TOXIC TWEET CLASSIFICATION WITH NATURAL LANGUAGE PROCESSING AND MACHINE LEARNING TECHNIQUES

A Technical Paper submitted to the Department of Computer Science
In Partial Fulfillment of the Requirements for the Degree
Bachelor of Science in Computer Science

By
Tanapol Kosolwattana

April 28, 2020

ADVISOR
N. Rich Nguyen, Department of Computer Science

Toxic Tweet Classification with Natural Language Processing and Machine Learning Techniques

--Category: Sentiment Analysis, Text Classification

Arty Kosolwattana

Department of Computer Science,

University of Virginia, USA

tk8mw@virginia.edu

***Abstract*— In recent years, recent research has been experimenting with different approaches to classify texts in sentimental ways. In this project, I will explore an alternative approach to classify tweets using the word embedding toolkit called Global Vector for Representation (GloVe) [1] to extract features and apply a matrix multiplication method to convert the tweet's features into vectors which will be trained in various classifiers, including the Support Vector Model (SVM), random forest classifier, and logistic regression.**

***Keywords*— GloVe Embedding, Matrix multiplication, sentiment analysis, text classification, machine learning, exploratory data analysis**

I. Introduction

Nowadays, in the online world, even though social platforms provide a freedom for users to share their thoughts through posts or messages, some users use these tools to attack other people's feelings with negative words or phrases, leading to cyberbullying, hate speech, and cyber harassment. Therefore, I would like to create a classifier that can provide a baseline result of which class a tweet belongs to. For the method, I propose an alternative approach to build a model such that the implementation on word embedding can provide a baseline result or improve the performance on text classification.

II. Related Work

A. Automated Hate Speech and Offensive Language Detection

Tingyan Xiang et al. propose a method to create a multi-class classification [2]. The authors use the Natural Language Toolkit (NLTK) library [2] to perform data cleaning and preprocessing steps. Then, they extract features of each tweet dataset using the n-gram model and the term frequency and document frequency (tf-idf) model. They use different classifiers to classify the features and return the results in a format of confusion metrics. They also compare the baseline model and parameter-tuned model to see whether they can provide a more accurate result in classifying multi-class tweets.

III. Preliminaries

A. Global Vector for Representation (GloVe)

GloVe is an unsupervised learning algorithm for obtaining vector representations for words [1]. The algorithm generates word embeddings by aggregating global word-word co-occurrence number of words from a matrix and present the result in the form of a dictionary of training words.

In building the GloVe model, there are two steps, including creating a co-occurrence matrix and fitting it into the GloVe algorithm. After training the GloVe object, a dictionary which contains index-value elements for each word is added to complete a model.

In figure 1, it illustrates a linear substructure of the word in a vector space. This application also provides a linear operation for vectors that can be used to find a relationship between words.

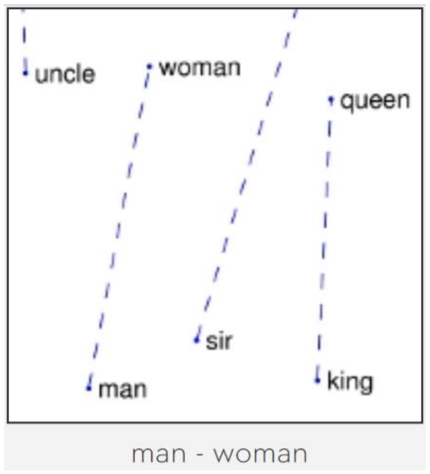
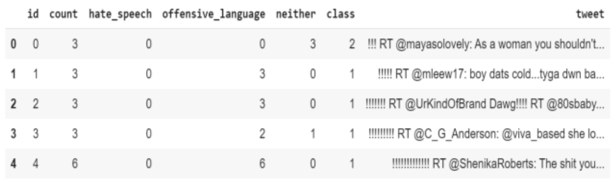


Figure 1: Relationship between words in a vector form [1]

IV. Method

A. Data Gathering

In the capstone project, I gather the data from a data.world website. The dataset is owned and used by Tom Davidson et al. for a published work called Automated Hate Speech Detection and the Problem of Offensive Language [3]. After downloading the dataset, I save them in the file name labeled_data.csv.



	id	count	hate_speech	offensive_language	neither	class	tweet
0	0	3	0	0	3	2	!!! RT @mayasolovely: As a woman you shouldn't...
1	1	3	0	3	0	1	!!!!! RT @mleew17: boy dats cold...tyga dwn ba...
2	2	3	0	3	0	1	!!!!!!! RT @UrKindOfBrand Dawg!!!! RT @80sbaby...
3	3	3	0	2	1	1	!!!!!!!!! RT @C_G_Anderson: @viva_based she lo...
4	4	6	0	6	0	1	!!!!!!!!!!!!! RT @ShenikaRoberts: The shit you...

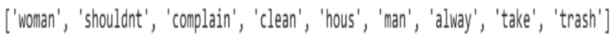
Figure 2: A table represents column labels and top 5 rows of the dataset.

As shown in figure 2, the data column names contain id, count, hate_speech, offensive_language, neither, class, and tweet. The 'count' represents the number of judges who vote to label each tweet in the dataset. The 'hate_speech', 'offensive_language', and 'neither' represent the number of voters who judge a tweet to be hate speech, offensive, and neither. The 'class' represents the label which majority of voters select the type of tweet, and 'tweet' represents a tweet text.

B. Text Cleaning and Preprocessing

After obtaining the tweet data from the data.world website, I conduct text cleaning and preprocessing steps to remove the unnecessary words that are not considered in classification and to increase the performance of GloVe embedding.

In a text cleaning process, I tokenize each tweet into a list of words and turn all letters in each word into lowercase and convert numbers into words and remove punctuation and white space (in front or at the end of a word). In a text preprocessing process, I stem and lemmatize words so that each word is in the most original form. Then, I convert them back into the list of words so that each list represents each tweet in the dataset. The example of the list is shown in figure 3.

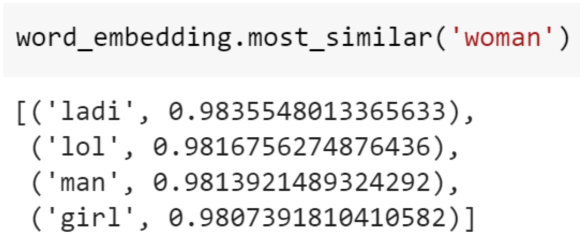


```
['woman', 'shouldnt', 'complain', 'clean', 'hous', 'man', 'alway', 'take', 'trash']
```

Figure 3: An example of the list of words that is used in GloVe embedding

C. GloVe Embedding

After cleaning the dataset by transforming words in each tweet into a more usable form of the list words, I apply GloVe embedding to extract features from each word. During the step, I set up a dimension of vector to 50. The dimension number determines the number of features inside each word in a GloVe dictionary. As mentioned in the preliminaries section, the word trained by GloVe embedding can be used to conduct a word-analogy test to find the closest words that relate to it. In figure 4, the function most_similar() provides the top 4 words that are considered to be most related to the selected word. For each row of the GloVe matrix, it contains the numbers that represent the feature values that are obtained from a training process. They are used to find features in a tweet dataset which later on is converted to a feature matrix.



```
word_embedding.most_similar('woman')
```

```
[('ladi', 0.9835548013365633),
 ('lol', 0.9816756274876436),
 ('man', 0.9813921489324292),
 ('girl', 0.9807391810410582)]
```

Figure 4: The word similarity for 'woman'

D. Matrix Multiplication

First, for each word in a GloVe dictionary, I check if it is in each tweet dataset (which is now a list of words). If it is in a tweet, I put 1 in a vector. If not, I put 0 in a vector. So, the resulting vector for each tweet after doing this step is the vector that contains only 0 and 1. I combine all resulting vectors into a feature matrix with the dimension of n x w by which n stands for the number of tweets in the dataset and w stands for the number of words in GloVe matrix. Then, I multiple a feature matrix with a GloVe matrix which has the dimension of w x d. ('d' stands for the number of features of a vector in a GloVe matrix) The result has the dimension of n x d which in this case is 24783 x 50.

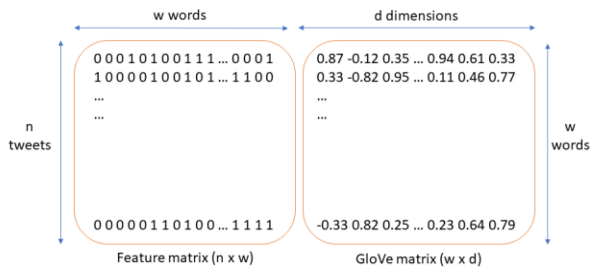


Figure 5: A diagram demonstrates the multiplication between two matrices.

E. Applying Classifiers

After obtaining the matrix that has a dimension of 24783 x 50, I delete the index values in the 1st column and save it in the file name mod_data.csv. Before training, I use a Panda dataframe toolkit to read that file and put a feature matrix as the training value 'x'. For the training value 'y', I read the data from a column 'class' in labeled_data.csv. Then, I train the (x, y) in different classifiers, including the Support Vector Model (SVM), random forest classifier, and logistic regression. After training, I apply some parameter tuning to improve the accuracy of each classifier. The result of the classifier will be explained in the result section.

V. Results

	Accuracy	Precision	Recall	F-Score
Linear SVM	0.8605	0.8294	0.8605	0.8363
Logistic Regression	0.8571	0.8287	0.8571	0.8358
Random Forest	0.8456	0.8209	0.8456	0.8307

Figure 6: Table of evaluation metrics for each model

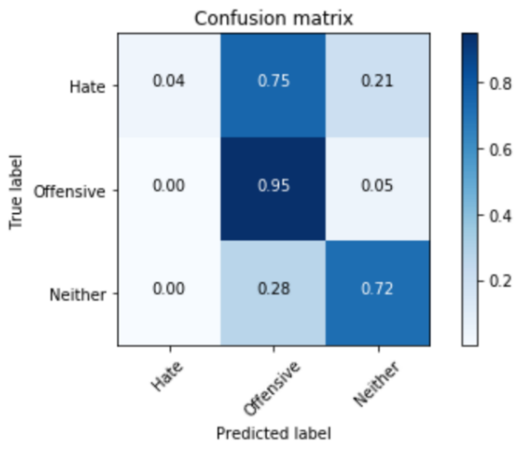
As shown in figure 6, it appears that Linear SVM has the best performance for accuracy at 86.05% and precision at 82.94%. This means this model correctly classifies around 86.05 % of all tweet classification and 82.94% of all toxic tweets are correctly classified. Linear SVM has given the most satisfactory result since it can perform well when there are many features which in this case are features in each tweet of the dataset. Also, the mapping to a higher dimensional space does not really improve the performance [5].

	Accuracy	Precision	Recall	F-Score
Linear SVM	0.8655	0.8479	0.8655	0.8418
Logistic Regression	0.8571	0.8280	0.8571	0.8362
Random Forest	0.8544	0.8293	0.8544	0.8321

Figure 7: Table of evaluation metrics for each model after tuning parameters

After getting the baseline results, I tune some parameters in each classifier. For linear SVM, I change the loss function from squared hinge to hinge. For logistic Regression, I change the penalty to 'None' and change the solver to 'lbfgs' to handle multinomial loss for multiclass problems. For Random Forest, I add the max depth from none to 10. As illustrated in figure 7, the result shows a better performance for Linear SVM and Random Forest. However, for

Logistic Regression, there is a very slight



improvement (about 0.4%) on f-score.

Figure 8: The confusion matrix for the Random Forest Classifier [2]

Another result that I look for is a confusion matrix since it can determine the performance of a classification model on a set of test data for which the true values are known [6]. In figure 8, it shows that the Random Forest Classifier have the best performance for determining the offensive case (0.95) and neither (0.72) case. However, it still has a poor performance on hate speech case (0.04) with 0.75 on a Hate-Offensive grid. This means that the classifier cannot accurately separate the hate type from offensive type.

VI. LIMITATIONS & EXTENSIONS

A. Limitations

The matrix multiplication step requires a large amount of time of processing since it takes two large matrices which are a feature matrix (converted from tweet dataset) and co-occurrence matrix from GloVe embedding step. Also, there might be some limitations of classifying processes regarding the tone of language whether it is trolling or sarcastic. Tweets with these tones can be considered either offensive or non-offensive when labeling manually, but hard to identify when training with classifiers. Therefore, there might be a method to assign the tones of some words before the training in classifiers.

B. Extensions

The experiment can be extended to more sentiment analysis after training with GloVe algorithm. Also, there can be an improvement in a text preprocessing process so that it can improve a performance in both GloVe training period and classifier training period.

VII. CONCLUSION

In the capstone project, it provides a baseline result which can be improved with more systematic ways of text processing. With the objective of being an alternative approach, I hope this project can inspire audience to try different methods for solving the classification and text mining problems. Also, I hope that this project can raise the awareness of people to mitigate the toxicity and create healthy and hate-free online communities.

REFERENCES

[1] J. Pennington, R. Socher, and C. Manning, “Glove: Global Vectors for Word Representation,” in Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Doha, Qatar, 2014, pp. 1532–1543.

[2] “yueqiusun/Twitter-hate-speech-classifier,” GitHub. [Online]. Available: https://github.com/yueqiusun/Twitter-hate-speech-classifier. [Accessed: 11-Dec-2019].

[3] “Natural Language Toolkit — NLTK 3.4.5 documentation.” [Online]. Available: https://www.nltk.org/. [Accessed: 11-Dec-2019].

[4] T. Davidson, “Hate Speech and Offensive Language - dataset by thomasrdavidson,” data.world. [Online]. Available: https://data.world/thomasrdavidson/hate-speech-and-offensive-language. [Accessed: 11-Dec-2019].

[5] C.-W. Hsu, C.-C. Chang, and C.-J. Lin, “A Practical Guide to Support Vector Classification,” p. 16.

[6] “Confusion Matrix in Machine Learning,” GeeksforGeeks, 15-Oct-2017.

[7] Alexandre KOWALCZYK, “Linear Kernel: Why is it recommended for text classification?” SVM Tutorial, 19-Oct-2014.

[8] J. S. Chawla, “Word Vectorization using GloVe,” Medium, 24-Apr-2018. [Online]. Available: https://medium.com/@japneet121/word-vectorization-using-glove-76919685ee0b. [Accessed: 04-Dec-2019].

[9] D. Monsters, “Text Preprocessing in Python: Steps, Tools, and Examples,” Medium, 15-Oct-2018. [Online]. Available: https://medium.com/@datamonsters/text-preprocessing-in-python-steps-tools-and-examples-bf025f872908. [Accessed: 11-Dec-2019].

[10] R. Vickery, “Detecting Hate Speech in Tweets: Natural Language Processing in Python for Beginners,” Medium, 11-Feb-2019. [Online]. Available: https://medium.com/vickdata/detecting-hate-speech-in-tweets-natural-language-processing-in-python-for-beginners-4e591952223. [Accessed: 11-Dec-2019].