Natural Language Processing
Toxic Comment Classification on a Broad Data Set

Hilders, M. `i6169337` and **Teeuwen, C. J.** `i6169583`

Abstract

A Random Forest Classifier is used to identify different types of toxic comments from a data set. In addition to this, the positive affection of the performance of this algorithm by using parameter optimization is described. The optimized Random Forest Classifier is tested on whether it is possible for the algorithm to become a proper and useful toxic comment filter for day-to-day users. After conducting the test it was determined that the results were inconclusive and it is not possible for the model to become a proper toxic comment filter for the specified user group.

1 Introduction

Natural Language Processing has a lot of applications: The technology is used as a base for spam filters, spell check and even voice text messaging. Having such a strong function, one becomes interested in the technology's capabilities; specifically in the field of internet discussion.

To reach this field, the decision was made to delve into a sentiment analysis of internet comments, specifically on recognizing toxicity. This has obvious applications in the digital environment in monitoring of websites; where the challenge of removing toxic comments often exists. An algorithm that picks out potential bad comments could speed up this task tremendously.

The topic is interesting to research because it is a widespread problem on the social platforms. Identifying these comments could have actual value, as it would improve the user experience of many websites. Therefore, for this paper, the research question reads as follows: *Is it possible to construct a language classifier which eradicates toxic comments, in such a way that no useful data will be lost?*

2 Prior research

The research conducted around sentiment analysis, to get a good view of what techniques would be applicable, lead to "Sentiment Classification Using an Ensemble of Convolutional Neural Networks with Distant Supervision" by Deriu et all(2). Which explores a technique in which convolutional neural networks are combined with a Random Forest Classifier to perform sentiment analysis. Digging through this, the Random Forest Classifier -from here on abbreviated as RFC-, emerged as something which could be highly applicable to this data set; as the technique can process a lot of data quickly and the algorithm leaves room for a high degree of customization. This customization can be achieved by using random grid techniques, which will be elaborated on later on in this paper.

Another article which caught the attention was "Deceiving Googles Perspective API Built for Detecting Toxic Comments" by Hosseini(4). In this paper, the main engine talked about is the 'Google Perspective API'; a toxic comment detection API made in such a way that is easy to implement for any user. This way, everyone harmed by hateful comments can use the API to reduce them significantly. The technique Google used for this API mainly revolves around implementing a deep neural network, which proved to be successful. However, as also found by the production team, the algorithm is not perfect. The team identified ways to trick the neural network into giving false positives and false negatives. To elaborate, the algorithm labeled comments which were toxic as non-toxic and vice-versa. This gives the sense that it will be hard to make a model which will have a high enough precision to be used as a tool to negate toxic comments for e.g. children; while favorable and useful comments might frequently be filtered.

3 Data Set

As a data set for the language model, a pre-existing data set is used provided by Kaggle.com, a challenge website. The challenge(5) it comes with is very similar to our task of identifying toxic comments. The data set consists of remarks, typically shorter than three sentences. Furthermore, The sentences are pulled from a wide variety of sources, which includes citations from scientific papers but also comments on Youtube videos; which makes it extremely diverse. Because the data set is so varied, confidence raises in the sense that a model trained by this data can be used to classify many types of texts for toxicity.

The training set consist of more than 500.000 lines of sentences labeled by: Toxic, severe-toxic, obscene, threat, insult and identity-hate. In the training set, for each of the sentences, these labels are given a binary value; coherent to it being true or not.

4 Approach

In short, the approach used is a RFC(7), and can be found in the GitHub of this project(1). The said classifier generates an array of decision trees, and decides the classification based on those trees. However, the RFC has to have a clean input. Therefore, the provided comments in the data set can not be in their original form and have to be cleaned. For this cleaning process, several steps are performed. As a first, stopwords are removed from the data provided by the Natural Language Toolkit(6). However, there are some stopwords which were deemed important for the toxic comment detection, namely 'not', 'you', 'your', 'you're' and 'are'. To elaborate, this decision was made while 'not' can negate a toxic statement and the versions of 'you' can be used to direct a -toxic-statement to an individual. For this reason these words were added to the whitelist. For the further steps, the decision was made to substitute abbreviations such as: isnt to its full spelling -'is not'-, remove punctuation plus numerals and setting every capital letter to its lowercase equivalent as there is a strong believe that these factors do not have any impact on the classification of a potentially toxic comment. With that being done, the cleaning process is finished.

While the RFC will only accept floats as an input, all comments are put in vector form using Word2vec provided by the Gensim library (8). Each comment is given a score based on the average vector values of the words in that sentence. The vector ultimately created has a length of 300.

A random sample sized $N = 25000$ of the adapted data is taken and used to build the RFC tree. This tree is built from this subset using a process where it takes a random selection of features sized M from the data and, hereafter, splits the node on the combination of variables that give the best prediction of the score of the algorithm. No pruning is implemented in exception of the situation where the tree builder reaches maximum depth. When the algorithm attempts to classify a new piece of text, it will run it through all previously created trees. Hereafter, these trees give a classification for each of the categories. The classification which is given most frequently by the trees is the classification of 'the forest'.

As it would stand, the RFC used is initialized with its default parameters. However, to get better results, it is possible to optimize the algorithm by customizing it to the data set; while making sure that the model is not over fitted. In order to do this a random grid technique called *RandomizedSearchCV*(7) is used in order to find the best values for the so-called hyperparameters of the RFC. The hyperparameters chosen for optimization were:

- The amount of trees in the forest; -$nestimators$-
- The maximum number of features the tree-building algorithm takes into account when splitting at a node; -$maxfeatures$-
- The maximum depth of the trees in the forest; -$maxdepth$-
- If a bootstrap is used or not. This determines whether you use a replacement in the subset for a tree or if you simply take the original set. -$bootstrap$-

The *RandomizedSearchCV* algorithm randomly selects a value for each of the parameters in a given range multiple times, and returns the best ones. As parameters for the *RandomizedSearchCV*, it was decided to fit 2 folds for each of 50 candidates, totalling in 100 fits. With this input, the *RandomizedSearchCV* algorithm will then optimize the parameters by using cross validation to score different configurations. To elaborate, it will select a number of random subsets of the training set and use some of them to train the classifier, whilst using the remaining one as a test set. The test set is hereafter rotated a couple of times and the average is taken from these scores, this to counteract high variance in the process of selecting the right hyperparameter values. After this process is finished the parameters found to be optimal are retrieved from the model an subsequently saved.

5 Experiments and Evaluation

For testing purposes, the decision was made to first conduct the tests with the non-optimized
-read default- RFC. This, in order to get a threshold value which can be compared to the optimized RFC's test results. The tests are run with a subset sized $N = 25000$ as a base. For the default RFC, this resulted in the following scores, see *-Table1, Appendix-*.

The table is inserted with the results of the classification report. From this, it can be deducted that the algorithm is able to achieve a precision with a micro average of 0.57. The micro average can be seen as the most important statistic while it is deemed to be more representative than any other summed statistic. Furthermore, the micro average was seen as the go-to average while big discrepancies, between the frequencies of the classes, occur in both the train and test set -represented in the 'support' column-; which the micro average deals best with. Therefore, it can be said that when the RFC classifies a comment in one of the classes provided, there's about a 57 percent chance it is correct in placing it there.

Whilst a high precision is positive, it is unwanted for the algorithm to be too reluctant to put samples in a class, as that would mean there would be many false negatives. Unfortunately, the micro average of 'recall' is very low, only 0.14. To elaborate, only 14 percent of all cases in which a class applies, it is correctly spotted. This in turn gives the algorithm a very low f1 score $f1 = 0.20$, which scores the algorithm based on both the precision and recall by the following calculation: $2 * \frac{precision*recall}{precision+recall}$.

In order to reach the goal of the project, to have a general classifier for all toxic comments, precision and recall are both important to optimize. The f1 metric gives a good indication of both these scores. The algorithm gave a very low f1 score, which indicates low precision and/or low recall. In this case, the recall is primarily responsible for the lower-than-desired score. Because the recall is so low at this stage, it can be said that the current configuration is not suited for the identification of toxic comments, as it doesn't identify the applicable classes in 0.86 percent of all cases.

Another test value which is important to be looked at is the Receiver Operator Characteristic -ROC- and its subsequent Area Under Curve -AUC- metric. In order to do this, the training set is cross-validated 10 folds. Each of these folds give an AUC score as an output, which is a combination of the True Positive rate and False Positive rate. to elaborate, the higher the AUC score, the better it can distinguish between a comment being toxic or non-toxic. After computing the 10 folds, the mean of the results is taken, which,in this case, gives a score of 0.67. Although this seems to bee a good AUC score, which would mean the model is good at distinguishing toxic and non-toxic comments, it can not yet be deemed as a good result. This, while first the comparison has to be made between the default and optimized model.

For the second experiment, the RFC was initialized by the found optimum hyperparameters *-RandomizedSearchCV, see Section: Approach-*. To get a view of the performance of this model, the RFC was tested on the same data set as the first experiment.

Looking at *Table2, Appendix*, it can be seen that even with the optimized parameters, the algorithm continues to have a low recall value; where the micro average precision jumped from 0.57 to 0.67, the recall stayed at a meager 0.12. As discussed before, this remains a problem. In addition and mainly due to this, the f1 score remains not unsurprisingly low: 0.21.

However, noteworthy is the result gained by the cross-validated AUC score; this improved significantly, rising from the previously mentioned average of 0.67 to 0.77. To elaborate, this indicates that the optimized algorithm is even better at distinguishing between toxic and non-toxic than the standard model.

6 Conclusion

From all the test results, facts can be derived. The first and obvious fact would be that both RFCs have low f1 scores. Which inevitably means that the models are bad at making predictions for the comment's classes. However, on the contrary, the models do have a high AUC score, with the optimized model even obtaining a score of 77 percent. What can be concluded from this are a few things, as a first, one can comfortably say that the optimized model is the best model. In addition to this, the model is good at distinguishing between toxic and non-toxic comments; however, bad at classifying it more specifically. Therefore, this paper has showed that the random forest classifier is a competent algorithm for toxic comment classification, although when keeping the classification at a binary level -read toxic or nontoxic-. An experiment for the future could be removing these distinctions from the data entirely and only using a binary classification saying if the given comment is toxic or not. Giving the model more specificity to work with and ultimately not having it deal with all the different sub classifications is expected to result in better performance . In conclusion, the model lacks good precision to be used as a good means to filter toxic comments while the false positives of the model will result in non-toxic and useful comments to be filtered.

Therefore, *Is it possible to construct a language classifier which eradicates toxic comments, in such a way that no useful data will be lost?* has to be answered with a no for this RFC.

References

[1] *The github repository containing our code*. https://github.com/MartijnHilders/NLP-Project.

[2] J. Deriu, M. Gonzenbach, F. Uzdilli, A. Lucchi, V. D. Luca, and M. Jaggi, *Swiss-cheese at semeval-2016 task 4: Sentiment classification using an ensemble of convolutional neural networks with distant supervision*, Proceedings of the 10th International Workshop on Semantic Evaluation, (2016), pp. 1124–1128.

[3] S. V. Georgakopoulos, S. K. Tasoulis, A. G. Vrahatis, and V. P. Plagianakos, *Convolutional neural networks for toxic comment classification*, in Proceedings of the 10th Hellenic Conference on Artificial Intelligence, SETN '18, New York, NY, USA, 2018, ACM, pp. 35:1–35:6.

[4] H. Hosseini, S. Kannan, B. Zhang, and R. Poovendran, *Deceiving google's perspective API built for detecting toxic comments*, CoRR, abs/1702.08138 (2017).

[5] Jigsaw, *Conversation ai, https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge*.

[6] E. Loper and S. Bird, *Nltk: The natural language toolkit*, in Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics - Volume 1, ETMTNLP '02, Stroudsburg, PA, USA, 2002, Association for Computational Linguistics, pp. 63–70.

[7] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, *Scikit-learn: Machine learning in Python*, Journal of Machine Learning Research, 12 (2011), pp. 2825–2830.

[8] R. Řehůřek and P. Sojka, *Software Framework for Topic Modelling with Large Corpora*, in Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks, Valletta, Malta, May 2010, ELRA, pp. 45–50. http://is.muni.cz/publication/884893/en.

7 Appendix

Test Results

Classification Report for the un-optimized algorithm

Classification	Precision	Recall	f1-score	Support
Toxic	0.55	0.14	0.22	798
Severe-toxic	0.46	0.07	0.13	81
Obscene	0.61	0.15	0.24	418
Threat	0.00	0.00	0.00	24
Insult	0.59	0.12	0.20	409
Identity-hate	1.00	0.01	0.02	88
micro avg	0.57	0.12	0.20	1818
macro avg	0.53	0.08	0.13	1818
weighted avg	0.58	0.12	0.20	1818

Table 1: default RFC

Classification Report for the optimized algorithm

Classification	Precision	Recall	f1-score	Support
Toxic	0.65	0.13	0.21	798
Severe-toxic	0.58	0.09	0.15	81
Obscene	0.69	0.16	0.26	418
Threat	0.00	0.00	0.00	24
Insult	0.69	0.12	0.21	409
Identity-hate	1.00	0.01	0.02	88
micro avg	0.67	0.12	0.21	1818
macro avg	0.60	0.08	0.14	1818
weighted avg	0.67	0.12	0.21	1818

Table 2: Optimized RFC