Advanced Topics in Web Science Seminar

Winter Semester 2018/2019

Final Report

Toxic Comment Detection

Author :
Marouene Zouaoui

Instructor:
Dr. Adamantios Koumpis

February 9, 2019

Contents

List of Figures

1 Abstract

Toxic comment classification has become an active research field with many recently proposed approaches. However, while these approaches address some of the task's challenges others still remain unsolved and directions for further research are needed. To this end, we compare different deep learning and shallow approaches on a new, large comment dataset and propose an ensemble that outperforms all individual models. Further, we validate our findings on a second dataset. The results of the ensemble enable us to perform an extensive error analysis, which reveals open challenges for state-of-the-art methods and directions towards pending future research. These challenges include missing paradigmatic context and inconsistent dataset labels.

2 Introduction

With the recent growth of people on the internet, civil conversations are seeing a decline. "Whatever intelligent observations do lurk there are often drowned out by obscenities, ad-hominem attacks, and off-topic rants." These things are forcing many online platforms which once flourished with intellectual discussions to close the comment sections. To facilitate meaningful conversations on their online platform The New York Times employed full-time moderators who moderate nearly 11,000 comments per day on the selected article(roughly 10% of Times articles). However, for small firms operating people for these tasks might be out of scope. To aid, the Conversation AI team, a research initiative founded by **Jigsaw** and **Google** (both a part of **Alphabet**) are working on tools to help improve online conversation. One area of focus is the study of negative online behaviors, like toxic comments[1] (i.e. comments that are rude, disrespectful or otherwise likely to make someone leave a discussion). So far they have built a range of publicly available models served through the **Perspective API**[2], including toxicity. But the current models[3] still make errors, and they do not allow users to select which types of toxicity they are interested in finding (e.g. some platforms may be fine with profanity, but not with other types of toxic content)..

So in this paper, we will build a multi-headed model that is capable of detecting different types of toxicity like threats, obscenity, insults, and identity-based comments from Wikipedia's talk page edits. On these datasets we propose an ensemble of state-of-the-art classifiers. By analyzing false

negatives and false positives of the ensemble we get insights about open challenges that all of the approaches share.

3 Problem Statement

The exact problem statement was thus given a group of sentences or paragraphs, used as a comment by a user in an online platform, classify it to belong to one or more of the following categories: toxic, severe-toxic, obscene, threat, insult or identity-hate with either approximate probabilities or discrete values (0/1).

4 Methodology

In this section, I will explain the different steps of the methodology that I have followed in order to solve the problem.

4.1 Multilabel vs Multiclass classification ?

As the task was to figure out whether the data belongs to zero, one, or more than one categories out of the six listed above, the first step before working on the problem was to distinguish between multi-label and multi-class classification.

In multi-class classification, we have one basic assumption that our data can belong to only one label out of all the labels we have. For example, a given picture of a fruit may be an apple, orange or guava only and not a combination of these.

In multi-label classification, data can belong to more than one label simultaneously. For example, in our case a comment may be toxic, obscene and insulting at the same time. It may also happen that the comment is non-toxic and hence does not belong to any of the six labels.

Hence, I had a multi-label classification problem to solve. The next step was to gain some useful insights from data which would aid further problem solving.

4.2 Pipeline of the solution

The steps that I have followed to solve this problem is as below:

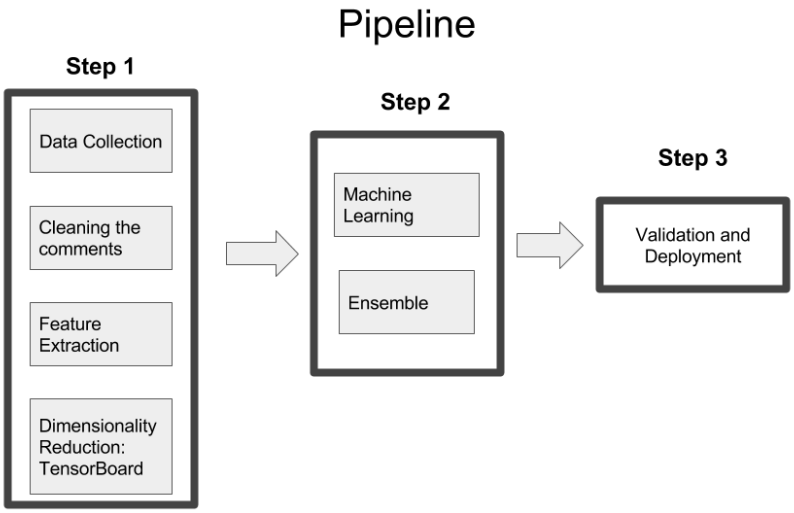

Figure 1: Different steps for solving the problem

4.3 Data Collection

The source of the data-set is the competition [4] that is launched by Jigsaw on Kaggle.

The data-set consists of a large number of Wikipedia comments which have been labeled by human raters for toxic behavior. The types of toxicity are:

- toxic
- severe_toxic
- obscene
- threat
- insult
- identity_hate

These labels are distributed as below :

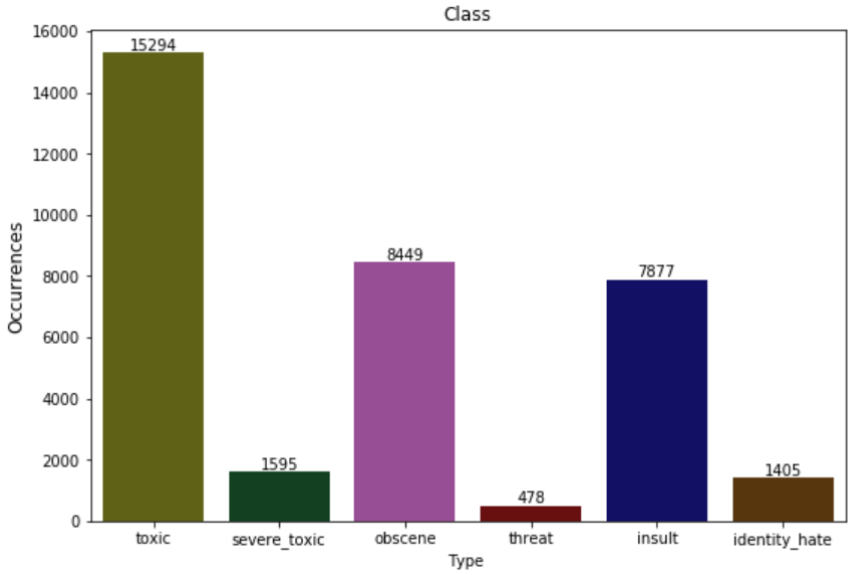

Figure 2: Occurrence of each label in the data-set

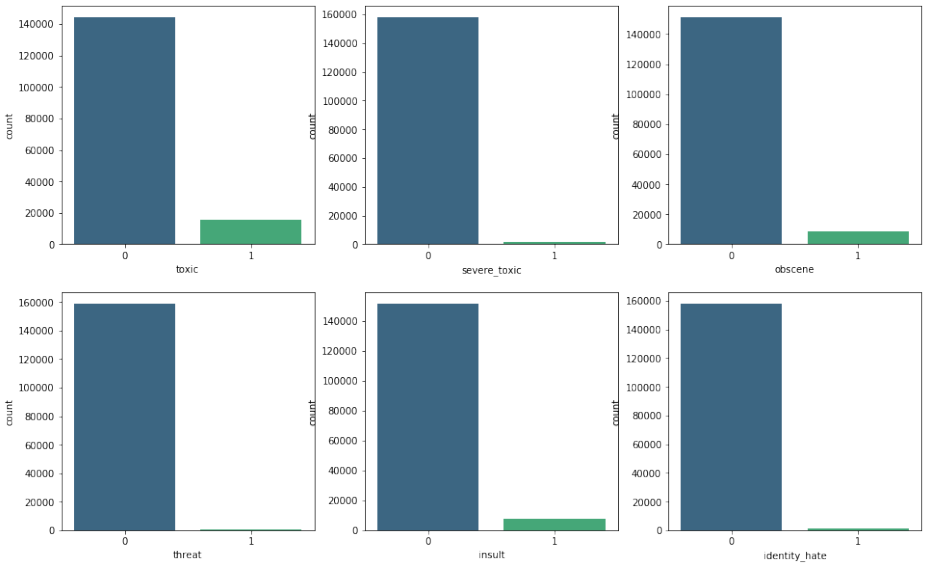

Figure 3: Occurrence of each label comparing to the whole data-set

4.4 Data Cleaning

1. **Preparation for removal of punctuation marks:** I imported the string library comprising all punctuation characters and appended the numeric digits to it, as those were required to be removed too.

2. **Updating the list of stop words :** Stop words are those words that are frequently used in both written and verbal communication and thereby do not have either a positive/negative impact on our statement. E.g. is, this, us, etc.Python has a built-in dictionary of stop words. I used the same and also appended the single letters like 'b', 'c' .... to it, which might be pre-existing or have generated during data preprocessing.

3. **Stemming and Lemmatising :** Stemming is the process of converting inflected/derived words to their word stem or the root form. Basically, a large number of similar origin words are converted to the same word. E.g. words like "stems", "stemmer", "stemming", "stemmed" are based on "stem". This helps in achieving the training process with a better accuracy.

   Lemmatising is the process of grouping together the inflected forms of a word so they can be analyzed as a single item. This is quite similar to stemming in its working but not exactly same. Lemmatising depends on correctly identifying the intended part of speech and meaning of a word in a sentence, as well as within the larger context surrounding that sentence, such as neighboring sentences or even an entire document. I used the word-net library in nltk for this purpose. Stemmer and Lemmatizer were imported from nltk.

4.5 Feature Extraction

Feature engineering is the process of using domain knowledge of the data to create features that make data mining algorithms work. If feature engineering is done correctly, it increases the predictive power of machine learning algorithms by creating features from raw data that facilitate the clustering process. It is the most important art in machine learning which creates the huge difference between a good model and a bad model.

For this project, I tried to use different built-in feature extractors for each approach of classification such as:

- **Tf-idf Vectorizer :** The Tfidf-Vectorizer (Frequency–inverse document frequency vectorizer), is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. In practice, we imported the class *TfidfVectorizer* [5] from *sklearn.feature_extraction.text*.

- **Word2Vec :** Word2Vec is a group of related models that are used to produce word embeddings. I used this technique with deep learning model as a shallow layer.

4.6 Classification Approaches

In this project, I have used two different approaches for solving this problem.

- **Simple Machine Learning Approach :** In this approach, I tried to use simple machine learning algorithm such as Linear Regression after a lot of testing of different algorithms. I got the best result with Logistic Regression with Tf-Idf Vectorizer.

- **Deep Learning Approach :** In this approach, and based on a paper talking about toxic comment detection [7], I tried to use Convolution Neural Networks with Word Word2Vec to include context and position of the words.

4.6.1 Simple Machine Learning Approach

Since we have mutlitple labels, we are dealing with Mulit-label classification model. We can approach this problem in different way, some of which used in this project are:

- **Binary Relevance:** In this labels are treated idependently i.e., for each label, a classifier is trained on the input data. Since, we have six labels, we would have six different classifers

- **Classifier Chains:** In this, the first classifier is trained just on the input data and one label and then each next classifier is trained on the input space and all the previous classifiers in the chain.

  Example: Lets say we have our data represented as,

X	y1	y2	y3	y4
x1	0	1	1	0
x2	1	0	0	0
x3	0	1	0	0

Figure 4: Example Problem

The problem would be transformed into 4 different single label problems as shown below. Yellow colour portion is the input space White coloured portion represent the target/label variable

X	y1
x1	0
x2	1
x3	0

Classifier 1

X	y1	y2
x1	0	1
x2	1	0
x3	0	1

Classifier 2

X	y1	y2	y3
x1	0	1	1
x2	1	0	0
x3	0	1	0

Classifier 3

X	y1	y2	y3	y4
x1	0	1	1	0
x2	1	0	0	0
x3	0	1	0	0

Classifier 4

Figure 5: Solve Problem With Classifier Chain

I tried these two methods with Linear Regression Classifier and Tf-Idf Vectorizer and the change in percentage of the accuracy is represented as below:

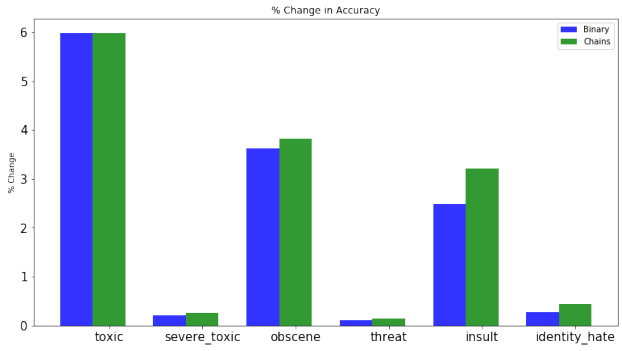

Figure 6: Binary Relevance VS Chain Classifiers Methods

4.6.2 Deep Learning Approach

In this approach, I used CNN as a classifier. CNN have been widely applied to image classification problems due to their inner capability to exploit the two statistical properties that characterize image data, namely 'local stationarity' and 'compositional structure' [8]. Local stationarity structure can be interpreted as the attribute of an image to present dependency between neighboring pixels that is reasonably constant in local image regions. Local stationarity is exploited by the CNNs' convolution operator. We may claim that for text classification problems the original raw data also present the aforementioned statistical properties based on the fact that neighboring words in a sentence present dependency, however, their processing is not straight forward. The components of an image are simply pixels represented by integer values within a specific range. On the other hand the components of a sentence (the words) have to be encoded before fed to the CNN. For this purposed we may use a vocabulary.

- **Word Embeddings :** For the vocabulary, I used the **Glove** [9] **Twitter** vectors with 300 dimensions. Other options were pre-trained vectors from Word2Vec or Fasttext. I tried **Word2Vec**, and like others, Glove worked better for me. I did not get to applying **Fasttext** [10], which is a promising prospect mainly because Fasttext has vectors for 'fractions' of words, and that might be useful for misspelled words (or other OOV terms) commonly found in comments.

- **CNN Model :** The CNN model used consists of one 1-dimensional convolutional layer across the concatenated word embedding for each input comment. The convolutional layer has 128 filters with a kernel size of 5 so that each convolution will consider a window of 5 word embeddings. The next layer is a fully connected layer with 50 units which is then followed by the output layer.

5 Results

I submitted the two solutions on Kaggle. Because Jigsaw already provided some specific criteria for evaluating the solutions. So not only the accuracy is taken into consideration but also some hidden criteria. And after submitting the solution, a score over 100% will be calculated for each submission and will be included in the ranking.

After submitting the two solution and even though I submitted after the challenge is finished, I have got into **top 17%** of the leaderboard.

- I have got a score of 95,55% by submitting the Simple Machine Learning Approach.
- But by submitting the Deep Learning Approach, I have got a score of 97,7% and that's the reason I have got into this good rank.

The results are shown as below:

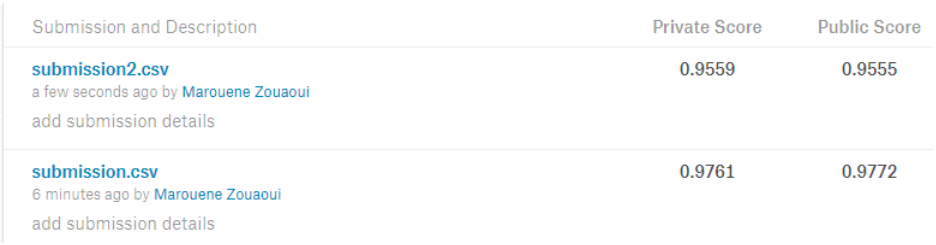

Figure 7: Submission on Kaggle

6 Conclusion and Future Plans

By working with different types of classification models , we could conclude which models may be better suited for the task of toxic comment classification. We found that the best model in our case was the CNN model with word embedding. Although its performance accuracy is marginally better than the Simple Machine Learning model, it could gain a better categorization of toxic comment accuracy.

My future plan is to enhance the Deep Learning approach by using other type of word embeddings like FastText because it helps to handle misspelled words and that would add some accuracy for sure to the solution.

Deploy the code on the internet as a Web Application so that the users reviews and the label(toxic or non toxic) specified by users are stored in database which can be used to train the model on in order to maintain the continuous learning.

References

[1] Online Harassment Pew Research Center: Internet, Science & Tech

[2] Jigsaw Perspective API
`http://www.perspectiveapi.com/`

[3] Get Creative With Perspective API on Github – Jigsaw – Medium

[4] Kaggle: Toxic Comment Classification Challenge

[5] 1.4. support vector machines - scikit-learn 0.19.2 documentation
`scikit-learn.org/../sklearn.feature_extraction.text.TfidfVectorizer.html`

[6] Word2vec
`https://en.wikipedia.org/wiki/Word2vec`

[7] Convolutional Neural Networks for Toxic Comment Classification

[8] Deep Convolutional Networks on Graph-Structured Data

[9] GloVe : Global Vectors for Word Representation

[10] FastText: English word vectors