IDF features.

The highest accuracy is achieved with only 2-gram features in the tokens counting and using TF-IDF.

The results for all classifiers tested with bi-gram tokens in the tokens counting and TF-IDF as feature extractors are shown in the figure 6. The classifier with higher precision and accuracy is the Logistic Regression, tied with the Linear SVC in the accuracy. The other two chosen classifiers were the ones that have the best recall, which is the Decision Tree and the Multi-layer Perceptrons.

F. Second testing phase

The four classifiers that are tested in this stage, as explained in the previous subsection, are Multi-layer Perceptrons, Decision Tree, Linear SVC and Logistic Regression.

The extensive results can be found the appendix B.

1) *Multi-layer Perceptrons*: On the Multi-layer Perceptrons classifier, the following hyperparameters were chosen:

- Activation function: ReLU (Rectified Linear Unit);
- Solver: ADAM;
- 500 epochs for training, with early stopping.

The hyperparameters chosen to vary were:

- Network configuration;

⁶https://en.wikipedia.org/wiki/Zipf's_law

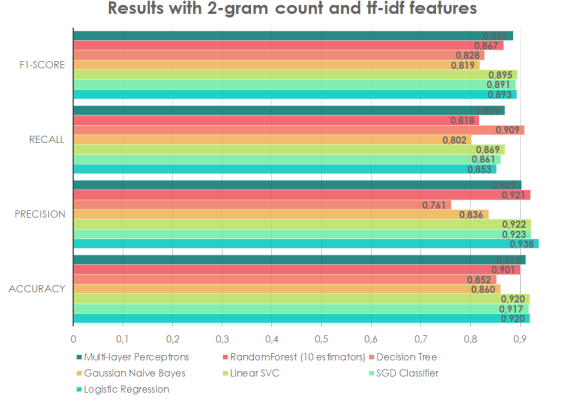


Fig. 6. Full results of all classifiers with bi-gram tokens in the tokens counting and TF-IDF features.

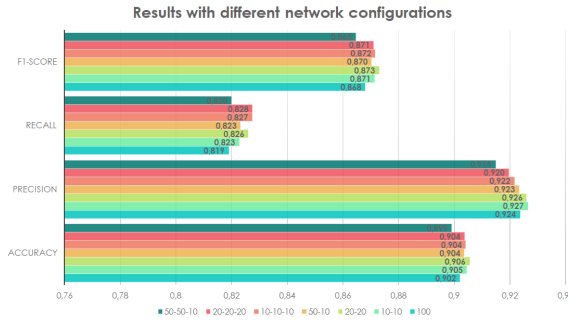


Fig. 7. Multi-layer Perceptrons results using different network configurations.

- Lambda value.

The network configuration varied among one, two and three layers, with different numbers of neurons: 100, 10x10, 20x20, 50x10, 10x10x10, 20x20x20 and 50x50x10. The lambda value chosen to regularize the activation function in each neuron varied among 0, 0.0001, 0.001, 0.01, 0.1 and 1.

The figure 7 shows the results obtained when using different network configurations. The results are similar among the different configurations, but it can be perceived a small improvement in the precision with smaller networks, with the exception of the configuration with just one layer.

In the figure 8 it can be seen the results of the classifier when tested with different lambda values. The results seem uncorrelated and the variations are too small to take any conclusion.

In the figure 9 the recall of the model is shown with different network configurations and lambda values. The recall is the chosen metric to represent because it is the most important metric to evaluate in this problem and in the previous tests showed worst results when compared with other metrics. Our goal is to maximize the recall, since the overall accuracy, F1-score and precision showed equally better results. It can be seen that the results with different network configurations and lambda values do not change dramatically among each other. However, the network with two layers and twenty neurons in each layer and a lambda value of 0.001 showed the highest recall. These parameters are the parameters chosen to calculate the final test result.

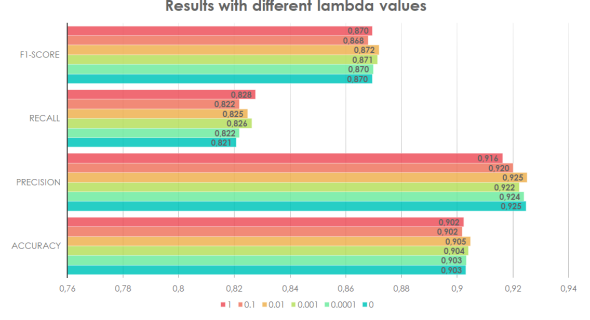


Fig. 8. Multi-layer Perceptrons results using different lambda values.

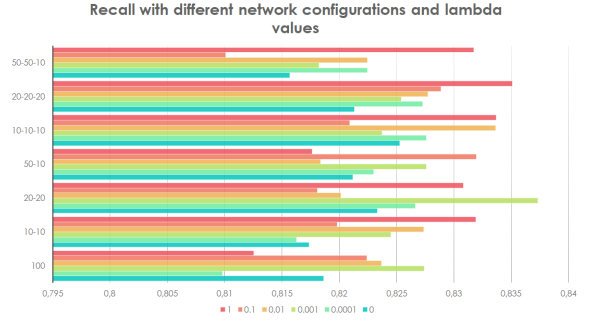


Fig. 9. Multi-layer Perceptrons recall using different network configurations and lambda values.

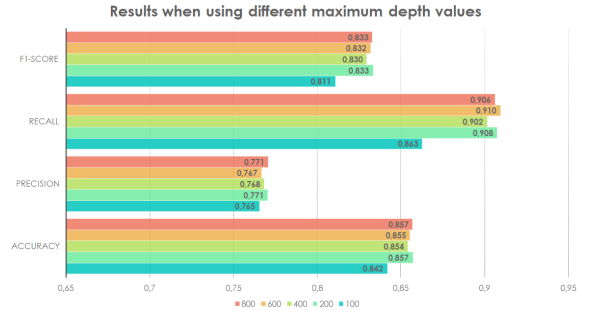


Fig. 10. Decision Tree results using different maximum depth values.

2) *Decision Tree*: On the Decision Tree classifier, the hyperparameters to vary were the following:

- Criterion;
- Maximum depth.

The first hyperparameters specifies the function to measure the quality of a split. While "gini" uses the Gini impurity, "entropy" uses the information gain. The second hyperparameter specifies the maximum depth of the tree. A few parameters were tested: 100, 200, 400, 600 and no maximum depth (more than 750 tree depth for both entropies tested, represented by 800).

The results when varying the maximum depth of the tree are presented in the figure 10. No conclusion can be taken from the results, because there is no clear trend in the data, the variations of results are minimal and seem totally random.

In the figure 11 are compared the results when using entropy and gini criterion for splitting nodes. Entropy criterion

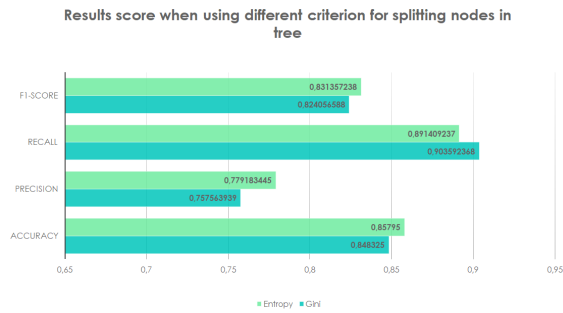


Fig. 11. Decision Tree results using different criteria for splitting nodes.

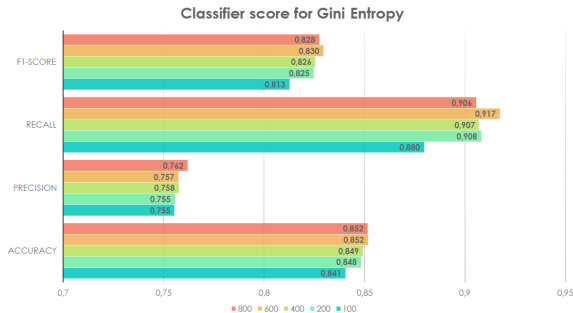


Fig. 12. Decision tree results with gini criterion and varying the maximum tree depth.

presents better results for accuracy, precision, and F1-score. However, gini criterion has a higher recall. As recall is the most important metric to evaluate, the gini criterion will be chosen for the following tests on the classifier.

With gini as a criterion decision, the figure 12 presents the results varying the maximum depth of the tree. It can be perceived a small improvement on the result with a higher tree depth. Particularly, the maximum depth chosen for the classifier is 600, because of the higher recall when compared with other tree depths.

3) *Linear SVC*: On the Linear SVC, the hyperparameters to vary were the following:

- Penalty;
- C.

The first hyperparameter specifies the norm used in the penalization. It was tested with 'l1' and 'l2' penalties. The second hyperparameter specifies the penalty parameter C of the error term. It determines the influence of the misclassification on the objective function. It was tested for the following values: 0.1, 1, 5, 10, 25, 50, 100, 500 and 1000.

In the figure 13 it is shown the accuracy for l1 and l2 penalties with different C values. The highest accuracy value achieved for the l2 penalty was with C=5 and for the l1 penalty was with C=1. For the same penalty, the difference is insignificant between the values of C, except for C=0.1 using l1 penalty. We can conclude that the l2 penalty has better accuracy compared to the l1 penalty for almost all C values. For this reason, it is clear that the classifier will produce better results when applied l2 normalization.

In the figure 14 it is shown the accuracy, precision, recall and f1-score values for different C values with l2 penalty. All

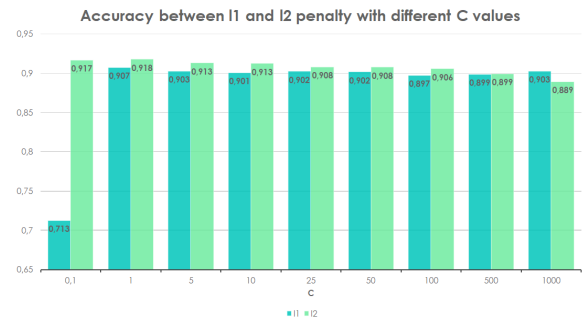


Fig. 13. Linear SVC accuracy for l1 and l2 penalty with different C values.



Fig. 14. Linear SVC accuracy, precision, recall and f1-score for l2 penalty with different C values.

parameters tend to decrease as C increases, with the exception of the recall. As accuracy and recall are the most important parameters in this problem, the most balanced combination must be evaluated. The highest accuracy value achieved was with C=0.1 but it has the worst recall. C=5 has the second highest accuracy result and a high recall. For this reason, we can conclude that C=5 offers the best relation between the two most important parameters referred above.

Based on the results shown, the best parameter values are l2 penalty and C=5.

4) *Logistic Regression*: On the Logistic Regression, the hyperparameters to vary were the following:

- Penalty;
- C.

The first hyperparameter specifies the norm used in the penalization. It was tested with 'l1' and 'l2' penalties. The second hyperparameter specifies the penalty parameter C of the error term. It determines the influence of the misclassification on the objective function. It was tested for the following values: 0.1, 1, 5, 10, 25, 50, 100, 500 and 1000.

In the figure 15 it is shown the accuracy for l1 and l2 penalties with different C values. The highest accuracy value achieved for the l2 penalty was with C=5 and for the l1 penalty was with C=5 and C=10. We can conclude that the l2 penalty is the best choice because it has better accuracy compared to the l1 penalty for all C values.

In the figure 16 it is shown the accuracy, precision, recall and f1-score values for different C values. As referred before, accuracy and recall are the most important parameters in this problem, so it is preferred the most balanced combination.

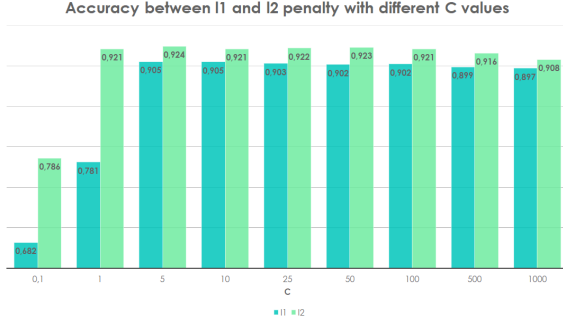


Fig. 15. Logistic Regression accuracy for l1 and l2 penalty with different C values.

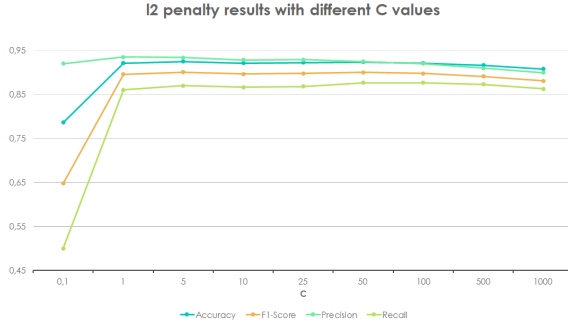


Fig. 16. Logistic Regression accuracy, precision, recall and f1-score for l2 penalty with different C values.

The highest accuracy value achieved was with $C=1$ but the difference is insignificant when compared to $C=5$. The highest recall value achieved was with $C=50$ or $C=100$. $C=5$ offers not only better accuracy than $C=50$ and $C=100$, but also better recall than $C=1$. It also has better accuracy than $C=10$ and $C=25$ and better recall than $C=10$, being almost the same as $C=25$. If we consider precision and f1-score, $C=5$ has the highest values overall. For these reasons, $C=5$ is considered the best C value, offering a balanced relation between accuracy and recall besides having the highest precision and f1-score.

Based on the results shown, the best parameter values are l2 penalty and $C=5$.

G. Final results

In this subsection are presented the results of the final tests. These tests were made with 20% of test data not used in the training phase, using the other 80% data for training.

All results have improved in the final tests, due to the large amounts of data used for training. If a larger dataset was available, it would be interesting to train the classifiers with more data to test if the results improved even more.

In the table I it is possible to see the final results. The Logistic Regression presents the highest accuracy and f1-score, followed close by Linear SVC. The Decision Tree has the worst accuracy, precision and f1-score, but it presents the best recall. However, the Logistic Regression and LinearSVC recall are not far from the Decision Tree recall. The Multi-layer perceptrons have the higher precision value, but the worst recall.

TABLE I
FINAL RESULTS OF THE FOUR BEST CLASSIFIERS

| Classifier | Accuracy | Precision | Recall | F1-Score |
|-------------------------|----------|-----------|--------|----------|
| Logistic Regression | 0.956 | 0.940 | 0.946 | 0.943 |
| Linear SVC | 0.952 | 0.924 | 0.952 | 0.938 |
| Decision Tree | 0.871 | 0.764 | 0.959 | 0.850 |
| Multi-layer Perceptrons | 0.944 | 0.943 | 0.907 | 0.925 |

From the analysis of the results, the Multi-layer Perceptrons would not be considered to the final classifier due to the low recall. The Logistic Regression and Linear SVC are two classifiers that could be used to classify offensive tweets. While its recall is not the best when compared to the Decision Tree recall, its results are close to the best and have a much higher accuracy than both other classifiers. Overall, if the recall was crucial when dealing with classification of offensive tweets, even if that meant a lot of false positives, Decision Tree was the chosen classifier. Otherwise, if overall accuracy and f1-score results were more important than recall, Logistic Regression or Linear SVC would be the best classifiers to solve the problem.

VI. CONCLUSION

In this report, it was shown an approach to build an offensive tweet classifier. The final results show that it is possible to build a classifier with high accuracy (95.6%) or high recall (95.9%) using a small dataset (20001 tweets) by applying classical machine learning techniques and feature and model tuning. In the explored dataset, the token count with one-gram tokens decreases the overall accuracy if having more than 5000 tokens as features to the counter. However, using two-gram tokens improves the overall classifiers results logarithmically with an increase in the number of tokens. This result may indicate that in this specific dataset, the most used tokens used in both classes of tweets are similar, but if we use token proximity we can better differentiate both classes. In the four classifiers chosen to vary the hyperparameters, the impact of the hyperparameters variations was reduced. This may happen due to the high number of tokens used to extract features (25000 tokens) and consequently the high number of features used in the classifiers (almost 40000 features). The results can be further improved using other techniques. Using bigger n-grams and possibly joining with bi-gram tokens can help to higher the general accuracy. More labelled data can also help to differentiate between both classes. Using recurrent deep learning networks may also detect long-range dependencies not explored in this work.

REFERENCES

- [1] S. Shah, "Twitter safe search will now automatically hide 'sensitive' content on Android," 2017. [Online]. Available: <https://www.digitaltrends.com/social-media/twitter-safety-search-replies/>
- [2] A. V. A. P. Shylaja S S, Abhishek Narayanan, "Document Embedding Generation for Cyber-Aggressive Comment Detection using Supervised Machine Learning Approach," *Proc. of ICON*, pp. 348–355, 2017.

- [3] J. B. E. D. C. G. S. A. V. Despoina Chatzakou, Nicolas Kourtellis, "Mean Birds: Detecting Aggression and Bullying on Twitter," *WebSci '17 Proceedings of the 2017 ACM on Web Science Conference*, pp. 13–22, 2017.

APPENDIX A

FIRST TESTING PHASE RESULTS

This appendix section contains seven pages, each one with different being tested.

- The first page contains the results only with 1-gram counter
- The second page contains the results with 2-gram counter
- The third page contains the results with 1-gram and 2-gram counter
- The fourth page contains the results with TF-IDF
- The fifth page contains the results with 1-gram counter and TF-IDF
- The sixth page contains the results with 2-gram counter and TF-IDF
- The seventh page contain the results with 1-gram and 2-gram counter and TF-IDF

Each table contains:

- Number of iteration
- Number of features tested
- Time to train (s)
- True Positives (TP)
- True Negatives (TN)
- False Positives (FP)
- False Negatives (FN)
- Accuracy
- Precision
- Recall
- F1-Score

APPENDIX B

SECOND TESTING PHASE RESULTS

This appendix contains four pages, each one with the testing results with the changing of the hyperparameters of a classifier: Logistic Regression, Linear SVC, Decision Tree and Multi-layer Perceptrons, respectively.

Each table contains:

- Number of iteration
- Number of features tested
- Time to train (s)
- True Positives (TP)
- True Negatives (TN)
- False Positives (FP)
- False Negatives (FN)
- Accuracy
- Precision
- Recall
- F1-Score

[illegible]

Chen, M. C.

Decision Tree

Features Counter with 2-gram and tf-idf
Maximum 25000

criterion gini
max_depl 100

| Feature number | Time to train | TP | TN | FP | FN | Accuracy | Precision | Recall | F1-Score |
|----------------|---------------|-------------|------|-------|-------|----------|-----------|---------|----------|
| 1 | 39527 | 429.4031811 | 1114 | 1594 | 381 | 111 | 0.84625 | 0.74515 | 0.90339 |
| 2 | 39527 | 419.2259033 | 1080 | 1590 | 346 | 184 | 0.83438 | 0.75736 | 0.85443 |
| 3 | 39527 | 260.6240873 | 1153 | 1613 | 324 | 110 | 0.86438 | 0.78064 | 0.91291 |
| 4 | 39527 | 93.09207344 | 1103 | 1563 | 388 | 146 | 0.83313 | 0.73977 | 0.88311 |
| 5 | 39527 | 102.1296356 | 1084 | 1555 | 353 | 208 | 0.82469 | 0.75435 | 0.83901 |
| Average | 260.8949761 | 1106.8 | 1583 | 388.4 | 151.8 | 0.84056 | 0.75545 | 0.87977 | 0.81265 |

criterion gini
max_depl 200

| Feature number | Time to train | TP | TN | FP | FN | Accuracy | Precision | Recall | F1-Score |
|----------------|---------------|-------------|--------|------|-------|----------|-----------|---------|----------|
| 1 | 39527 | 137.3900383 | 1162 | 1540 | 378 | 120 | 0.84438 | 0.75455 | 0.9064 |
| 2 | 39527 | 132.9106324 | 1134 | 1547 | 400 | 119 | 0.83781 | 0.73924 | 0.90503 |
| 3 | 39527 | 132.292778 | 1172 | 1568 | 351 | 109 | 0.85625 | 0.76853 | 0.91491 |
| 4 | 39527 | 114.8673561 | 1128 | 1611 | 356 | 105 | 0.85594 | 0.76011 | 0.91484 |
| 5 | 39527 | 132.3715451 | 1119 | 1591 | 365 | 125 | 0.84688 | 0.75404 | 0.89952 |
| Average | 129.9737699 | 1143 | 1571.4 | 370 | 115.6 | 0.84825 | 0.75549 | 0.90814 | 0.82479 |

criterion gini
max_depl 400

| Feature number | Time to train | TP | TN | FP | FN | Accuracy | Precision | Recall | F1-Score |
|----------------|---------------|-------------|--------|-------|-----|----------|-----------|---------|----------|
| 1 | 39527 | 132.86023 | 1132 | 1573 | 364 | 131 | 0.84531 | 0.75683 | 0.90628 |
| 2 | 39527 | 133.6318045 | 1125 | 1598 | 365 | 112 | 0.85094 | 0.75503 | 0.90946 |
| 3 | 39527 | 136.7623962 | 1162 | 1547 | 370 | 121 | 0.84656 | 0.76949 | 0.90569 |
| 4 | 39527 | 160.9458862 | 1153 | 1585 | 357 | 105 | 0.85563 | 0.76358 | 0.91653 |
| 5 | 39527 | 150.1404922 | 1136 | 1578 | 370 | 116 | 0.84813 | 0.75432 | 0.90735 |
| Average | 143.2681226 | 1141.6 | 1576.2 | 365.2 | 117 | 0.84931 | 0.75762 | 0.90706 | 0.82562 |

criterion gini
max_depl 600

| Feature number | Time to train | TP | TN | FP | FN | Accuracy | Precision | Recall | F1-Score |
|----------------|---------------|-------------|--------|-------|-----|----------|-----------|---------|----------|
| 1 | 39527 | 156.025345 | 1122 | 1600 | 381 | 97 | 0.85063 | 0.74651 | 0.92043 |
| 2 | 39527 | 151.3102052 | 1150 | 1584 | 377 | 89 | 0.85438 | 0.75311 | 0.92817 |
| 3 | 39527 | 157.8437214 | 1180 | 1529 | 369 | 122 | 0.84656 | 0.76178 | 0.9063 |
| 4 | 39527 | 157.46083 | 1121 | 1583 | 372 | 124 | 0.845 | 0.75084 | 0.9004 |
| 5 | 39527 | 154.000463 | 1200 | 1562 | 350 | 88 | 0.86313 | 0.77419 | 0.93168 |
| Average | 155.7415508 | 1154.6 | 1571.6 | 369.8 | 104 | 0.85194 | 0.75729 | 0.91739 | 0.82964 |

criterion gini
max_depl MAX (~800)

| Feature number | Time to train | TP | TN | FP | FN | Accuracy | Precision | Recall | F1-Score |
|----------------|---------------|-------------|--------|-------|-------|----------|-----------|---------|----------|
| 1 | 39527 | 153.7395238 | 1180 | 1556 | 346 | 118 | 0.855 | 0.77326 | 0.90909 |
| 2 | 39527 | 166.029485 | 1138 | 1540 | 393 | 129 | 0.83688 | 0.74331 | 0.89818 |
| 3 | 39527 | 147.8999882 | 1119 | 1617 | 343 | 121 | 0.855 | 0.76539 | 0.90242 |
| 4 | 39527 | 187.4034297 | 1122 | 1614 | 346 | 118 | 0.855 | 0.76431 | 0.90484 |
| 5 | 39527 | 122.3378849 | 1140 | 1599 | 353 | 108 | 0.85594 | 0.76356 | 0.91346 |
| Average | 165.4520635 | 1139.8 | 1585.2 | 356.2 | 118.8 | 0.85156 | 0.76197 | 0.9056 | 0.82758 |

criterion entropy
max_depl 100

| Feature number | Time to train | TP | TN | FP | FN | Accuracy | Precision | Recall | F1-Score |
|----------------|---------------|-------------|--------|-------|-----|----------|-----------|---------|----------|
| 1 | 39527 | 91.57152629 | 1066 | 1651 | 298 | 185 | 0.84906 | 0.78152 | 0.85212 |
| 2 | 39527 | 92.02545428 | 994 | 1659 | 346 | 201 | 0.82906 | 0.74179 | 0.8318 |
| 3 | 39527 | 88.18655762 | 1079 | 1629 | 305 | 187 | 0.84625 | 0.77962 | 0.85229 |
| 4 | 39527 | 94.66225863 | 1064 | 1597 | 311 | 228 | 0.83156 | 0.77382 | 0.82353 |
| 5 | 39527 | 85.87591124 | 1120 | 1632 | 279 | 169 | 0.86 | 0.80057 | 0.86889 |
| Average | 90.46474161 | 1064.6 | 1633.6 | 307.8 | 194 | 0.84319 | 0.77947 | 0.84573 | 0.80902 |

criterion entropy
max_depl 200

| Feature number | Time to train | TP | TN | FP | FN | Accuracy | Precision | Recall | F1-Score |
|----------------|---------------|-------------|--------|-------|-------|----------|-----------|---------|----------|
| 1 | 39527 | 103.6859341 | 1165 | 1614 | 306 | 115 | 0.86844 | 0.79198 | 0.91016 |
| 2 | 39527 | 106.8158304 | 1133 | 1639 | 324 | 104 | 0.86625 | 0.77763 | 0.91593 |
| 3 | 39527 | 103.9283895 | 1110 | 1654 | 297 | 139 | 0.86375 | 0.78891 | 0.88871 |
| 4 | 39527 | 108.5672193 | 1149 | 1621 | 320 | 110 | 0.86563 | 0.78216 | 0.91263 |
| 5 | 39527 | 104.0673733 | 1150 | 1621 | 311 | 118 | 0.86594 | 0.78713 | 0.90694 |
| Average | 105.4128413 | 1141.4 | 1629.8 | 311.6 | 117.2 | 0.866 | 0.78556 | 0.90687 | 0.84182 |

criterion entropy
max_depl 400

| Feature number | Time to train | TP | TN | FP | FN | Accuracy | Precision | Recall | F1-Score |
|----------------|---------------|-------------|--------|------|-------|----------|-----------|---------|----------|
| 1 | 39527 | 107.7346308 | 1152 | 1583 | 342 | 123 | 0.85469 | 0.77108 | 0.90353 |
| 2 | 39527 | 107.9468927 | 1112 | 1669 | 302 | 117 | 0.86906 | 0.78602 | 0.9048 |
| 3 | 39527 | 102.2973237 | 1124 | 1626 | 306 | 144 | 0.85938 | 0.78601 | 0.88644 |
| 4 | 39527 | 109.0716522 | 1119 | 1615 | 326 | 140 | 0.85438 | 0.77439 | 0.8888 |
| 5 | 39527 | 101.5984204 | 1133 | 1614 | 324 | 129 | 0.85844 | 0.77763 | 0.89778 |
| Average | 105.729784 | 1128 | 1621.4 | 320 | 130.6 | 0.85919 | 0.77911 | 0.89627 | 0.83356 |

criterion entropy
max_depl 600

| Feature number | Time to train | TP | TN | FP | FN | Accuracy | Precision | Recall | F1-Score |
|----------------|---------------|-------------|--------|-------|-------|----------|-----------|---------|----------|
| 1 | 39527 | 114.749965 | 1122 | 1605 | 325 | 148 | 0.85219 | 0.77754 | 0.88346 |
| 2 | 39527 | 105.3174195 | 1110 | 1657 | 317 | 116 | 0.86469 | 0.77786 | 0.90538 |
| 3 | 39527 | 107.0216668 | 1161 | 1595 | 317 | 127 | 0.86125 | 0.78552 | 0.9014 |
| 4 | 39527 | 106.5190003 | 1142 | 1631 | 321 | 106 | 0.86656 | 0.78059 | 0.91506 |
| 5 | 39527 | 110.5286832 | 1139 | 1581 | 358 | 122 | 0.85 | 0.76086 | 0.90325 |
| Average | 108.8273469 | 1134.8 | 1613.8 | 327.6 | 123.8 | 0.85894 | 0.77604 | 0.90171 | 0.83413 |

criterion entropy
max_depl MAX (~800)

| Feature number | Time to train | TP | TN | FP | FN | Accuracy | Precision | Recall | F1-Score |
|----------------|---------------|-------------|------|-------|-------|----------|-----------|---------|----------|
| 1 | 39527 | 108.3459818 | 1150 | 1608 | 307 | 135 | 0.86168 | 0.78929 | 0.89494 |
| 2 | 39527 | 105.8573973 | 1132 | 1628 | 320 | 120 | 0.8625 | 0.7961 | 0.90415 |
| 3 | 39527 | 108.5925591 | 1160 | 1617 | 312 | 111 | 0.86781 | 0.78804 | 0.91267 |
| 4 | 39527 | 107.6706195 | 1127 | 1637 | 316 | 120 | 0.86375 | 0.78101 | 0.90377 |
| 5 | 39527 | 107.1375256 | 1135 | 1605 | 357 | 103 | 0.85625 | 0.76072 | 0.9168 |
| Average | 107.5208167 | 1140.8 | 1619 | 322.4 | 117.8 | 0.86244 | 0.77974 | 0.90647 | 0.83826 |

