Text Mining and Sentiment Analysis of Twitter Tweets
ECE 569A Artificial Intelligence - Project Report
By Suseela Pattamatta
V00949525

August 29, 2020

1 Problem statement and motivation

The objective of this project/experiment is to detect hate speech in tweets. Tweets that contain either racist or sexist sentiments are classified as hate speech related tweets. In the present global situation, social media plays not only a powerful but also an influential role. Hate crimes are not new in the society. Suspects in several recent hate-crimes terror attacks had an extensive social media history of hate related posts, suggesting that social media contributes to their radicalization[1]. The abundance of online forums provides users ample resources to express themselves freely and anonymously. Freedom of speech is misconstrued, thereby leading to the expression of hate through these online forums. Popular online social media websites like Facebook, YouTube, Twitter, etc consider hate speech harmful and have rules in place to remove such content.

COVID 19 situation provides a perfect opportunity for some users to express racial intolerance. Detecting hate speech is a very challenging task. While some content can be considered hate speech by some, the same content can be considered safe by others. To understand this complexity, I chose text mining for sentiment analysis and classification of Twitter feeds to detect hate speech.

2 Related work

Several papers have been published about hate speech detection in online content. The experiments mentioned in these papers are conducted using Natural Language Processing (NLP) and classification techniques like Support Vectore Machines (SVMs), Recurrent Neural Networks (RNNs) and Long Short Term Memory (LSTM) [2][3][4]

3 Problem formulation

The dataset constists of a training sample of tweets and labels, where label '1' denotes a sexist/racist tweet and label '0' denotes a non-sexist/non-racist tweet. The objective is to predict the accuracy of the classification of tweets using different Machine learning techniques. The following are the details of the dataset used.

The dataset is available at :
https://www.kaggle.com/arkhoshghalb/twitter-sentiment-analysis-hatred-speech
The datset used is train.csv

The dataset contains 31,963 samples.
The dataset is split into training and testing datsets in the ratio of 80/20.

4 Evaluation methodology

The methodology followed for this project is divided into three parts:

1. Text analysis using NLTK

   Importing data into Pandas dataframe
   Converting all tweets to lower case
   Data cleanup - Including removal of certain symbols like @,# and sentences starting like https, removal of certain garbage words
   Tokenization
   Remove stopwords
   Join the tokens of words into corresponding tweets

In Text classification, we have texts and their corresponding labels. Since we can't directly use text in our model, converting the text into vectors is preferred.

2. Perform Sentiment analysis using text classification

   Feature generation using ***Bag of words*** using ***CountVectorize***
   Feature generation using ***word2vec*** word embeddings
   Upsampling
   Model building and evaluation using Bag of words and word2vec models

3. Different techniques are used to analyze the classification

   Random forest
   Naive Bayes
   Logistic regression
   Support Vector Machine (SVM)

The dataset used for my experiment(s) is available on kaggle.com[5].

4.1 Part 1 : Data pre-processing and cleanup

In part 1, the experiments start by loading the data into Pandas dataframe, performing data pre-processing techniques like coverting all the tweets to lowercase. Next datacleanup activities like removing certain symbols, garbage words, etc are performed. Till this stage the tweets are not yet tokenized. In the next step, the process of Tokenization is performed. Here, tweets are split into tokens of words and then searched for stopwords. All stopwords are removed by using nltk toolkit as the basis of reference for stopwords in the English language. After this stage, we still have tokens of words instead of tweets. In the next step, each tweet's tokens are joined back to form a tweet, i.e. the list of words are joined back into a sentence. This step is very important since CountVectorizer needs a sentence rather than a list of words as input

4.2 Part 2 : Feature extraction and upsampling

4.2.1 Bag of words

In part 2, feature generation is performed using Bag of words as well as word2vec word embeddings. The bag-of-words model is a way of representing text data when modeling text with machine learning algorithms. The bag-of-words model is simple to understand and implement and has seen great success in problems such as language modeling and document classification. A problem with

modeling text is that it is messy, and techniques like machine learning algorithms prefer well defined fixed-length inputs and outputs. Machine learning algorithms cannot work with raw text directly; the text must be converted into numbers. Specifically, vectors of numbers. This is called feature extraction or feature encoding. A popular and simple method of feature extraction with text data is called the bag-of-words model of text. A bag-of-words model, or BoW for short, is a way of extracting features from text for use in modeling, such as with machine learning algorithms. The approach is very simple and flexible, and can be used in a myriad of ways for extracting features from documents. A bag-of-words is a representation of text that describes the occurrence of words within a document. It involves two things:

1. A vocabulary of known words.
2. A measure of the presence of known words.

It is called a "bag" of words, because any information about the order or structure of words in the document is discarded. The model is only concerned with whether known words occur in the document, not where in the document. The bag-of-words can be as simple or complex as you like. The complexity comes both in deciding how to design the vocabulary of known words (or tokens) and how to score the presence of known words.

For Bag of words, CountVectorizer is used to conver the tweets into a matrix of token counts. This implementation provides a sparse representation of the counts using scipy.sparse.csr_matrix. Later this sparse matrix is converted into a dense representation using pandas to_dense() method. The following parameters are passed to the CountVectorizer:

ngram_range = (1,1)
max_features = 1000

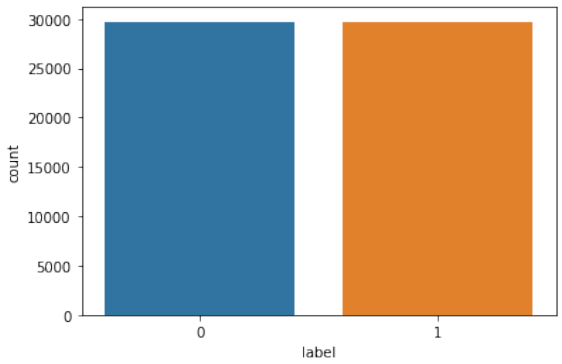
4.2.2 word2vec

For word2vec, the gensim modue is used. Word embedding algorithms like word2vec are key to the state-of-the-art results achieved by machine learning models on natural language problems. A word embedding is an approach to provide a dense vector representation of words that capture something about their meaning. Word embeddings are an improvement over simpler bag-of-word model word encoding schemes like word counts and frequencies that result in large and sparse vectors (mostly 0 values) that describe documents but not the meaning of the words. Word embeddings work by using an algorithm to train a set of fixed-length dense and continuous-valued vectors based on a large corpus of text. Each word is represented by a point in the embedding space and these points are learned and moved around based on the words that surround the target word. It is defining a word by the company that it keeps that allows the word embedding to learn something about the meaning of words. The vector space representation of the words provides a projection where words with similar meanings are locally clustered within the space.

4.2.3 Upsampling

In the dataset, the number of samples with label == 1, i.e. tweets which are labelled as racist/sexist, are 2,242, while those with label == 0 are 29,720. This leads to an imbalance in the dataset whereby the number of samples of one class (major class) are significantly more than those of the other class (minor class). Any classification performed on imbalances data sets is skewed in favor of the major class since the features of the major class are learned more than those of the minor class. Imbalanced classes put "accuracy" out of business. This is a surprisingly common problem in machine learning (specifically in classification), occurring in datasets with a disproportionate ratio of observations in each class. Standard accuracy no longer reliably measures performance, which makes model training much trickier. Upsampling is the process of randomly duplicating observations from the minoriy class in order to regain accuracy in the classification process.

In my experiements, upsampling has been performed to balance the dataset to conatin equal number of samples of both labels, i.e. each label has now 29,720 samples. Below is a representation of the same.



4.3 Part 3 : Machine learning classification techniques to train the models using Bag of words and word2vec

In this part, different machine learning clasification techniques are used to train the models on Ba of words and word2vec word embeddings.
Machine learning is a field of study and is concerned with algorithms that learn from examples. Classification is a task that requires the use of machine learning algorithms that learn how to assign a class label to examples from the problem domain. An easy to understand example is classifying emails as “spam” or “not spam.” Classification is a supervised learning technique. Supervised learning is the machine learning task of learning a function that maps an input to an output based on example input-output pairs. It infers a function from labeled training data consisting of a set of training examples. In supervised learning, each example is a pair consisting of an input object (typically a vector) and a desired output value (also called the supervisory signal). A supervised learning algorithm analyzes the training data and produces an inferred function, which can be used for mapping new examples. An optimal scenario will allow for the algorithm to correctly determine the class labels for unseen instances. This requires the learning algorithm to generalize from the training data to unseen situations in a ”reasonable” way.

In my experiements, I used the following Machine learning techniques for classification:

1. Random Forest classification
2. Naive Bayes classification
3. Logistic Regression
4. Support Vector Machine (SVM)

The upsampled data is used for training. The training to testing data ratio is considered as 80/20, i.e. 80% of the upsampled dataset is used for training the model(s) while 20% of the upsampled dataset is used for testing the model(s). An important aspect of data processing is used at this juncture, i.e. ***data shuffling***. This technique has been chosen because upsampled data consists of initially all the tweet labeled as ’0’ followed by tweets labeled as ’1’.

4.3.1 Random Forest classification

Random forests or random decision forests are an ensemble learning method for classification that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes of the individual trees. As we know that a forest is made up of trees and more trees means more robust forest. Similarly, random forest algorithm creates decision trees on

data samples and then gets the prediction from each of them and finally selects the best solution by means of voting. It is an ensemble method which is better than a single decision tree because it reduces the over-fitting by averaging the result.

The internal working of Random Forest algorithm is :

Step 1 - Start with the selection of random samples from a given dataset.

Step 2 - Next, this algorithm will construct a decision tree for every sample. Then it will get the prediction result from every decision tree.

Step 3 - In this step, voting will be performed for every predicted result.

Step 4 - At last, select the most voted prediction result as the final prediction result.

scikit-learn has a module for Random Forests, i.e. sklearn.ensemble.RandomForestClassifier. The hyperparameters passed to the classifier are as follows:

n_estimators = 1,000.
criterion is set to default, i.e. "gini".

n_estimators is a representation of the number of trees in the forest. In my experiment, I used 1,000 trees in the forest. Initially, a model is initialized as a RandomForestClassifier with 1,000 trees in the forest. Next, model fitting is performed using the training data and training labels followed by prediction of label of the tweets in the testing dataset. Accuracy score is calculated based on the actual labels and predicted labels of the testing dataset. Visual representation of the actual and predicted labels of the testing dataset is plotted in the form of a ***confusion matrix***. Confusion matrix plots actual labels vs predicted labels in the form of a specific table layout that allows for the visualization of the performance of the algorithm. Each row of the matrix represents the instances in a predicted class while each column represents the instances in an actual class (or vice versa). The name stems from the fact that it makes it easy to see if the system is confusing two classes (i.e. commonly mislabeling one as another).

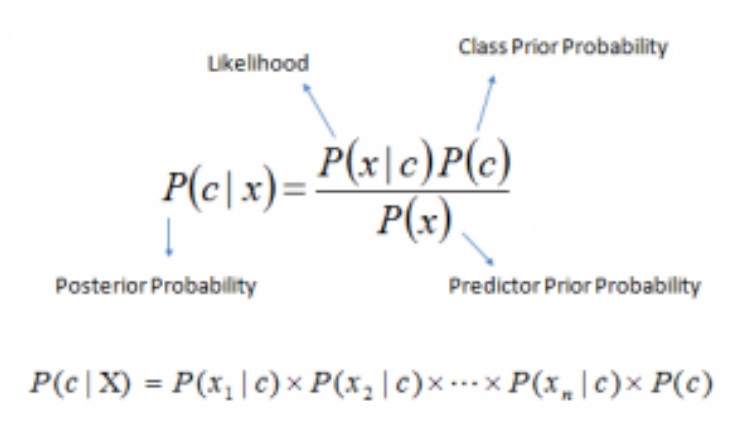
4.3.2 Naive Bayes classification

Naive Bayes algorithm is based on Bayes' theorem with an assumption of independence among predictors. A Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. Naive Bayes is easy to build and particularly useful for very large datasets. Using Naive Bayes text classifiers might be a really good idea, especially if there's not much training data available and computational resources are scarce. Usually, results are pretty competitive in terms of performance if features are well engineered. The Naive Bayes algorithm works as follows.

Step 1 - Convert the data set into a frequency table

Step 2 - Create Likelihood table by finding the probabilities

Step 3 - Use Naive Bayes equation to calculate the posterior probability for each class. The class with the highest posterior probability is the outcome of the prediction.



scikit-learn has a module for Naive Bayes, i.e. sklearn.naive_bayes.MultinomialNB. The hyperparameters passed to the classifier are as follows:

alpha = 0.00001.
fit_prior = True.
class_prior = None.

Initially, a model is initialized as a MultinomialNB with the above provided hyperparameters. Next, model fitting is performed using the training data and training labels followed by prediction of label of the tweets in the testing dataset. Accuracy score is calculated based on the actual and predicted labels of the testing dataset. Finally, a confusion matrix is plotted to visualize the performance of the classifier.

We can use Naive Bayes classification only with Bag of words, but not with word2vec, since word2vec contains negative values and these are inappropriate to be passed for model fitting as Naive Bayes classifier works on probabilities whose values should be positive.

4.3.3 Logistic Regression

Logistic Regression is a statistical model that in its basic form uses a logistic function to model a binary dependent variable. A binary logistic model has a dependent variable with two possible values, such as pass/fail which is represented by an indicator variable, where the two values are labeled as "0" and "1". In my experiments, since I have only two output variables, one for non-racist/sexist tweets and the other for racist/exist tweets, my model is a binary logistic regression model. In my experiments, I used a form of regularization called l2-regularization. ***Regularization*** is one of the techniques to help prevent overfitting. Overfitting happens when a model learns the details and noise in the training data to the extent that it negatively impacts the performance of the model on unseen data. This means that the noise or random fluctuations in the training data is picked up and learned as concepts by the model. The problem is that these concepts do not apply to new data and negatively impact the model's ability to generalize. Regularization in machine learning is the process of regularizing the parameters that constrain, regularize, or shrink the coefficient estimates towards zero. In other words, this technique discourages learning a more complex or flexible model, avoiding the risk of Overfitting.

In L2 regularization, the regularization term is the sum of square of all feature weights (not including the bias term) as shown in the below equation :

$$\min_{w\in\mathbb{R}^d, b\in\mathbb{R}} \quad \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} \ell\Big(y_i, \sigma(\langle w, x_i\rangle + b)\Big),$$

where:

- $\ell(y, \hat{y}) = -y \log \hat{y} - (1-y)\log(1-\hat{y})$ is the cross-entropy loss;
- $\sigma(z) = \frac{1}{1+e^{-z}}$ is the sigmoid function;
- $\|w\|^2 = \sum_{j=1}^{d} w_j^2$ is the squared Euclidean norm of the vector w.

Here, $C \geq 0$ is the regularization parameter. Higher C means *less* regularization

scikit-learn has a module for Logistic Regression, i.e. sklearn.linear_model.LogisticRegression. The hyperparameters passed to the classifier are as follows:

Ten different values of C, the regularization parameter.
solver = lbfgs.
penalty = l2.
max_iter = 1,000.
random_state = 42.

Initially, a model is initialized as a LogisticRegression with the above provided hyperparameters. Next, model fitting is performed using the training data and training labels for different value of the regularization hyperparameter. Accuracies are calculated for each of the ten trials. From the results, the hyperparameters that provide the best accuracy are used to predict the label of the tweets in the testig dataset. Finally, confusion matrix is plotted to visualize the performance of the logistic regression model.

4.3.4 Support Vector Machine (SVM)

SVM is a linear model for classification problems. It can solve linear and non-linear problems and works well for many practical problems. The idea of SVM is very simple : The algorithm creates a line or a hyperplane which separates the data into classes. The algorithm takes the data as an input and outputs a line that separates those classes, if possible.

The SVM algorithm works as follows:

Step 1 - Initially, we find the points closest to the line from both the classes. These points are called ***support vectors***.

Step 2 - We compute the distance between the line and the support vectors. This distance is called ***the margin***.

Step 3 - Our goal is to maximize the margin. The hyperplane for which the margin is maximum is the ***optimal hyperplane***.
scikit-learn has a module for SVM, i.e. sklearn.SVM. The hyperparameters passed to the classifier are as follows:

kernel = "linear".
C = 0.75.
max_iter = 1,000.

Initially, some preprocessing is required to normalize the data inorder to decrease the runtime of the model. I used the sklearn.preprocessing.MinMaxScaler to normalize the data to the range (0,1). Next, a model is initialized as a SVM with the above provided hyperparameters. Next, model fitting is performed using the training data and training labels followed by prediction of label of the tweets in the testing dataset. Accuracy score is calculated based on the actual and predicted labels of the testing dataset. Finally, a confusion matrix is plotted to visualize the performance of the classifier.

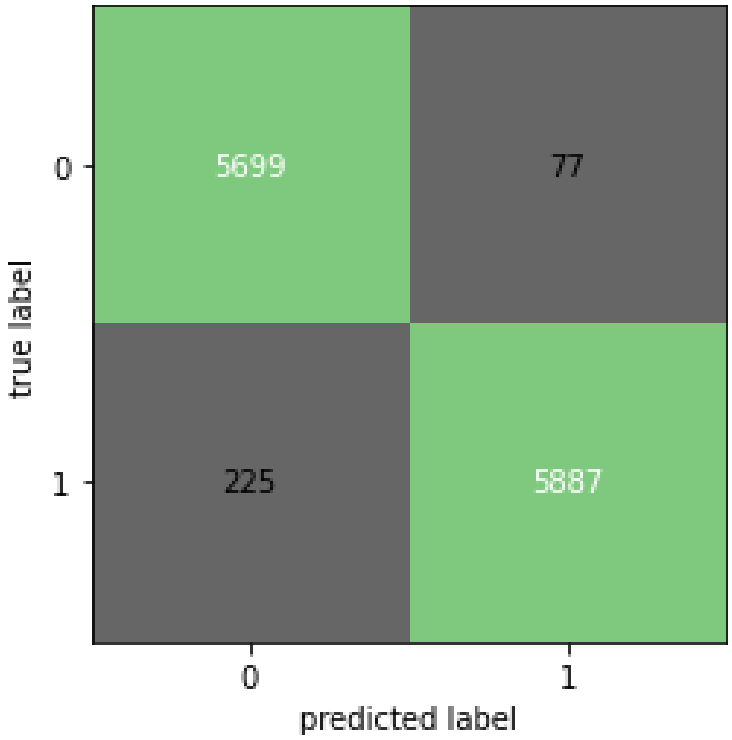
5 Results and discussions

The above section explains in detail the methodologies used for analysis and the different hyperparameters used for different classifiers. A lot of information is provided about how the experiements have been conducted in the above section. In this section, the results are being presented along with challenges faced and solutions for the challenges.

5.1 Bag of words

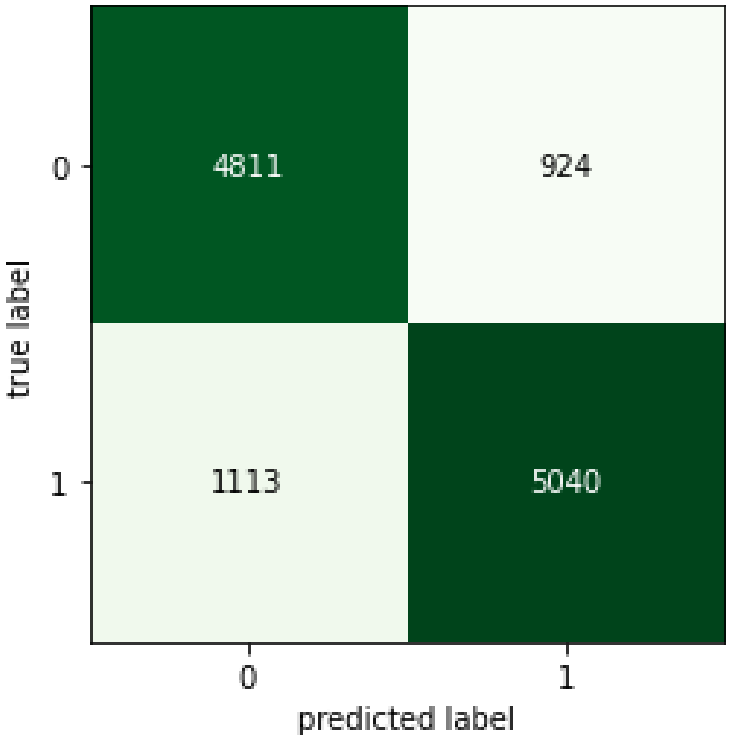
5.1.1 Random Forest

The confusion matrix, discussed in the Evaluation methodology section, for Random Forest classifier using Bag of words is as follows:



5.1.2 Naive Bayes

The confusion matrix, discussed in the Evaluation methodology section, for Naive Bayes classifier using Bag of words is as follows:

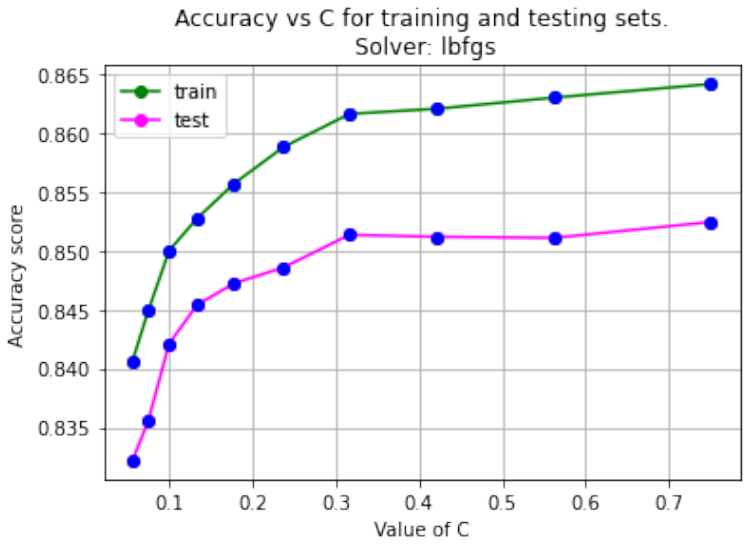


5.1.3 Logistic Regression

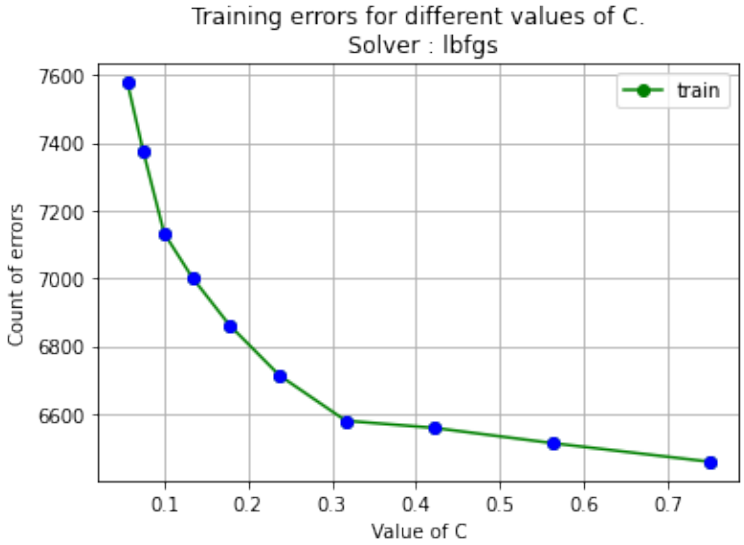
For classificatin using Logistic Regression, the below table depicts the best and average accuracies of the model using different regularization hyperparameters.

Train/Test	Best	Average
lbfgs Training	0.8641907806191117	0.8553835800807537
lbfgs Testing	0.8524562584118439	0.8457183714670256

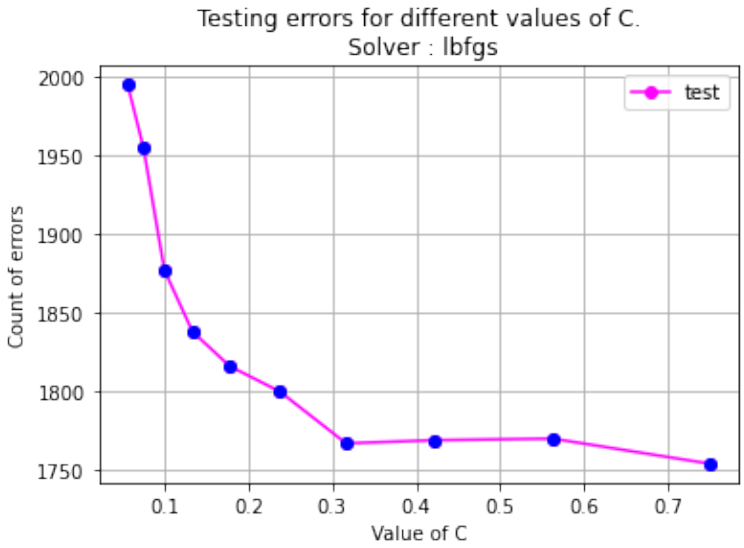
The below figure depicts the training vs testing accuracies for different values of regularization parameter.



The below figure depicts the training errors for different values of regularization parameter. It is clear from the figure that at $C = 0.75$, the number of errors is the least.

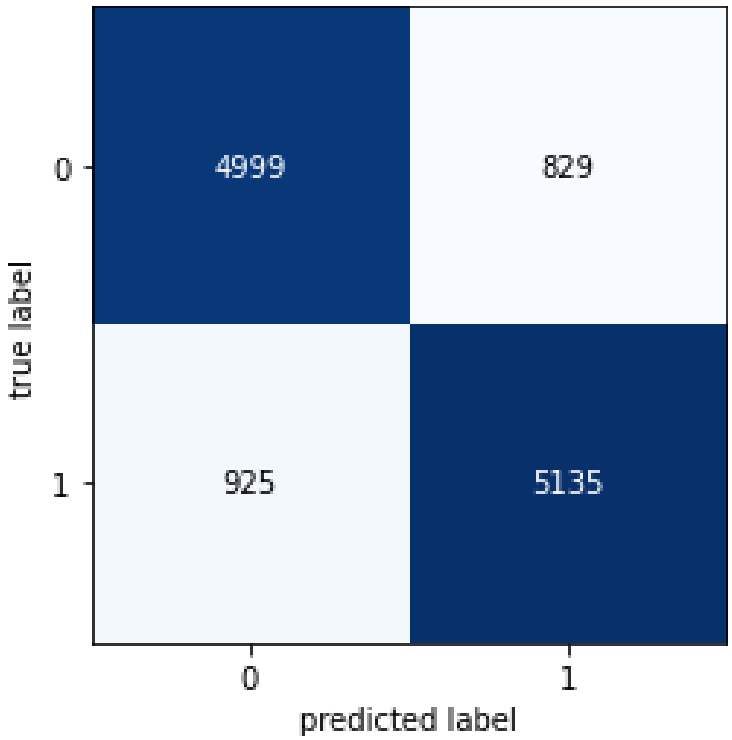


The below figure depicts the testing errors for different values of regularization parameter. It is clear from the figure that at $C = 0.75$, the number of errors is the least.



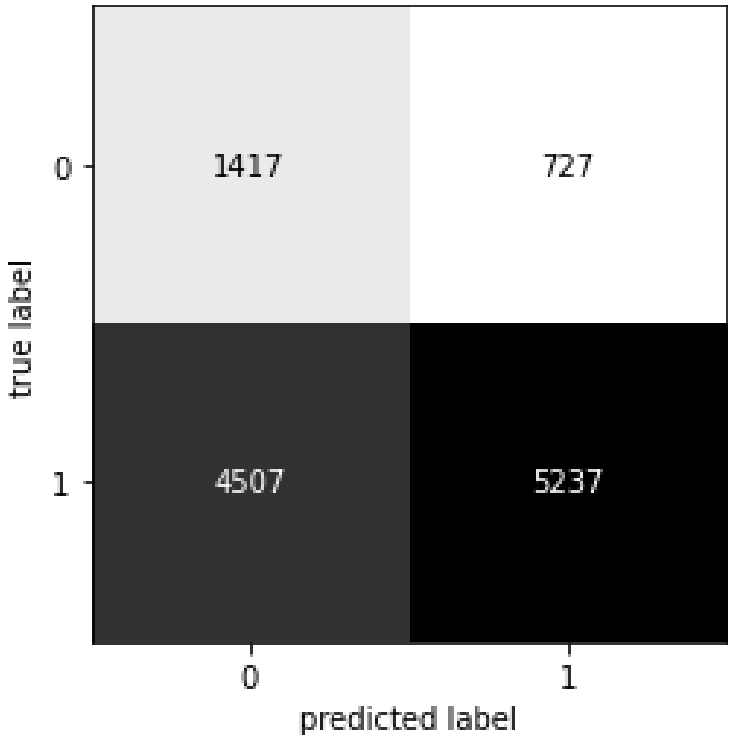
From the above observations, it is clear that at $C = 0.75$ the best accuracy is obtained. Hence, the model is trained using this hyperparameter value.

The confusion matrix, discussed in the Evaluation methodology section, for Logistic Regression classifier using Bag of words is as follows:



5.1.4 SVM

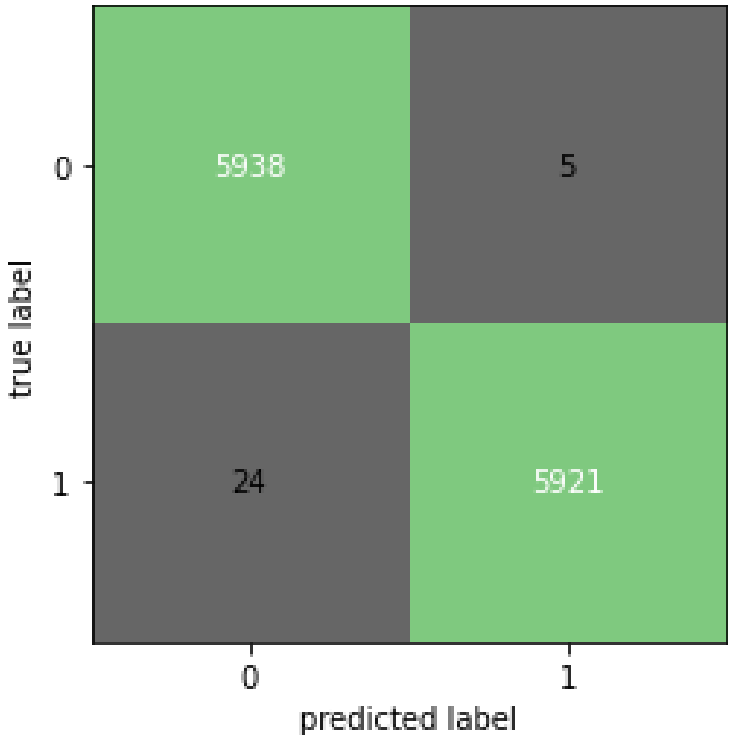
The confusion matrix, discussed in the Evaluation methodology section, for SVM classifier using Bag of words is as follows:



5.2 word2vec

5.2.1 Random Forest

The confusion matrix, discussed in the Evaluation methodology section, for Random Forest classifier using word2vec is as follows:



5.2.2 Naive Bayes

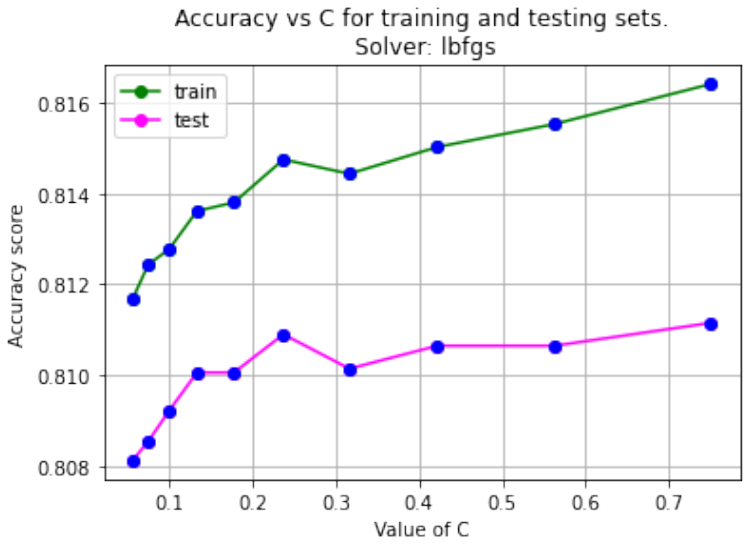
We can use Naive Bayes classification only with Bag of words, but not with word2vec, since word2vec contains negative values and these are inappropriate to be passed for model fitting as Naive Bayes classifier works on probabilities whose values should be positive.

5.2.3 Logistic Regression

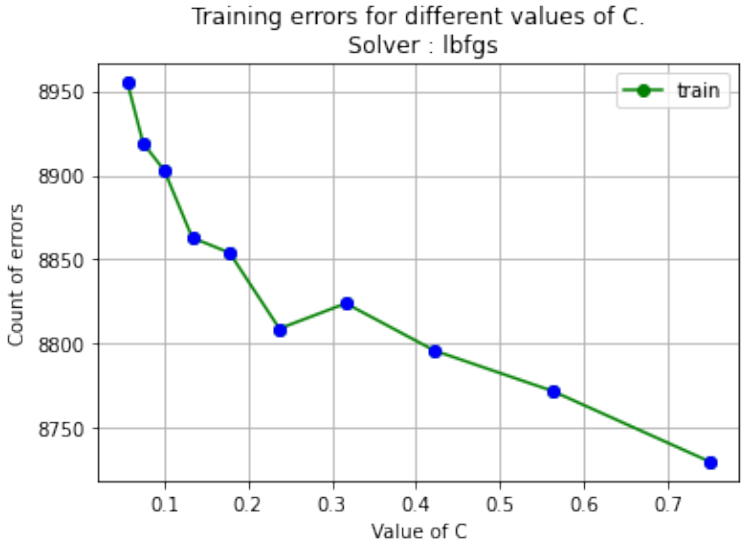
For classificatin using Logistic Regression, the below table depicts the best and average accuracies of the model using different regularization hyperparameters.

Train/Test	Best	Average
lbfgs Training	0.8164115074024226	0.8140456763122477
lbfgs Testing	0.8111541049798116	0.8099512113055182

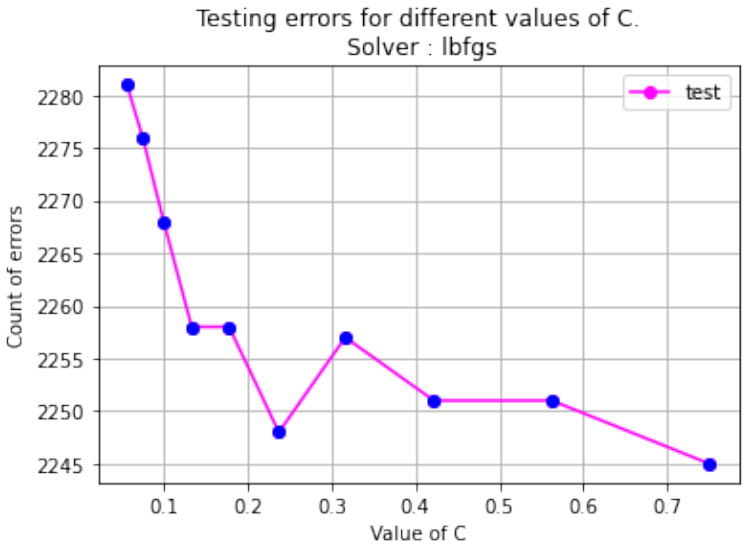
The below figure depicts the training vs testing accuracies for different values of regularization parameter.



The below figure depicts the training errors for different values of regularization parameter. It is clear from the figure that at $C = 0.75$, the number of errors is the least.

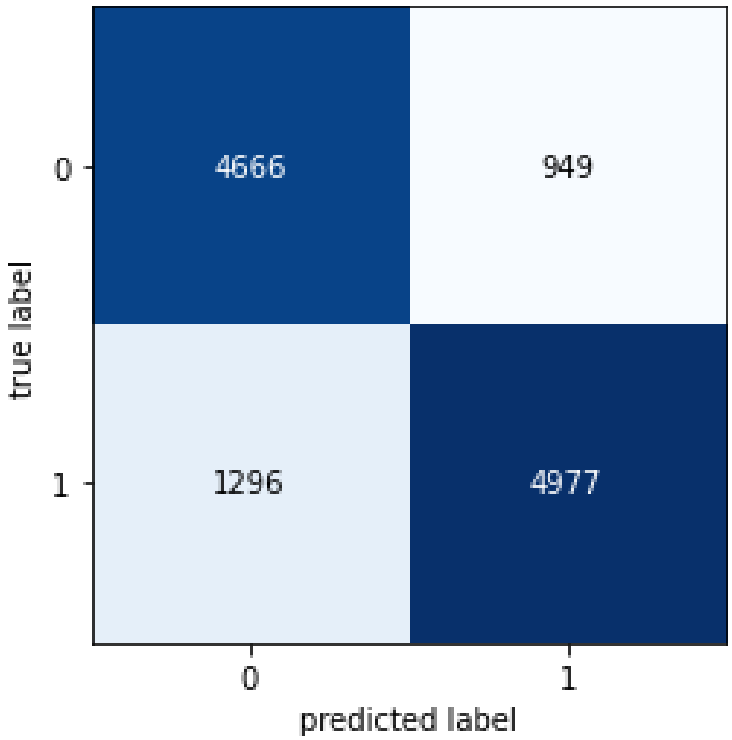


The below figure depicts the testing errors for different values of regularization parameter. It is clear from the figure that at $C = 0.75$, the number of errors is the least.



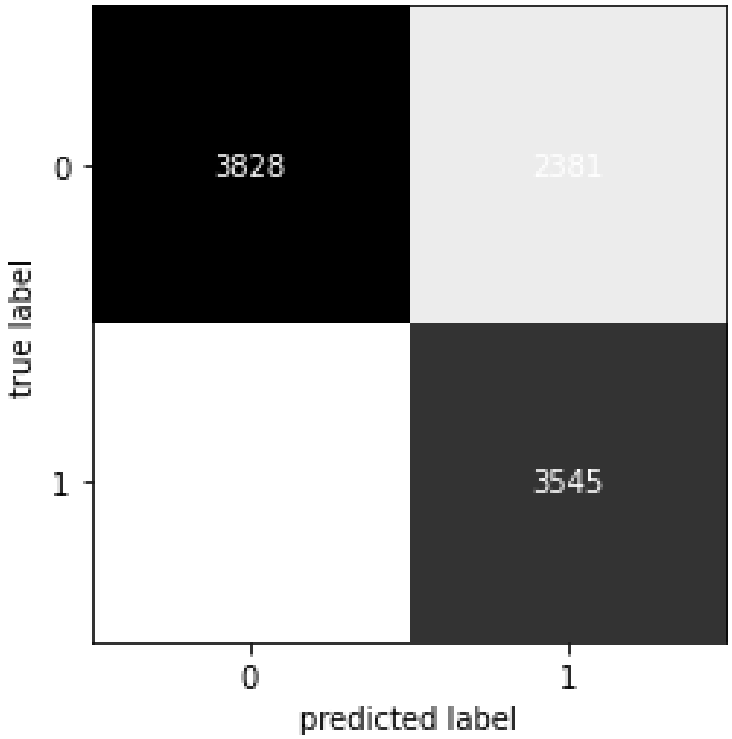
From the above observations, it is clear that at $C = 0.75$ the best accuracy is obtained. Hence, the model is trained using this hyperparameter value.

The confusion matrix, discussed in the Evaluation methodology section, for Logistic Regression classifier using word2vec is as follows:



5.2.4 SVM

The confusion matrix, discussed in the Evaluation methodology section, for SVM classifier using word2vec is as follows:



5.3 Discussion

The below table depicts the performance of the four models, i.e. Random Forest, Naive Bayes, Logistic Regression and SVM for Bag of words and word2vec.

ML Classifier Accuracy	Bag of words	word2vec
Random Forest	0.9745962314939435	0.9975605652759085
Naive Bayes	0.8286507402422612	--
Logistic Regression	0.8524562584118439	0.8111541049798116
SVM	0.5597240915208613	0.6202052489905787

Random forest using word2vc provides the best accuracy (approx. 0.9976) followed by Random forest using Bag of words (approx. 0.9746). For the chosen hyperparameters, Logistic Regression performed better than SVM and Naive Bayes. SVM provided the least accuracies among all the classifiers. The reason Random forest provides better accuracy is that Random forest classifiers are suitable for dealing with high dimensional noisy data in text classification. A Random forest model comprises a set of decision trees each of which is trained using random subsets of features. Given an instance, the prediction by the RF is obtained via majority voting of the predictions of all the trees in the forest. SVm might have failed because Bag of words has a maximum number of features of 1,000 while word2vec is having 100 features. It is widely known that SVMs are inefficient if the number of features are very huge compared to the training samples (in our case the number of training samples are 47,552 only).

5.4 Challenges and solutions

The following are the challenges faced and their solutions I found :

1. After performing tokenization and removal of stopwords, each tweet is a list of words. CountVectorizer needs a sentence as an input. So, joining the list of words into its corresponding tweet, after removal of stopwords, resolved this issue.

2. As part of data pre-processing, I tried to implement Lemmatization, however lemmatization has been modifying basic words like was to wa and has to ha, etc. So, lemmatization has been removed from the process of data pre-processing.

3. The initial training dataset is a highly imbalanced dataset (29,720 data samples labeled "0" and 2,242 data samples labeled as "1"). This led to skewed results. Using Upsamplng technique, the dataset is balanced. This provided better results than the imbalanced datasets.

4. Training SVM models has been quite time consuming. To overcome this challenge, I used the MinMaxScaler available in scikit-learn to normalize the features to a range of (0,1). This reduced the runtime of SVM classifiers manifold.

6 Conclusion and future work

Twitter tweets are analyzed for racist/sexist tweets using Python programming language, NLTK toolkit and Machine learning algorithms available through the scikit-learn module of Python. An initial data pre-processing phase has been followed by creating the much needed Bag of words as well as word2vec word embeddings. Due to the imbalance in the datasets, a method called Upsampling has been used to balance the datasets, thereby facilitating fair classification. Four Machine learning algorithms - Random Forest, Naive Bayes, ogistic Regression and Support Vector Machine - have been used to perform classification of the tweets. For the classification process, 80% of the data is dedicated as the training dataset while 20% is dedicated as the testing dataset. Accuracies are calculated for each classifier and tabulated. Conclusions have been made about the accuracy of the classifiers and discussions have been made about the performance of the classifiers.
Though it is observed that Random forest classifier performed better than the other three classifiers for this data, there is scope to train more models on more different machine learning algorithms to better understand how those models will perform on the Twitter dataset. Further, future work includes analyzing emoticons and idioms in the tweets.

7 References

[1] Robertson C, Mele C, Tavernise S. 11 Killed in Synagogue Massacre; Suspect Charged With 29 Counts. 2018;

[2] Hate speech detection : Challenges and solutions, By Sean MacAvaney, Hao-Ren Yao, Eugene Yang, Katina Russell, Nazli Goharian and Ophir Frieder;

[3] The datafication of hate : Expectations and challenges in automated hate speech monitoring, By Salla-Maaria Laaksonen, Jesse Haapoja, Teeemu Kinnunen, Matti Nelimarkka and Reeta Poyhtari;

[4] Detection of hate speech in social networks : A survey on multilingual corpus, By Arrej Al-Hassan and Hmood Al-Dossari;

[5] https://www.kaggle.com/arkhoshghalb/twitter-sentiment-analysis-hatred-speech;