# Classification Of Profane And Racist Tweets Using Natural Language Processing

IST 664 / CIS 668

Submitted By Group 9 :-
1) Shashwat Bhatt (SUID:701024219)
2) Sandesh Bhat (SUID: 324270666)
3) Siddhant Bandiwadekar (SUID: 364845292)
4) Yeshwanth Chandrashekaraiah  (SUID: 645047527)

## Contents

## Introduction

Most social media platforms grant users freedom of speech by allowing them to freely express their thoughts, beliefs, and opinions. Although this represents incredible and unique communication opportunities, it also presents important challenges. Indeed, some users misuse the medium to promote offensive and hateful language, which mars the experience of regular users, affects the business of online companies, and may even have severe real-life consequences. In this project we tried to detect racism and sexism through various tweets and evaluate them qualitatively and quantitatively, we decided to create a set of deep neural network-based classifiers. For this purpose, we used the most common dataset published for Tweet classification. The purpose of this model is to be able to identify tweets as racist, sexist or neither.

Abusive comments also referred to as hate speech on social media websites have become a serious problem. A direct impact of this kind of speech has been on teenagers which has led to increase in the number of cases of suicides, depression etc. This issue can be resolved by identifying the users that promote hate speech on social media websites which are actively dealing with such racist/sexist comments. We have considered racist and sexist tweets to be hate speech and will be classifying the data based on these features. We have taken a training sample of tweets and labels, where label '1' denotes the tweet is racist/sexist and label '0' denotes the tweet is not racist/sexist, the objective of this project is to predict the labels on the given test dataset.

## Data Collection:

Data collection phase started with going through some of the tweets between 2016-2020, specific to the events involving racism, sexism, etc. and some of the profane tweets across United States. In the next part, we searched through the Kaggle to search for the dataset.

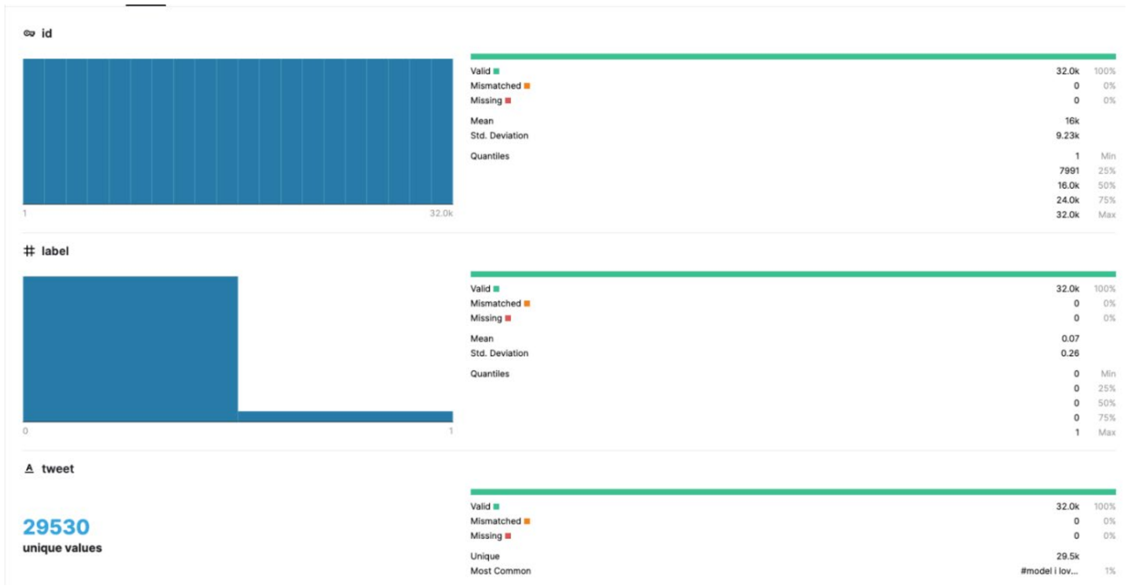
- train.csv - The train data contains 31,962 tweets in the form of a csv file with each line storing a tweet id, its label and the tweet.
- test_tweets.csv - The test data file contains 17,197 rows of tweet ids and the tweet text with each tweet in a new line.

Data collection is one of the most integral parts of our project because we wanted to know which kind of data we wanted analyze. This data collection phase will guide the pre-processing phase to label and id the data.

Here are some of the facts that we took into the consideration while the team worked in the data collection phase:

- There are a total of 1.3 billion accounts on Twitter
- 23% American population are on Twitter
- Since 2016 there has been a drastic rise in the hate, racist and sexist tweets.

It also needs to be considered that the data might considered alphanumeric characters, emojis and some of the special characters, etc. The data is in the form a csv file and the separation was done in two parts i.e. the training data and the testing data.



id

| | | |
|---|---|---|
| Valid 🟩 | 32.0k | 100% |
| Mismatched 🟧 | 0 | 0% |
| Missing 🟥 | 0 | 0% |
| Mean | 16k | |
| Std. Deviation | 9.23k | |
| Quantiles | 1 | Min |
| | 7991 | 25% |
| | 16.0k | 50% |
| | 24.0k | 75% |
| | 32.0k | Max |

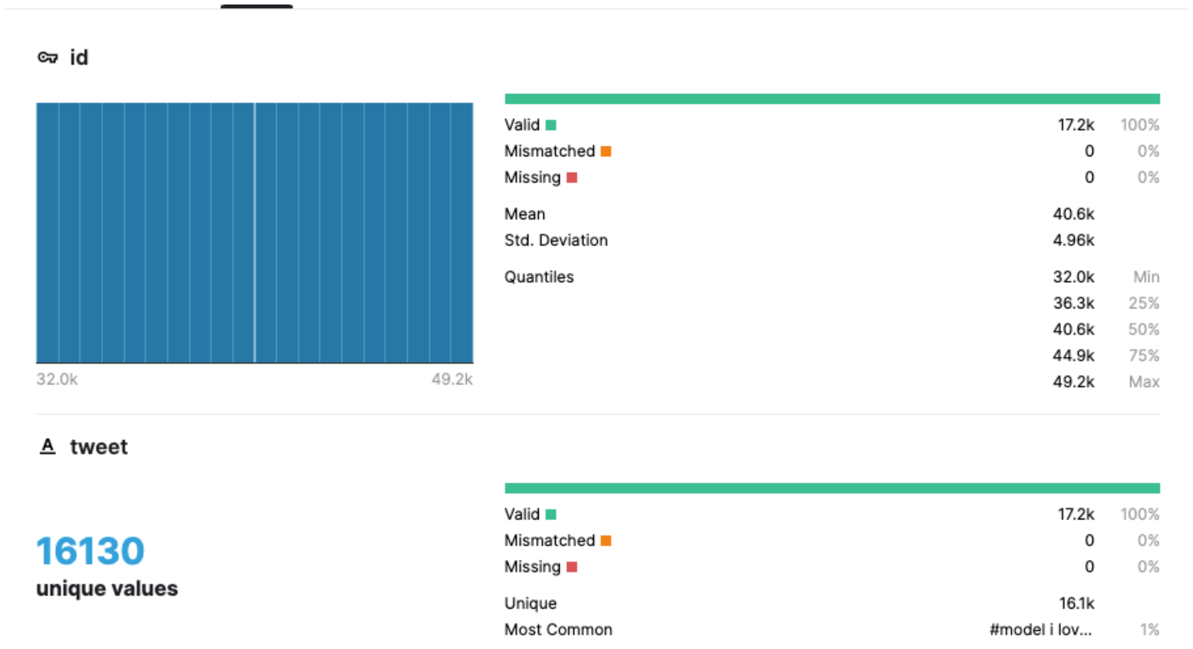
label

| | | |
|---|---|---|
| Valid 🟩 | 32.0k | 100% |
| Mismatched 🟧 | 0 | 0% |
| Missing 🟥 | 0 | 0% |
| Mean | 0.07 | |
| Std. Deviation | 0.26 | |
| Quantiles | 0 | Min |
| | 0 | 25% |
| | 0 | 50% |
| | 0 | 75% |
| | 1 | Max |

tweet

29530
unique values

| | | |
|---|---|---|
| Valid 🟩 | 32.0k | 100% |
| Mismatched 🟧 | 0 | 0% |
| Missing 🟥 | 0 | 0% |
| Unique | 29.5k | |
| Most Common | #model i lov... | 1% |



id

| | | |
|---|---|---|
| Valid 🟩 | 17.2k | 100% |
| Mismatched 🟧 | 0 | 0% |
| Missing 🟥 | 0 | 0% |
| Mean | 40.6k | |
| Std. Deviation | 4.96k | |
| Quantiles | 32.0k | Min |
| | 36.3k | 25% |
| | 40.6k | 50% |
| | 44.9k | 75% |
| | 49.2k | Max |

tweet

16130
unique values

| | | |
|---|---|---|
| Valid 🟩 | 17.2k | 100% |
| Mismatched 🟧 | 0 | 0% |
| Missing 🟥 | 0 | 0% |
| Unique | 16.1k | |
| Most Common | #model i lov... | 1% |

## Preprocessing:

The preprocessing of the content information is a basic advance as it prepares the crude content for mining, i.e., it gets simpler to extricate data from the content and apply AI calculations to it. In the event that we avoid this progression, at that point there is a higher possibility that you are working with loud and conflicting information. The target of this progression is to clean clamor those are less pertinent to discover the opinion of tweets, for example, accentuation, uncommon characters, numbers, and terms which don't convey much weightage in setting to the content. The data has 3 columns id, label, and tweet. label is the binary target variable and tweet contains the tweets that we will clean and preprocess.

Beginning information cleaning prerequisites that we can consider subsequent to taking a gander at the main 5 records:
The Twitter handles are as of now covered as @user because of protection concerns. Thus, these Twitter handles are not really giving any data about the idea of the tweet.

- We can likewise consider disposing of the accentuations, numbers and even extraordinary characters since they wouldn't help in separating various types of tweets.

- Most of the more modest words don't add a lot of significant worth. For instance, 'pdx', 'his', 'all'. In this way, we will attempt to eliminate them too from our information.

- Once we have executed the over three stages, we can part every tweet into singular words or tokens which is a fundamental advance in any NLP task.

- In the fourth tweet, there is a word 'love'. We may likewise have terms like loves, cherishing, adorable, and so on in the remainder of the information. These terms are frequently utilized in a similar setting. In the event that we can decrease them to their root word, which is 'love', at that point we can lessen the absolute number of interesting words in our information without losing a lot of data.

For data preprocessing, we used the following techniques:

## Tokenization:

Tokenization is the process by which a stream of text is split into smaller meaningful components called tokens, that can be viewed as a single logical entity. A token, for instance, may be a word, phrase or a symbol.

## Removing Stopwords:

Filtering out useless data is one of the big steps of pre-processing. These words are referred to as stop words. A stop word is a widely used word that has no sense of its own (such as the", "a", "an", "in"). Storing stop words takes up precious space and time to process. For this purpose, stop words can be easily eliminated by storing a list of words that you consider to be stop words, so that we can easily delete them. We may also add our own terms to the list of stop words depending on the context that normally occurs in our dataset, which we don't want to process or evaluate.

## Lemmatization:

The process of grouping the various inflected forms of a word together is lemmatization, so that they can be evaluated as a single object. It binds words to one word with a similar significance. Lemmatization is favored over stemming and morphological analysis of the words is achieved by lemmatization.

## Stemming (PorterStemmer):

Stemming is the linguistic normalization process, in which a word's variant forms are reduced to a common form. Stemming algorithms strip words such as {'ing', 'ly', 'es', 's'} from their suffixes, leaving only the word stem.

## Bag of Words Model

### Count Vectorizer

Count Vectorizer is the simplest vectorization technique. Count Vectorizer first builds the vocabulary dictionary where keys are the words available in the corpus and value is the number of occurrences of this word in the corresponding sentence. Finally, we end up with our dataset of size $n_{samples}\ x\ size\ of\ vocabulary$

## Classification:

Machine learning requires the creation of a model that is trained on some training data and can then process additional information to make predictions.

### Bagging

Bootstrap Aggregating is a technique for an ensemble. Second, we produce random replacement samples of the training data set. Then for each sample, we create a model. Finally, using average or plurality voting, the outcomes of these models are combined. Since each model is exposed to a different subset of data and we use their cumulative output at the end, we ensure that the overfitting issue is taken care of by not sticking too closely to our training data collection. Bagging allows us to reduce the error of variance. An example of a bagging pattern is Random Forest.

### Voting Classifier

Using various algorithms, we can train and assemble the data set to predict the final output. By majority vote, the final performance on a prediction is taken. For instance, if classifier 1 predicts class 0, classifier 2 predicts class 0, and classifier 3 predicts class 1, we will classify the sample as 'class 0' by majority vote.

## Classification Models:

A classification model draws conclusions from the input values given for training. Based on the training the model will predict the class labels/categories for new data. A binary classification is one with two possible prediction outcomes. Multiclass classification is one with more than two possible prediction outcomes.
For classification we used the techniques mentioned below:

### Random Forest:

It is a classifier which makes use of several decision tress on the subsets of datasets and then takes the average of all the decision trees to obtain an improved predictive accuracy of the dataset. Random Forest is bagging technique and bagging techniques makes use of base learner models and in random forest this base learner model Is called as the decision tree. So, in random forest it chooses certain row sample and column sample or also called as feature sample and supplies it to the decision tree 1. The decision tree will be trained on this particular supplied dataset. Similarly, for decision tree 2 we provide row samples and feature sample, and this happens with replacement so it may happen that some of the rows and features we provided for the decision tree1 may be again provided for decision tree 2. After which the decision tree 2 will be trained on the supplied dataset.

We do this process for all the decision trees after which the decision tree will be trained on the complete dataset. Then every decision tree will provide us with the accuracy, then on the accuracy majority votes technique will be applied to obtain the best possible accuracy

When we create a single decision tree to its complete depth it leads to something called as overfitting but in random forest, we are combining several decision trees with respect to the majority votes, so the high variance error of the decision tree will be turned to low variance error thereby giving us high accuracy.

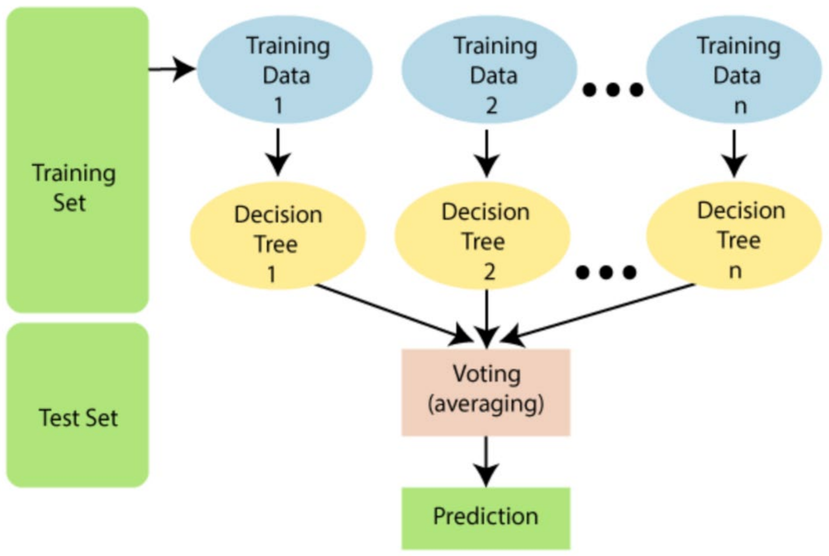


***Fig***: *Working of Random Forest*

**Assumptions made in Random Forest:**

In random forest we are using multiple decision trees to predict the accuracy so it’s possible that some of the decision trees predict correct accuracy value while some may predict the wrong value. All the decision trees together predict the correct accuracy value. In order to make the random forest a better classifier we make the following assumptions:
While providing the features sets to the decision tree there should be some actual values in the dataset for the decision tree to predict the accuracy rather than guessing it.
There should be low correlation (from the decision tree)

**Step wise working of Random Forest Algorithm:**
From the datasets we select the random data points.
Then the selected datasets (subsets) will be provided to the decision tree.
Then select the number of decision trees we want
Repeat the steps 1 & 2
For the new datasets we then find the accuracy from each decision tree
Then choose the accuracy based on majority voting

**Advantages of Random Forest:**
It can be used for classification as well as for regression.
It can easily handle large datasets which have high dimensionality.
It prevents the overfitting.
It improves the accuracy.

**Accuracy Obtained: 0.9992**

**Logistic Regression:**

It is a type of supervised learning model. Given a set of independent variables it is used for prediction of categorical dependent variable. As logistic regression type is used for predicting the output of categorical dependent variable therefore the output will be a discrete value. Discrete value meaning 1 or 0, yes or no etc. but instead of giving exact value it gives a probabilistic value which lie in the range of 0 to 1. It is very similar to that of linear regression, but the application is different as the linear regression is used to solve regression problems while this solves the classification problems. It is one of the significant algorithms in machine learning as it can provide probability while classifying new data using continuous or discrete datasets. There are several types of logistic regression like:

Binomial: which has only 2 possible dependent variables like 0 or 1
Multinomial: which can have 3 or more possible unordered dependent variables like cat, dog, goat.
Ordinal: which can have 3 or more possible ordered dependent variables like low, medium, high

**Assumptions made in Logistic Regression:**
The nature of dependent variable must be categorical.

Multi-collinearity must not exist in independent variables.

**Advantages of Logistic Regression:**
It is simple to implement and efficient to train.
It classifies the unknown records at a very high speed.
It performs well in the case where there are linearly separable datasets and also gives good accuracy in case of simple datasets.

**Accuracy Obtained: 0.9851**

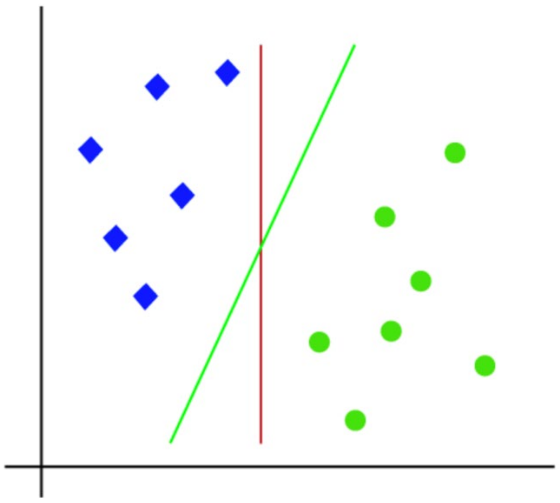
**SVM**
It creates a decision boundary which can be used to segregate the n-dimensional space into classes so that in future it can easily put the new data point in correct category. By using the extreme points or also named as the support vectors, it generates the hyperplane. It is one of the popular supervised learning algorithms used for problems of classification as well as regression. There are two types of SVM:

**Linear SVM:** It is mainly used for linearly separable data meaning the type of data which can be separated by a single straight line.

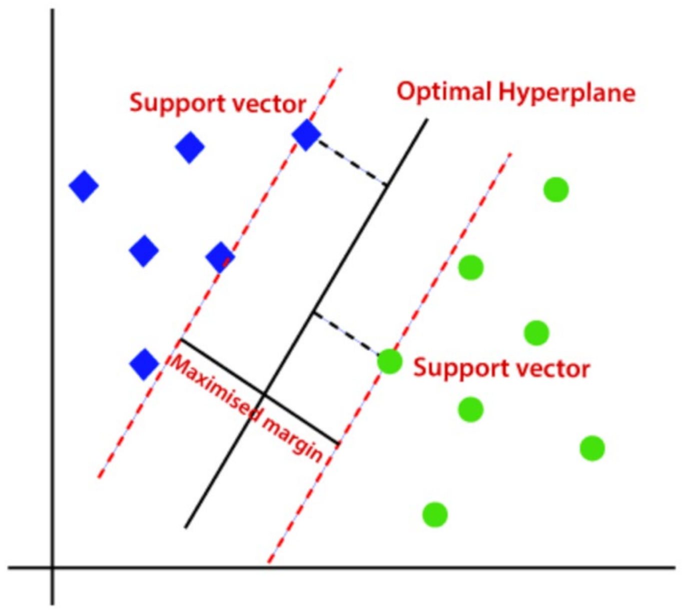
**Non-Linear SVM:** It is mainly used for non-linearly separable data meaning the type of data which cannot be separated by a single straight line.

**Working of SVM:**

Let's consider the below example where we have to classify the red and the green. So, we have to find a line which separates the red and the green that is separate two datasets. But the problem is we don't have a unique line which separates both the datasets there can be infinite number of lines which can do the same.



So, the support vector machine finds the best line or also called as decision boundary. The boundary which is the best one is called as the hyperplane. The work of SVM is to find the closest point from the line of both green and blue. These points which are closest to the boundary are also called as the support vectors. The margin is the distance between the vectors and the hyperplane. The optimal hyperplane would be the hyperplane with maximum margin.



**Advantages of Support Vector Machine:**
It works comparatively better when there is clear separation between the classes.
It is most effective when used for high dimensional spaces and also when the sample number is less than the number of dimensions.
It is comparatively more efficient than other algorithms.

**Accuracy Obtained: 0.9781**

## Decision Tree Classifier:

In order to build a decision tree, it makes use of two nodes which are decision node and the leaf node or also called as the final output node. The decision nodes can be further used to make decision and have further branches. Decision tree can be basically defined as a tree structure classifier whose internal nodes are a representation of the features set while the branches are used to represent the rules of the decision and each leaf node is used to represent the result or the outcome. It makes use of Classification And Regression Tree Algorithm (CART) to build a tree.

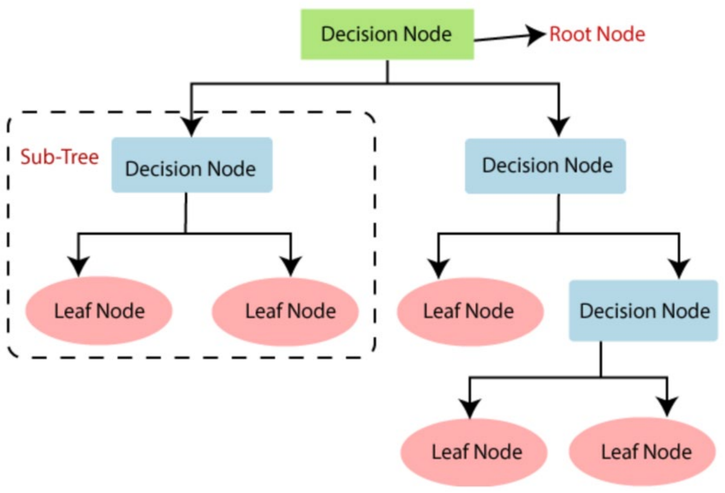


*Fig*: *Working of Decision Tree*

**Working of Decision Tree:**
The decision tree begins with the root node which consists of the entire dataset.
Then it finds the attributes of the dataset
Then it divides the above datasets into subsets which might have some or all of the values of the above found attributes.
Then it recursively makes decision for various subsets of the main dataset which we have created above.
It continues the above process until we have reached the leaf node.

**Advantages of Decision Tree:**
It is simple to understand and easy to implement.
Most effective when used for solving decision related problems.
As the decision tree opens every node it helps us to understand all the possible solutions.
When compared to the other algorithm it can handle even the less cleaned dataset well.

**Accuracy Obtained: 0.9991**

## Model Evaluation

We have evaluated the models based on the performance metrics such as training accuracy, f1-score and evaluation accuracy which are obtained using the confusion matrix. Other aspects of evaluation that were considered were speed, robustness, scalability, and interpretability.

### Accuracy

Accuracy is simply a ratio of correctly predicted observation to the total observations. High accuracy doesn’t necessarily great model performance. Accuracy is a great measure when datasets are symmetric i.e. where values of FPs and FNs are almost the same.

**Precision**

Precision is the ratio of positive observations predicted correctly to the total positive observations predicted. The low false positive rate leads to high precision.

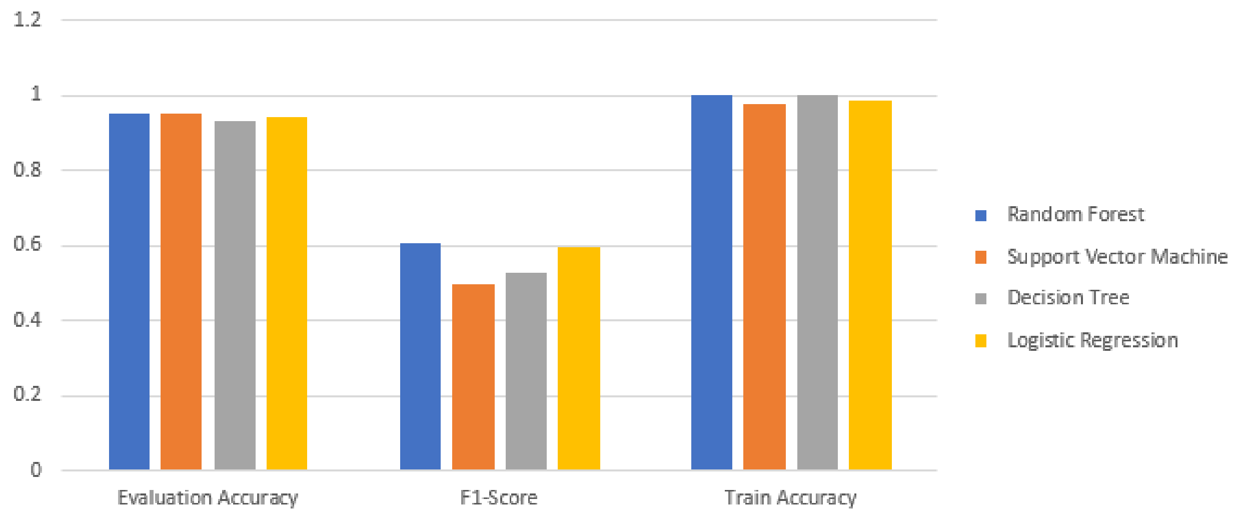
$$Precision = \frac{TP}{TP + FP}$$

**F-1 Score**

The F1 Score is the Precision and Recall Weighted Average. It takes into account both false positives and false negatives. If there is an uneven class distribution, F1 is typically more useful than precision. Accuracy functions well if there are equal costs for false positives and false negatives. If the expense of false positives and false negatives are somewhat different, both Precision and Recall are easier to look at.

$$\boldsymbol{F1\ score = \frac{2 * Recall * Precision}{Recall + Precision}}$$

| Model Evaluation | Random Forest | Logistic Regression | Decision Tree | Support Vector Machine |
|---|---|---|---|---|
| Train Accuracy | 0.9992 | 0.9851 | 0.9991 | 0.9781 |
| Evaluation Accuracy | 0.9521 | 0.9416 | 0.9309 | 0.9521 |
| F1-Score | 0.6069 | 0.5933 | 0.5282 | 0.4986 |



## Conclusion

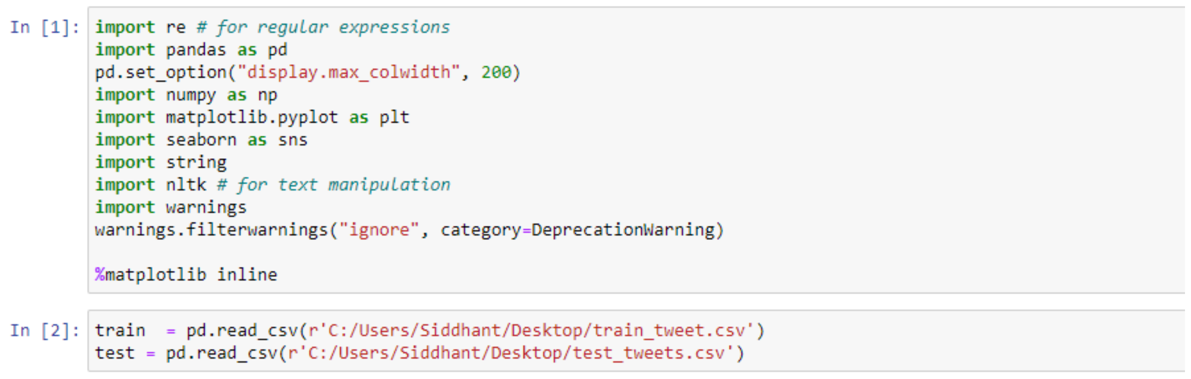
We used several training models in this project to make a prediction and tried to find the most suitable model on our dataset. During the process, in the NLTK and sci-kit-learn library, we learned to evaluate steps and training models that are helpful and easy to use. It produced distinct results with distinct vectorizers and Count vectorizer. For instance, by removing very low or high frequency words, we can enhance the model to make the prediction model filter outliers and boost the model's accuracy.

For the assessment of models such as Logistic Regression, SVM, Random Forest and Decision Tree on different extracted features such as Bag of Words, Word2Vec, Doc2Vec.

From the analysis of all algorithms, we can observe that using Random Forest Algorithm we are getting the highest values of evaluation metrics when compared to the other classification algorithms. Since F Measure is the highest (0.6069) and gives us the harmonic mean of precision and recall, it helps us in giving accurate results on the dataset as it depends on values of other important metrics. Also, the accuracy of the model if higher compared to the other models as it has the highest correctly predicted instances. Overall, by comparing the results of all the classification algorithms we can conclude the Random Forest shows the best performance having significantly higher values of F Score and accuracy.

## Code & Output

Important Libraries and storing the training and testing dataset in the train and test variables respectively.



```
In [1]: import re # for regular expressions
        import pandas as pd
        pd.set_option("display.max_colwidth", 200)
        import numpy as np
        import matplotlib.pyplot as plt
        import seaborn as sns
        import string
        import nltk # for text manipulation
        import warnings
        warnings.filterwarnings("ignore", category=DeprecationWarning)

        %matplotlib inline

In [2]: train  = pd.read_csv(r'C:/Users/Siddhant/Desktop/train_tweet.csv')
        test = pd.read_csv(r'C:/Users/Siddhant/Desktop/test_tweets.csv')
```

Checking out the negative comments from the train set

```
In [6]: train[train['label'] == 0].head(10)
```

Out[6]:

| | id | label | tweet |
|---|---|---|---|
| **0** | 1 | 0 | @user when a father is dysfunctional and is so selfish he drags his kids into his dysfunction. #run |
| **1** | 2 | 0 | @user @user thanks for #lyft credit i can't use cause they don't offer wheelchair vans in pdx. #disapointed #getthanked |
| **2** | 3 | 0 | bihday your majesty |
| **3** | 4 | 0 | #model i love u take with u all the time in urð□□±!!! ð□□□ð□□□ð□□□ð□□□ð□□¦ð□□¦ð□□¦ |
| **4** | 5 | 0 | factsguide: society now #motivation |
| **5** | 6 | 0 | [2/2] huge fan fare and big talking before they leave. chaos and pay disputes when they get there. #allshowandnogo |
| **6** | 7 | 0 | @user camping tomorrow @user @user @user @user @user @user @user dannyâ□¦ |
| **7** | 8 | 0 | the next school year is the year for exams.ð□□¯ can't think about that ð□□ #school #exams #hate #imagine #actorslife #revolutionschool #girl |
| **8** | 9 | 0 | we won!!! love the land!!! #allin #cavs #champions #cleveland #clevelandcavaliers â□¦ |
| **9** | 10 | 0 | @user @user welcome here ! i'm it's so #gr8 ! |

Checking out the positive comments from the train set

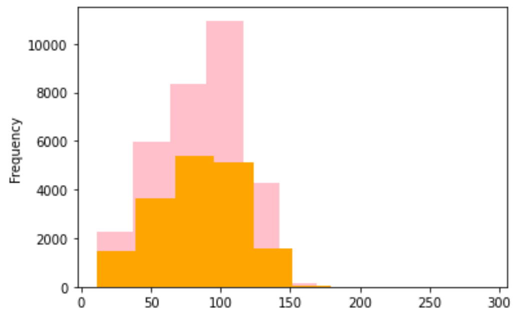
```
In [7]: train[train['label'] == 1].head(10)
```

Out[7]:

| | id | label | tweet |
|---|---|---|---|
| **13** | 14 | 1 | @user #cnn calls #michigan middle school 'build the wall' chant " #tcot |
| **14** | 15 | 1 | no comment! in #australia #opkillingbay #seashepherd #helpcovedolphins #thecove #helpcovedolphins |
| **17** | 18 | 1 | retweet if you agree! |
| **23** | 24 | 1 | @user @user lumpy says i am a . prove it lumpy. |
| **34** | 35 | 1 | it's unbelievable that in the 21st century we'd need something like this. again. #neverump #xenophobia |
| **56** | 57 | 1 | @user lets fight against #love #peace |
| **68** | 69 | 1 | ð□□©the white establishment can't have blk folx running around loving themselves and promoting our greatness |
| **77** | 78 | 1 | @user hey, white people: you can call people 'white' by @user #race #identity #medâ□¦ |
| **82** | 83 | 1 | how the #altright uses &amp; insecurity to lure men into #whitesupremacy |
| **111** | 112 | 1 | @user i'm not interested in a #linguistics that doesn't address #race &amp; . racism is about #power. #raciolinguistics bringsâ□¦ |

Checking the distribution of tweets in the data

```
In [9]: length_train = train['tweet'].str.len().plot.hist(color = 'pink', figsize = (6, 4))
        length_test = test['tweet'].str.len().plot.hist(color = 'orange', figsize = (6, 4))
```



A column was added in order to represent the length of the tweet and printing the first 10 records

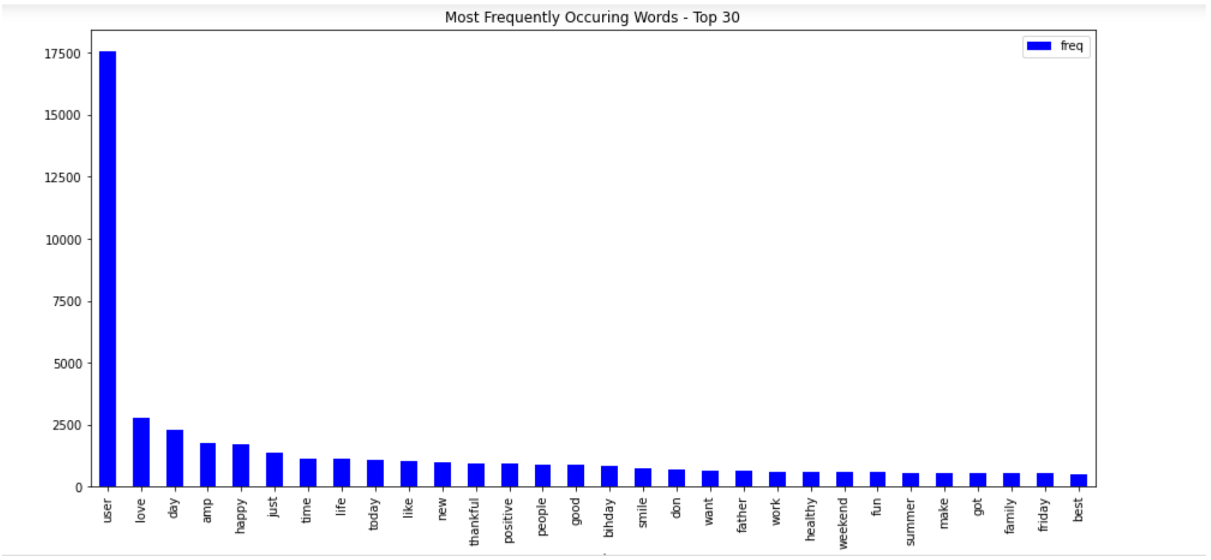
```
In [10]: train['len'] = train['tweet'].str.len()
         test['len'] = test['tweet'].str.len()
         train.head(10)
```

Out[10]:

| | id | label | tweet | len |
|---|---|---|---|---|
| 0 | 1 | 0 | @user when a father is dysfunctional and is so selfish he drags his kids into his dysfunction. #run | 102 |
| 1 | 2 | 0 | @user @user thanks for #lyft credit i can't use cause they don't offer wheelchair vans in pdx. #disapointed #getthanked | 122 |
| 2 | 3 | 0 | bihday your majesty | 21 |
| 3 | 4 | 0 | #model i love u take with u all the time in urð□□±!!! ð□□□ð□□□ð□□□ð□□□ð□□¦ð□□¦ð□□¦ | 86 |
| 4 | 5 | 0 | factsguide: society now #motivation | 39 |
| 5 | 6 | 0 | [2/2] huge fan fare and big talking before they leave. chaos and pay disputes when they get there. #allshowandnogo | 116 |
| 6 | 7 | 0 | @user camping tomorrow @user @user @user @user @user @user dannyâ□¦ | 74 |
| 7 | 8 | 0 | the next school year is the year for exams.ð□□¯ can't think about that ð□□ #school #exams #hate #imagine #actorslife #revolutionschool #girl | 143 |
| 8 | 9 | 0 | we won!!! love the land!!! #allin #cavs #champions #cleveland #clevelandcavaliers â□¦ | 87 |
| 9 | 10 | 0 | @user @user welcome here ! i'm it's so #gr8 ! | 50 |

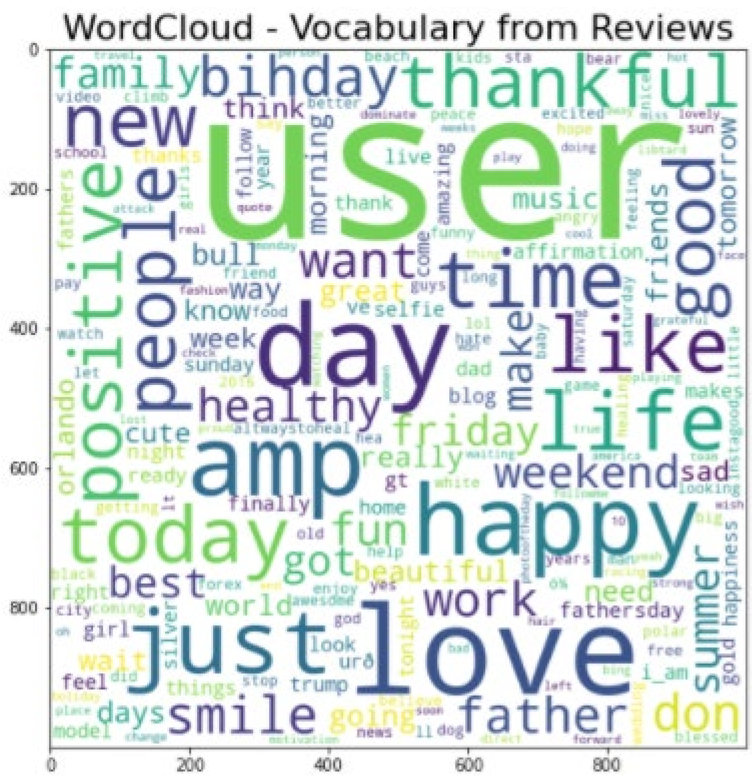
Showing the most frequently occurring words (Top 30)

```
In [13]: from sklearn.feature_extraction.text import CountVectorizer


         cv = CountVectorizer(stop_words = 'english')
         words = cv.fit_transform(train.tweet)

         sum_words = words.sum(axis=0)

         words_freq = [(word, sum_words[0, i]) for word, i in cv.vocabulary_.items()]
         words_freq = sorted(words_freq, key = lambda x: x[1], reverse = True)

         frequency = pd.DataFrame(words_freq, columns=['word', 'freq'])

         frequency.head(30).plot(x='word', y='freq', kind='bar', figsize=(15, 7), color = 'blue')
         plt.title("Most Frequently Occuring Words - Top 30")

Out[13]: Text(0.5, 1.0, 'Most Frequently Occuring Words - Top 30')
```

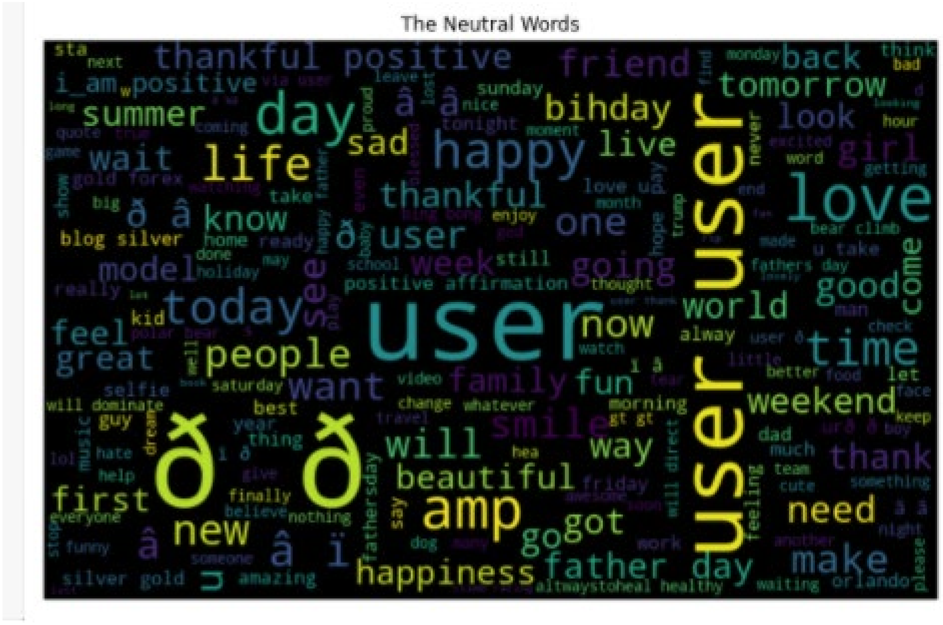


Creating word cloud for positive negative and neutral words:

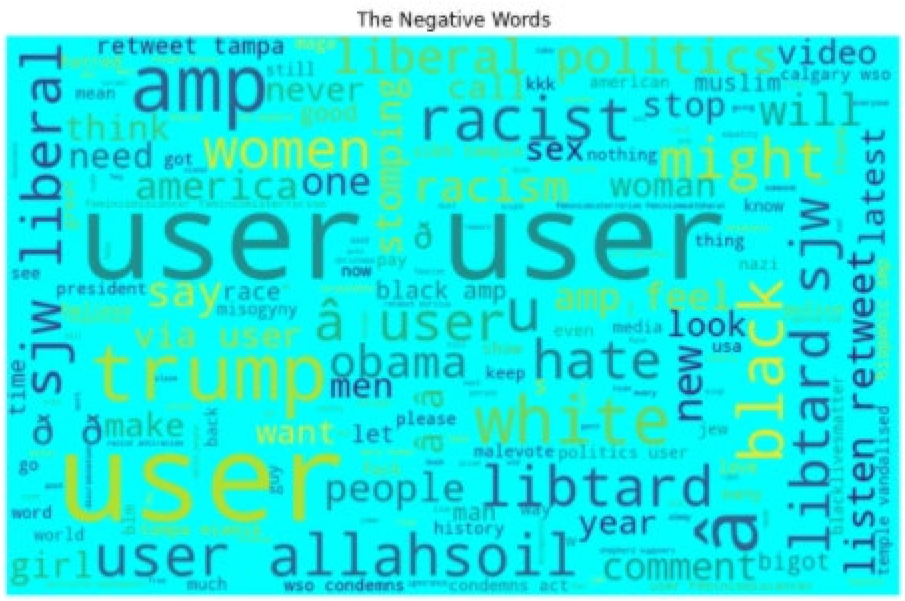
```
In [15]: from wordcloud import WordCloud

         wordcloud = WordCloud(background_color = 'white', width = 1000, height = 1000).generate_from_frequencies(dict(words_freq))

         plt.figure(figsize=(10,8))
         plt.imshow(wordcloud)
         plt.title("WordCloud - Vocabulary from Reviews", fontsize = 22)

Out[15]: Text(0.5, 1.0, 'WordCloud - Vocabulary from Reviews')
```



```
In [16]: normal_words =' '.join([text for text in train['tweet'][train['label'] == 0]])

         wordcloud = WordCloud(width=800, height=500, random_state = 0, max_font_size = 110).generate(normal_words)
         plt.figure(figsize=(10, 7))
         plt.imshow(wordcloud, interpolation="bilinear")
         plt.axis('off')
         plt.title('The Neutral Words')
         plt.show()
```



```
In [17]: negative_words =' '.join([text for text in train['tweet'][train['label'] == 1]])

         wordcloud = WordCloud(background_color = 'cyan', width=800, height=500, random_state = 0, max_font_size = 110).generate(negative_
         plt.figure(figsize=(10, 7))
         plt.imshow(wordcloud, interpolation="bilinear")
         plt.axis('off')
         plt.title('The Negative Words')
         plt.show()
```

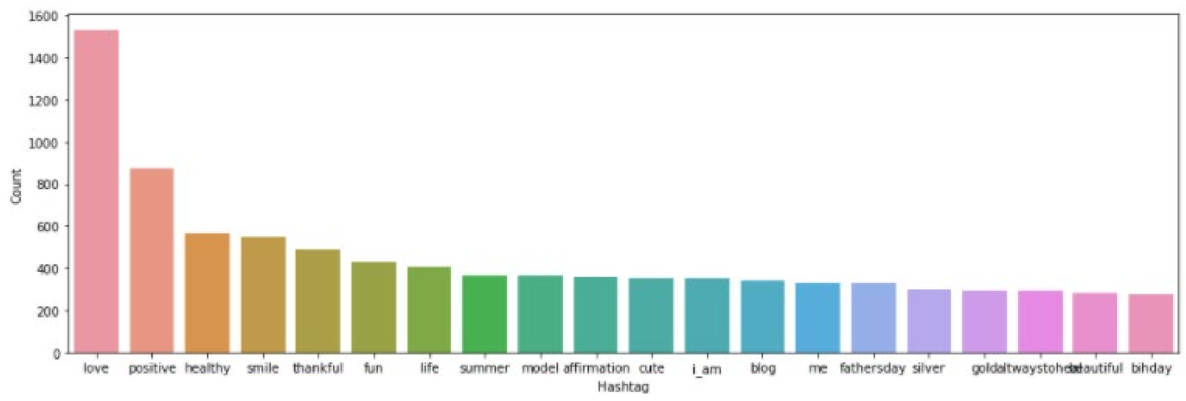


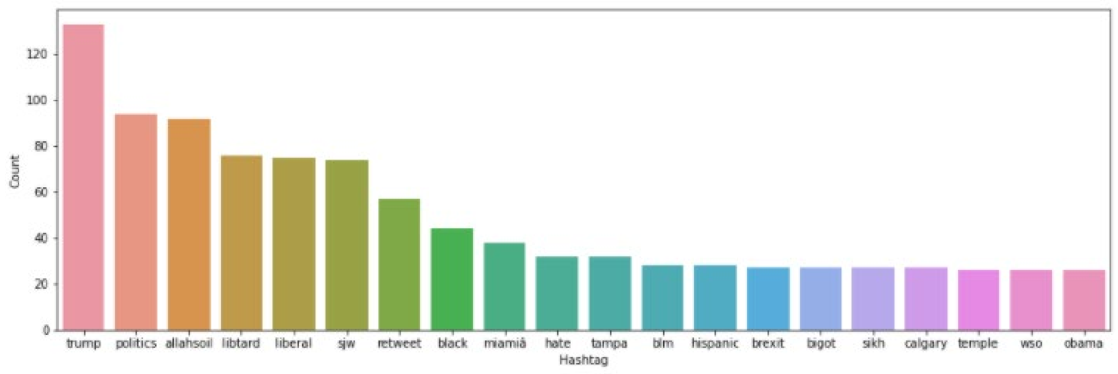
Collecting hashtags from non-racist/ non-sexist tweets and racist/ sexist tweets, then displaying the top 20 most frequent hashtags.

```
In [18]: def hashtag_extract(x):
             hashtags = []

             for i in x:
                 ht = re.findall(r"#(\w+)", i)
                 hashtags.append(ht)

             return hashtags
```

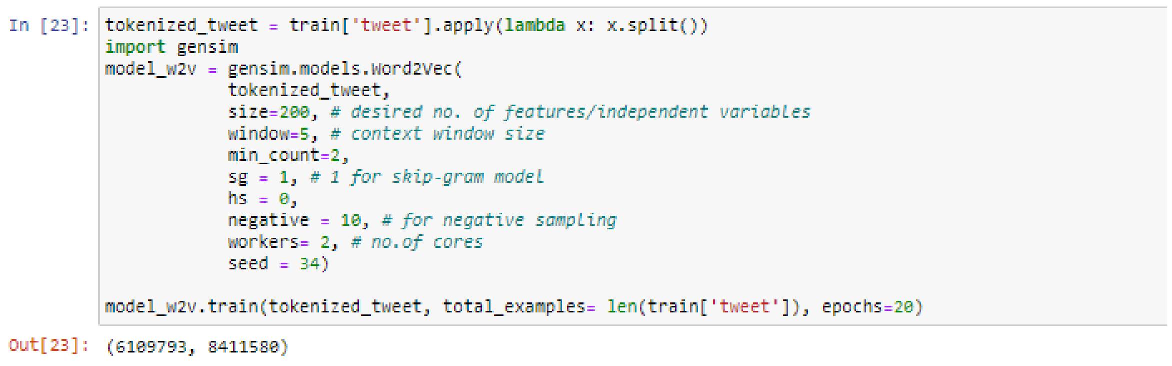
```
In [19]: HT_regular = hashtag_extract(train['tweet'][train['label'] == 0])
         HT_negative = hashtag_extract(train['tweet'][train['label'] == 1])
         HT_regular = sum(HT_regular,[])
         HT_negative = sum(HT_negative,[])
```

```
In [20]: a = nltk.FreqDist(HT_regular)
         d = pd.DataFrame({'Hashtag': list(a.keys()),
                           'Count': list(a.values())})
         d = d.nlargest(columns="Count", n = 20)
         plt.figure(figsize=(16,5))
         ax = sns.barplot(data=d, x= "Hashtag", y = "Count")
         ax.set(ylabel = 'Count')
         plt.show()
```



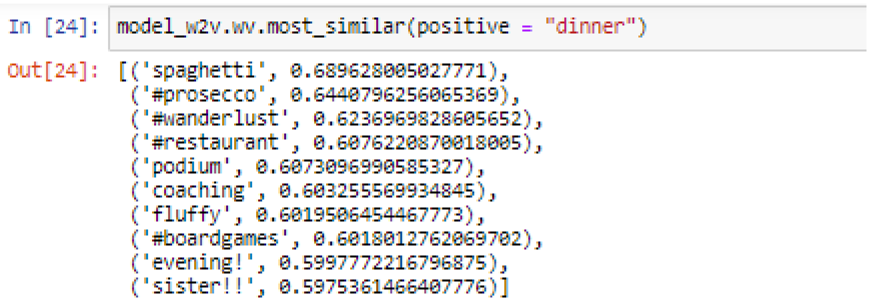


Tokenizing the words present in the training set and creating a word to vector model.

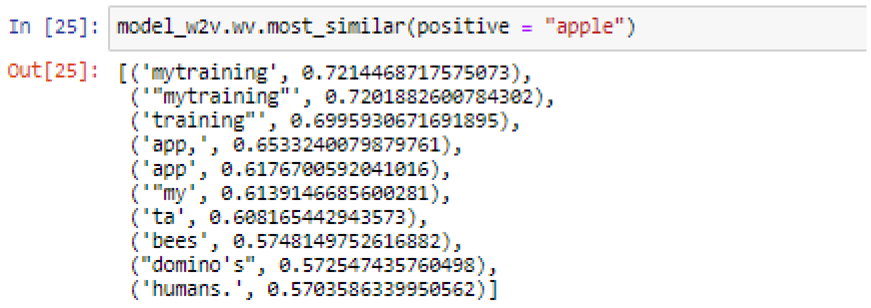


```
In [23]: tokenized_tweet = train['tweet'].apply(lambda x: x.split())
         import gensim
         model_w2v = gensim.models.Word2Vec(
                     tokenized_tweet,
                     size=200, # desired no. of features/independent variables
                     window=5, # context window size
                     min_count=2,
                     sg = 1, # 1 for skip-gram model
                     hs = 0,
                     negative = 10, # for negative sampling
                     workers= 2, # no.of cores
                     seed = 34)

         model_w2v.train(tokenized_tweet, total_examples= len(train['tweet']), epochs=20)

Out[23]: (6109793, 8411580)
```

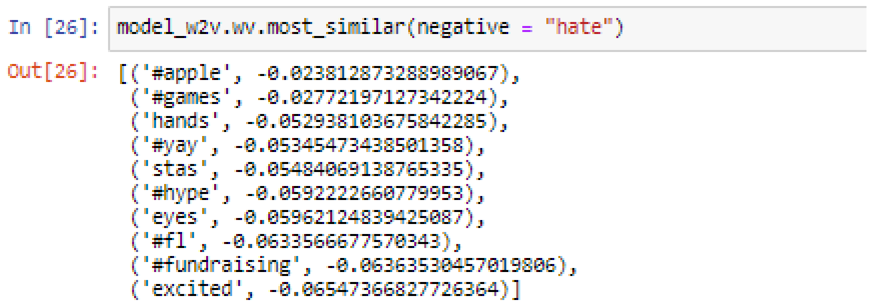
Testing the word to vector model



```
In [24]: model_w2v.wv.most_similar(positive = "dinner")

Out[24]: [('spaghetti', 0.689628005027771),
          ('#prosecco', 0.6440796256065369),
          ('#wanderlust', 0.6236969828605652),
          ('#restaurant', 0.6076220870018005),
          ('podium', 0.6073096990585327),
          ('coaching', 0.603255569934845),
          ('fluffy', 0.6019506454467773),
          ('#boardgames', 0.6018012762069702),
          ('evening!', 0.5997772216796875),
          ('sister!!', 0.5975361466407776)]
```

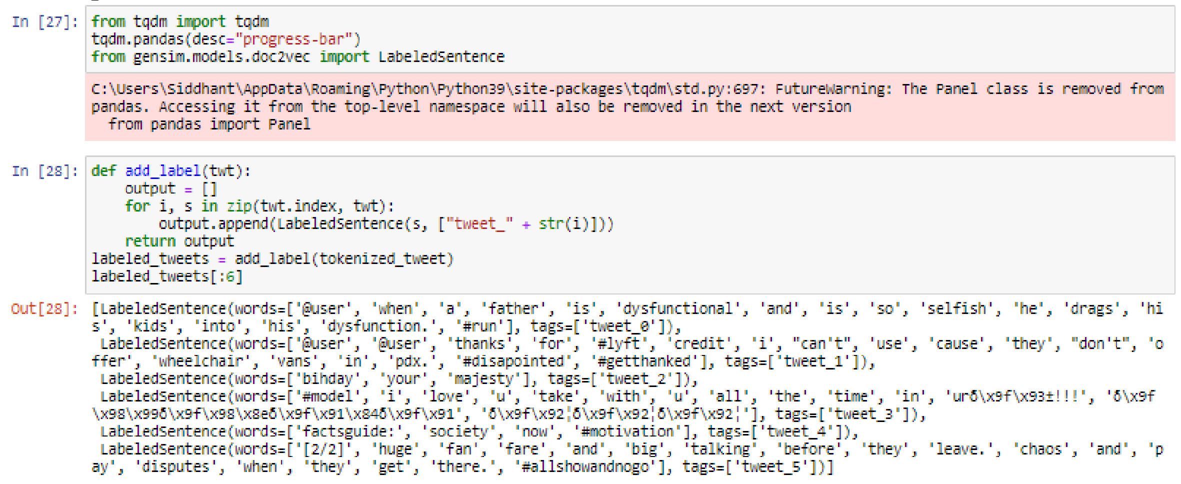


```
In [25]: model_w2v.wv.most_similar(positive = "apple")

Out[25]: [('mytraining', 0.7214468717575073),
          ('"mytraining"', 0.7201882600784302),
          ('training"', 0.6995930671691895),
          ('app,', 0.6533240079879761),
          ('app', 0.6176700592041016),
          ('"my', 0.6139146685600281),
          ('ta', 0.608165442943573),
          ('bees', 0.5748149752616882),
          ("domino's", 0.572547435760498),
          ('humans.', 0.5703586339950562)]
```



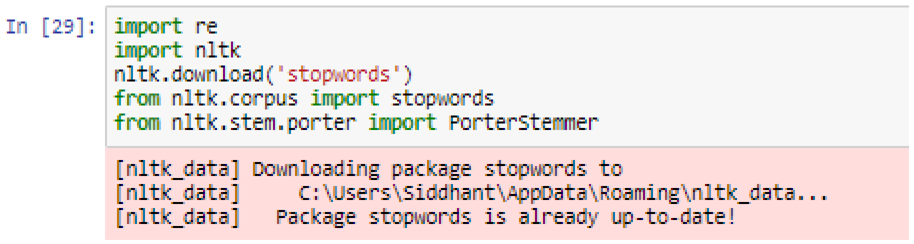
```
In [26]: model_w2v.wv.most_similar(negative = "hate")

Out[26]: [('#apple', -0.023812873288989067),
          ('#games', -0.02772197127342224),
          ('hands', -0.052938103675842285),
          ('#yay', -0.05345473438501358),
          ('stas', -0.05484069138765335),
          ('#hype', -0.0592222660779953),
          ('eyes', -0.059621248394250087),
          ('#fl', -0.0633566677570343),
          ('#fundraising', -0.06363530457019806),
          ('excited', -0.06547366827726364)]
```

Labelling all the tweets.



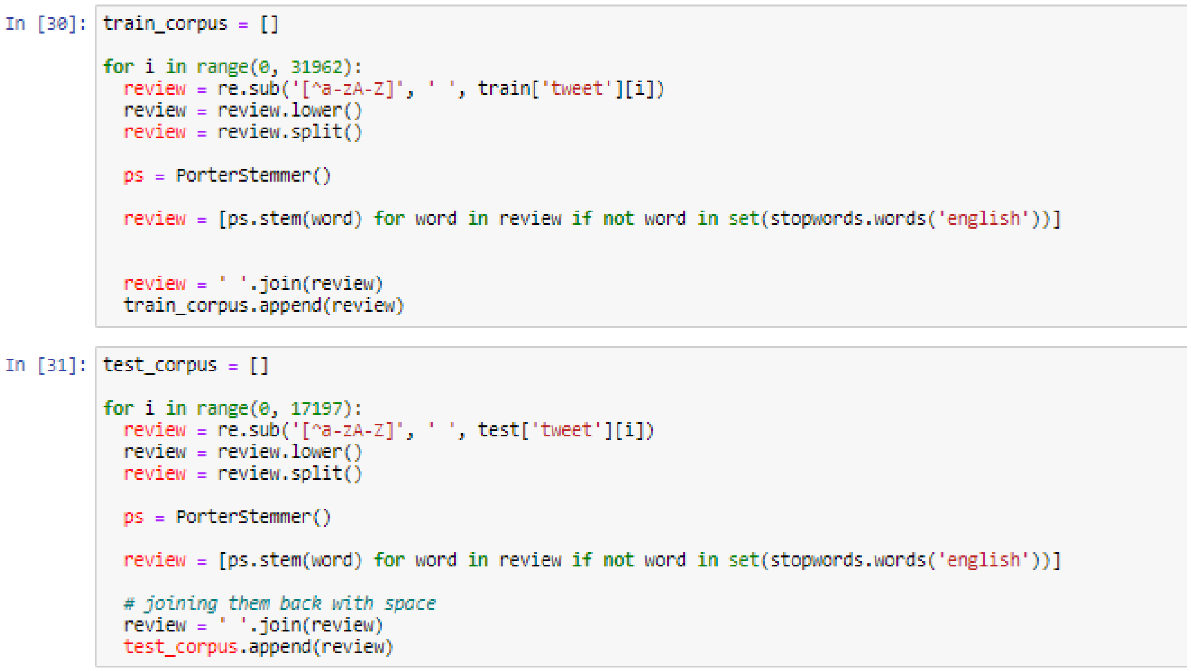
```
In [27]: from tqdm import tqdm
         tqdm.pandas(desc="progress-bar")
         from gensim.models.doc2vec import LabeledSentence

         C:\Users\Siddhant\AppData\Roaming\Python\Python39\site-packages\tqdm\std.py:697: FutureWarning: The Panel class is removed from
         pandas. Accessing it from the top-level namespace will also be removed in the next version
           from pandas import Panel

In [28]: def add_label(twt):
             output = []
             for i, s in zip(twt.index, twt):
                 output.append(LabeledSentence(s, ["tweet_" + str(i)]))
             return output
         labeled_tweets = add_label(tokenized_tweet)
         labeled_tweets[:6]

Out[28]: [LabeledSentence(words=['@user', 'when', 'a', 'father', 'is', 'dysfunctional', 'and', 'is', 'so', 'selfish', 'he', 'drags', 'hi
         s', 'kids', 'into', 'his', 'dysfunction.', '#run'], tags=['tweet_0']),
          LabeledSentence(words=['@user', '@user', 'thanks', 'for', '#lyft', 'credit', 'i', "can't", 'use', 'cause', 'they', "don't", 'o
         ffer', 'wheelchair', 'vans', 'in', 'pdx.', '#disapointed', '#getthanked'], tags=['tweet_1']),
          LabeledSentence(words=['bihday', 'your', 'majesty'], tags=['tweet_2']),
          LabeledSentence(words=['#model', 'i', 'love', 'u', 'take', 'with', 'u', 'all', 'the', 'time', 'in', 'urð\x9f\x93±!!!', 'ð\x9f
         \x98\x99ð\x9f\x98\x8eð\x9f\x91\x84ð\x9f\x91', 'ð\x9f\x92¦ð\x9f\x92¦ð\x9f\x92¦'], tags=['tweet_3']),
          LabeledSentence(words=['factsguide:', 'society', 'now', '#motivation'], tags=['tweet_4']),
          LabeledSentence(words=['[2/2]', 'huge', 'fan', 'fare', 'and', 'big', 'talking', 'before', 'they', 'leave.', 'chaos', 'and', 'p
         ay', 'disputes', 'when', 'they', 'get', 'there.', '#allshowandnogo'], tags=['tweet_5'])]
```

Removing unwanted patterns from the data.



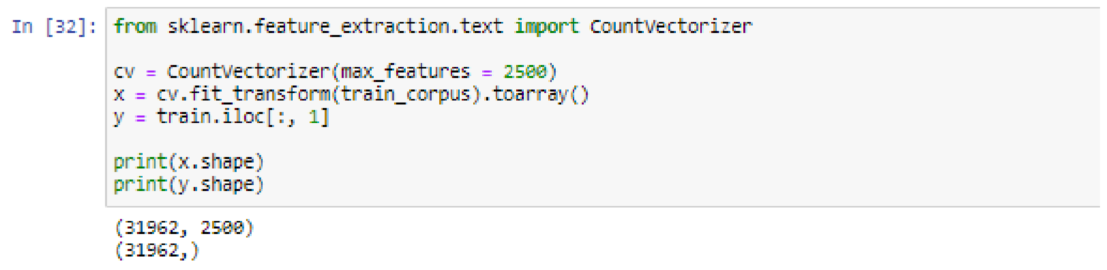
```
In [29]: import re
         import nltk
         nltk.download('stopwords')
         from nltk.corpus import stopwords
         from nltk.stem.porter import PorterStemmer

         [nltk_data] Downloading package stopwords to
         [nltk_data]     C:\Users\Siddhant\AppData\Roaming\nltk_data...
         [nltk_data]   Package stopwords is already up-to-date!
```

Using the Porter Stemmer for stemming the words and then joining them back with a space.

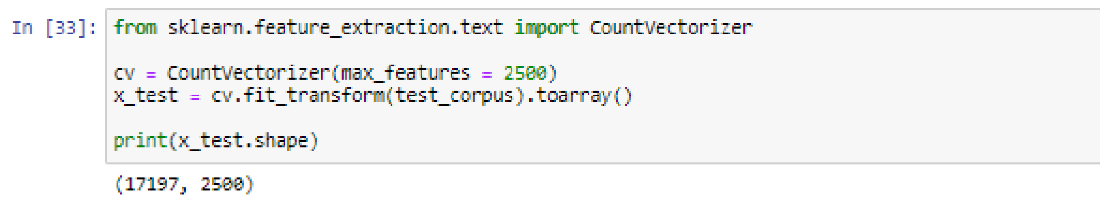


```
In [30]: train_corpus = []

         for i in range(0, 31962):
           review = re.sub('[^a-zA-Z]', ' ', train['tweet'][i])
           review = review.lower()
           review = review.split()

           ps = PorterStemmer()

           review = [ps.stem(word) for word in review if not word in set(stopwords.words('english'))]


           review = ' '.join(review)
           train_corpus.append(review)

In [31]: test_corpus = []

         for i in range(0, 17197):
           review = re.sub('[^a-zA-Z]', ' ', test['tweet'][i])
           review = review.lower()
           review = review.split()

           ps = PorterStemmer()

           review = [ps.stem(word) for word in review if not word in set(stopwords.words('english'))]

           # joining them back with space
           review = ' '.join(review)
           test_corpus.append(review)
```

## Creating Bag of Words

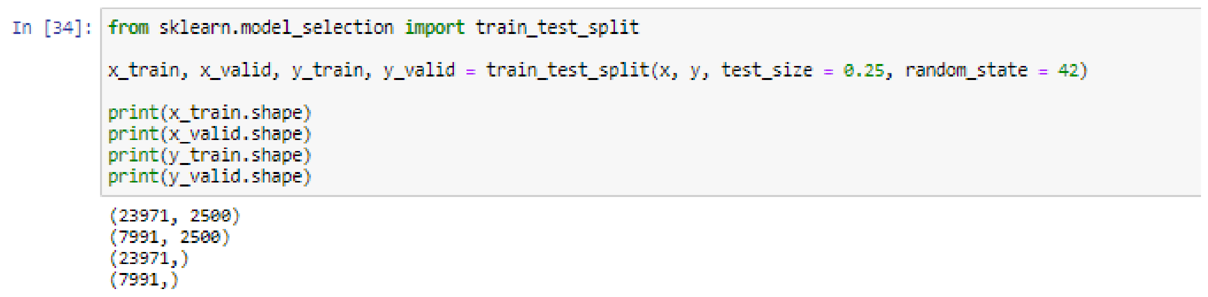


```
In [32]: from sklearn.feature_extraction.text import CountVectorizer

         cv = CountVectorizer(max_features = 2500)
         x = cv.fit_transform(train_corpus).toarray()
         y = train.iloc[:, 1]

         print(x.shape)
         print(y.shape)

         (31962, 2500)
         (31962,)
```



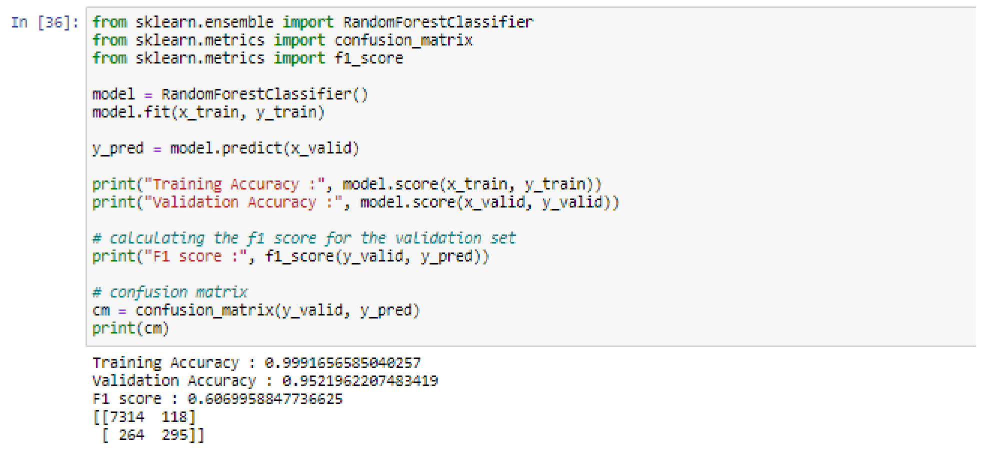
```
In [33]: from sklearn.feature_extraction.text import CountVectorizer

         cv = CountVectorizer(max_features = 2500)
         x_test = cv.fit_transform(test_corpus).toarray()

         print(x_test.shape)

         (17197, 2500)
```

## Splitting the training data into train and valid sets



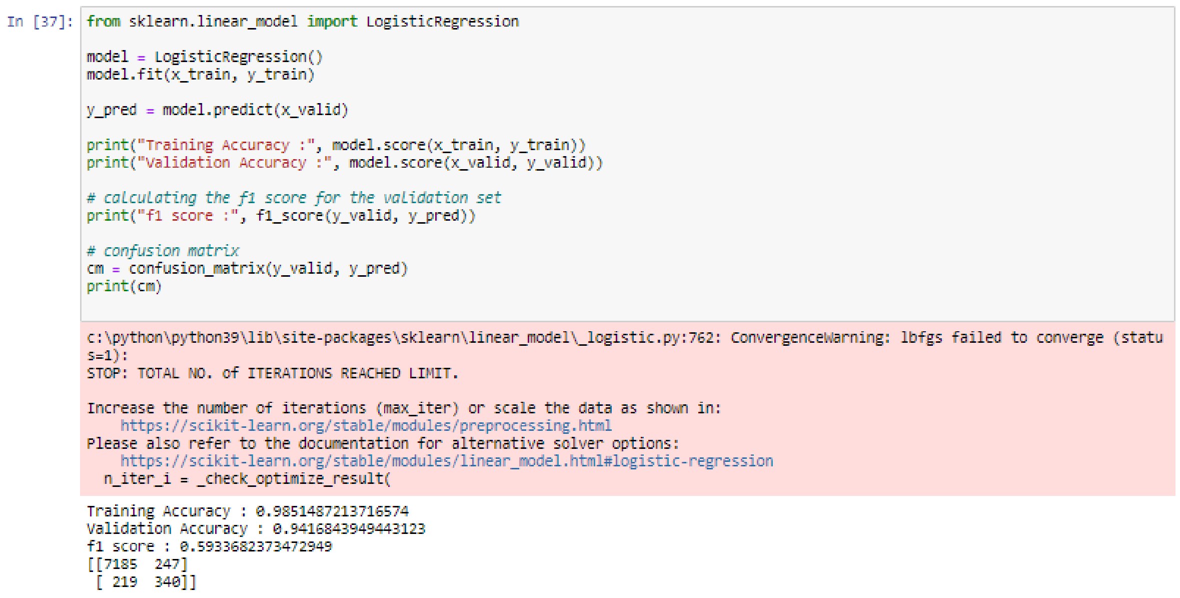
```
In [34]: from sklearn.model_selection import train_test_split

         x_train, x_valid, y_train, y_valid = train_test_split(x, y, test_size = 0.25, random_state = 42)

         print(x_train.shape)
         print(x_valid.shape)
         print(y_train.shape)
         print(y_valid.shape)

         (23971, 2500)
         (7991, 2500)
         (23971,)
         (7991,)
```

## Implementing Random Forest Classifier



```
In [36]: from sklearn.ensemble import RandomForestClassifier
         from sklearn.metrics import confusion_matrix
         from sklearn.metrics import f1_score

         model = RandomForestClassifier()
         model.fit(x_train, y_train)

         y_pred = model.predict(x_valid)

         print("Training Accuracy :", model.score(x_train, y_train))
         print("Validation Accuracy :", model.score(x_valid, y_valid))

         # calculating the f1 score for the validation set
         print("F1 score :", f1_score(y_valid, y_pred))

         # confusion matrix
         cm = confusion_matrix(y_valid, y_pred)
         print(cm)

         Training Accuracy : 0.9991656585040257
         Validation Accuracy : 0.9521962207483419
         F1 score : 0.6069958847736625
         [[7314  118]
          [ 264  295]]
```

## Implementing Logistic Regression Model

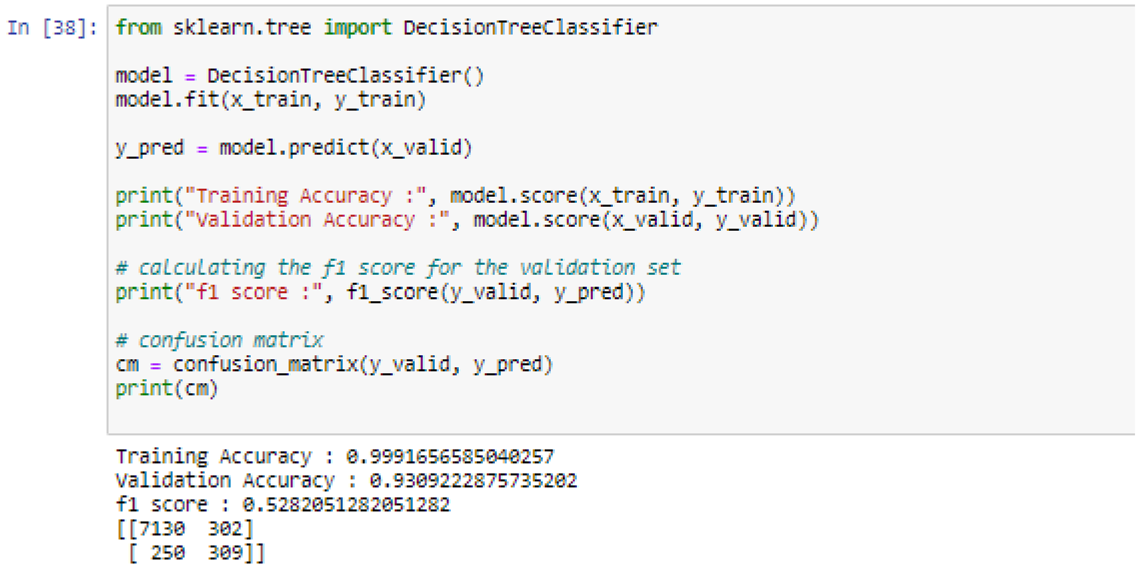


```
In [37]: from sklearn.linear_model import LogisticRegression

         model = LogisticRegression()
         model.fit(x_train, y_train)

         y_pred = model.predict(x_valid)

         print("Training Accuracy :", model.score(x_train, y_train))
         print("Validation Accuracy :", model.score(x_valid, y_valid))

         # calculating the f1 score for the validation set
         print("f1 score :", f1_score(y_valid, y_pred))

         # confusion matrix
         cm = confusion_matrix(y_valid, y_pred)
         print(cm)
```

```
c:\python\python39\lib\site-packages\sklearn\linear_model\_logistic.py:762: ConvergenceWarning: lbfgs failed to converge (statu
s=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
Training Accuracy : 0.9851487213716574
Validation Accuracy : 0.9416843949443123
f1 score : 0.5933682373472949
[[7185  247]
 [ 219  340]]
```

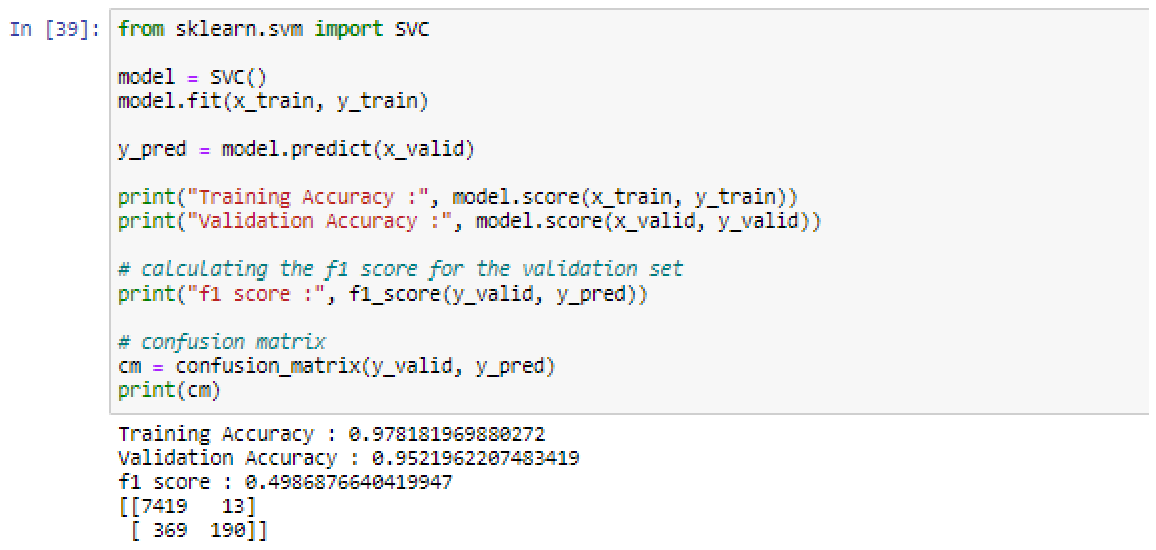
## Implementing Decision Tree Classifier



```
In [38]: from sklearn.tree import DecisionTreeClassifier

         model = DecisionTreeClassifier()
         model.fit(x_train, y_train)

         y_pred = model.predict(x_valid)

         print("Training Accuracy :", model.score(x_train, y_train))
         print("Validation Accuracy :", model.score(x_valid, y_valid))

         # calculating the f1 score for the validation set
         print("f1 score :", f1_score(y_valid, y_pred))

         # confusion matrix
         cm = confusion_matrix(y_valid, y_pred)
         print(cm)
```

```
Training Accuracy : 0.9991656585040257
Validation Accuracy : 0.9309222875735202
f1 score : 0.5282051282051282
[[7130  302]
 [ 250  309]]
```

Implementing Support Vector Machine



```
In [39]: from sklearn.svm import SVC

         model = SVC()
         model.fit(x_train, y_train)

         y_pred = model.predict(x_valid)

         print("Training Accuracy :", model.score(x_train, y_train))
         print("Validation Accuracy :", model.score(x_valid, y_valid))

         # calculating the f1 score for the validation set
         print("f1 score :", f1_score(y_valid, y_pred))

         # confusion matrix
         cm = confusion_matrix(y_valid, y_pred)
         print(cm)

         Training Accuracy : 0.978181969880272
         Validation Accuracy : 0.9521962207483419
         f1 score : 0.4986876640419947
         [[7419   13]
          [ 369  190]]
```

## References

Anandarajan, M., Hill, C., & Nolan, T. (2018). Introduction to Text Analytics. *Practical Text Analytics Advances in Analytics and Data Science,* 1-11. Sidoi:10.1007/978-3-319-95663-3_1

Fersini, E. (2017). Sentiment Analysis in Social Networks. *Sentiment Analysis in Social Networks,* 91-111. doi:10.1016/b978-0-12-804412-4.00006-1

Tan, P., Karpatne, A., Kumar, V., & Steinbach, M. (2020). *Introduction to data mining*. Harlow: Pearson.

Singh, T., Kumari, M. (2016). Role of Text Pre-processing in Twitter Sentiment Analysis. Procedia Computer Science, 89, 549-554. doi:10.1016/j.procs.2016.06.095

Yogish, D., Manjunath, T. N., &amp; Hegadi, R. S. (2019). Review on Natural Language Processing Trends and Techniques Using NLTK. Communications in Computer and Information Science Recent Trends in Image Processing and Pattern Recognition, 589-606. doi:10.1007/978-981-13-9187-3_53

Beatty, M. (2020). Classification Methods for Hate Speech Diffusion: Detecting theSpread of Hate Speech on Twitter. Proceedings of the 6th World Congress on Electrical Engineering and Computer Systems and Science. doi:10.11159/cist20.105

BOUKKOURI, H. (2020, September 12). Text Classification: The First Step Toward NLP Mastery. Retrieved December 10, 2020, from https://medium.com/data-from-the-trenches/text-classification-the-first-step-toward-nlp-mastery-f5f95d525d73

Shivam BansalShivam Bansal is a data scientist with exhaustive experience in Natural Language Processing and Machine Learning in several domains. He is passionate about learning and always looks forward to solving challenging analytical problems. (2019, July 26). A Comprehensive Guide to Understand and Implement Text Classification in Python. Retrieved December 10, 2020, from https://www.analyticsvidhya.com/blog/2018/04/a-comprehensive-guide-to-understand-and-implement-text-classification-in-python/

Sentiment Classification using Machine Learning Techniques. (2016). International Journal of Science and Research (IJSR), 5(4), 819-821. doi:10.21275/v5i4.nov162724