Toxic Comment Classification

Maria Novik
Department of Computer Science
Universität Konstanz
Konstanz, Germany
novik.maria@gmail.com

Ahmet Melih Çelik
Department of Computer Science
Universität Konstanz
Konstanz, Germany
ahmet.celik@ceng.deu.edu.tr

Mohnish Deshpande
Department of Computer Science
Universität Konstanz
Konstanz, Germany
deshpande.mohnish@gmail.com

***Abstract*—Nowadays, communication online strive due to amount of available forums and platforms etc., but online discussion may not always be constructive. By exploring two traditional machine learning models and one deep learning algorithm, this paper presents an approach able to identify and classify toxic online comments. To achieve this task., we propose Logistic Regression and SVM as traditional models, exploiting the canonical bag-of-word model for text representation. As deep learning algorithm we choose LSTM and use words as vectors model for word embedding. We also resolve the class imbalance data issues by employing resampling. Results show that traditional models have high overall accuracy with relatively low cost, compared to our deep learning algorithm.**

***Index Terms*—machine, learning, model, deep, SVM, logistic, regression, LSTM, word, vectorization, embedding, TF-IDF**

I. Introduction

The idea behind is to identify and classify toxic online comments. With an intention of creating a model capable of classifying user comments, we train such traditional machine learning models as support vector machines, logistic regression, and compare their results with deep neural network LSTM.

II. Data Exploration

A. Class Distribution

The dataset used for the project are comments from the Wikipedia talk page edits. Comments are manually annotated or classified using crowd-sourcing. The types of toxic labels used for classification are toxic, severe toxic, obscene, threat, insult, identity hate. The data is of multi-labelled, i.e, a comment can belong to one or more label tags. Any comments that belong to a class has "1" marker in the label column else "0".

B. Stats About Classes

The entire dataset has 178.394 comments marked with these six labels.

TABLE I
Class Distributions

Clean	Toxic	Severe Toxic	Obscene	Insult	Identity Hate	Threat
143.300	15.290	1.595	8.449	7.877	478	1.405

From class distribution statistic, it was observed that the dataset has significant class imbalance. Due to limited computational power available, we decided to use a subset of the entire data for training our models. Since class distribution for the subset samples remains proportional to the original dataset, showing similar imbalance, we applied resampling.

C. What Was Discovered

Here is a list of what was discovered after EDA:

1) The distribution remains pretty much the same on 10.000 and 170.000 samples.
2) There are comments with multiple tagging.
3) Most amount of comments have length of less than 200 words.
4) Top 10 most common words are related to Wikipedia, the origin of the dataset.
5) Some comments are duplicates, or very similar to each other.
6) Some comments contain IP addresses, usernames and some mysterious numbers.
7) 8 out of 10 top bigrams are toxic, which shows that toxic comments tend to have similar words more frequent.

III. Data Preprocessing

In order to train the models, all comments had to be processed and then subsequently transformed into numeric vectors to be fed into the models. The steps involved were:

- converting all the text data into lower case;
- tokenizing the text (breaking down each sentence by splitting into words);
- erasing the punctuation;
- removing stop words such as "and", "but", "the" etc, which don't influence the meaning of a sentence drastically;
- removing words with two or fewer characters like "of", "to" etc
- removing words with fifteen or more characters;
- lemmatizing the word corpus (converting word stems like tenses, infinitives, etc into their original word forms);

An example of preprocessing would be:
Comment before: "Explanation Why the edits made under my username Hardcore Metallica Fan were reverted?"
Comment after preprocessing: "explanation why edits make username hardcore metallica fan revert"

Effects of preprocessing can be seen in Figure 1, where the most frequent words are plotted.

Fig. 1. Effects of data preprocessing. Left side is for raw data, right side is for preprocessed data.

IV. Traditional Machine Learning Models

A. Shaping the task

First, we need to decide either to use multi-class classification or one-vs-all approach. Since the classes are not mutually exclusive, which means that there are comments which can belong to more than 1 class (for example, when a comment is both, toxic and obscene), we need to use one-vs-all approach. In the initial dataset the data is labeled by 6 classes: toxic, severe toxic, obscene, threat, insult and identity hate. For training we took three classes: toxic, obscene and insult. That points out that we would have three logistic regression classifiers.

B. Coping With Imbalance And Splitting the Data

Class imbalance issue has proven to be very impactful in the model training process, so we decided to resample the input dataset. We take 3 classes that has the biggest numbers of comments: ”toxic”, ”insult” and ”obscene”. For each class we take as much comments as we have, only for ”toxic” we use about 11.000 samples instead of all. Figure 2 shows the resulting proportion that we take for training.

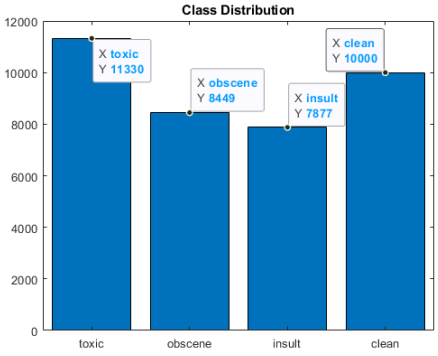

Fig. 2. Number of labled and unlabled comments of all 22184 samples that we use.

We tried to train the model without resampling with the same proportion of classes as in an initial dataset and figured out that it helps to increase the performance.

Having resampled the data, we split data of total 22184 comments into training set and into testing set with the proportion:

- 70% - for training and validation.
- 30% - for testing the model.

C. Text Vectorization and TF-IDF

After resampling the input data, splitting it into training set and testing set, preprocessing, we need to do text vectorization, because machine learning algorithms cannot work with raw text directly. Rather, the text must be converted into vectors of numbers. We use a natural language processing approach, which is called a bag of words model, that is a common technique for extracting features from text to place all of the words that occur in the text in a “bag”. It’s referred to as a “bag” of words because any information about the structure of the sentence is lost. We create a dictionary of words and their occurrence for each document in the corpus. After that, we remove words with low counts, in our case 10, from bag-of-words model, which decreases the number of features. Having this done, our ‘train bag’ has the vocabulary of 4074 words and 15529 comments. These words are features and comments are training examples for our models.

Next, we score the words by how relevant a word is to a document in a collection of documents. For this purpose, TF-IDF (term frequency-inverse document frequency) measure is used. TF-IDF is a statistical measure that does this by multiplying two metrics: how many times a word appears in a document (TF – term frequency), and the inverse document frequency of the word across a set of documents (IDF).

$$TF - IDF = TF * IDF$$

As a result, we have a table with 15529 rows (comments) and 4074 columns (words), cells are TF-IDF weight of each word. Now our data is fully prepared for training.

D. Logistic Regression

First traditional model that we choose is Logistic Regression. To implement this model we use ‘fitclinear’ function from Statistics and Machine Learning Toolbox. We train a linear classification model by using ”fitclinear”, because it reduces computation time on a high-dimensional data set that includes many predictor variables. As a learner we specify Logistic Regression with ”lasso” regularization type.

$$\lambda \sum_{j=1}^{\rho} |\beta_j|$$

We also specify objective function minimization technique as SpaRSA (Sparse Reconstruction by Separable Approximation), Lower the tolerance on the gradient of the objective function to 10^{-8} and set up a regularization term Lambda. We get the optimal value of this hyper-parameter after manual tuning. In next section we discuss how we tune Lambda.

E. Tuning Hyper-Parameter Lambda for Logistic Regression

To train a machine learning model, we must tune hyper-parameters. We implement 5-fold cross-validation to find optimal parameter Lambda for Logistic Regression. We have 5 folds: 4 for training and 1 is test fold. These steps were implemented as in the follwing list.

- Create a set of 15 logarithmically-spaced regularization strengths from 10^{-6} through $10^{-0.5}$, which are used to create a series of different models.
- Cross-validate the models.
- Estimate the cross-validated classification error and plot it.
- Choose the index of lambda that has low classification error.

For each class we estimate optimal lambda separately. We choose lambda by looking at the classification error rates and finding the one with smallest prediction error. Sometimes taking the previous to our "optimal" lambda produce slightly better results, so we decided to choose this one as an optimal.

F. Results for Logistic Regression

To show the results we Plot ROC Curve and also report AUC score to report the total area lying beneath. However, using AUC as a single model performance metric does not give us a full picture. Here we introduce precision and recall scores to explore the model performance further. Figure 3 shows ROC curves and Confusion matrixes for toxic class.

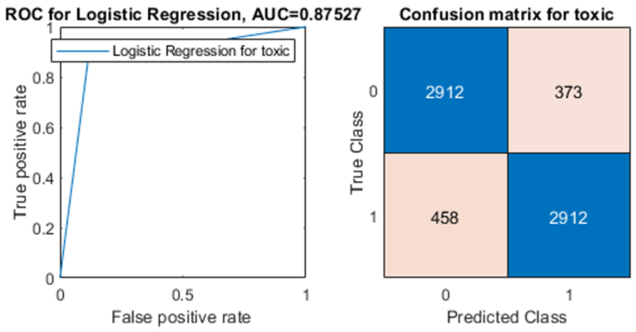

Fig. 3. Classification errors during cross-validation for one class, for Logistic Regression.

Table II shows precision, recall, AUC scores and the chosen value of Lambda. An indication for a descent classifier is that it shows not only high values in both precision and recall but also a balance between them.

TABLE II
LOGISTIC REGRESSION SCORES

Class	Precision	Recall	AUC	Lambda
Toxic	0.88645	0.86409	0.87527	0.00022
Obscene	0.954	0.875	0.86547	0.00056
Insult	0.9108	0.81649	0.76925	0.00056

G. Support Vector Machines

The second model we choose to identify and classify toxic online comments is SVM. For this model we take the same amount of comments, we resample and prepare the data in the same way and use the same vectorization technique as with the Logistic Regression. To implement this model, we use "fitcsvm" function from Statistics and Machine Learning Toolbox. "fitcsvm" trains or cross-validates a support vector machine (SVM) model on moderate-dimensional predictor data set. It supports mapping the predictor data using kernel functions, and sequential minimal optimization (SMO).

We find hyper-parameters that minimize five-fold cross-validation loss by using automatic hyper-parameter optimization. We optimize two parameters: "BoxConstraint" and "KernelScale", and also set the number of iteration to 20. The optimization attempts to minimize the cross-validation loss by varying the parameters.

- "BoxConstraint" is the parameter C in the soft margin SVM.
- "KernelScale" tunes sigma in Gaussian kernel function.

H. Results for SVM

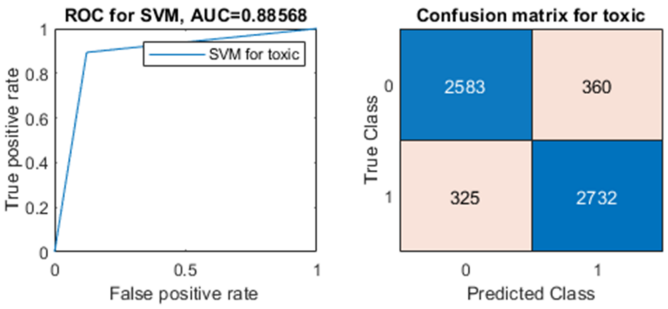

Fig. 4. Classification errors during cross-validation for one class, for Support Vector Machines.

Table III shows the scores for SVM.

TABLE III
SVM SCORES

Class	Precision	Recall	AUC	Box Constraint	Kernel Scale
Toxic	0.87767	0.88823	0.88568	434.90921	916.57752
Obscene	0.94722	0.891988	0.88043	53.3302	110.49374
Insult	0.90742	0.82973	0.78093	34.47461	95.08248

V. DEEP LEARNING ALGORITHM

A. Long Short Term Memory (LSTM)

For this challenge, a deep learning algorithm called Long Short Term Memory (LSTM) was used. LSTM is a special kind of recurrent neural networks. The main difference of LSTM is that, it has feedback connections unlike other neural networks algorithms. Also, it has Long Term memory, which can't be found on a typical RNN. These properties of LSTM

results that the algorithm can remember even older information with its long-term memory.

Figure 5 shows the general architecture of LSTM that was used.

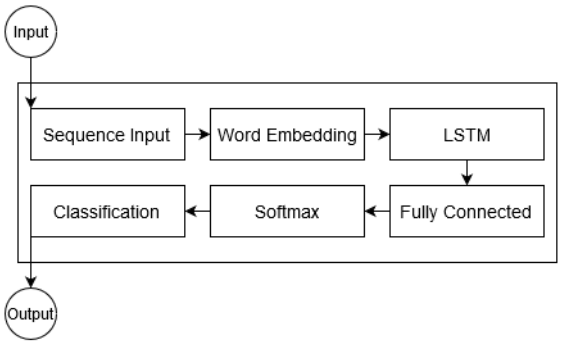

Fig. 5. Architecture of LSTM.

We have used the model shown in the figure above and same data set. Training was done in 10 epochs, with the initial learning rate of 0.01. Number of features we have are the number of words, and we have 33986 words from 10850 comments. The number of features are actually bigger than what can be seen on traditional models, because in traditional models bag-of-words structure were used. As a benefit of that, infrequent words was removed from the bag after preprocessing. With tokenized documents on the other hand, infrequent words weren't removed from the words because there was no way of knowing counts of the words. Also, since converting bag-of-words to documents would result in loss of some data, documents were left with infrequent words.

B. Issuing the Imbalance In the Data

As discussed before, dataset is imlabanced and needed resampling. Resampled data is 10000 entries, however there were at least 15% toxic comments in the dataset to make sure the network can train better. As a future work, with an online computational service or a better system, the whole dataset can be fetched to the network to train.

Figure 6 shows the difference between original data with only first 10000 entries and random selected data with at least %15 identity hate comments included.

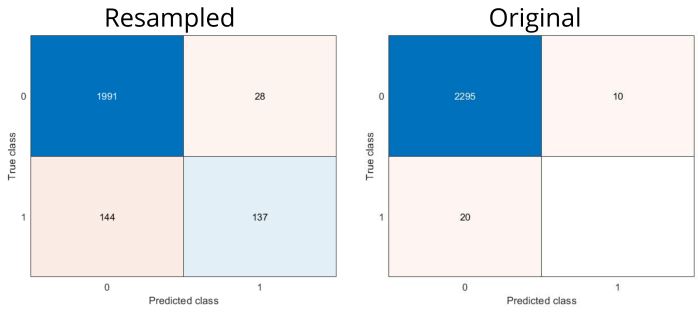

Fig. 6. Difference between resampled and raw data.

It can be seen from the confusion matrices in Figure 6, that the original dataset had only 20 comments classified as identity hate therefore resulting in no correct predictions of identity hate comments. Resampled data on the other hand has better prediction results since it has more entries classified as identity hate in the first place.

C. Results of LSTM

Overall, our LSTM network did not work great and it had classification problems with all of the classes. Results of recall, precision, F1 score and also area under the performance curve are given in Table IV. From the table we can see that there aren't any consistency with the scores and when examined the area under the performance curve, we see that performance of LSTM wasn't really good. We believe that this was caused by the lack of training caused by limited computational power.

TABLE IV
LSTM SCORES

Class	Precision	Recall	AUC	F1 Score
Toxic	0.96368	0.83799	0.53934	0.89645
Severe Toxic	0.99798	0.86144	0.50056	0.90364
Obscene	0.99211	0.82967	0.5123	0.90364
Threat	0.98684	0.96026	0.52467	0.97337
Insult	0.98	0.83088	0.51625	0.8993
Identity Hate	0.98613	0.93255	0.73	0.95859

VI. CONCLUSION

Based on the results, we can say that, SVM and Logistic regression can give better results compared to our RNN modelt. After exploring the dataset, we observed a significant class imbalance. The deep learning model trained on the original dataset performed poorer than expected. A resampled dataset was created to balance out the skewness. In case of LSTM, despite lower accuracy scores, models trained on this resampled dataset, performed significantly better on new, unseen test data (manually selected from reddit). As for traditional models we can say that they have similar average scores, Logistic Regression AUC - 83.67% and SVM AUC - 85.37%. Results show that traditional models have high overall accuracy with relatively low computational cost. SVM performs better, but Logistic Regression model has way more lower computational time.

Source code for this project is available here: https://github.com/netereal/toxic-comment-classification

REFERENCES

[1] Kaggle challenge page. [Online]. Available: https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge
[2] Text Analytics Toolbox Matlab. [Online]. Available: https://www.mathworks.com/products/text-analytics.html
[3] Kaggle notebook about LSTM. [Online]. Available: https://www.kaggle.com/jhoward/improved-lstm-baseline-glove-dropout
[4] 'fitcsvm' documentation and examples. [Online]. Available: https://www.mathworks.com/help/stats/fitcsvm.html
[5] 'fitclinear' documentation and examples. [Online]. Available: https://www.mathworks.com/help/stats/fitclinear.html
[6] Blog post with solved kaggle chalenge. [Online]. Available: https://kyleongmachinelearning.wordpress.com/2018/03/15/toxic-comments-classification-a-project-for-kaggle-competition/