CPTS 570: Machine Learning Fall 2021
Toxic Comment Classification

Harsh Tyagi
Jasmeet Singh

Abstract

People express themselves freely and without reluctance at online platforms only when they feel comfortable. Any threat of abuse or harassment will make them leave the conversation and prohibit them from participating in any good conversation in future. It is hence a vital requirement for any organization or community to have an automated system which can identify such toxic comments and report/block the same immediately. This problem falls under the category of Natural Language Processing where we try to identify the intention of the speaker, and act accordingly. Research on modelling a solution to this problem has already begun. We personally feel it is important to handle any such nuisance and create a more user-friendly experience regarding online conversation for each one of us. Only then will people enjoy participating in discussions. The idea of our project and the dataset has been taken from kaggle. Under the name of toxic comment classification challenge, it aims to identify and classify online toxic comments. It is being conducted as research by the Conversation AI team, which is an initiative by Jigsaw and Google.

Introduction

The motivation behind it is the multitude of online forums, where-in people participate actively and make comments. Comments can be derogatory, abusing and they should be managed accordingly for wellbeing of all users. Blocking or reporting can be further action after classifying. Our problem is a multi-class and multi-label problem. A multi-label classification problem differs from a multi-class classification problem (in which each sample can only be assigned to one of the many-labels). Hence in our problem each comment can belong to more than one label for ex, a comment can be obscene, toxic and insult, all at the same time.

Some of the **main challenges** associated with the problem are:

a. Sarcastic and Ironic Comments: Detection of this type of text is hard task and increases difficulty of classifying comments because comments states opposite of what is written.
   For example: 'hope you're proud of yourself. Another milestone in idiocy'
b. Toxicity without swear words: In some cases, the toxic or hate comments might not contain any swear words. For example: "Her looks resembles a horse."
c. Spelling mistakes and rare words: There would be some misspelled sentences and the same word would have many misspelled versions which is hard to detect. Which would make it a rare word.
d. Imbalance: Some categories of comments have very less proportion as compared to the complete dataset.

Solution approach:

a. We started our project with exploring the dataset, looking how features are distributed and how much they correlate with each other.
b. Explored various feature(data) engineering techniques to improve accuracy score
c. By using multinomial Naïve Bayes (because it performed well in fortune cookie assignment) created a baseline score.
d. Evaluate the efficiency of various machine learning algorithms and choose the best suited.
e. Choose the best working model and did hyper-parameter and boosting with that for making it perform better.

Problem Setup

The problem in our hand is that online platforms struggle to effectively facilitate conversation, leading many communities to limit or completely shut down user comments. The threat of abuse and harassment online means that many people stop expressing themselves and give up on seeking different opinions.

For this we have tried building a multi-headed model that can detect distinct types of toxicity like threats, obscenity, insults, and identity-based hate. The dataset used is from Wikipedia's talk page edits.

Solution Approach

As I stated earlier, we started our project with exploring the dataset, looking how features are distributed and how much they correlate with each other. In sequence, the Figure (1) shows us the frequency of different labels in the complete dataset, Figure (2) shows us the distribution of comment length size. Then as **suggested by Professor** we detected words and phrases that corresponds to toxic and non-toxic comments and created a word cloud of them as shown in the Figure (3) below. Then in the fourth picture we looked at the correlation between different features.

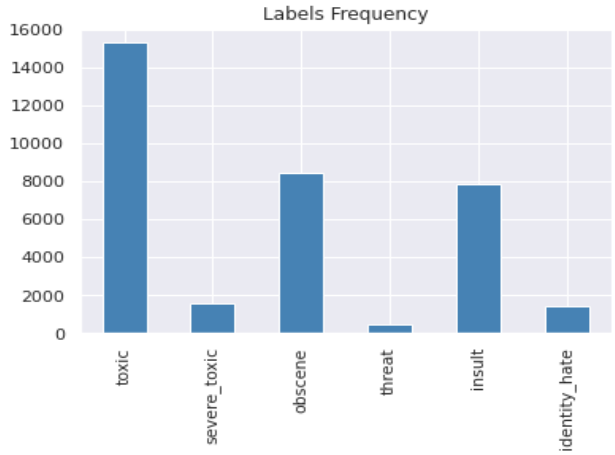
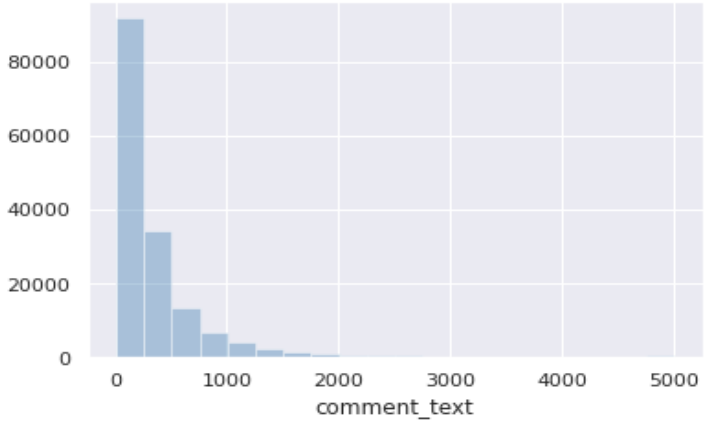


Figure (1): Frequency of Labels



Figure(2): Length of Comments in dataset

Figure (3): Word-Cloud of Toxic words

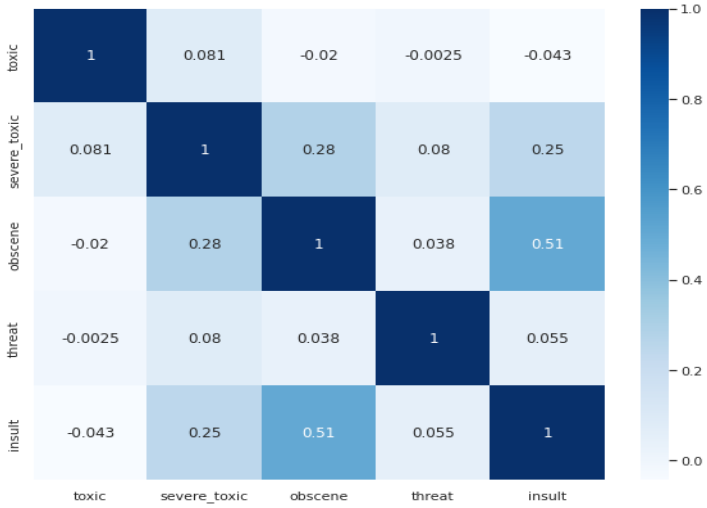


Figure (4): Correlation Matrix of different features

Solution and Refinement

Initially, we created the classifier without any data pre-processing and classified resulting in an exceptionally low evaluation metric, it **helped us understand** that pre-processing part is a key step in creating a classifier, initially we did not address the problem of stop words and different punctuation words that do not account for any clean or toxic comment and needs to be removed. Since all our data are text comments, we used tokenize() function, **removing punctuations** and **special characters**, **lemmatizing** the comments, and filtering out comments with length below 3. After benchmarking between different vectorizers (TFIDFVectorizer and CountVectorizer), we chose **TFIDFVectorizer** which **provided** us with **better performance**. We also tried CountVectorizer. However, it is not performing as well as TFIDF. The TfidfVectorizer is actually a CountVectorizer followed by TfidfTransformer. TfidfTransformer transforms a count matrix to a normalized Term-Frequency or TermFrequency-InverseDocumentFrequency representation. The goal of using TFIDF instead of the raw frequencies of occurrence of a token in each document is to scale down the impact of tokens that occur very frequently in a given corpus and that are hence empirically less informative than features that occur in a small fraction of the training corpus. That's why we can improve the accuracy here. Besides lemmatization, we also tried **stemming** but did not get a better result. Then we further **refined** solution by **hyper-tuning** the best performing classifier and then using **ensemble** method to make the classifier stronger and efficient.

Drawbacks of solution:

- **Spelling mistakes and rare words**: There are misspelled words, and the same word would have many misspelled versions which is hard to detect. They can be analyzed in future improvements with Symmetric Delete Spelling Correction.
- **Sarcasm and Toxicity without swear words** is also a drawback since in these types of sentences, the words used in both toxic and clean comments

Experiments and Results

Dataset that we used for our project is readily available in Kaggle. The dataset consists of the following fields:

1. id: A unique combination of character for each person who wrote that comment.
2. comment_text: A field containing the main comment.
3. toxic: A label with value 0 and 1 (0 for False, 1 for True)
4. severe_toxic: A label with value 0 and 1
5. obscene: A label with value 0 and 1
6. threat: A label with value 0 and 1
7. insult: A label with value 0 and 1
8. identity_hate: A label with value 0 and 1

Initially, we did Data Preprocessing on our dataset where for comment_text field we reduced each comment into its base form and also perform some other normalizations such as converting sentences to lowercase, removing punctuations, removing stopwords, removing all non-ascii characters. This preprocessed text was then fitted into different classifiers to predict weather that belonged to either one or more than one labels. The first five samples of our dataset looks like the following:

	id	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate
0	0000997932d777bf	Explanation\nWhy the edits made under my usern...	0	0	0	0	0	0
1	000103f0d9cfb60f	D'aww! He matches this background colour I'm s...	0	0	0	0	0	0
2	000113f07ec002fd	Hey man, I'm really not trying to edit war. It...	0	0	0	0	0	0
3	0001b41b1c6bb37e	"\nMore\nI can't make any real suggestions on ...	0	0	0	0	0	0
4	0001d958c54c6e35	You, sir, are my hero. Any chance you remember...	0	0	0	0	0	0

Base learners: As mentioned in the project proposal, we implement two basic ML classification algorithms called Logistic Regression and SVM, apart from this we also implemented Naïve-Base Classifier. We performed a comparison between these classifiers using Cross Validation to find out the best classifier which performed best in our text classification.

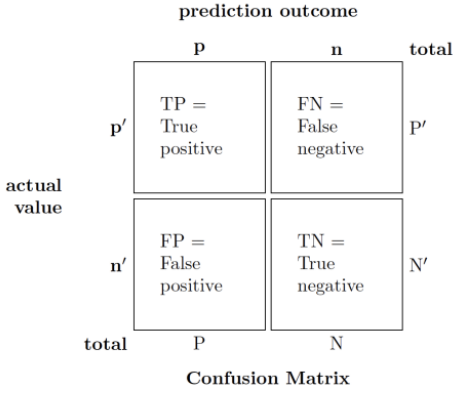
Out[32]:

	Model	Label	Recall	F1
0	MultinomialNB	toxic	0.483066	0.636650
1	MultinomialNB	severe_toxic	0.021336	0.041091
2	MultinomialNB	obscene	0.469168	0.622147
3	MultinomialNB	threat	0.000000	0.000000
4	MultinomialNB	insult	0.367144	0.511521
5	MultinomialNB	identity_hate	0.007837	0.015355
6	LogisticRegression	toxic	0.610500	0.731340
7	LogisticRegression	severe_toxic	0.256395	0.351711
8	LogisticRegression	obscene	0.636997	0.747361
9	LogisticRegression	threat	0.125665	0.210778
10	LogisticRegression	insult	0.523678	0.638237
11	LogisticRegression	identity_hate	0.199970	0.309119
12	LinearSVC	toxic	0.680528	0.759304
13	LinearSVC	severe_toxic	0.267693	0.355258
14	LinearSVC	obscene	0.695230	0.773980
15	LinearSVC	threat	0.223936	0.326625
16	LinearSVC	insult	0.576235	0.662899
17	LinearSVC	identity_hate	0.275400	0.384760

Performance metric(s):

To evaluate our classifier, we use three performance metrics for the course of the project to measure the success of our classifiers. They are Confusion-Matrix, F1-Score (or F score) and Hamming Loss.

1. Confusion Matrix: Confusion Matrix is a useful performance measure for classification task. Each element in the cell of our matrix represents how many items with label i are classified as label j (where i: row, j: Column). In an ideal case, we look for a diagonal in the confusion matrix which insinuates that no item is miss-classified.



Confusion Matrix

2. F1-Score: This is our main metric for measuring the model performance. F1-Score is a metric that combines the precision and recall of a classifier into a single metric by taking the harmonic mean. It is used to measure the accuracy of a classifier and it can also be used to compare the performance of two classifiers.

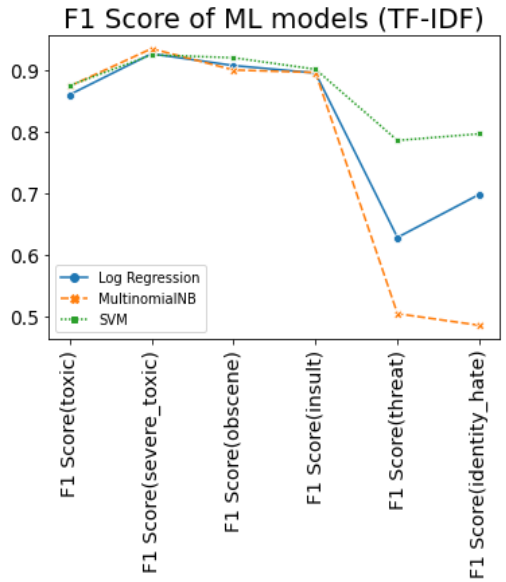
$$F = (1 + \beta)^2 \frac{\text{Precision.recall}}{((\beta^2.\text{Precision}) + \text{recall})}$$

$$F = (1 + \beta)^2 \frac{Precision.recall}{((\beta^2.Precision) + recall)}$$

$$Recall = \frac{TP}{P'}, Precision = \frac{TP}{P'}$$

3. Recall: Recall is a metric that measures how many true positives were recalled (found) by a classifier.
4. Hamming Loss: Hamming loss represents the fraction of wrong labels to fraction to total number of labels.

Since our dataset contains 6 labels for each sample. The problem becomes a multilabel classification problem. We will train each of our classifier six times for each label for a particular sample. Based on this we will decide the performance of each classifier and we will also decide the overall best classifier that performed the best. Then we checked the performance of these classifiers on the test dataset to see if they could perform correct predictions. After predicting we noticed that overall SVM model performed better. We came to this conclusion after looking at the performance metrics, confusion matrix and a plot. The following plot shows the result:



After checking the performance of our baseline models, we then tried to improve the accuracy of our model by performing some hyper-parameter tuning.

After calculating the performance of our classifiers, we noticed that comments that are toxic (and other forms of toxicity) make up less than 10% of the comments in the data. Meaning that the number of toxic comments were far more than the comments with other labels. This created the issue of class imbalance. Now since there is no uniformity, we dealt with class imbalance by manually adjusting class_weight for models. After training our model we noticed that by adjusting class_weight, we got way better results as compared to our basic models. We observe that SVM outperforms Logistic Regression by approximately 1%.

	Model	F1	Recall	Hamming_Loss	Training_Time
0	LogisticRegression	0.947921	0.934050	0.065950	2.137849
1	LinearSVC	0.951508	0.941634	0.058366	7.478050

We did further hyper-parameter tuning by doing a grid search to seek for the **optimal hyperparameters** for our chosen models. To achieve optimal time, tuning each model for each label would be time expensive as we have six different labels. Therefore, we chose to use only the most common label i.e Toxic to tune our hyperparameters. After getting the best parameters for our models we did a comparison between them based on their tuned hyperparameters to see which model performs the best.

	Model	F1	Recall	Hamming_Loss	Traing_Time
0	LinearSVC	0.971706	0.971524	0.028476	5.029654
1	LogisticRegression	0.973227	0.974330	0.025670	13.031119

We further tried to improve the accuracy of models using Ensembling learning that is used to improve the model's results by merging several models and enables the production of a better predictive model performance compared to single models. To ensemble several models, we initially tried some models based on tree boosting, then we used a **Voting** classifier to ensemble one of the boosting models with our models. For tree-based boosting model, we used **AdaBoostClassifier**, **GradientBoostingClassifier**, **XGBClassifier**. We got the following scores for these boosting models:

	Model	F1	Recall	Hamming_Loss	Traing_Time
0	AdaBoostClassifier	0.967605	0.969771	0.030229	50.761416
1	GradientBoostingClassifier	0.969075	0.971748	0.028252	204.453572
2	XGBClassifier	0.967563	0.971790	0.028210	68.613414

We noticed that gradient boosting outperforms other two boosting models. Then we apply the Voting classifier and got following result:

	Model	F1	Recall	Hamming_Loss	Training_Time
0	Ensemble	0.973026	0.974119	0.025881	64.728463

Therefore, we saw that our model worked really well. Compared to our initial basic models where Naïve bias, Logistic Regression and SVM had F1-Scores 30%, 49.33%, 53.88% respectively our final model scored 97.3%. Which is a particularly good improvement.

Finally, this is our first project in Machine Learning, and we got the opportunity to learn a lot from this. First, we got the opportunity to apply different Machine Learning concepts that we learnt in class and how they impact a real-life scenario application, we were able to map our knowledge. We applied different feature engineering techniques like converting to lowercase, removal of stop, **lemmatization**, **stemming** some of which were new to us. We got to know about non-uniformity in data and the impact of it. In addition to that we learnt how can we make a classifier more efficient using **ensemble** and tuning their **hyper-parameter**. We improved our problem-solving skills. Got more familiarity with machine learning libraries in Python.

Conclusions and Future Work

In terms of evaluation metric, SVM performs the best. After tuning hyperparameters and ensembling, we got better results. Besides, SVM trains model the fastest. Referring to interpretability, SVM is also easier to understand. Therefore, we choose SVM as our optimal model. Further, we concluded on our above experiments that **Data Engineering**, **hyperparameter tuning** and **ensembling** are powerful techniques to improve a classifier.

In further analysis of misclassified words, we found that most of the words that were misclassified were either mis-spelled or rarely used words, as it can be seen in the below word cloud. This was the challenge we addressed in the beginning.

Most common words from misclassified

Future Work that can be done to this project:

- Try more ways of vectorizing text data.
- Go deeper on feature engineering : Spelling corrector, Sentiment scores, etc.
- Advanced models (e.g., lightgbm).
- Advanced Ensemble model (e.g., stacking).
- Deep learning model (e.g., LSTM).
- Bayesian Optimization.

Acknowledgments

The project group members would like to thanks following people for the successful completion of project:

- Professor Janardhan Rao Doppa for giving his valuable inputs in the project and his materials and classes that helped us in completing the project.
- StackOverflow and Github for providing necessary help while facing implementation problems.

References

1. Wikipedia : https://en.wikipedia.org/wiki/Multi-label_classification
2. Kaggle challenge page for datasets and ideas : https://www.kaggle.com/c/ jigsaw-toxic-comment-classification-challenge
3. Conversation AI git page : https://conversationai.github.io/
4. Research Paper titled “Multi-label Classification: Problem Transformation methods in Tamil Phoneme classification” : https://ac.els-cdn.com/ S1877050917319440/1-s2.0-S1877050917319440-main.pdf?_tid=eced1a38-f8fa-11e
5. - b8ef-00000aab0f27&acdnat=1515914406_0f244d3e6313bb049c435bf43 504bd52 7. Research Paper titled “Benchmarking Multi-label Classification Algorithms” : http://ceur-ws.org/Vol-1751/AICS_2016_paper_33.pdf
6. . Problem Transformation Methods : https://www.analyticsvidhya.com/blog/ 2017/08/introduction-to-multi-label-classification/