Paris-Dauphine University

IASD Master

# NLP Project: Toxic Comments Classification

*Author:*
Nassim AIT ALI BRAHAM
Théo Delemazure
Jean Dupin

May 16, 2020

## 1 Introduction

Moderation of online discussions/comments has become a key area of focus both for technology companies whose products include text interaction between users and from regulators since the nefarious aspect of negative online speech has shown serious impact on mental health. While technology companies employ armies of moderators, advancements in NLP using neural networks may provide larger degrees of automation and reduce the need for human moderation so that focus can be directed towards other content (videos, deepfake, propaganda) that cannot be automatically processed as efficiently as text.

In this project, we explore the use of various neural architectures to spot and classify messages or comments that contain hateful speech, bullying, or other kinds of toxicity.

## 2 Data

### 2.1 Data description

We use the Wikipedia Comment Dataset containing 159,563 comments with a category of toxicity for each one, annotated by human moderator. There are 6 categories of toxicity (see Table 1). Any comment can fall into zero, one or multiple categories. These are the labels for the input data.

| | | |
|---|---|---|
| `toxic` | `severe_toxic` | `obscene` |
| `threat` | `insult` | `identity_hate` |

Table 1: Categories of toxicity

As a result, we are dealing here with a multi-class and multi-label classification problem since there can be 0, 1 or multiple categories for any particular comment. See below an example from the dataset in Table 2 (beware the example may include offensive content..).

| | |
|---|---|
| **ID** | 0036621e4c7e10b5 |
| **Comment** | "Would you both shut up, you don't run wikipedia, especially a stupid kid." |
| **Multi-class label** | 1,0,0,0,1,0 |

Table 2: An example from the dataset

### 2.2 Data cleaning

The dataset is provided raw and requires various steps of cleaning before it can be processed. An important first step is therefore to inspect the data and remove parts that are irrelevant, confusing, creating duplicates or may not be properly processed. The steps below demonstrate some of the editing that was applied.

We apply elementary cleaning steps, such as converting everything to lowercase to remove duplicates, removing numbers, removing stopwords and punctuation, and properly separating english verb contractions (e.g "don't" become ["don",”'t"]).

The raw dataset is also littered with online links (identified as starting with "http://"), IP adresses (also identified with a regular expression), or user handles that need to be removed.

### 2.3 Embeddings

We tried our models with both embeddings trained directly on the classification task, and with pre-trained embeddings. We used *GLoVe* embeddings [8] trained on Twitter, with 27 billion tokens and with an embedding size from 25 to 300. In most of our experiments we used the embeddings with 50 features, for the purpose of time/space-complexity efficiency.

The benefit of using *GLoVe* embeddings trained on Twitter is that, by nature, the vocabulary used is idiosyncratic of online exchanges between human users. Also, while Twitter moderates targeted abuse or harassment, the use of foul language or offensive language is permitted, providing us with embeddings trained in context for exactly the sort of words that are contained in the toxic comments we aim to identify.

## 3 Models

### 3.1 Simple models

It is useful to initially explore the use of elementary models that will provide a baseline against which more complex models can be compared. The simplest pipeline to construct a text classification model consists of the two following steps i) compute a (convenient) representation for the input sentences; ii) feed these representations into a classical classification algorithm.

Word embedding algorithms such as Word2Vec [7] and *GLoVe* [8] surpassed previous approaches and allowed to capture some semantics by preserving word similarity in the embedding vector space. Earlier techniques such as Bag of Words relied on word occurrence statistics to represent a text. Specifically, a sentence is represented as a vector of dimension $|V|$ [1], where the $i^{th}$ entry specifies whether $w_i$ is present in the sentence or not (or how many times it occurred). In our implementation of the baselines, we rely on TF-IDF (Term Frequency - Inverse Document Frequency), a slight variation of this concept that allows to deal with common uninformative words. Formally, TF-IDF is defined as follows,

$$tf - idf(d,t) = tf(d,t)\log(\frac{d}{df(t)})$$

where $tf(d,t)$ denotes the frequency of the word $t$ in document $d$ and $df(t)$ is the number of documents containing the term $t$.

Following this pipeline, we try multiple classification algorithms, including, Naive Bayes Classifier, Logistic Regression, Support Vector Machine (SVM), Gradient Boosting and Random Forest.

### 3.2 More complex models

In this section, we detail the neural models that were implemented to classify the different types of toxic comments. We focused on the most widely used architectures while keeping the number of parameters reasonable enough to train the models from scratch with low computational resources.

#### 3.2.1 Multilayer Perceptron

The simplest neural network architecture for text classification is the multilayer perceptron. It consists of an embedding layer (which can be trained from scratch or freezed using pre-trained embeddings like *GLoVe* or *Word2Vec*), followed by one or multiple hidden layers and

[1] where $V$ is a fixed vocabulary.

a final output layer. In our experiments, we instantiated a MLP with a single hidden layer of 50 neurons and trained the architecture with and without *GloVe* embeddings.

We also implemented the N-gram features proposed in [4] which consists in concatenating (as tokens) to each comment all the N-grams that appeared in it. In our experiments, we tested this approach with bi-grams.

#### 3.2.2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) allowed important breakthroughs in computer vision thanks to their ability to capture the spatial and temporal dependencies in an image using convolutions with small filters. The convolution operation was also generalized to textual data using a sliding windows over the neighbors of each words, allowing to capture a notion of context.

Vanilla CNNs are typically not the most powerful models for text classification, but they are computationally efficient and they provide a good baseline for comparison.

#### 3.2.3 Recurrent Neural Networks

Recurrent Neural Networks (RNN) are very suitable architectures for processing sequential data thanks to their ability to capture temporal dependencies in the input. In our experiments, we tested multiple variants of RNNs including,

- **Long Short-Term Memory (LSTM):** LSTMs were introduced in [3] to solve the problem of vanishing gradient in RNNs by introducing memory cells that give the network a *long term memory*. We implemented a classical LSTM architecture where the output of the recurrent part (the text representation) is fed into a fully connected layer followed by a linear output layer. We also implemented an attention mechanism to enhance performance and used bidirectional LSTMs in the experiments.
- **Gated Recurrent Units (GRU):** GRUs [2] are simplified versions of LSTMs with fewer parameters. We implemented two GRU architectures (denoted by *GRU1* and *GRU2*) that differ in the final layers, i.e., the way the intermediate representation computed by the recurrent part is processed: i) *GRU1:* the output states of the recurrent part for every word in the sentence are concatenated and fed into a 3-layers convolutional neural network and then a final linear output layer; ii) *GRU2:* the output of the recurrent part for the last word of the sentence is fed into a multilayer perceptron with a single hidden layer. We need to add the information of the sentence length when we compute the labels.
- **Recurrent Convolutional Neural Networks (RCNN):** RCNNs [5] combine the recurrent structure of RNNs with the pooling operation of CNNs to obtain the advantages of both architectures. Thus, the model captures contextual information with the recurrent structure and constructs the representation of the text with a CNN.

#### 3.2.4 Transformers

Transformers were introduced in [9] and have become the basic building block of multiple state-of-the-art NLP systems, outperforming GRUs and LSTMs in many tasks. In our implementation, we use pre-trained RoBERTa (Robustly optimized BERT approach) fine-tuned on our dataset.

### 3.3 Model with rationales

While detecting toxic comments is the aim of the classification task, it would be beneficial to highlight the specific part of the comment that is seen as toxic so as to provide a *rationale* for the classification. This would facilitate the work of moderators if they can directly see which part of the comment is problematic, as shown in Table 3.

| |
|---|
| yeah thats a really good way to get a life ! fucking fag ! |
| heh , look , not everything is on google you are a dumbass |
| i care about the public being misinformed about his lazy , boring ass album go cry to an episode of degrassi you fucking twat |

Table 3: Showing which part of the comment is problematic could help moderators.

In [6], the authors proposed a way to learn *rationales* for a classification task in an unsupervised fashion (i.e. target rationales are not provided during training). Additionally, they also show that training a classifier and a rationales generator can improve the accuracy of the classifier.

The main idea of this approach is to train concurrently an *encoder* and a *generator* that aim, respectively, at identifying the sentiment associated to the input and to extract a subsection of the input that has the same predictive power. In particular, the overall loss function is designed in such a way that we want to select only a few words and that those words need to be consecutive (i.e. they should form phrases). We used a bidirectional LSTM for the generator, as suggested in the paper.

We adapted the code from [6] to our project, but the results were not satisfying. We could probably fix this with some work, but we preferred to focus on other models to improve our results on other tasks. For instance, we asked for a rationale when the comment was clean and it would be better to only ask for it when the comment is toxic.

### 3.4 Training

Our training routine is inspired from the ones presented in the book *Natural Language Processing with Pytorch*. We split the dataset into 70% training set, 20% validation set and 10% test set, and we cut the sentences at 800 words because not many comments have more than 800 words (the maximum length being around 1400 words) and it enables to save time and RAM.

We used the Binary Cross Entropy Loss for both binary case and multilabel case. Indeed, since we can have several labels for the same comment, we apply a binary cross entropy loss for each possible label. We used the Adam optimizer with a learning rate of $10^{-3}$ and a scheduler to reduce the learning rate when the loss is reaching a plateau.

## 4 Experimental Results

### 4.1 Data exploration

In this part, we expose our preliminary exploration of the dataset. This study gave us insights about the distribution of the data, the most frequent tokens, which data pre-processing should be applied, etc. Naturally, all the statistics were computed on the raw data before any cleaning had been performed yet.

Figure 1 shows the distribution of the most frequent words in the dataset. Not surprisingly, the dataset is mostly dominated by uninformative stopwords. These tokens are irrelevant to the classification task, so we simply remove them in the pre-processing phase.

Figure 2a displays the comments' classes distribution in the dataset. We clearly see that the class distribution is heavily skewed, with approximately 90% of clean comments. Moreover, the subcategories of toxic comments are not equally represented. For instance, there are (fortunately) only 340 threat comments. In order to alleviate this issue, we modify the

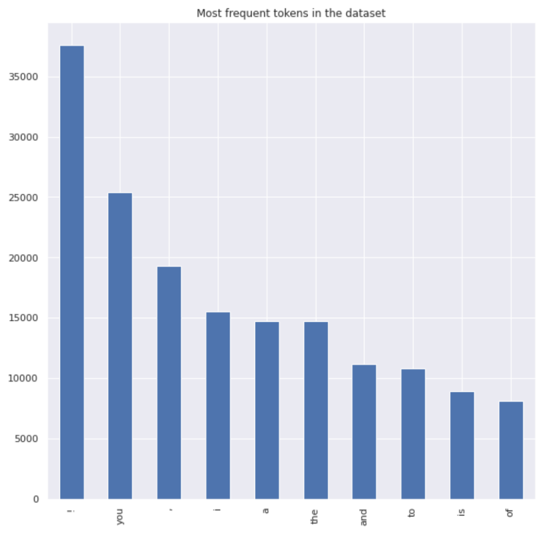
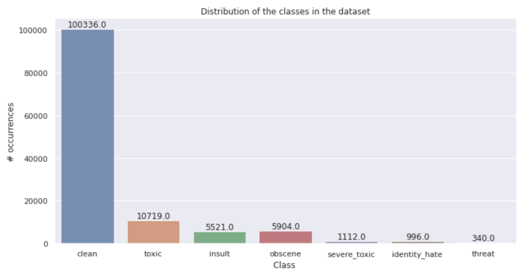
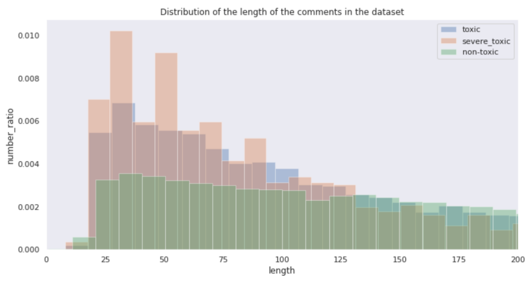


Figure 1: Histograms of the most frequent words in the dataset.

sampling procedure of our training algorithm by assigning higher weights to toxic comments to obtain balanced batches.



(a) Class distribution



(b) Length distribution

Figure 2: Distribution of the classes and comments' length in the dataset.

Figure 2b shows the distribution of the length of the comments in the data. We observe that clean comments tend to be longer, which is also totally expected. It is highly uncommon to see insults of 1000 characters..

## 4.2 Binary task

In this section, we report the results obtained on the binary classification task using all the models presented above. In other words, the goal is to determine whether a comment is toxic [2] or not.

### 4.2.1 Baseline models

Table 4 summarizes the results we obtained using very simplistic baselines and TF-IDF bag of words to represent the comment.

One could be tempted to say that the results are not that bad since all the model achieve at least 91% of accuracy. However, remember that the data is heavily skewed. Indeed, since

[2] Toxic in the broad sense, including all the toxicity subclasses of the datset.

Table 4: Results obtained on multiple baselines

| Model | Accuracy | Recall | F1 Score | AUC |
|---|---|---|---|---|
| Naive Bayes | 91.88 | 20.51 | 33.62 | 88.26 |
| Logistic Regression | 95.28 | 58.33 | 71.31 | 97.34 |
| SVM | 95.83 | 69.8 | 77.27 | - |
| Gradient Boosting | 94.44 | 49.3 | 64.24 | 92.87 |
| Random Forest | 95.04 | 56.45 | 69.70 | - |

90% of the comments are clean, then the trivial classifier that always returns "Clean" also achieves 90% of accuracy. Therefore, the interesting statistic to look at is the recall that quantifies how many toxic comments are missed.

We also report the F1 Score [3] and the Area Under the (ROC) Curve (AUC), which is the score that was used to rank the competitors in the Kaggle challenge.

We can see that the Naive Bayes classifier has very poor performance (20% of the toxic comments detected). On the other hand, SVM achieves a surprisingly good score with almost 70% of recall and 77% for the F1 Score while keeping a good global accuracy.

Note that in this part we did not put any effort in class balancing, and this is indeed very clear from the statistics since the classifiers are encouraged to predict "Clean" most of the time.

#### 4.2.2 Neural models

In this subsection, we report the results that were obtained for the binary classification problem using various architectures. We found the binary case easier to analyze because, even-though there is an inherent trade-off between precision and recall [4], one can still rank different models according to a reasonable performance criterion (e.g., the F1 score). On the other hand, analyzing the multiclass (and multi-label) setting can be tricky since each model may favor different classes over the others.

The results we obtained are given in Table 5. At first glance, we can see that MLP and CNN are clearly dominated by recurrent models, which is indeed an empirically established fact. On the other hand, it is surprising to see how well FastText does (82% recall and 74% F1 score). This suggests that adding bi-grams in a MLP architecture can boost the performance, which makes sense since it allows to capture more complex interaction patterns between the words.

Regarding the recurrent models, it seems that LSTM with the attention mechanism and RCNN are performing slightly better than the rest. However, such a difference might be compensated with proper hyper-parameter tuning, or might even be caused by the randomness of the training procedure. Due to computational limitations, we could not afford to average multiple runs and perform proper hyper-parameter search.

Another apparent observation from the table is that character-level models (GRU1 and GRU2 in the bottom) seem to be less efficient than word-level models. Moreover, they required more epochs to converge.

[3]The F1 score is the geometric mean of precision and recall.

[4]and we cannot simply rely on the accuracy metric because the data is heavily skewed

Table 5: Results obtained with various neural network architectures on the binary classification problem.

| **Model** | **Epochs** | **Level** | **Accuracy** | **Loss** | **AUC** | **Recall** | **F1 Score** |
|---|---|---|---|---|---|---|---|
| MLP | 5 | words | 91.88 | 0.211 | 94.01 | 75.34 | 65.37 |
| FastText | 5 | words | 94.30 | 0.183 | 95.27 | 82.43 | 74.63 |
| CNN | 4 | words | 94.43 | 0.151 | 92.35 | 75.78 | 73.84 |
| GRU1 | 5 | words | 94.77 | 0.154 | 96.12 | 80.14 | 75.70 |
| GRU2 | 5 | words | 94.41 | 0.148 | 95.56 | 79.70 | 74.76 |
| LSTM | 2 | words | 94.30 | 0.157 | 95.46 | 83.41 | 74.84 |
| LSTM-ATT | 2 | words | 95.24 | 0.137 | 97.08 | 80.70 | 77.52 |
| RCNN | 3 | words | 94.87 | 0.134 | 96.76 | 81.06 | 76.26 |
| GRU1 | 6 | char | 92.16 | 0.151 | 94.34 | 79.78 | 67.43 |
| GRU2 | 6 | char | 92.33 | 0.174 | 93.76 | 77.93 | 67.38 |
| Transformer | 2 | words | 96.9 | 0.081 | - | 86.3 | 84.16 |

Finally, we see that the transformer significantly outperforms all the models above, with an F1 score of 84.16% even-though we only performed minimal fine-tuning on our dataset without any class balancing. Well, with hundred millions of parameters..

## 4.3 Multi-label task

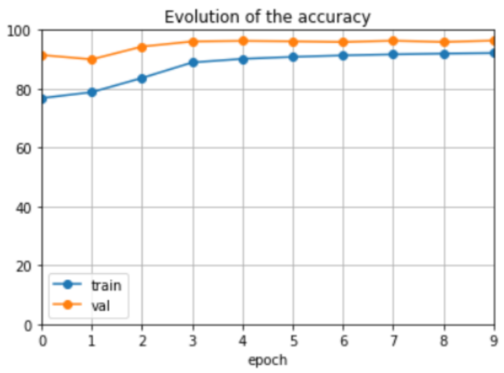
The multi-labeling task is more complex because we need a more precise output for each comment. Since there is a small fraction of comments that fall in each category (recall that 90% of the comments in the dataset are totally clean), we need to explicitly address the skewness issue. To do so, we use two simple strategies: i) we discard an important fraction of the clean comments to get a balanced (smaller) dataset with 50% of toxic comments; ii) we use a weighted sampler during training to favour toxic comments. By doing that, we reduce the accuracy but the recall increases. One could argue that when dealing with toxic comments and showing them to moderator, it is better to have false positive than false negative. Moreover, even-though the first approach might seem brutal, it has the advantage of reducing the computational cost of the training phase. The results obtained using the first (resp. second) method are given in Table 6 (resp. Table 7).

We trained multiple models on this task and we also added a 7th label, which is simply *"clean"* and is equal to 1 when the comment is clean. We thought it could help the learning. For each model, the overall accuracy is around $97-98\%$, as one can see on figure 3. To compute this accuracy, we compute the fraction of correct entries in the prediction [5]. However, this metric is not very informative since most of the labels are "0". That's the reason why we report the Recall for each label on Table 6 and 7. The percentages shown are the proportion of comments having the label '1' that are detected.

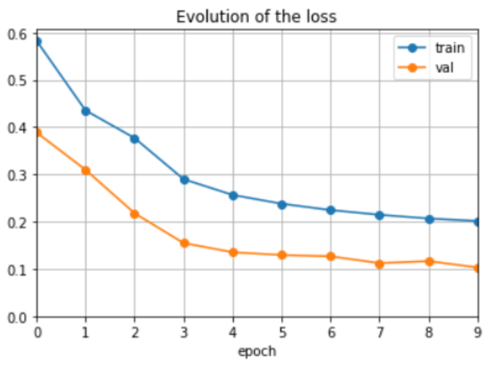
We remark from Table 6 that reducing the size of the dataset generates many false alarms ("Clean" column). On the other hand, this improves the recall of the classifier (in the sense that it is able to detects more toxic comments). Nevertheless, the weighted sampling technique is probably more adapted since it makes use of all the dataset while giving a flexible way to balance the number of elements per class observed during training. We also point-out

[5] For instance, if the target is $[0,0,0,0,0,0,0,0]$ and the prediction is $[1,0,0,0,0,0,0]$, then the accuracy of is 6/7

that LSTMs in this setting performed very poorly, probably because of the lack of data that led to massive overfitting (especially for LSTM with attention).



(a) Evolution of the Accuracy



(b) Evolution of the Loss

Figure 3: Evolution of loss and accuracy during training of the GRU2 classifier on the multi labeling task.

| Model | Clean | Identity hate | Insult | Obscene | Severe toxic | Threat | Toxic |
|---|---|---|---|---|---|---|---|
| MLP | 83 | 0.6 | 53 | 55 | 5 | 0 | 85 |
| CNN | 89 | 0 | 73 | 81 | 7 | 0 | 94 |
| GRU1 | 90 | 0 | 75 | 87 | 50 | 0 | 86 |
| GRU2 | 92 | 10 | 77 | 84 | 28 | 0 | 92 |
| LSTM | 92 | 0 | 73 | 82 | 0 | 0 | 92 |
| LSTM-ATT | 57 | 0 | 2 | 2 | 4 | 0 | 87 |
| RCNN | 91 | 28 | 74 | 82 | 20 | 0 | 92 |

Table 6: Recalls obtained with various neural architectures on the multi-class labeling task when keeping 1/9 of clean comments.

In Figure 3, we can see that loss of the validation set is lower than the one of the training set. This is due to the fact that we are forcing the batches to be balanced, while the validation set is not (it contains 90% of clean comments).

As for the binary case, we see from both Table 6 and Table 7 that the best models are obviously the RNN-based classifier. Moreover, the transformer still outperforms all our models[6]. Also note that the classifier systematically fails for some categories like *identity hate* and *threat*. This is very likely due to the lack of comments falling in these categories (less than 1% for *identity hate* and less than 0.5% for *threat*).

## 4.4 Fine-grained models

We noticed that a lot of toxic users try to change the spelling of toxic words, by concatenating them, or putting random space inside the words, like *"fu cky ou"*. This motivated us to try an approach in which tokens are characters instead of words.

For each model, the fine-grained approach took more training epochs to reach good performance. However, results were a bit below what we obtained with the coarse grained

[6]The transformer was trained without class balancing.

Table 7: Recalls obtained with various neural network architectures on the multi-label classification problem with weighted sampling

| **Model** | **Level** | **Clean** | **Hate** | **Insult** | **Obscene** | **Severe** | **Threat** | **Toxic** |
|---|---|---|---|---|---|---|---|---|
| MLP1 | words | 92.87 | 2.02 | 67.97 | 72.08 | 7.99 | 0.00 | 77.07 |
| FastText | words | 93.38 | 0.34 | 64.49 | 71.67 | 6.51 | 0.00 | 79.89 |
| CNN1 | words | 94.86 | 0.00 | 68.59 | 78.51 | 3.26 | 0.00 | 82.50 |
| GRU1 | words | 95.36 | 33.33 | 66.65 | 80.42 | 33.43 | 15.53 | 77.60 |
| GRU2 | words | 96.81 | 31.42 | 59.24 | 73.12 | 38.99 | 28.16 | 76.82 |
| LSTM | words | 96.69 | 15.25 | 64.45 | 78.73 | 31.75 | 0.00 | 79.88 |
| LSTM-ATT | words | 95.63 | 0.34 | 65.34 | 73.88 | 51.19 | 0.98 | 81.80 |
| RCNN | words | 96.91 | 27.27 | 56.67 | 73.76 | 42.90 | 21.36 | 74.08 |
| GRU1 | char | 93.91 | 29.29 | 65.17 | 78.77 | 39.64 | 18.45 | 75.48 |
| Transformer | words | 98.19 | 79.49 | 47.59 | 86.50 | 0.00 | 74.87 | 73.00 |

approach. We obtained an accuracy of around 90% and a loss of 0.31 using the *GRU2* model for instance.

## 4.5 Bias

It has been previously shown that toxic comment detectors can be biased towards protected minorities [1]. It is mainly due to the fact that a lot of toxic comments contain hate towards these communities, like the ones displayed on Table 8.

| |
|---|
| i 'm not homophobic or anything , but you are fucking gay |
| leave our country we do n't want you arabs here |
| your a faggot and a nigger |
| i am gonna cut off zhanzhao 's fucking indian chinese bastard balls |

Table 8: Some toxic comments from the dataset categorized as *identity hate*

Consequently, if we don't take care of it, our detector can associate hate speech to minorities vocabulary instead of toxicity vocabulary. The Figure 4 show how our simple MLP classifier classify *"I am white"* as clean and *"I am black"* as toxic. The toxicity score of *"I am homosexual"* is even of 0.99.

```
Sample : white people
Toxicity:(p=0.14)
Sample : black people
Toxicity:(p=0.63)
Sample : I am heterosexual
Toxicity:(p=0.69)
Sample : I am homosexual
Toxicity:(p=0.99)
Sample : I am gay
Toxicity:(p=0.98)
```

Figure 4: Outputs of a biaised detector for different sentences.

Even if the bias is less important for complex models, we need to take care of it. We can for instance add some artificial clean comments with protected vocabulary. However, when using pre-trained embeddings like *GLoVe*, there is almost no bias, probably because their construction requires a significant amount of data, and most importantly, more diverse data.

## 5 Conclusion

This project has been an opportunity to put into practice what we have learned during the NLP class. We implemented a lot of models seen in class, from the simple Multi Layer Perceptron to Transformers, in both binary and multilabel classification.

In the end, our best classifier seems to be pretty good, and it would be very interesting to continue working on rationales generation, which are for us a real plus for website moderators.

## References

[1] How automated tools discriminate against black language. 2019.

[2] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[3] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[4] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*, 2016.

[5] S. Lai, L. Xu, K. Liu, and J. Zhao. Recurrent convolutional neural networks for text classification. In *Twenty-ninth AAAI conference on artificial intelligence*, 2015.

[6] T. Lei, R. Barzilay, and T. S. Jaakkola. Rationalizing neural predictions. *CoRR*, abs/1606.04155, 2016.

[7] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[8] J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[9] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.