ISyE 6740 - Fall 2022
Project Report

Team Member Names: Syed Mehlial Hassan Kazmi.

Project Title: Detecting Social Media toxicity using Word2vec and Ensemble learning.

1 **Problem Statement** Social media is an integral part of contemporary world. Individuals use it to share information, to have conversations or to create content. As one can share, say or comment on anything so along with positive material there is a lot of negativity on social media as well. According to numerous studies negative content, comments and conversations can have an adverse effect on individuals. This is imperative for major social media firms to quickly identify this type of content and remove it so that social media is a safe space for everyone. Hence, here we will try to identify whether a certain comment in English Wikipedia talks data-set is toxic (negative contribution to conversation) or not. We would employ different tools such as Word2Vec, SMOTE, classifiers like Logistic regression, Naïve Bayes, KNN and ensemble learning to achieve it.

2 **Data Source** The data has been taken from **Wikipedia Detox** project. This project has multiple data sets but we will be using toxicity annotated data-set. we have around 160k labeled comments from English Wikipedia by approximately 10 annotators via Crowdflower on a spectrum of how toxic the comment is (perceived as likely to make people want to leave the discussion) to how healthy to conversation the contribution is. The data can be found here. We have two set of data *toxicity_annotations.tsv* **(1)** and *toxicity_annotated_comments.tsv* **(2)** details of which can be found below. Schema for ***toxicity_annotations.tsv***:

- **rev_id**: MediaWiki revision id of the edit that added the comment to a talk page (i.e. discussion).
- **worker_id**: Anonymized crowd-worker id.
- **toxicity_score**: Categorical variable ranging from very toxic (-2), to neutral (0), to very healthy (2)
- **toxicity**: indicator variable for whether the worker thought the comment is toxic. The annotation takes on the value 1 if the worker considered the comment toxic (i.e worker gave a **toxicity_score** less than 0) and value 0 if the worker considered the comment neutral or healthy (i.e worker gave a **toxicity_score** greater or equal to 0). Takes on values in {0, 1}.

Schema for ***toxicity_annotated_comments.tsv***:

- **rev_id**: MediaWiki revision id of the edit that added the comment to a talk page (i.e. discussion).
- **comment**: Comment text. Consists of the concatenation of content added during a revision/edit of a talk page. MediaWiki markup and HTML have been stripped out. To simplify tsv parsing, \n has been mapped to NEWLINE_TOKEN, \t has been mapped to TAB_TOKEN and " has been mapped to '.
- **year**: The year the comment was posted in.
- **logged_in**: Indicator for whether the user who made the comment was logged in. Takes on values in 0, 1.
- **ns**: Namespace of the discussion page the comment was made in. Takes on values in {user, article}.

– **sample**: Indicates whether the comment came via random sampling of all comments, or whether it came from random sampling of the 5 comments around a block event for violating WP:npa or WP:HA. Takes on values in {random, blocked}.

– **split**: For model building in our paper we split comments into train, dev and test sets. Takes on values in train, dev, test.

3 **Methodology**:

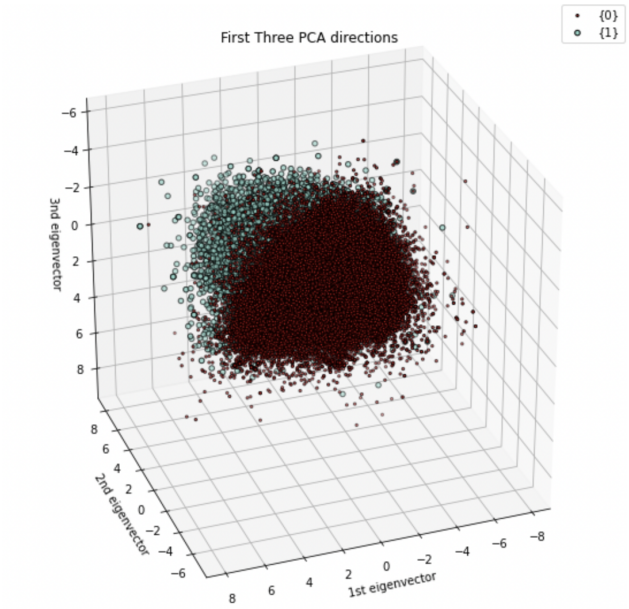
a ***Data Preparation and Preprocessing*** : We have two data sets and we joined both on rev_id. As we had multiple annotaters and each comment was labelled multiple times, to get one label for every comment we took the mean of the toxicity_score column for every comment. If the mean was less than 0 this was labelled as a toxic comment and vice versa.
Furthermore, we removed all non-alphanumeric characters such as new line and tab token, and stop-words from our comment column. As we would be using words in comments as features for our models and these stop words are commonly used words which would not be informative for polarity identification hence we removed them. To get an overview of our data we can take a look at Wordcloud below:

Seems like that words such as article, talk, people and page were most frequently used in comments. This is an important observation as these words would be prime features in determining the polarity of a certain comment.

b ***Word2Vec*** Since most of the ML models take numerical inputs we will be using Word2Vec to convert our comments in to a numerical representation. These numerical representations are called word embeddings and are formally defined as technique for feature learning, in which the words of vocabulary are converted to vectors of continuous real numbers with low dimensions. It is a distributional vector representation of the words and it is also called semantic vector space since it captures both the semantic (meaning) and syntactic (structure) information of words from the context in which they are used.
Word2Vec is a powerful tool developed by Google which efficiently computes word embeddings. Word2vec has two neural network architectures: continuous bag-of-words (CBOW) and skip-gram. In CBOW, the target word (center word) is predicted based on the surrounding words. The context is represented by multiple words, and to predict the missing target word, the sequence of context words must be given,then try to predict the missing word in the context according to the window size. This can be applied to different sizes of windows. Moreover the idea of skip-gram

is that in each estimation step you are taking one word as a center word, and you are going to try and predict words in its context out to some window size, and the model is going to define a probability distribution that is the probability of a word appearing in the context given this center word. And we are going to choose vector representation of words so we can try and maximize that probability distribution.
Initially we thought of using a pre-trained Google news Word2vec model but there was one caveat. If Word2vec model doesnt have a word in its training then we can not have its embedding. Pre-trained Word2vec model had around 3M words and our data had around 5M words but common words in both of them were just around 60K and we were losing a lot of information, hence, instead of using the pre-trained model we trained Word2vec (with window size 5) model by ourselves, and created 300 dimensional vector for every word. Since we wanted to get the embedding for the whole sentence rather than just the word so we took mean of all the embeddings of all the words in a single sentence and deduced the embeddings.

c ***PCA*** After creating the 300 dimensional embeddings we used PCA to extract the first three principal components and plotted them to gauge if we are able to segregate the two classes based on the words embeddings. We can find the plot below.



It can be easily seen that two classes are being cleanly separated in three dimensional space. Using word2vec is working as intended those comments that are semantically and syntactic similar are plotted closer to each other. We had 300 dimensional feature vector for our models but these were a lot of features for training hence we employed PCA to find the first 20 principal components and used those as features for our models.

d ***SMOTE***: As we can see above that there is a clear class imbalance. We had 130140 instances of '0' class and 29450 instances of '1' class. Therefore we would use SMOTE to cater this problem. SMOTE over-samples the minority classes by adding synthetic samples based on feature-space similarities between existing minority examples. For each example x_i, where $x_i \in S_{min(minority\_class)}$, it considers its k-nearest neighbors. SMOTE computes the difference between the sample under consideration and its nearest neighbor and multiplies it by a random number between zero and one. Most important parameter for SMOTE is the number of nearest neighbors. For our experimentation we set it to 5. Moreover, for the purposes of this project we would try models both with and without SMOTE and compare the performance.

e ***Models***: We had two sets of data, imbalanced and balanced. We used 80/20 split for both and reserved 80% of our data for training and 20% for testing. We used the following models:

* **Logistic Regression**: Logistic regression estimates the probability of an event occurring, such as whether the comment is toxic or not, based on a given dataset of independent variables. Since the outcome is a probability, the dependent variable is bounded between 0 and 1. In logistic regression, a logit transformation is applied on the odds—that is, the probability of success divided by the probability of failure. We can set a threshold and if the predicted probability is greater than the threshold then that comment would fall into one class else the other. Since we had already selected features using PCA so here we would just try the base regression without any regularization.
* **Naïve Bayes**: Naïve Bayes Classifier is a probability-based classifier that utilizes Bayes' theorem to calculate the classification problem. The sample belongs to the class with the highest calculated conditional probability. However, this classifier assumes that all features shall be independent so as to apply Bayes' theorem. Since our features would be vector representations of our words then this is a fair assumption to make.
* **KNN**: K-Nearest Neighbor Classifier is a classifier that is based on the distance of the current sample to its K nearest neighbors. The class that the sample belongs to depends on the most common class of its neighbors. We would try it with a different number of K's [3,5,7] and select the one which performs the best.
* **Randomforest**: Random Forest Classifier is created as a collection of decision trees. The benefit of RF is that it can overcome the overfitting problem in decision trees by using multiple trees. Therefore, the prediction from RF is the average value from different underlining randomly chosen decision trees. In classification, it would be the mode of all the underlying chosen decision trees. We would play with different parameters of the random forest to choose the best one. We ended up choosing the RF with hyper-paramaters of max_depth = 12 and n_estimators = 300 as it gave the best results among other random forests.
* **Adaboost**: Ada-boost combines multiple classifiers to increase the accuracy of classifiers. AdaBoost is an iterative ensemble method. AdaBoost classifier builds a strong classifier by combining multiple poorly performing classifiers so that you will get high accurate strong classifier. The basic concept behind Adaboost is to set the weights of classifiers and train the data sample in each iteration such that it ensures accurate predictions of unusual observations. Any machine learning algorithm can be used as a base classifier if it accepts weights on the training set. For the purposes of the project we would just use the decision tree as the default weak learner. For this model we tuned our hyper-parameter of n_estimators to a number of 545.

4 **Evaluation and Final Results** We can find the results below. First table have results for imbalanced data-set while the second one have results for the balanced data-set. We evaluated our models across 5 metrics.

Accuracy:

$$\frac{TP + TN}{TP + FP + TN + FN} \times 100$$

Precision:

$$\frac{TP}{TP + FP} \times 100$$

Recall:

$$\frac{TP}{TP + FN} \times 100$$

F1 score

$$2 \times \frac{Precision \times Recall}{Precision + Recall}$$

AUC ROC stands for Receiver Operating Characteristics. It is a graph of True Positive Rate (TPR) vs False Positive Rate(FPR). Using ROC we would find AUC. The Area Under the Curve (AUC) is the measure of the ability of a classifier to distinguish between classes and is used as a summary of the ROC curve. The higher the AUC, the better the performance of the model at distinguishing between the positive and negative classes. We would just report AUC.

Results					
Models	Accuracy	Precision	Recall	F1 Score	AUC
Logistic Regression	87.4%	77.2%	45.7%	57.4%	88.4%
KNN ($k = 3$)	86.11%	66.0%	51.3%	57.9%	80.0%
KNN ($k = 5$)	86.9%	70.6%	50.5%	58.8%	82.7%
KNN ($k = 7$)	87.3%	72.9%	49.99%	59.3%	84.2%
Gaussian Naïve Bayes	87.3%	83.8%	38.8%	53.0%	85.1%
Random Forest	87.8%	84.9%	41.6%	55.8%	88.3%
AdaBoost	87.7%	73.6%	52.3%	61.2%	88.0%

Results with SMOTE					
Models	Accuracy	Precision	Recall	F1 Score	AUC
Logistic Regression	80.8%	81.9%	79.2%	80.5%	89.1%
KNN ($k = 3$)	82.6%	77.6%	91.8%	84.1%	89.2%
KNN ($k = 5$)	82.1%	77.4%	90.6%	83.5%	90.1%
KNN ($k = 7$)	81.9%	77.5%	89.8%	83.2%	90.33%
Gaussian Naïve Bayes	72%	67.8%	83.8%	72.9%	83.5%
Random Forest	83.3%	83.9%	82.5%	83.2%	91.5%
AdaBoost	81.6%	81.9%	81.2%	81.6%	89.8%

For the results without SMOTE, accuracy is not a good measure since the data is imbalanced. F1 score and AUC would be more informative measures since former considers both precision and recall and latter is a summary of plot of false positive rate vs true positive rate. In AUC space all models performed comparatively but AdaBoost stood out in terms of F1 score.
While after applying SMOTE performance of all the models improved in terms of F1 score and Recall. Though, if we look holistically random forest really stands out. Overall ensemble models out-performed the base models.

5 **Conclusion** We used novel techniques for detecting toxicity on social media just by using the words employed in the comments. We experimented with different models and concluded that ensemble models generally perform better than the base classifiers. We also noted that SMOTE really helps with model training. We practically observed that imbalanced data-set can have an adverse impact on model training. We would like to continue to experiment with different ML models, SMOTE techniques, fine tuning Word2Vec models and also try out things like Doc2Vec for creating document and sentence embeddings rather than just using the word embeddings and deducing the sentence embeddings.

6 **References**

- Al-Saqqa, Samar Awajan, Arafat. (2019). The Use of Word2vec Model in Sentiment Analysis: A Survey. 10.1145/3388218.3388229.
- Tarik Sabri, Omar El Beggar, Mohamed Kissi, Comparative study of Arabic text classification using feature vectorization methods, Procedia Computer Science, Volume 198, 2022, Pages 269-275, ISSN 1877-0509, https://doi.org/10.1016/j.procs.2021.12.239.

– Al-Azani, Sadam El-Alfy, El-Sayed. (2017). Using Word Embedding and Ensemble Learning for Highly Imbalanced Data Sentiment Analysis in Short Arabic Text. Procedia Computer Science. 109. 359-366. 10.1016/j.procs.2017.05.365.