SENTIMENT ANALYSIS OF UNION BUDGET 2019

GROUP MEMBERS:

- ARIJIT RAY, THE HERITAGE ACADEMY, 172131010016 OF 2017-2018
- SANKHA MISRA, THE HERITAGE ACADEMY, 172131010080 OF 2017-2018
- SACHIN KUMAR ROY, MCKV INSTITUTE OF ENGINEERING, 171160110039 OF 2017-2018
- DIPAK GUPTA, MCKV INSTITUTE OF ENGINEERING, 171160110021 OF 2017-2018
- RIDDHINATH GANGULY, MCKV INSTITUTE OF ENGINEERING, 171160110038 OF 2017-2018

TABLE OF CONTENTS

ACKNOWLEDGEMENT:

I take this opportunity to express my profound gratitude and deep regards to my faculty **Prof. Arnab Chakraborty** for his exemplary guidance, monitoring and constant encouragement throughout the course of this project. The blessing, help and guidance given by him time to time shall carry me a long way in this journey of life on which I am about to embark.

I am obliged to my project team members for the valuable information provided by them in their respective fields. I am grateful for their cooperation during the period of my assignment.

Sachin Kumar Roy,

Riddhinath Ganguly,

Sankha Misra,

Dipak Gupta,

Arijit Ray

PROJECT OBJECTIVE:

Social media has become an effective tool in any organisation's public reception. People are more engaged in the social life media nowadays thanks to availability of internet data. Online social media analytics is a powerful tool to boost e-commerce, politics etc. In marketing field companies use it to develop their strategies, to understand customers' feelings towards products or brand, how people respond to their campaigns or product launches and why consumers don't buy some products. Sentiment analysis also is used to monitor and analyse social phenomena, for the spotting of potentially dangerous situations and determining the general mood of the blogosphere. In political field, it is used to keep track of political view, to detect consistency and inconsistency between statements and actions at the government level. It can be used to predict election results as well! In this project we have a dataset of Union Budget 2019 of India that consists of tweets of public about Budget and label that opinion is positive or negative.

Target variable:

Label: The tweet has label (4 = Positive and 0 = Negative)

Goals:

- Predict the tweets on Union Budget are positive or negative.
- Prepare dataset to train and to test on different classifier.
- Create 4 different types of models on data – **Naïve Bayes-Multinomial Classifier, SVM-Linear Classifier, SVM-SGD Classifier, Random Forest Classifier** based on the training set.
- Apply the test set on this Classifier and **compare the difference in the accuracies.**
- Plot different module results and comparison using matplotlib.pyplot, seaborn.

Way to approach:

1. We have collected the required twitter sentiment dataset for training and testing from online resources.
2. We have train this data in different classifier and check the accuracy result of those.
3. We apply the testing data into our classifier to get the labelling of positives and negative values of the test data.
4. Then we check the amount of positive and negative to have a firm knowledge about how well it is taken by the public.

PROJECT SCOPE:

Query: The Union Budget 2019 is well accepted or rejected according to tweets at twitter

Disclaimer: Our sentiment analysis is purely based on collected data from tweets on the social networking platform “Twitter”. The data is collected on an unbiased method. This analysis is solely for educational purposes.

Scope: In the following project, we use the sentiment140 dataset and uniondata dataset as training and testing data respectively. Firstly we perform the Lexicon-Based Sentiment Analyser to find out the count of positive and negative tweets from the union data.

Then comes the Machine Learning approach for sentiment analysis. In this technique, firstly the dataset are splited in X_train and X_test dataset to conduct training with X_train and test the performance with X_test. Following this, we clean up the test and training dataset and construct the feature vector of the training dataset. Then, we implement the classifiers constructing individual pipelines for each of them. Next, we train the dataset and store the performance metrics in a list. Lastly we show the positive and negative sentiment count for each classifier and see which classifier gives the best result.

Finally our aim is to answer our query, “The people like the new budget or not”.

REQUIREMENT SPECIFICATIONS:

System requirements:

1. CPU: 2 x 64-bit 2.8 GHz 8.00 GT/s CPUs
2. RAM: 32 GB (or 16 GB of 1600 MHz DDR3 RAM)
3. Storage: 300 GB. (600 GB for air-gapped deployments.) Additional space recommended if the repository will be used to store packages built by the customer. With an empty repository, a base install requires 2 GB.
4. Internet access to download the files from Anaconda Cloud or a USB drive containing all of the files you need with alternate instructions for air gapped installations

Software requirements:

1. Anaconda Navigator v3.6.4
2. SciKit learn package for Python
3. Google Chrome with Web Scrapper Extension
4. Twitter Account

DATA DESCRIPTION:

Sources of data:

Training Data Source:

Description: Sentiment140 Dataset (link: http://help.sentiment140.com/for-students)

Dataset name: train123.csv

Format:

The data is a CSV with emoticons removed. Data file format has 6 fields:
0 - The polarity of the tweet (0 = negative, 4 = positive)
1 - The id of the tweet (2087)
2 - The date of the tweet (Sat May 16 23:58:44 UTC 2009)
3 - The query (lyx). If there is no query, then this value is NO_QUERY.
4 - The user that tweeted (robotickilldozr)
5 - The text of the tweet (Lyx is cool)

Test Data:

Description: This is a data collected specifically for this project.

Dataset name: test12.csv

Format:

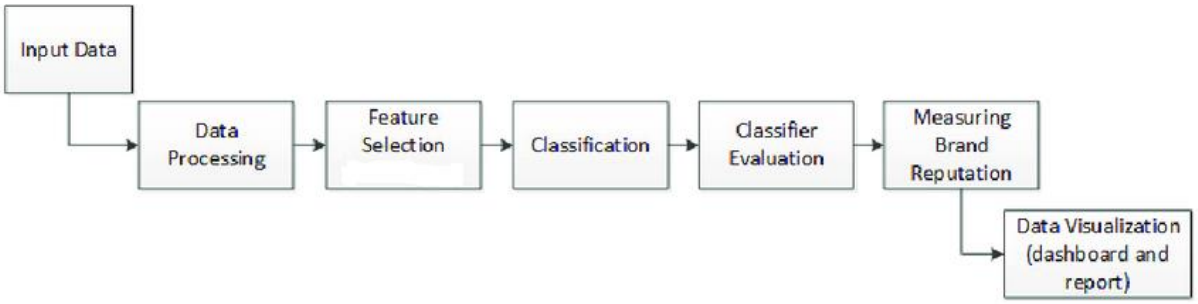
1. 709 rows
2. 1 Column (tweets)

Requirements for data scraping:

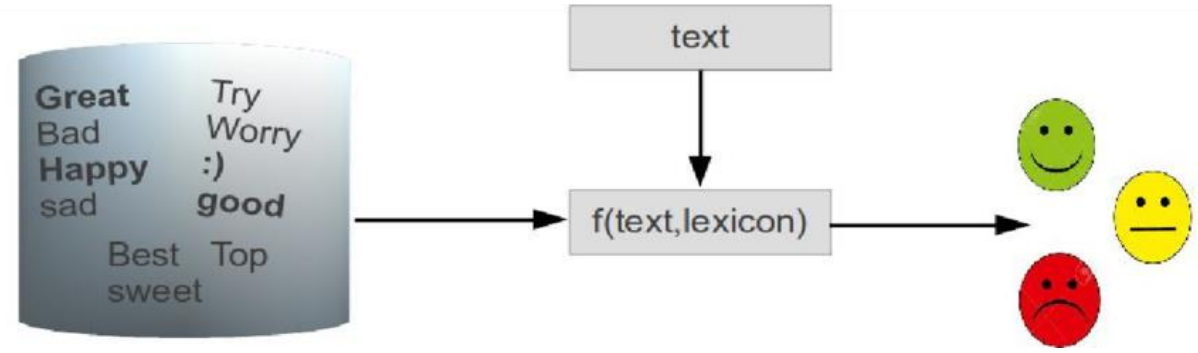
- The Advanced Search URL for tweets on Twitter for Union Budget 2019
- Google Chrome with Web Scraper extension
- Twitter Advanced Search Scraper.
  (Link:https://gist.github.com/scrapehero/d0305d8d15b0e447dcefdf548a9846e9)

WORKFLOW:

Machine Learning Approach:



Lexicon Based Approach:



MODEL BUILDING:

Lexicon-Based Approaches:

Description:

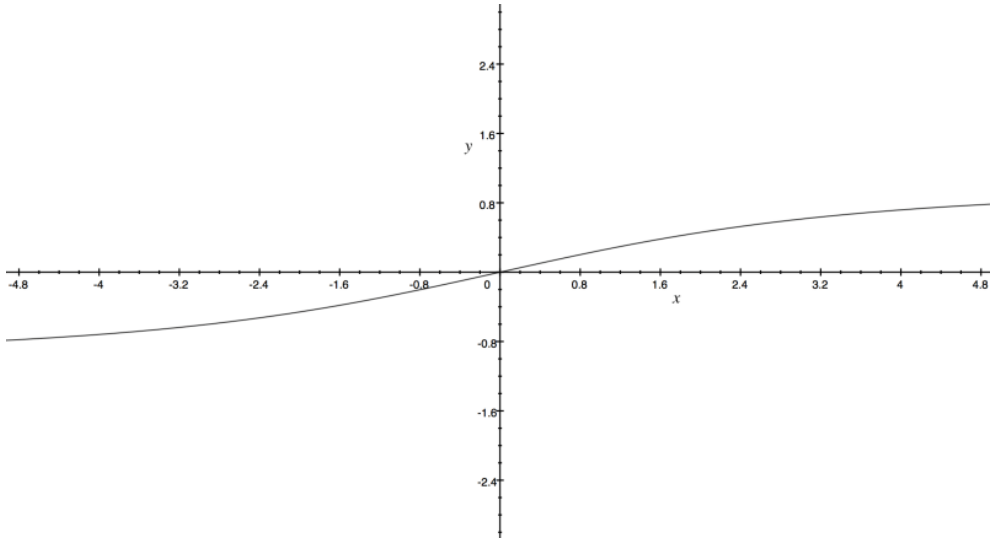
Lexical approaches aim to map words to *sentiment* by building a lexicon or a 'dictionary of sentiment.' We can use this dictionary to assess the sentiment of phrases and sentences, without the need of looking at anything else. Sentiment can be categorical – such as {negative, neutral, positive} – or it can be numerical – like a range of intensities or scores. Lexical approaches look at the sentiment category or score of each word in the sentence and decide what the sentiment category or score of the whole sentence is. The power of lexical approaches lies in the fact that we do not need to train a model using labelled data, since we have everything we need to assess the sentiment of sentences in the dictionary of emotions. VADER is an example of a lexical method.

Primarily, VADER sentiment analysis relies on a dictionary which maps lexical features to emotion intensities called sentiment scores. The sentiment score of a text can be obtained by summing up the intensity of each word in the text. It returns a sentiment score in the range -1 to 1, from most negative to most positive. The sentiment score of a sentence is calculated by summing up the sentiment scores of each VADER-dictionary-listed word in the sentence. Cautious readers would probably notice that there is a contradiction: individual words have a sentiment score between -4 to 4, but the returned sentiment score of a sentence is between -1 to 1. They're both true. The sentiment score of a sentence is the sum of the sentiment score of each sentiment-bearing word. However, we apply normalization to the total to map it to a value between -1 to 1.

The normalization used by VADER is

$$\frac{x}{\sqrt{x^2+\alpha}}$$

where x is the sum of the sentiment scores of the constituent words of the sentence and α is a normalization parameter that we set to 15. The normalization is graphed below.



We see here that as x grows larger, it gets more and more close to -1 or 1. To similar effect, if there are a lot of words in the document you're applying VADER sentiment analysis to, you get a score

close to -1 or 1. Thus, VADER sentiment analysis works best on short documents, like tweets and sentences, not on large documents.

Five Simple Heuristics of VADER Lexicon:

Lexical features aren't the only things in the sentence which affect the sentiment. There are other contextual elements, like punctuation, capitalization, and modifiers which also impart emotion. VADER sentiment analysis takes these into account by considering five simple heuristics. The effect of these heuristics is, again, quantified using human raters.

The first heuristic is **punctuation**. Compare "I like it." and "I like it!!!" It's not really hard to argue that the second sentence has more intense emotion than the first, and therefore must have a higher VADER sentiment score.

VADER sentiment analysis takes this into account by amplifying the sentiment score of the sentence proportional to the number of exclamation points and question marks ending the sentence. VADER first computes the sentiment score of the sentence. If the score is positive, VADER adds a certain empirically-obtained quantity for every exclamation point (0.292) and question mark (0.18). If the score is negative, VADER subtracts.

The second heuristic is **capitalization**. "AMAZING performance." is definitely more intense than "amazing performance." And so VADER takes this into account by incrementing or decrementing the sentiment score of the word by 0.733, depending on whether the word is positive or negative, respectively.

The third heuristic is the use of **degree modifiers**. Take for example "effing cute" and "sort of cute". The effect of the modifier in the first sentence is to increase the intensity of cute, while in the second sentence, it is to decrease the intensity. VADER maintains a booster dictionary which contains a set of boosters and dampeners.

The effect of the degree modifier also depends on its distance to the word it's modifying. Farther words have a relatively smaller intensifying effect on the base word. One modifier beside the base word adds or subtracts 0.293 to the sentiment score of the sentence, depending on whether the base word is positive or not. A second modifier from the base word adds/subtract 95% of 0.293, and a third adds/subtracts 90%.

The fourth heuristic is the **shift in polarity due to "but"**. Oftentimes, "but" connects two clauses with contrasting sentiments. The dominant sentiment, however, is the latter one. For example, "I love you, but I don't want to be with you anymore." The first clause "I love you" is positive, but the second one "I don't want to be with you anymore." is negative and obviously more dominant sentiment-wise.

VADER implements a "but" checker. Basically, all sentiment-bearing words before the "but" have their valence reduced to 50% of their values, while those after the "but" increase to 150% of their values.

The fifth heuristic is **examining the tri-gram before a sentiment-laden lexical feature to catch polarity negation**. Here, a tri-gram refers to a set of three lexical features. VADER maintains a list of negator words. Negation is captured by multiplying the sentiment score of the sentiment-laden lexical feature by an empirically-determined value -0.74.

Installing VADER Lexicon:

The VADER sentiment analyzer is an open source lexicon available with the **nltk** library in Python. To install, use pip command line installer to write:

```
!pip install vadersentiment
```

Machine Learning Approach:

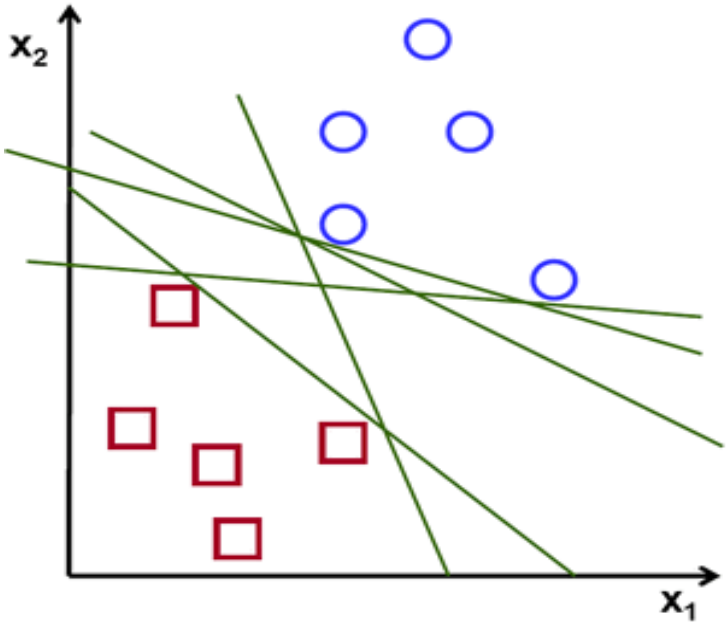
Here we use four types of models: SVM-Linear Classifier, SVM-SGD Classifier, Random Forest Classifier and Naïve Bayes-Multinomial Classifier.

SVM-Linear Classifier:

A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, given labelled training data (*supervised learning*), the algorithm outputs an optimal hyperplane which categorizes new examples.

In which sense is the hyperplane obtained optimal? Let's consider the following simple problem:

For a linearly separable set of 2D-points which belong to one of two classes, find a separating straight line.



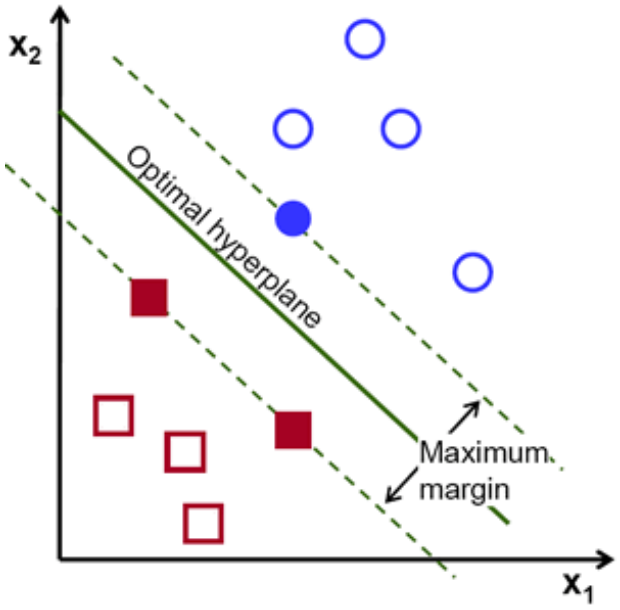
Note: In this example we deal with lines and points in the Cartesian plane instead of hyperplanes and vectors in a high dimensional space. This is a simplification of the problem. It is important to understand that this is done only because our intuition is better built from examples that are easy to imagine. However, the same concepts apply to tasks where the examples to classify lie in a space whose dimension is higher than two.

In the above picture you can see that there exists multiple lines that offer a solution to the problem. Is any of them better than the others? We can intuitively define a criterion to estimate the worth of the lines:

A line is bad if it passes too close to the points because it will be noise sensitive and it will not generalize correctly. Therefore, our goal should be to find the line passing as far as possible from all points.

Then, the operation of the SVM algorithm is based on finding the hyperplane that gives the largest minimum distance to the training examples. Twice, this distance receives the important name

of **margin** within SVM's theory. Therefore, the optimal separating hyperplane *maximizes* the margin of the training data.



Calculating Optimal Hyperplane:

Let's introduce the notation used to define formally a hyperplane:

$$f(x) = \beta_0 + \beta^T x,$$

where β is known as the *weight vector* and β_0 as the *bias*.

The optimal hyperplane can be represented in an infinite number of different ways by scaling of β and β_0. As a matter of convention, among all the possible representations of the hyperplane, the one chosen is

$$|\beta_0 + \beta^T x| = 1$$

where x symbolizes the training examples closest to the hyperplane. In general, the training examples that are closest to the hyperplane are called **support vectors**. This representation is known as the **canonical hyperplane**.

Now, we use the result of geometry that gives the distance between a point x and a hyperplane (β, β_0):

$$\text{distance} = \frac{|\beta_0 + \beta^T x|}{\|\beta\|}.$$

In particular, for the canonical hyperplane, the numerator is equal to one and the distance to the support vectors is

$$\text{distance}_{\text{ support vectors}} = \frac{|\beta_0 + \beta^T x|}{\|\beta\|} = \frac{1}{\|\beta\|}.$$

Recall that the margin introduced in the previous section, here denoted as M, is twice the distance to the closest examples:

$$M = \frac{2}{\|\beta\|}$$

Finally, the problem of maximizing M is equivalent to the problem of minimizing a function $L(\beta)$ subject to some constraints. The constraints model the requirement for the hyperplane to classify correctly all the training examples x_i. Formally,

$$\min_{\beta,\beta_0} L(\beta) = \frac{1}{2}\|\beta\|^2 \text{ subject to } y_i(\beta^T x_i + \beta_0) \geq 1 \ \forall i,$$

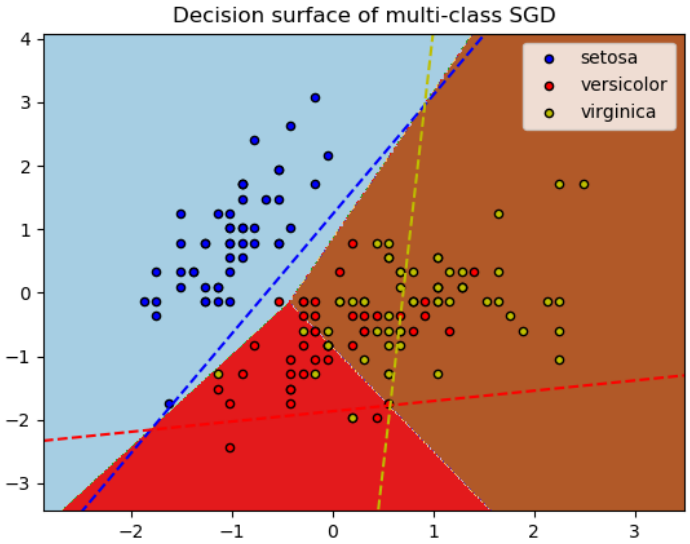
where y_i represents each of the labels of the training examples.

This is a problem of Lagrangian optimization that can be solved using Lagrange multipliers to obtain the weight vector β and the bias β_0 of the optimal hyperplane.

Stochastic Gradient Descent Classifier Model:

Stochastic gradient descent in continuous time (SGDCT) provides a computationally efficient method for the statistical learning of continuous-time models, which are widely used in science, engineering, and finance. The SGDCT algorithm follows a (noisy) descent direction along a continuous stream of data. SGDCT performs an online parameter update in continuous time, with the parameter updates θt satisfying a stochastic differential equation. We prove that limit→∞∇g¯(θt)=0 where g¯ is a natural objective function for the estimation of the continuous-time dynamics. The convergence proof leverages ergodicity by using an appropriate Poisson equation to help describe the evolution of the parameters for large times. SGDCT can also be used to solve continuous-time optimization problems, such as American options. For certain continuous-time problems, SGDCT has some promising advantages compared to a traditional stochastic gradient descent algorithm. As an example application, SGDCT is combined with a deep neural network to price high-dimensional American options (up to 100 dimensions).

SGD Classifier supports multi-class classification by combining multiple binary classifiers in a “one versus all” (OVA) scheme. For each of the K classes, a binary classifier is learned that discriminates between that and all other K−1classes. At testing time, we compute the confidence score (i.e. the signed distances to the hyperplane) for each classifier and choose the class with the highest confidence. The Figure below illustrates the OVA approach on the iris dataset. The dashed lines represent the three OVA classifiers; the background colors show the decision surface induced by the three classifiers.



SGD Classifier supports averaged SGD (ASGD). Averaging can be enabled by setting `average=True`. ASGD works by averaging the coefficients of the plain SGD over each iteration over a sample. When using ASGD the learning rate can be larger and even constant leading on some datasets to a speed up in training time.

Mathematical Calculation:

Gradient descent in one dimension:

To get going, consider a simple scenario in which we have one parameter to manipulate. Let's also assume that our objective associates every value of this parameter with a value. Formally, we can say that this objective function has the signature f:R→R. It maps from one real number to another.

Note that the domain of *f* is in one-dimensional. According to its Taylor series expansion as shown below, we have

$f(x+\epsilon) \approx f(x) + f'(x)\epsilon.$

Substituting $\epsilon\epsilon$ with $-\eta f'(x)$ where η is a constant, we have

$f(x-\eta f'(x)) \approx f(x) - \eta f'(x)^2$

If ηη is set as a small positive value, we obtain

$f(x-\eta f'(x)) \leq f(x).$

In other words, updating x as

$x := x - \eta f'(x)$

may reduce the value of f(x) if its current derivative value $f'(x) \neq 0$. Since the derivative f'(x)is a special case of gradient in one-dimensional domain, the above update of xx is gradient descent in one-dimensional domain.

The positive scalar η is called the learning rate or step size. Note that a larger learning rate increases the chance of overshooting the global minimum and oscillating. However, if the learning rate is too small, the convergence can be very slow. In practice, a proper learning rate is usually selected with experiments.

Gradient descent over multi-dimensional parameters:

Consider the objective function $f:R^d \rightarrow R$that takes any multi-dimensional vector $x=[x1,x2,\ldots,x_d]^\top$as its input. The gradient of f(x) with respect to x is defined by the vector of partial derivatives:

$\nabla_x f(x) = [\partial f(x)/\partial x_1, \partial f(x)/\partial x_2, \ldots, \partial f(x)/\partial x_d]^\top$

To keep our notation compact we may use the notation ∇f(x) and ∇xf(x) interchangeably when there is no ambiguity about which parameters we are optimizing over. In plain English, each element $\partial f(x)/\partial x_i$ of the gradient indicates the rate of change for *f* at the point x with respect to the input x_i only. To measure the rate of change of *f*in any direction that is represented by a unit vector u, in multivariate calculus, we define the directional derivative of f at x in the direction of u as

$D_u f(x) = \lim_{h \rightarrow 0} (f(x+hu) - f(x))/h$

which can be rewritten according to the chain rule as

$D_u f(x) = \nabla f(x) \cdot u$

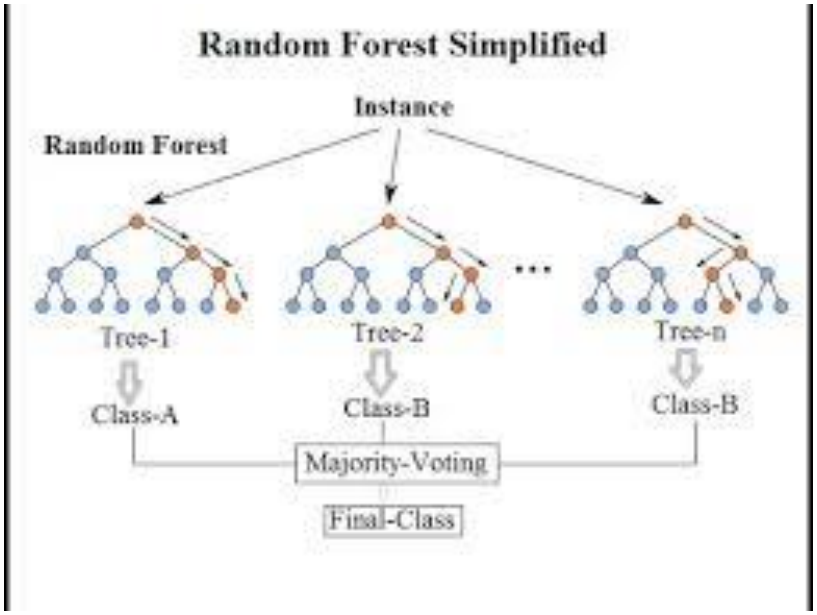
Since $D_u f(x)$ gives the rates of change of f at the point x in all possible directions, to minimize f, we are interested in finding the direction where f can be reduced fastest. Thus, we can minimize the directional derivative $D_u f(x)$ with respect to u. Since $D_u f(x) = \|\nabla f(x)\| \cdot \|u\| \cdot \cos(\theta) = \|\nabla f(x)\| \cdot \cos(\theta)$, where θ is the angle between $\nabla f(x)$and u, the minimum value of cos(θ) is -1 when $\theta = \pi$
Therefore, $D_u f(x)$ is minimized when u is at the opposite direction of the gradient $\nabla f(x)$. Now we can iteratively reduce the value of f with the following gradient descent update:

$x := x - \eta \nabla f(x)$,

where the positive scalar η is called the learning rate or step size.

Random Forest Classifier:

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of over fitting to their training set.



Decision trees are a popular method for various machine learning tasks. Tree learning "come[s] closest to meeting the requirements for serving as an off-the-shelf procedure for data mining", say Hastie *et al.*, "because it is invariant under scaling and various other transformations of feature values, is robust to inclusion of irrelevant features, and produces inspectable models. However, they are seldom accurate".

In particular, trees that are grown very deep tend to learn highly irregular patterns: they over fit their training sets, i.e. have low biases, but very high variance. Random forests are a way of averaging multiple deep decision trees, trained on different parts of the same training set, with the goal of reducing the variance. This comes at the expense of a small increase in the bias and some loss of interpretability, but generally greatly boosts the performance in the final model.

Bagging:

The training algorithm for random forests applies the general technique of bootstrap aggregating, or bagging, to tree learners. Given a training set $X = x_1,..., x_n$ with responses $Y = y_1,..., y_n$, bagging repeatedly (B times) selects a random sample with replacement of the training set and fits trees to these samples:

For $b = 1,..., B$:

- Sample, with replacement, n training examples from X, Y; call these X_b, Y_b.
- Train a classification or regression tree f_b on X_b, Y_b.
- After training, predictions for unseen samples x' can be made by averaging the predictions from all the individual regression trees on x':

  $$f=1/B(^{b} \sum_{b=1} f_b(x'))$$

  or by taking the majority vote in the case of classification trees.

This bootstrapping procedure leads to better model performance because it decreases the variance of the model, without increasing the bias. This means that while the predictions of a single tree are highly sensitive to noise in its training set, the average of many trees is not, as long as the trees are not correlated. Simply training many trees on a single training set would give strongly correlated trees (or even the same tree many times, if the training algorithm is deterministic); bootstrap sampling is a way of de-correlating the trees by showing them different training sets.

Additionally, an estimate of the uncertainty of the prediction can be made as the standard deviation of the predictions from all the individual regression trees on x':

$$\sigma = (^{B}\sum_{b=1}(f_b(x')-f)^2)^{1/2}$$

The number of samples/trees, B, is a free parameter. Typically, a few hundred to several thousand trees are used, depending on the size and nature of the training set. An optimal number of trees B can be found using cross-validation, or by observing the *out-of-bag error*: the mean prediction error on each training sample x_i, using only the trees that did not have x_i in their bootstrap sample. The training and test error tend to level off after some number of trees have been fit.

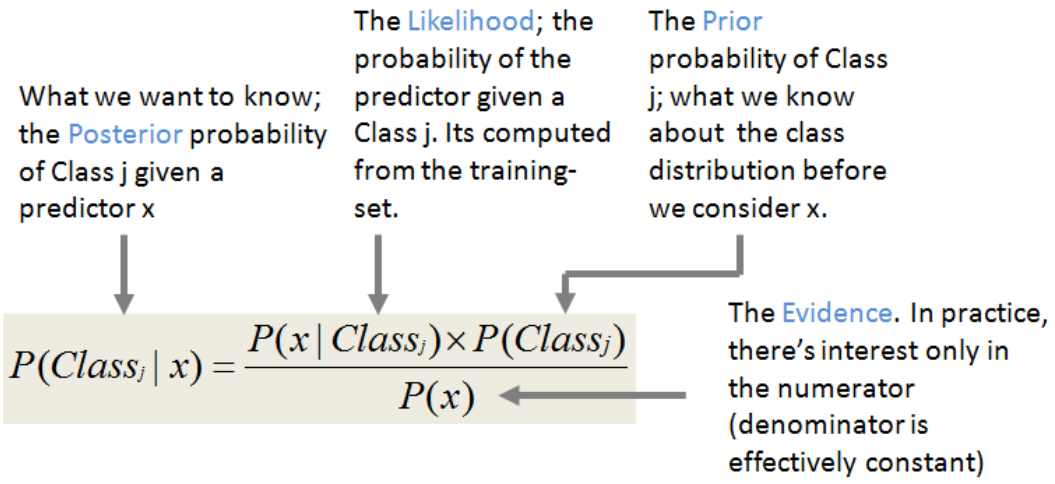
Naive-Bayes Multinomial Classifier:

Naive Bayes is a simple technique for constructing classifiers: models that assign class labels to problem instances, represented as vectors of feature values, where the class labels are drawn from some finite set. There is not a single algorithm for training such classifiers, but a family of algorithms based on a common principle: all naive Bayes classifiers assume that the value of a particular feature is independent of the value of any other feature, given the class variable. For example, a fruit may be considered to be an apple if it is red, round, and about 10 cm in diameter. A naive Bayes classifier considers each of these features to contribute independently to the probability that this fruit is an apple, regardless of any possible correlations between the color, roundness, and diameter features.

For some types of probability models, naive Bayes classifiers can be trained very efficiently in a supervised learning setting. In many practical applications, parameter estimation for naive Bayes models uses the method of maximum likelihood; in other words, one can work with the naive Bayes model without accepting Bayesian probability or using any Bayesian methods.

Despite their naive design and apparently oversimplified assumptions, naive Bayes classifiers have worked quite well in many complex real-world situations. In 2004, an analysis of the Bayesian classification problem showed that there are sound theoretical reasons for the apparently implausible efficacy of naive Bayes classifiers. Still, a comprehensive comparison with other classification algorithms in 2006 showed that Bayes classification is outperformed by other approaches, such as boosted trees or random forests.

An advantage of naive Bayes is that it only requires a small number of training data to estimate the parameters necessary for classification.



Applying the independence assumption

$$P(x \mid Class_j) = P(x_1 \mid Class_j) \times P(x_2 \mid Class_j) \times \ldots \times P(x_k \mid Class_j)$$

Substituting the independence assumption, we derive the Posterior probability of Class j given a new instance x' as...

$$P(Class_j \mid x') = P(x'_1 \mid Class_j) \times P(x'_2 \mid Class_j) \times \ldots \times P(x'_k \mid Class_j) \times P(Class_j)$$

A practical example:

In the example, we are given a sentence "A very close game", a training set of five sentences (as shown below), and their corresponding category (Sports or Not Sports). The goal is to build a Naive Bayes classifier that will tell us which category the sentence "A very close game" belongs to.

The author Bruno suggested that we could try applying a Naive Bayes classifier, thus the strategy would be calculating the probability of both "A very close game **is Sports**", as well as it's ***Not Sports.*** The one with the higher probability will be the result.

Expressed formally, this is what we would like to calculate **P(*Sports* | *A very close game*)**, i.e. the probability that the category of the sentence is *Sports* given that the sentence is "A very close game".

Bruno included a step-by-step guide to building a Native Bayes classifier to achieve this goal, calculating P(*Sports* | *A very close game*).

Text	Category
“A great game”	Sports
“The election was over”	Not sports
“Very clean match”	Sports
“A clean but forgettable game”	Sports
“It was a close election”	Not sports

Step 1: Feature Engineering

In the first step, feature engineering, we focus on extracting features of text. We need numerical features as input for our classifier. So an intuitive choice would be word frequencies, i.e., counting the occurrence of every word in the document.

Then, we need to convert the probability that we wish to calculate into a form that can be calculated using word frequencies. Here, we adopt the properties of possibilities and Bayes’ Theorem to do the conversion.

Bayes’ Theorem is useful for dealing with conditional probabilities, since it provides a way for us to reverse them.

$$P(A|B) = \frac{P(B|A) \times P(A)}{P(B)}$$

In our case, the probability that we wish to calculate can be calculated as:

$$P(sports|a\ very\ close\ game) = \frac{P(a\ very\ close\ game|sports) \times P(sports)}{P(a\ very\ close\ game)}$$

Because we are only trying to find out which category (Sports or Not Sports) has a higher probability, it makes sense to discard the divisor P(a very close game), and compare only:

$$P(a\,very\,close\,game|Sports) \times P(Sports)$$

with

$$P(a\,very\,close\,game|Not\,Sports) \times P(Not\,Sports)$$

But we have a problem: In order to obtain P(*a very close game* | *Sports*) , we have to count the occurrence of "a very close game" in the Sports category. But it does not even appear in the training set at all, thus the probability is zero, and consequently making P(*a very close game* | *Sports*)zero as well. What shall we do to tackle this problem?

Step 2: Being naive

In the non-naive Bayes way, we look at sentences in entirety, thus once the sentence does not show up in the training set, we will get a zero probability, making it difficult for further calculations. Whereas for Naive Bayes, there is an assumption that every word is independent of one another. Now, we look at individual words in a sentence, instead of the entire sentence.

Here, we can rewrite the probability we wish to calculate accordingly:

$$P(a\,very\,close\,game) = P(a) \times P(very) \times P(close) \times P(game)$$

$$P(a\,very\,close\,game|Sports) = P(a|Sports) \times P(very|Sports) \times P(close|Sports) \times P(game|Sports)$$

Similarly,
P(*a very close game* | *Not Sports*) = P(*a* | *Not Sports*) x P(*very* | *Not Sports*) x P(*close* | *Not Sports*) x P(*game* | *Not Sports*)

Step 3: Calculating the probabilities

In the final step, we are good to go: simply calculating the probabilities and compare which has a higher probability: **P*(a very close game | Sports*) or P(*a very close game | Not Sports***).

Here, the word “close” does not exist in the category Sports, thus **P(*close |Sports*)** = 0, leading to **P(*a very close game | Sports*)**=0. It is problematic when a frequency-based probability is zero, because it will wipe out all the information in the other probabilities, and we need to find a solution for this.

A solution would be **Laplace smoothing**, which is a technique for smoothing categorical data. A small-sample correction, or **pseudo-count**, will be incorporated in every probability estimate. Consequently, no probability will be zero. This is a way of regularizing Naive Bayes, and when the pseudo-count is zero, it is called Laplace smoothing. While in the general case it is often called **Lidstone smoothing**.

CODE:

Importing libraries:

```
#Importing Important python module.
import pandas as pd
import numpy as np
import nltk
from nltk.corpus import stopwords
import re
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
import string
from sklearn.model_selection import train_test_split
from random import shuffle
from sklearn import svm
from sklearn.externals import joblib
from sklearn.pipeline import Pipeline
import pickle
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score,roc_curve,auc,f1_score,precision_score,recall_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import SGDClassifier
import matplotlib.pyplot as plt
from nltk.sentiment.vader import SentimentIntensityAnalyzer
```

As, we all know that how python library are useful in data analysis, application of Machine Learning Algorithm and so on. There are also many things we can do with the python library that are we can get classification report, efficiency of different ML algorithms. There are also some library to plot different type of graph that makes our visualization great. Ex:-Mathplotlib.pyplot, seaborn etc. some module like joblib are used to save the trained classifier that save our time to train each time the millions of data.

Loading Dataset and splitting it in Train and Test set:

```
#Loading Dataset and spliting it in Train and Test set.
train=pd.read_csv("train123.csv",header=None,encoding='ISO-8859-1')
test=pd.read_csv("uniondata.csv")
train.columns=["label","1","2","3","4","tweet"]
d=train[["tweet","label"]]
x=train["tweet"].tolist()
y=train["label"].tolist()
X_train, X_test, y_train, y_test = train_test_split(x,y,test_size=0.20, random_state=42)
x_test=test["tweets"].tolist()
print(train.head(10))
test.head(10)
```

By, using pd. read_csv that is a pandas attribute to read a dataset which should with .csv extension. Train123.csv is a 1.6 million dataset with columns "label","1","2","3","4","tweet" but in train dataset there are four columns which are not important that are "1","2","3","4".
So, these columns should drop. Now, x and y will contain the list of tweet and label respectively of train123.csv dataset. Now, with train_test_split x and y will split in 20% for test. Uniondata.csv is union budget dataset to find the sentiment of the tweet. x_test is the list of tweets of uniondata.csv.

Output of X_train.head(10):

```
   label          1                             2         3                4  \
0      0  1467810369  Mon Apr 06 22:19:45 PDT 2009  NO_QUERY  _TheSpecialOne_
1      0  1467810672  Mon Apr 06 22:19:49 PDT 2009  NO_QUERY    scotthamilton
2      0  1467810917  Mon Apr 06 22:19:53 PDT 2009  NO_QUERY         mattycus
3      0  1467811184  Mon Apr 06 22:19:57 PDT 2009  NO_QUERY          ElleCTF
4      0  1467811193  Mon Apr 06 22:19:57 PDT 2009  NO_QUERY           Karoli
5      0  1467811372  Mon Apr 06 22:20:00 PDT 2009  NO_QUERY         joy_wolf
6      0  1467811592  Mon Apr 06 22:20:03 PDT 2009  NO_QUERY          mybirch
7      0  1467811594  Mon Apr 06 22:20:03 PDT 2009  NO_QUERY             coZZ
8      0  1467811795  Mon Apr 06 22:20:05 PDT 2009  NO_QUERY  2Hood4Hollywood
9      0  1467812025  Mon Apr 06 22:20:09 PDT 2009  NO_QUERY          mimismo

                                               tweet
0  @switchfoot http://twitpic.com/2y1zl - Awww, t...
1  is upset that he can't update his Facebook by ...
2  @Kenichan I dived many times for the ball. Man...
3    my whole body feels itchy and like its on fire
4  @nationwideclass no, it's not behaving at all....
5                      @Kwesidei not the whole crew
6                                      Need a hug
7  @LOLTrish hey  long time no see! Yes.. Rains a...
8              @Tatiana_K nope they didn't have it
9                          @twittera que me muera ?
```

Output of x_test.head(10):

	tweets
0	Talk on Union Budget 2019!! pic.twitter.com/8D...
1	Here is a quick highlight on Union Budget 2019...
2	Union Budget 2019: \r\n\r\nAuto Industry Disap...
3	Mr. Dhiraj Jain, Director, Mahagun India, shar...
4	A 19 Years Old Teenager @tarak9999 breaks All ...
5	5 trillion for who ?? Ambani ?? Union Budget 2...
6	Union Budget 2019: Railways focus to be on saf...
7	.@nsitharaman to present Union Budget of Modi ...
8	The process of legislative reforms on Labour i...
9	#Moneypitara: Highlights of Budget 2019-20: #B...

Lexicon Based Approach

```
#importing test dataset
data = pd.read_csv("uniondata.csv")
#instanciating the SentimentIntensityAnalyzer object from vader sentiment analyzer
sid = SentimentIntensityAnalyzer()
listy = []
pos,neg=0,0
#calculating the polarity score of all the tweets
for index, row in data.iterrows():
    ss = sid.polarity_scores(row["tweets"])
    if ss['compound'] >= 0.0 :
        pos+=1
        score=1
    else:
        neg+=1
        score=0
    g=[ss,score]
    listy.append(g)
sentiment_scores={"Positive":pos,"Negetive":neg}
se = pd.DataFrame(listy,columns=["polarity","lexicon label"])
data=pd.concat([data, se], axis=1)
print("Positive: ",sentiment_scores["Positive"])
print("Negetive: ",sentiment_scores["Negetive"])
```

Output:

Positive: 608
Negative: 101

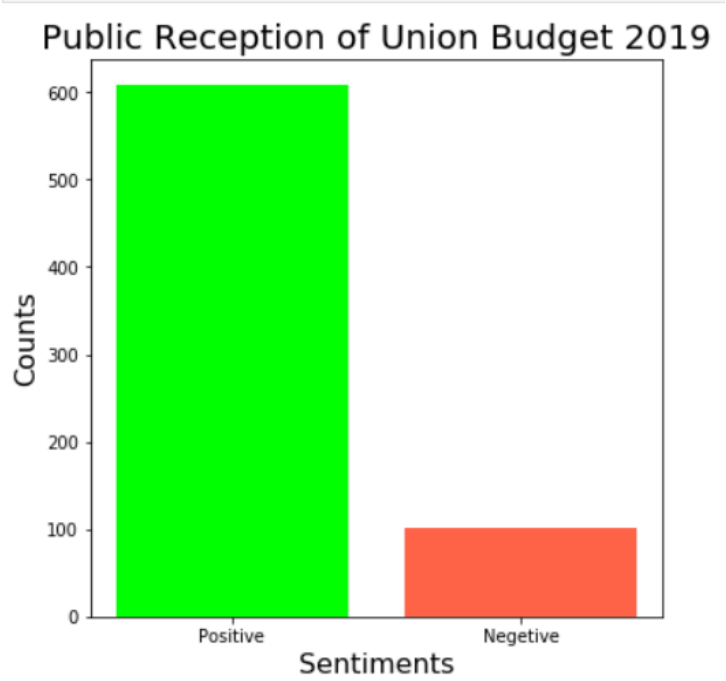
Lexicon Based Approach is an easier compared to the machine learning approach as there is no machine learning occurring here. VADER library will parse any string input by searching for polarity scores in each word in its lexicon and return a full score. In order to place this result in a dataframe, a

2-D list is created to store the sentiment scores and the labels of the tweets. A dictionary is created such that it can store the count of positively and negatively predicted tweets.

Count plot for positive and negative sentiments:

```
#countplot for positive and negetive sentiments
fig = plt.figure(figsize=(6,6))
plt.bar(range(len(sentiment_scores)),list(sentiment_scores.values()),tick_label=list(sentiment_scores.keys()),color=['lime', 'tomato', 'cyan'])
plt.xlabel('Sentiments',fontsize=16)
plt.ylabel('Counts',fontsize=16)
plt.title('Public Reception of Union Budget 2019', fontsize=20)
plt.show()
plt.savefig("C:\\Users\kools\\Desktop\\Assignment\\graph9.png",bbox_inches="tight",pad_inches=2)
```

Output:



Using the dictionary that stores the count of positively and negatively predicted tweets, a count plot is created to visualize the counts of the respective labels.

Machine Learning Approach:

Pre-processing and Cleaning of Tweets function. Like removing URL, @Username, Hash tag, Repeating etc.

```
#Pre-processing and Cleaning of Tweets. Like removing URL,@Username,Hagtag,Repating etc..
def preprocessTweets(tweet):
    #Convert www.* or https?://* to URL
    tweet = re.sub('((www\.[^\s]+)|(https?://[^\s]+))','URL',tweet)
    #Convert @username to __HANDLE
    tweet = re.sub('@[^\s]+','__HANDLE',tweet)
    #Replace #word with word
    tweet = re.sub(r'#([^\s]+)', r'\1', tweet)
    #trim
    tweet = tweet.strip('\'"')
    # Repeating words like akashhhhhhhhhhh
    rpt_regex = re.compile(r"(.)\1{1,}", re.IGNORECASE)
    tweet = rpt_regex.sub(r"\1\1", tweet)
    #Emoticons
    emoticons = \
    [
     ('__positive__',[ ':-)', ':)', '(:', '(-:', \
                       ':-D', ':D', 'X-D', 'XD', 'xD', \
                       '<3', ':\*', ';-)', ';)', ';-D', ';D', '(;', '(-;', ] ),\
     ('__negative__', [':-(', ':(', '(:', '(-:', ':,(',\
                       ':\'(', ':"(', ':((', ] ),\
    ]
    def replace_parenth(arr):
       return [text.replace(')', '[)}\]]').replace('(', '[({\[') for text in arr]

    def regex_join(arr):
        return '(' + '|'.join( arr ) + ')'

    emoticons_regex = [ (repl, re.compile(regex_join(replace_parenth(regx))) ) \
            for (repl, regx) in emoticons ]
```

Stemming of tweets

```
#Stemming of Tweets
def stem(tweet):
        stemmer = nltk.stem.PorterStemmer()
        tweet_stem = ''
        words = [word if(word[0:2]=='__') else word.lower() \
                    for word in tweet.split() \
                    if len(word) >= 3]
        words = [stemmer.stem(w) for w in words]
        tweet_stem = ' '.join(words)
        return tweet_stem
```

Pre-processing is considered to be the first step in text classification, and choosing the right pre-processing techniques can improve classification effectiveness. Pre-processing in this context is the procedure of cleansing and preparation of texts that are going to be classified. It is a fact that unstructured texts on the Internet —and in our case on Twitter— contain significant amounts of noise. By the term noise, we mean data that do not contain any useful information for the analysis at hand, i.e. sentiment analysis in our case. With tweets there are many noise that are numbers, punctuations, repetition of character in word, USER_NAME etc. these are irrelevant for sentiment analysis. In the function processTweets we get tweet the which first replace “web url” with URL and “@user_name” with _Handel by using re.sub() pre-defined function in module “re” i.e. is regular expression. After that return string of function re.sub() pass again in re.sub() to remove “#”. After that stem function is used to remove the whitespace from both side of tweet. Then repeating character of word and after that emotion character are removed. Then the tweet pass in stem function to convert the plural word in singular and soon by nltk.stem.porterstemmer().

Calling Pre-processing Function:

```
#Processing tweets.
def processTweets(X):
        X = [stem(preprocessTweets(tweet)) for tweet in X]
        return X
```

Passing each tweet form list of tweets and again store the pre-processed tweet in X.

Cleaning Data:

```
#Getting cleaned X_train and X_test Data.
X_train = processTweets(X_train)
```

```
#Getting cleaned X_test and x_test Data.
X_test = processTweets(X_test)
x_test = processTweets(x_test)
```

Preparing X_train, X_test, x_test for model testing by refining each and every tweet of dataset.

Vectorizations of data to train models:

```
#Vectorization of data to train it.
vec = TfidfVectorizer(min_df=5, max_df=0.95, sublinear_tf = True,use_idf = True,ngram_range=(1, 2))
```

TfidfVectorizer is a sklearn module for feature extraction of words. There are also other ways to extract but this is easy to implement and more efficient. TF-IDF stands for term frequency-inverse document frequency, and the tf-idf weight is a weight often used in information retrieval and text mining. This weight is a statistical measure used to evaluate how important a word is to a document in a collection or corpus. The importance increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus. Variations of the tf-idf weighting scheme are often used by search engines as a central tool in scoring and ranking a document's relevance given a user query. One of the simplest ranking functions is computed by summing the tf-idf for each query term; many more sophisticated ranking functions are variants of this simple model. Typically, the tf-idf weight is composed by two terms: the first computes the normalized Term Frequency (TF), aka. the number of times a word appears in a document, divided by the total number of words in that document; the second term is the Inverse Document Frequency (IDF), computed as the logarithm of the number of the documents in the corpus divided by the number of documents where the specific term appears.TF: Term Frequency, which measures how frequently a term occurs in a document. Since every document is different in length, it is possible that a term would appear much more times in long documents than shorter ones. Thus, the term frequency is often divided by the document length (aka. the total number of terms in the document) as a way of normalization. IDF: Inverse Document Frequency, which measures how important a term is. While computing TF, all terms are considered equally important. However it is known that certain terms, such as "is", "of", and "that", may appear a lot of times but have little importance. This also have an attribute stopword that will remove the irrelevant words from each tweets which are not important in sentiment analysis.

Here, we pass X_train and y_train to train our different classifier here we use pipeline to make it faster the first part of the pipeline if vectorize the tweet and the other is call the classifier function and after that fit the X_train and y_train in the classifier. And by module joblib we saved the trained classifier so that we doesn't need to train our classifier again and again with that huge amount of data.

Support Vector Machine Classifier Model

```
#Support Vector Machine classifier Model.
def classifier(X_train,y_train):
        svm_clf =svm.LinearSVC()
        vec_clf = Pipeline([('vectorizer', vec), ('pac', svm_clf)])
        vec_clf.fit(X_train,y_train)
        joblib.dump(vec_clf, 'svmClassifier.pkl', compress=3)
        return vec_clf
vec_clf1 = classifier(X_train,y_train)
```

Stochastic Gradient Descent Classifier Model

```
#Stochastic Gradient Descent classifier Model.
def classifiersgd(X_train,y_train):
        svm_clf =SGDClassifier()
        vec_clf = Pipeline([('vectorizer', vec), ('pac', svm_clf)])
        vec_clf.fit(X_train,y_train)
        joblib.dump(vec_clf, 'sgdClassifier.pkl', compress=3)
        return vec_clf
vec_clf2 = classifiersgd(X_train,y_train)
```

Random Forest Classifier Model

```
#Random Forest Classsifier Model
def classifierRFC(X_train,y_train):
        svm_clf =RandomForestClassifier(n_estimators=100,max_depth=5)
        vec_clf = Pipeline([('vectorizer', vec), ('pac', svm_clf)])
        vec_clf.fit(X_train,y_train)
        joblib.dump(vec_clf, 'RFCClassifier.pkl', compress=3)
        return vec_clf
vec_clf3 = classifierRFC(X_train,y_train)
```

Naive-Bayes Multinomial Model

```
#Naive-Bayes Multinomial Model
def classifierNB(X_train,y_train):
        svm_clf =MultinomialNB()
        vec_clf = Pipeline([('vectorizer', vec), ('pac', svm_clf)])
        vec_clf.fit(X_train,y_train)
        joblib.dump(vec_clf, 'MNBClassifier.pkl', compress=3)
        return vec_clf
vec_clf4 = classifierNB(X_train,y_train)
```

Trained Classifier for SVC Classifier and Their Produced Result

Loading SVC Classifier Model

```
#Loading SVC Classifier Model
classifier_f = open("svmClassifier.pkl", "rb")
classifierSVC = joblib.load(classifier_f)
classifier_f.close()
```

By using joblib.load() function we can call our trained classifier, which we trained before.Here svmclassifier has called.

Predict the X_test on SVM Classifier

```
#Predict the X_test on svmclassifier
y_predSVC = classifierSVC.predict(X_test)
Y_predSVC = classifierSVC.predict(x_test)
posSVC=[w for w in Y_predSVC if w==4]
negSVC=[w for w in Y_predSVC if w==0]
```

Now, by passing X_test dataset in classifierSVC.predict() function, the predicted value of the tweets obtained as a list of “0” for negative and “4” for positive in y_predSVC. Same with x_test dataset. And posSVC is the list of all positive predicted tweets and negSVC is the list of all negative predicted tweets.

Accuracy score, AUC and Classification report

```
#ACCURACY SCORE,AUC,AND CLASSIFICATION REPORT
print("The Accuracy Score is :",accuracy_score(y_test,y_predSVC))
fpr,tpr,threshold = roc_curve(y_test,y_predSVC,pos_label=4)
print("The False Positive Rate is {} and The True Positive Rate is {}".format(fpr,tpr))
print("The Accuracy of False Positive Rate and True Positive Rate is ",auc(fpr,tpr))
print("The classification Report is : ",end="\n")
print(classification_report(y_test,y_predSVC))
```

Here, The accuracy score means how much the classifier predict the correct. False positive rate is classifier predict positive but it is false. True positive rate means classifier predicted true and the originally it is true. After the classification report is obtained there are precision_score, recall_score, f1_score of the classifier which measure the performance of the classifier.

Classification report is of SVM Classifier.

```
The Accuracy Score is : 0.804896875
The False Positive Rate is [0.         0.19944951 1.        ] and The True Positive Rate is [0.         0.80921585 1.        ]
The Accuracy of False Positive Rate and True Positive Rate is  0.8048831728932834
The classification Report is :
              precision    recall  f1-score   support

           0       0.81      0.80      0.80    159494
           4       0.80      0.81      0.81    160506

    micro avg       0.80      0.80      0.80    320000
    macro avg       0.80      0.80      0.80    320000
 weighted avg       0.80      0.80      0.80    320000
```

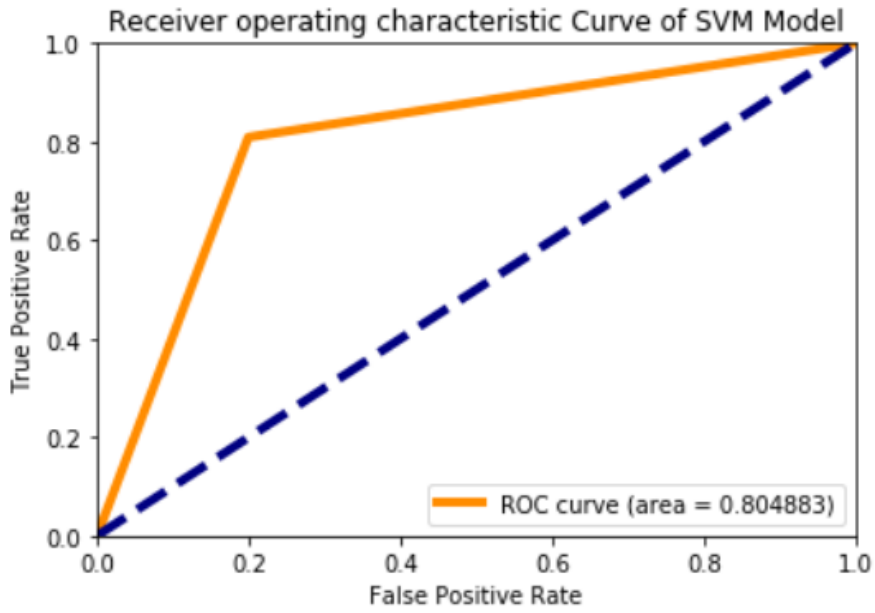
```
f1score.append(f1_score(y_test,y_predSVC,pos_label=4))
precission.append(precision_score(y_test,y_predSVC,pos_label=4))
recall.append(recall_score(y_test,y_predSVC,pos_label=4))
```

Here, the f1_score, precision_score, recall_score is append in the predefine list f1score,precission and recall respectively.

Plotting ROC-Curve of SVC Classifier:

```
#Ploting ROC-Curve.
plt.figure()
lw = 4
plt.plot(fpr, tpr, color='darkorange',lw=lw, label='ROC curve (area = %f)' % auc(fpr,tpr))
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic Curve of SVM Model')
plt.legend(loc="lower right")
plt.show()
plt.savefig("C:\\Users\kools\\Desktop\\Assignment\\sachin\\graph1.png",bbox_inches="tight",pad_inches=2)
```

Output:



Trained Classifier for SGD classifier and Their Produced Result:

Loading SGD Classifier

```
#Loading SGD Classifier.
classifier_f = open("sgdClassifier.pkl", "rb")
classifierSGD = joblib.load(classifier_f)
classifier_f.close()
```

By using joblib.load() function we can call our trained classifier, which we trained before. Here sgdclassifier has called.

Predict The X_test on SGD model

```
#Predict The X_test on SGD model.
y_predSGD = classifierSGD.predict(X_test)
Y_predSGD = classifierSGD.predict(x_test)
posSGD=[w for w in Y_predSGD if w==4]
negSGD=[w for w in Y_predSGD if w==0]
```

Now, by passing X_test dataset in classifierSGD.predict() function, the predicted value of the tweets obtained as a list of “0” for negative and “4” for positive in y_predSGD. Same with x_test dataset. And posSGD is the list of all positive predicted tweets and negSGD is the list of all negative predicted tweets.

Accuracy score, AUC and Classification report:

```
#ACCURACY SCORE,AUC,AND CLASSIFICATION REPORT
print("The Accuracy Score is :",accuracy_score(y_test,y_predSGD))
fpr,tpr,threshold = roc_curve(y_test,y_predSGD,pos_label=4)
print("The False Positive Rate is {} and The True Positive Rate is {}".format(fpr,tpr))
print("The Accuracy of False Positive Rate and True Positive Rate is ",auc(fpr,tpr))
print("The classification Report is : ",end="\n")
print(classification_report(y_test,y_predSGD))
confusion_matrix(y_test,y_predSGD)
```

Here, The accuracy score means how much the classifier predict the correct. False positive rate is classifier predict positive but it is false. True positive rate means classifier predicted true and the originally it is true. After the classification report is obtained there are precision_score, recall_score, f1_score of the classifier which measure the performance of the classifier.

Output:

```
The Accuracy Score is : 0.781009375
The False Positive Rate is [0.         0.23046008 1.        ] and The True Positive Rate is [0.         0.79240651 1.        ]
The Accuracy of False Positive Rate and True Positive Rate is  0.780973217197603
The classification Report is :
              precision    recall  f1-score   support

           0       0.79      0.77      0.78    159494
           4       0.78      0.79      0.78    160506

   micro avg       0.78      0.78      0.78    320000
   macro avg       0.78      0.78      0.78    320000
weighted avg       0.78      0.78      0.78    320000
```

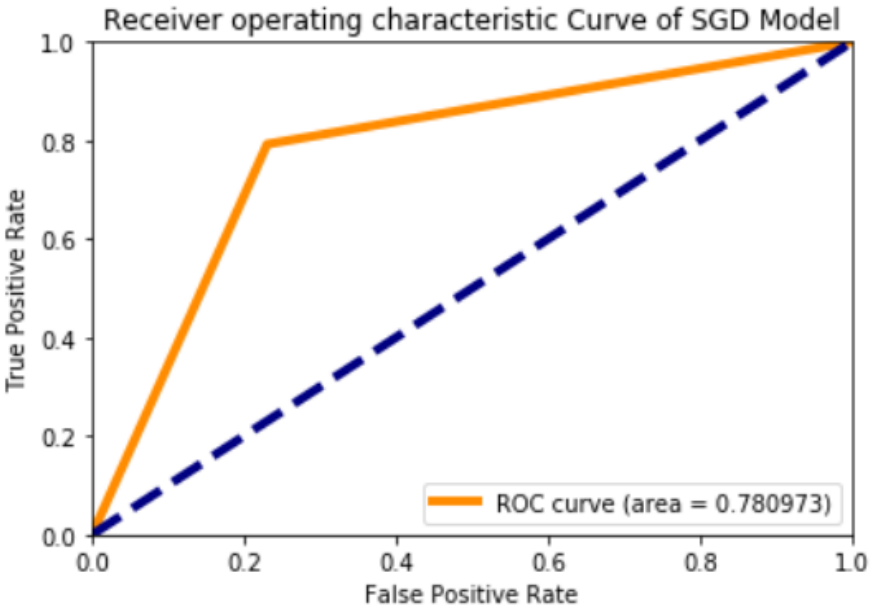
```
f1score.append(f1_score(y_test,y_predSVC,pos_label=4))
precission.append(precision_score(y_test,y_predSGD,pos_label=4))
recall.append(recall_score(y_test,y_predSGD,pos_label=4))
```

Here, the f1_score, precision_score, recall_score is append in the predefine list f1score,precission and recall respectively.

ROC-Curve of SGD Classifier:

```
#ROC-Curve of SGD Classifier.
plt.figure()
lw = 4
plt.plot(fpr, tpr, color='darkorange',lw=lw, label='ROC curve (area = %f)' % auc(fpr,tpr))
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic Curve of SGD Model')
plt.legend(loc="lower right")
plt.show()
plt.savefig("C:\\Users\kools\\Desktop\\Assignment\\graph2.png",bbox_inches="tight",pad_inches=2)
```

Output:



Trained Classifier for RFC Classifier and Their Produced Result

Loading RFC Classifier:

```
#Loading RFC Classifier.
classifier_f = open("RFCClassifier.pkl", "rb")
classifierRFC = joblib.load(classifier_f)
classifier_f.close()
```

By using joblib.load() function we can call our trained classifier, which we trained before .Here RFCclassifier has called.

Predict X_test on RFC Model:

```
#Predict X_test on RFC Model.
y_predRFC = classifierRFC.predict(X_test)
Y_predRFC = classifierRFC.predict(x_test)
posRFC=[w for w in Y_predRFC if w==4]
negRFC=[w for w in Y_predRFC if w==0]
```

Now, by passing X_test dataset in classifierRFC.predict() function, the predicted value of the tweets obtained as a list of “0” for negative and “4” for positive in y_predRFC. Same with x_test dataset. And posRFC is the list of all positive predicted tweets and negRFC is the list of all negative predicted tweets.

Accuracy score, AUC and Classification report

```
#ACCURACY SCORE,AUC,AND CLASSIFICATION REPORT
print("The Accuracy Score is :",accuracy_score(y_test,y_predRFC))
fpr,tpr,threshold = roc_curve(y_test,y_predRFC,pos_label=4)
print("The False Positive Rate is {} and The True Positive Rate is {}".format(fpr,tpr))
print("The Accuracy of False Positive Rate and True Positive Rate is ",auc(fpr,tpr))
print("The classification Report is : ",end="\n")
print(classification_report(y_test,y_predRFC))
```

Here, The accuracy score means how much the classifier predict the correct. False positive rate is classifier predict positive but it is false. True positive rate means classifier predict true and the originally it is true. After the classification report is obtained there are precision_score, recall_score, f1_score of the classifier which measure the performance of the classifier.

Output:

```
The Accuracy Score is : 0.692759375
The False Positive Rate is [0.         0.26506953 1.        ] and The True Positive Rate is [0.         0.65085417 1.        ]
The Accuracy of False Positive Rate and True Positive Rate is  0.6928923206397737
The classification Report is :
              precision    recall  f1-score   support

           0       0.68      0.73      0.70    159494
           4       0.71      0.65      0.68    160506

   micro avg       0.69      0.69      0.69    320000
   macro avg       0.69      0.69      0.69    320000
weighted avg       0.69      0.69      0.69    320000
```

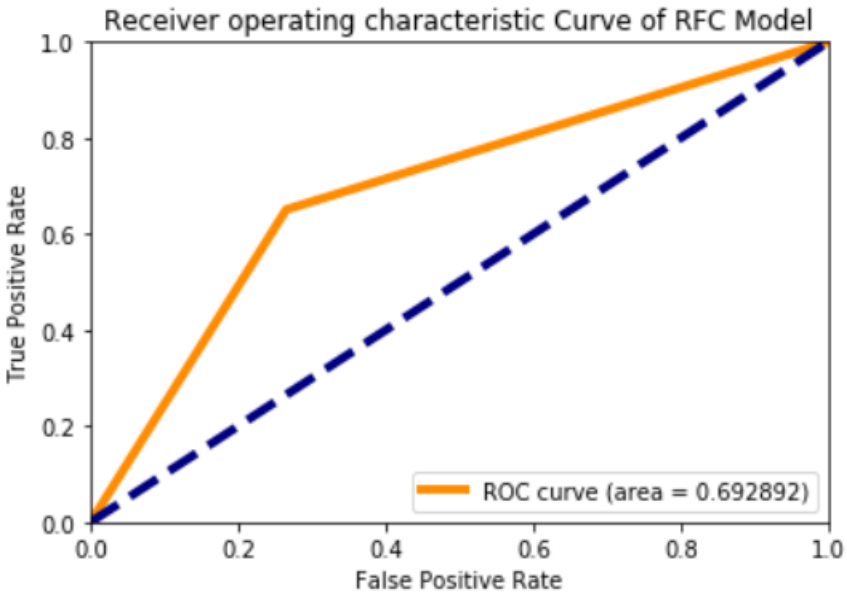
```
f1score.append(f1_score(y_test,y_predRFC,pos_label=4))
precission.append(precision_score(y_test,y_predRFC,pos_label=4))
recall.append(recall_score(y_test,y_predRFC,pos_label=4))
```

Here, the f1_score, precision_score, recall_score is append in the predefine list f1score, precission and recall respectively.

ROC Curve of RFC Classifier

```
#ROC_Curve of RFC Classifer.
plt.figure()
lw = 4
plt.plot(fpr, tpr, color='darkorange',lw=lw, label='ROC curve (area = %f)' % auc(fpr,tpr))
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic Curve of RFC Model')
plt.legend(loc="lower right")
plt.show()
plt.savefig("C:\\Users\kools\\Desktop\\Assignment\\graph3.png",bbox_inches="tight",pad_inches=2)
```

Output



Trained Classifier for Naive-Byes Multinomial Classifier and Their Produced Result

Loading Naïve-Bayes Multinomial Classifier

```
#Loading Naive-Bayes Multinomial Calssifier.
classifier_f = open("MNBClassifier.pkl", "rb")
classifierMNB = joblib.load(classifier_f)
classifier_f.close()
```

By using joblib.load() function we can call our trained classifier, which we trained before. Here MNBclassifier has called.

Predict X_test on Naive-Byes Model

```
#Pridict X_test on Naive-Byes Model.
y_predMNB = classifierMNB.predict(X_test)
Y_predMNB = classifierMNB.predict(x_test)
posMNB=[w for w in Y_predMNB if w==4]
negMNB=[w for w in Y_predMNB if w==0]
```

Now, by passing X_test dataset in classifierMNB.predict() function, the predicted value of the tweets obtained as a list of “0” for negative and “4” for positive in y_predMNB. Same with x_test dataset. And posMNB is the list of all positive predicted tweets and negMNB is the list of all negative predicted tweets.

Accuracy score, AUC and Classification report

```
#ACCURACY SCORE,AUC,AND CLASSIFICATION REPORT
print("The Accuracy Score is :",accuracy_score(y_test,y_predMNB))
fpr,tpr,threshold = roc_curve(y_test,y_predMNB,pos_label=4)
print("The False Positive Rate is {} and The True Positive Rate is {}".format(fpr,tpr))
print("The Accuracy of False Positive Rate and True Positive Rate is ",auc(fpr,tpr))
print("The classification Report is : ",end="\n")
print(classification_report(y_test,y_predMNB))
```

Here, The accuracy score means how much the classifier predict the correct. False positive rate is classifier predict positive but it is false. True positive rate means classifier predict true and the originally it is true. After the classification report is obtained there are precision_score, recall_score, f1_score of the classifier which measure the performance of the classifier.

Output

```
The Accuracy Score is : 0.799496875
The False Positive Rate is [0.         0.19571269 1.        ] and The True Positive Rate is [0.         0.79473665 1.
The Accuracy of False Positive Rate and True Positive Rate is  0.7995119769862711
The classification Report is :
              precision    recall  f1-score   support

           0       0.80      0.80      0.80    159494
           4       0.80      0.79      0.80    160506

   micro avg       0.80      0.80      0.80    320000
   macro avg       0.80      0.80      0.80    320000
weighted avg       0.80      0.80      0.80    320000
```

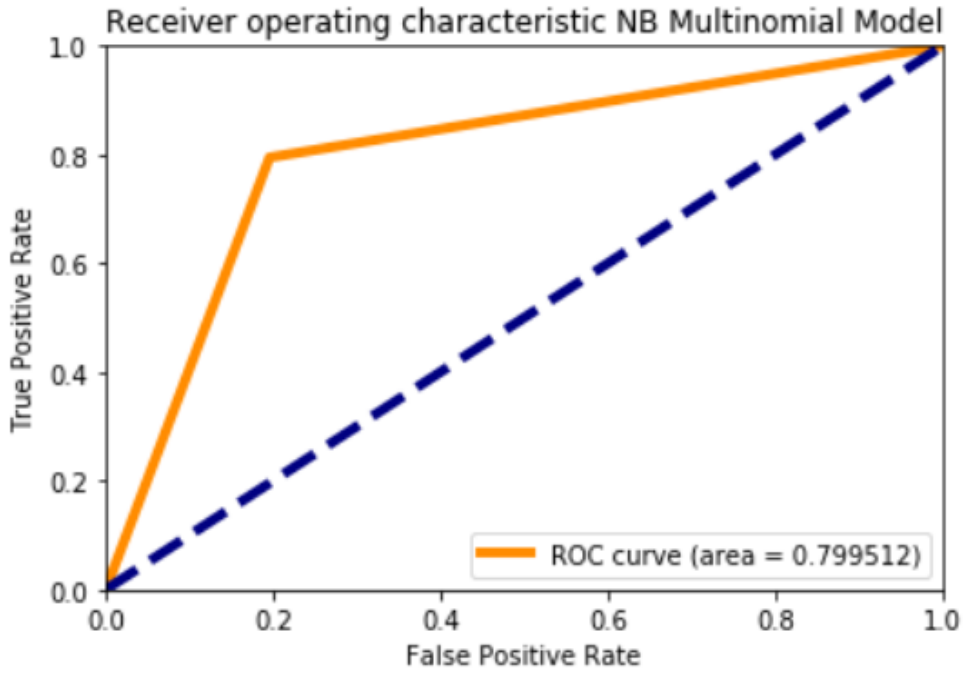
```
f1score.append(f1_score(y_test,y_predMNB,pos_label=4))
precission.append(precision_score(y_test,y_predMNB,pos_label=4))
recall.append(recall_score(y_test,y_predMNB,pos_label=4))
```

Here, the f1_score, precision_score, recall_score is append in the predefine list f1score,precission and recall respectively.

ROC-Curve of NB Multinomial Model

```
#ROC-Curve of NB Multinomial Model.
plt.figure()
lw = 4
plt.plot(fpr, tpr, color='darkorange',lw=lw, label='ROC curve (area = %f)' % auc(fpr,tpr))
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic NB Multinomial Model')
plt.legend(loc="lower right")
plt.show()
```

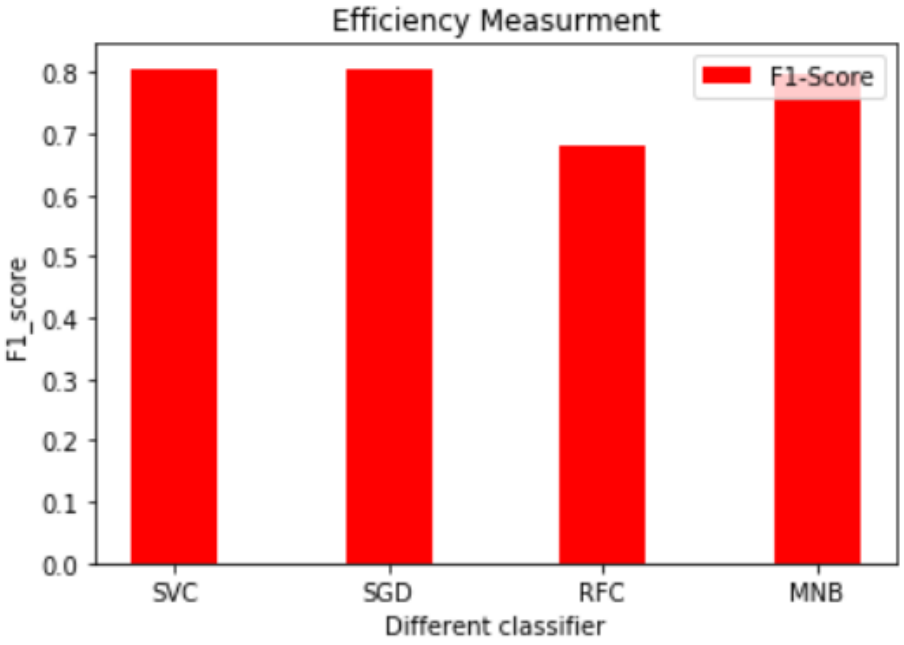
Output



Bar plot for F1-Score

```
#Bar plot for F1-Score
s=["SVC","SGD","RFC","MNB"]
xpos=np.arange(len(s))
plt.xticks(xpos,s)
plt.xlabel("Different classifier")
plt.ylabel("F1_score")
plt.title("Efficiency Measurment")
plt.bar(xpos-0,f1score,color='red',width=0.4,label='F1-Score')
plt.legend(loc="best")
```

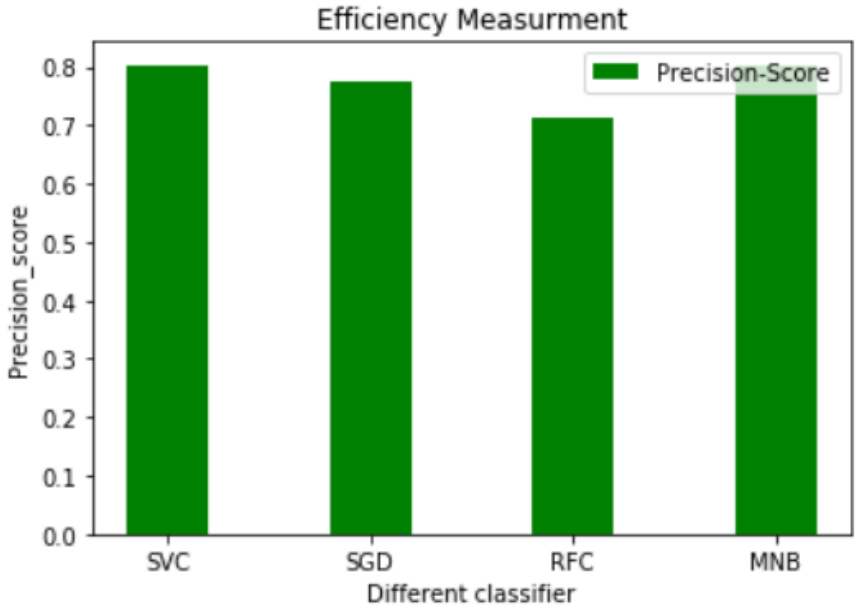
Output



Bar plot for Precision-Score

```
#Bar plot for Precision-Score
s=["SVC","SGD","RFC","MNB"]
xpos=np.arange(len(s))
plt.xticks(xpos,s)
plt.xlabel("Different classifier")
plt.ylabel("Precision_score")
plt.title("Efficiency Measurment")
plt.bar(xpos+0,precission,color='green',width=0.4,label='Precision-Score')
plt.legend(loc="best")
```

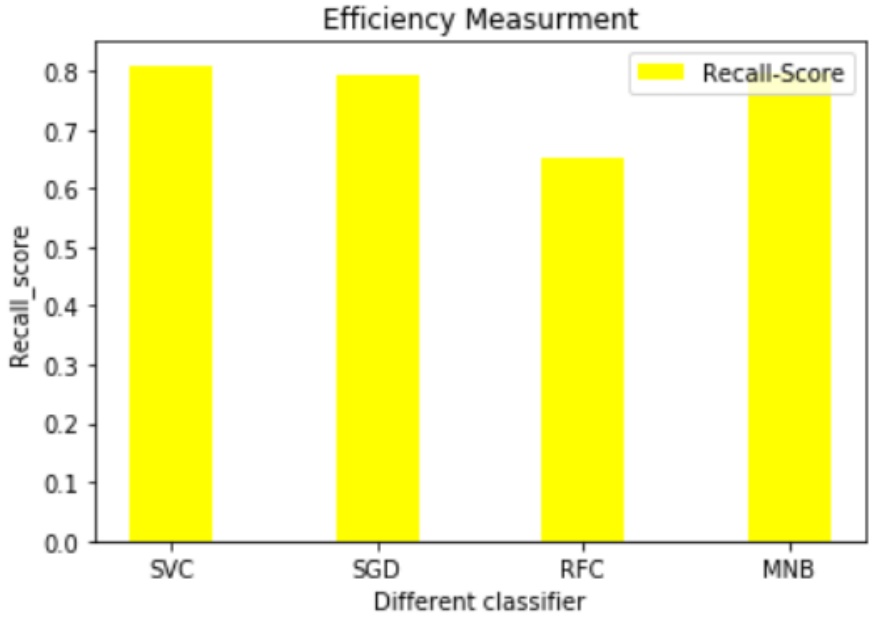
Output



Bar plot for Recall-Score

```
#Bar plot for Recall-Score
s=["SVC","SGD","RFC","MNB"]
xpos=np.arange(len(s))
plt.xticks(xpos,s)
plt.xlabel("Different classifier")
plt.ylabel("Recall_score")
plt.title("Efficiency Measurment")
plt.bar(xpos+0,recall,color='yellow',width=0.4,label='Recall-Score')
plt.legend(loc="best")
```

Output



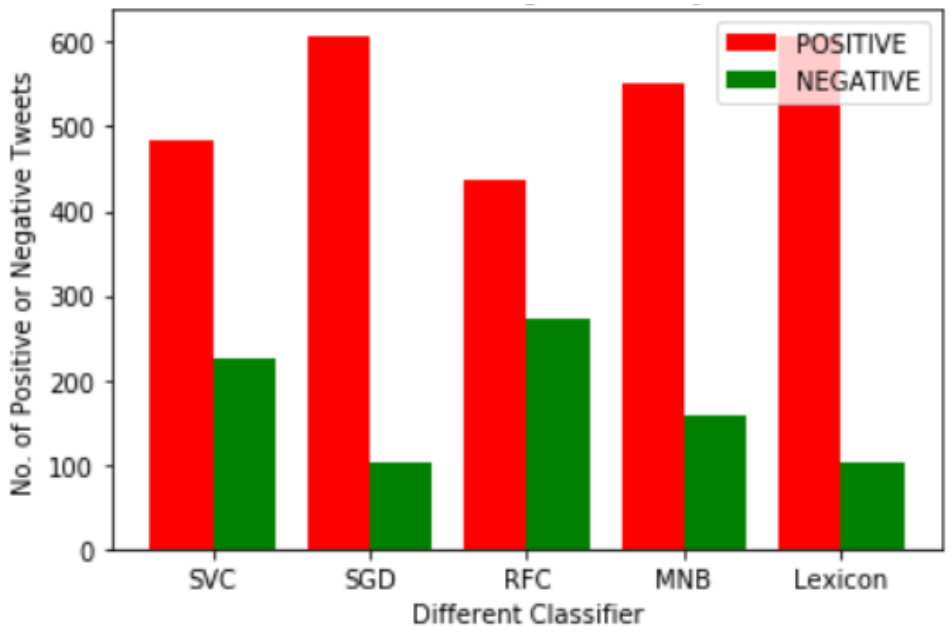
Classification of positive and negative for different Classifier:

```
finalPOS=[]
finalNEG=[]
finalPOS.append(len(posSVC))
finalPOS.append(len(posSGD))
finalPOS.append(len(posRFC))
finalPOS.append(len(posMNB))
finalNEG.append(len(negSVC))
finalNEG.append(len(negSGD))
finalNEG.append(len(negRFC))
finalNEG.append(len(negMNB))
finalPOS.append(sentiment_scores["Positive"])
finalNEG.append(sentiment_scores["Negetive"])
```

Plot for no. of Positive and Negative Tweets

```
#Plot for no. of positive and Negative Tweets.
s=["SVC","SGD","RFC","MNB","Lexicon"]
xpos=np.arange(len(s))
plt.xticks(xpos,s)
plt.xlabel("Different Classifier")
plt.ylabel("No. of Positive or Negative Tweets")
plt.title("Positive and Negative Analyzier")
plt.bar(xpos-0.2,finalPOS,color='red',width=0.4,label='POSITIVE')
plt.bar(xpos+0.2,finalNEG,color='green',width=0.4,label='NEGATIVE')
plt.legend(loc="best")
plt.savefig("C:\\Users\kools\\Desktop\\Assignment\\graph8.png",bbox_inches="tight",pad_inches=2)
```

Output



PERFORMANCE ANALYTICS:

Intuition:

After doing the usual Feature Engineering, Selection, and of course, implementing a model and getting some output in forms of a probability or a class, the next step is to find out how effective is the model based on some metric using test datasets. Different performance metrics are used to evaluate different Machine Learning Algorithms. We have used classification performance metrics such as Accuracy, AUC(Area under Curve) etc. Another example of metric for evaluation of machine learning algorithms is precision, recall, which can be used for sorting algorithms primarily used by search engines.

The metrics chosen to evaluate for any machine learning model is very important. Choice of metrics influences how the performance of machine learning algorithms is measured and compared.

In this project have used the following metrics for model evaluation:-

1. Precision:
   Precision looks at the ratio of correct positive observations.
   The formula is True Positives / (True Positives + False Positives).
   Note that the denominator is the count of all positive predictions, including positive observations of events which were in fact negative.
2. Recall:
   Recall is also known as **sensitivity** or **true positive rate**. It's the ratio of correctly predicted positive events.
   Recall is calculated as True Positives / (True Positives + False Negatives).
   Note that the denominator is the count of all positive events, regardless whether they were correctly predicted by the model.
3. F1 Score:
   The F1 Score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account. Intuitively it is not as easy to understand as accuracy, but F1 is usually more useful than accuracy, especially if the data has uneven class distribution. It works best if false positives and false negatives have similar cost. If the cost of false positives and false negatives are very different, it's better to look at both Precision and Recall.
   The formula for F1 Score is 2*(Recall * Precision)/(Recall + Precision)
4. Accuracy:
   Accuracy in classification problems is the number of correct predictions made by the model over all kinds predictions made.
   The formula is (True Positive+True Negetive)/(True Positive+True Negetive+False Negetive+False Posistive)
   Accuracy is a good measure when the target variable classes in the data are nearly balanced.
   Accuracy should NEVER be used as a measure when the target variable classes in the data are a majority of one class.

5. ROC-AUC:

   AUC (**Area Under The Curve**) ROC (**Receiver Operating Characteristics**) is a performance measurement for classification problem at various thresholds settings. ROC is a probability curve and AUC represents degree or measure of separability. It tells how much model is capable of distinguishing between classes. Higher the AUC, better the model is at predicting 0s as 0s and 1s as 1s. By analogy, Higher the AUC, better the model is at distinguishing between patients with disease and no disease.

   The ROC curve is plotted with True Positive Rate or Recall against the False Positive Rate (False Positive/True Negetive+False Positive) where TPR is on y-axis and FPR is on the x-axis. An excellent model has AUC near to the 1 which means it has good measure of separability. A poor model has AUC near to the 0 which means it has worst measure of separability. In fact it means it is reciprocating the result. It is predicting 0s as 1s and 1s as 0s. And when AUC is 0.5, it means model has no class separation capacity whatsoever.

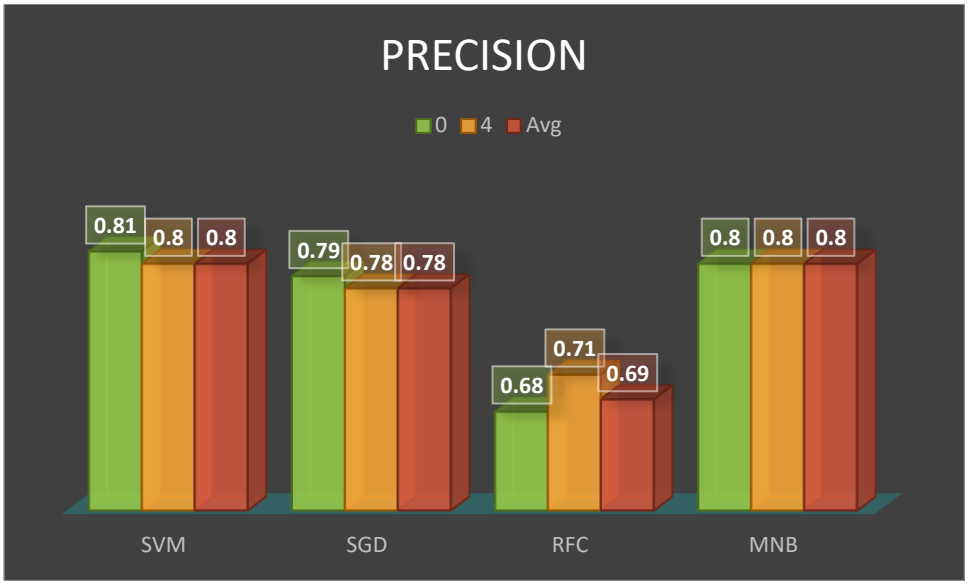
Performance of models:

Models:

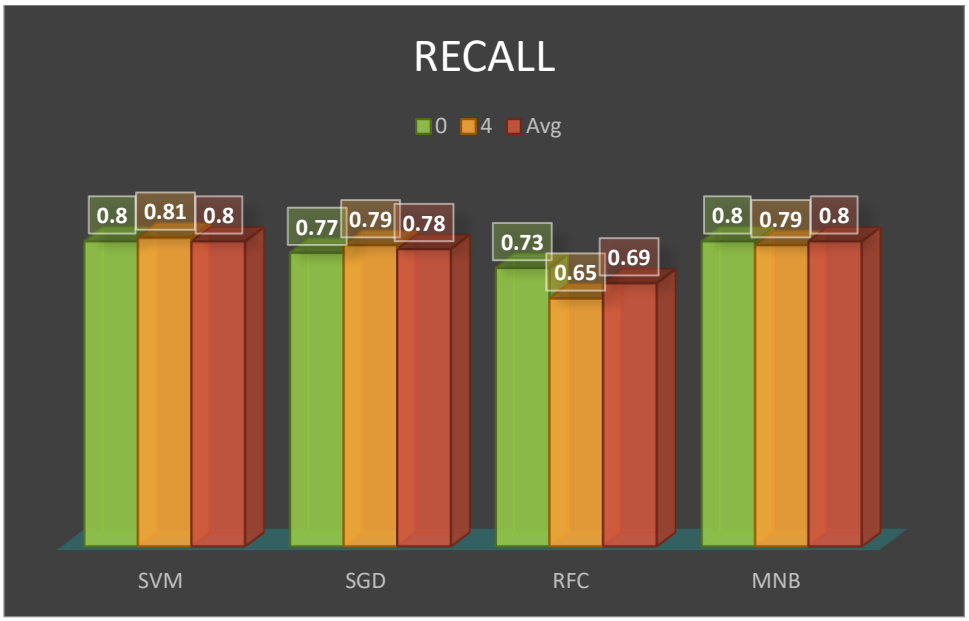
- SVC-Support Vector Machines
- SGD-Stochastic Gradient Descent
- RFC-Random Forest Classifier
- MNB-Multinomial Naïve Bayes

Metrics		*SVC*	*SGD*	*RFC*	*MNB*
Precision	0	0.81	0.79	0.68	0.8
	4	0.8	0.78	0.71	0.8
	Average	0.8	0.78	0.69	0.8
Recall	0	0.8	0.77	0.73	0.8
	4	0.81	0.79	0.65	0.79
	Average	0.8	0.78	0.69	0.8
f1-score	0	0.8	0.78	0.7	0.8
	4	0.81	0.78	0.68	0.8
	Average	0.8	0.78	0.69	0.8
Accuracy Score		0.804897	0.781009	0.692759	0.799497

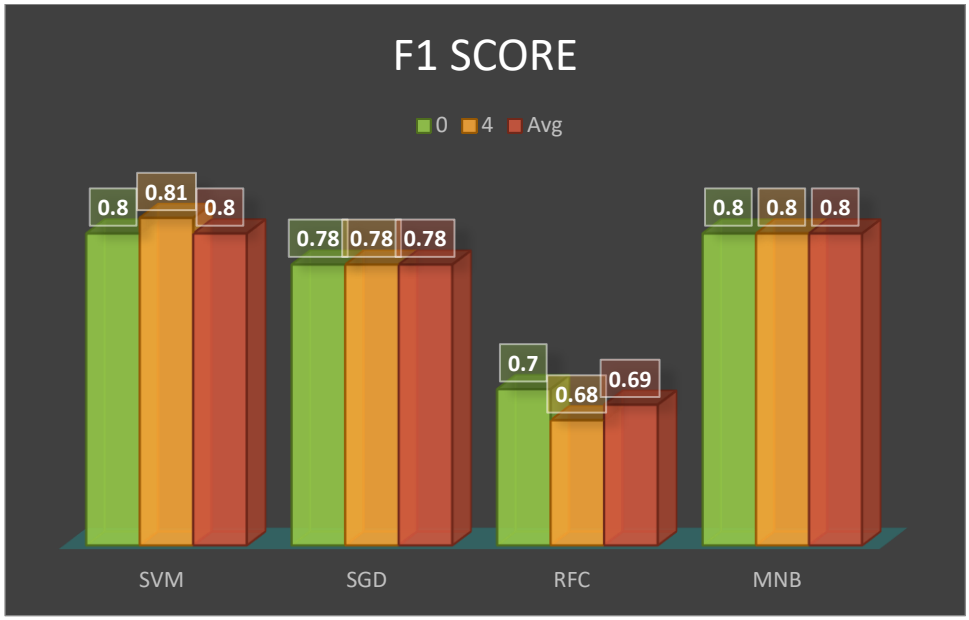
Precision:



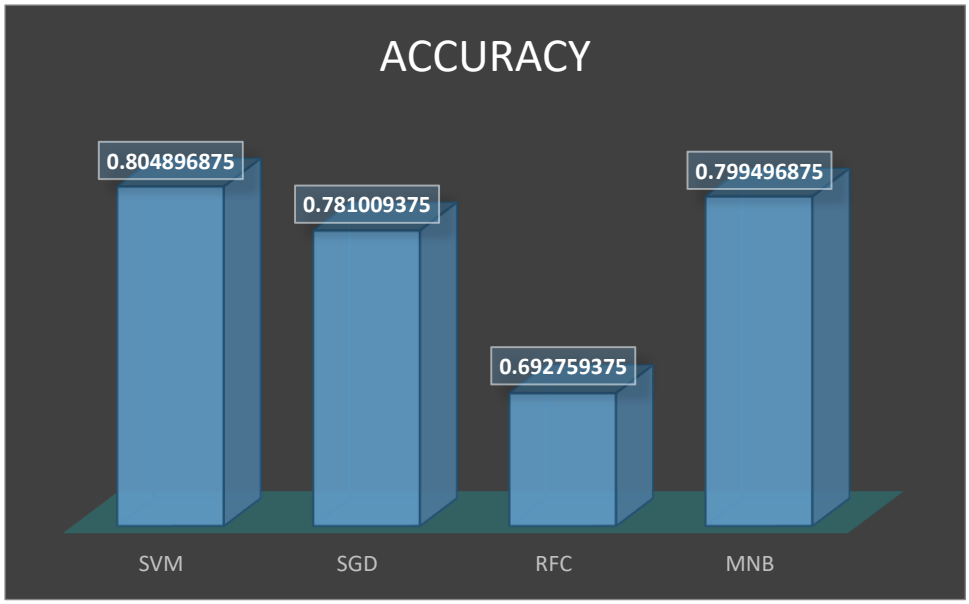
Recall:



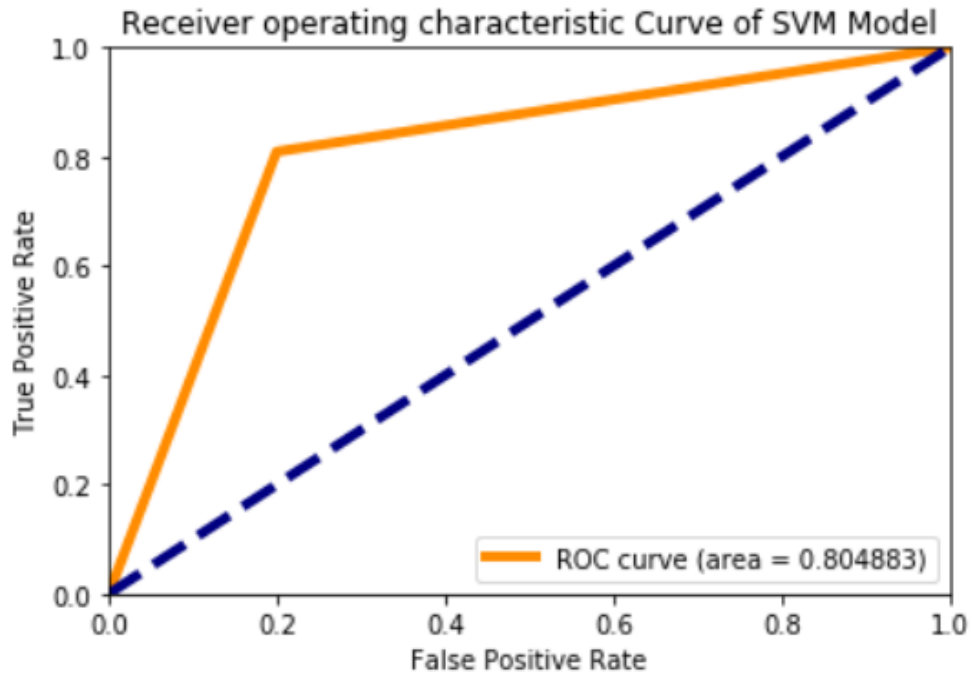
F1-Score:

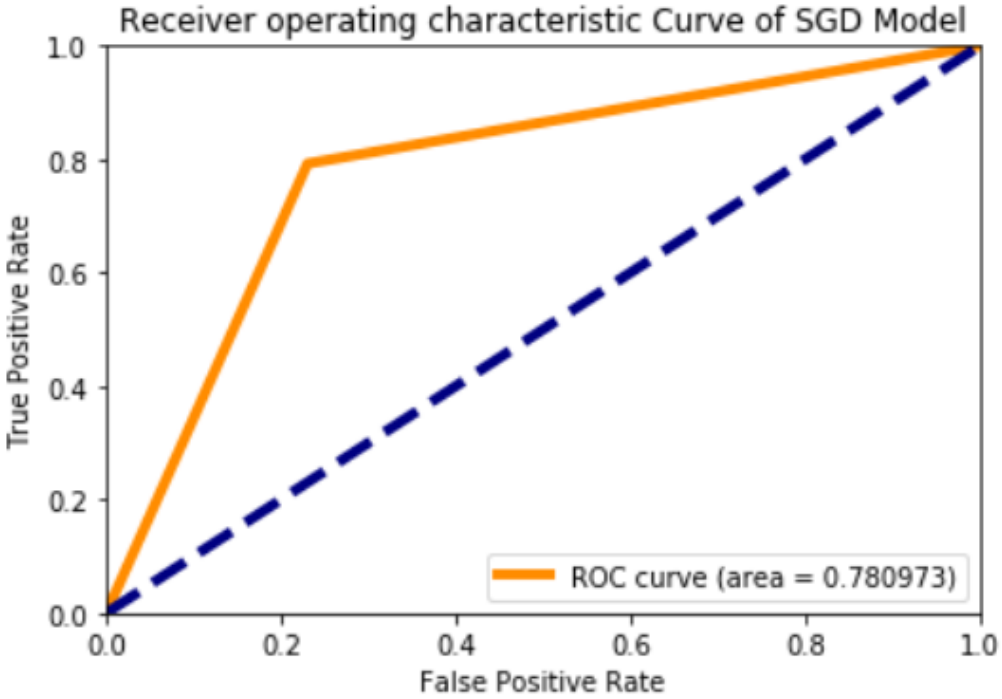


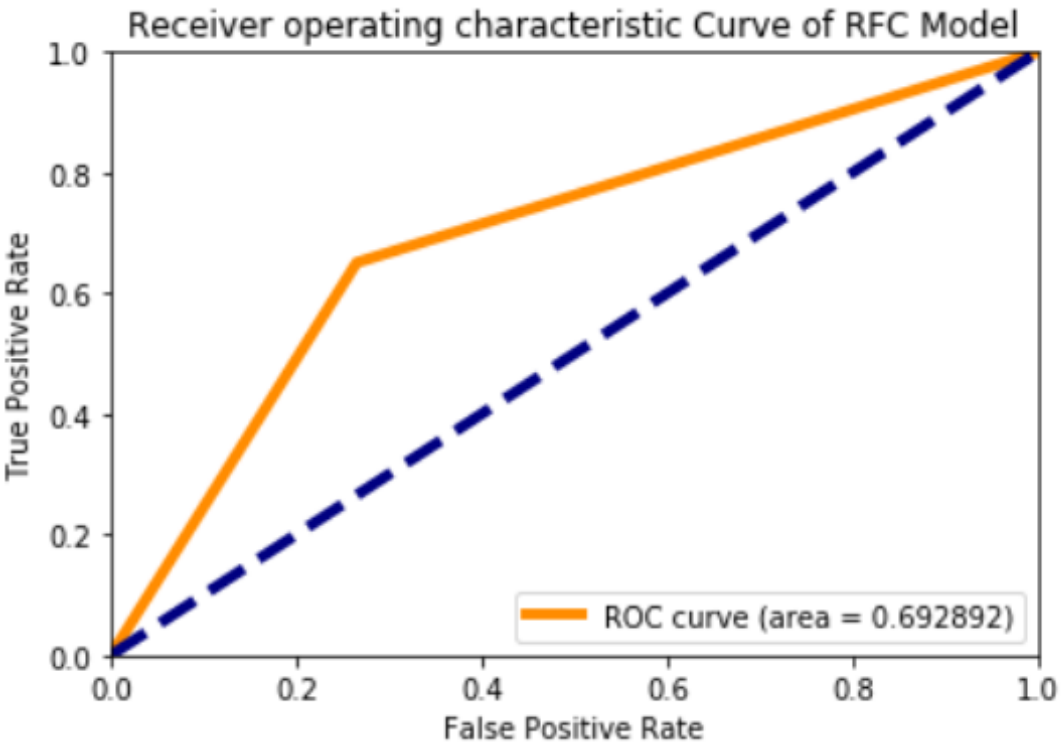
Accuracy:

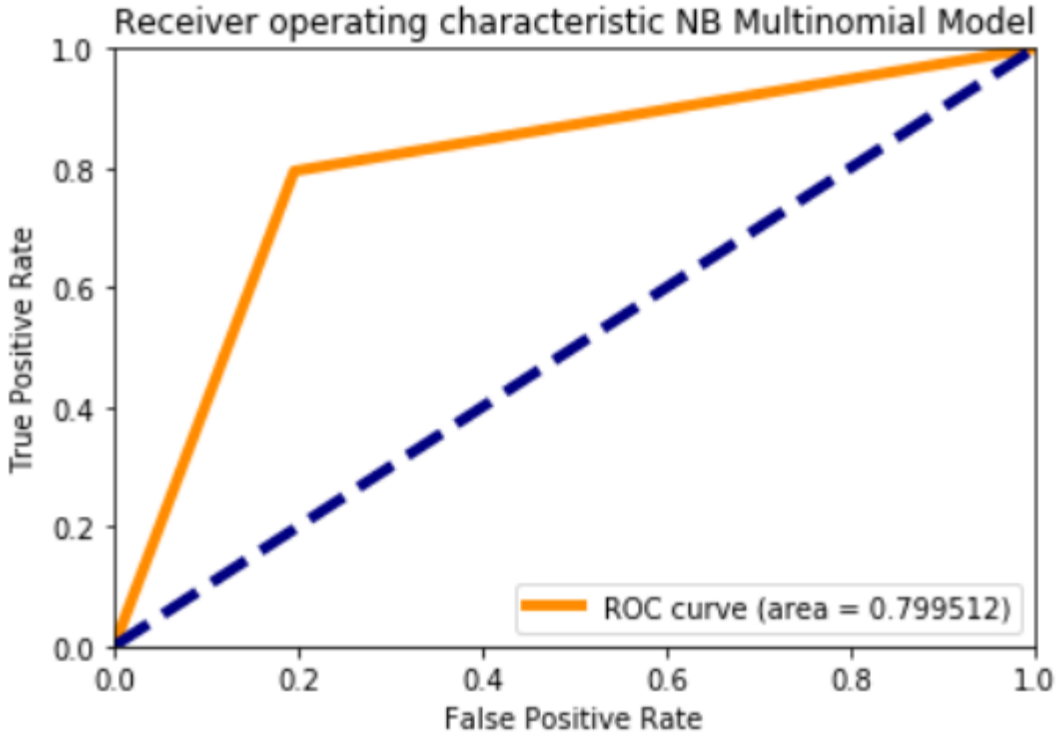


ROC-AUC:


Receiver operating characteristic Curve of SVM Model
True Positive Rate
False Positive Rate
ROC curve (area = 0.804883)
0.0
0.2
0.4
0.6
0.8
1.0


Receiver operating characteristic Curve of SGD Model
True Positive Rate
False Positive Rate
ROC curve (area = 0.780973)
0.0
0.2
0.4
0.6
0.8
1.0


Receiver operating characteristic Curve of RFC Model
True Positive Rate
False Positive Rate
ROC curve (area = 0.692892)
0.0
0.2
0.4
0.6
0.8
1.0


Receiver operating characteristic NB Multinomial Model
True Positive Rate
False Positive Rate
ROC curve (area = 0.799512)
0.0
0.2
0.4
0.6
0.8
1.0

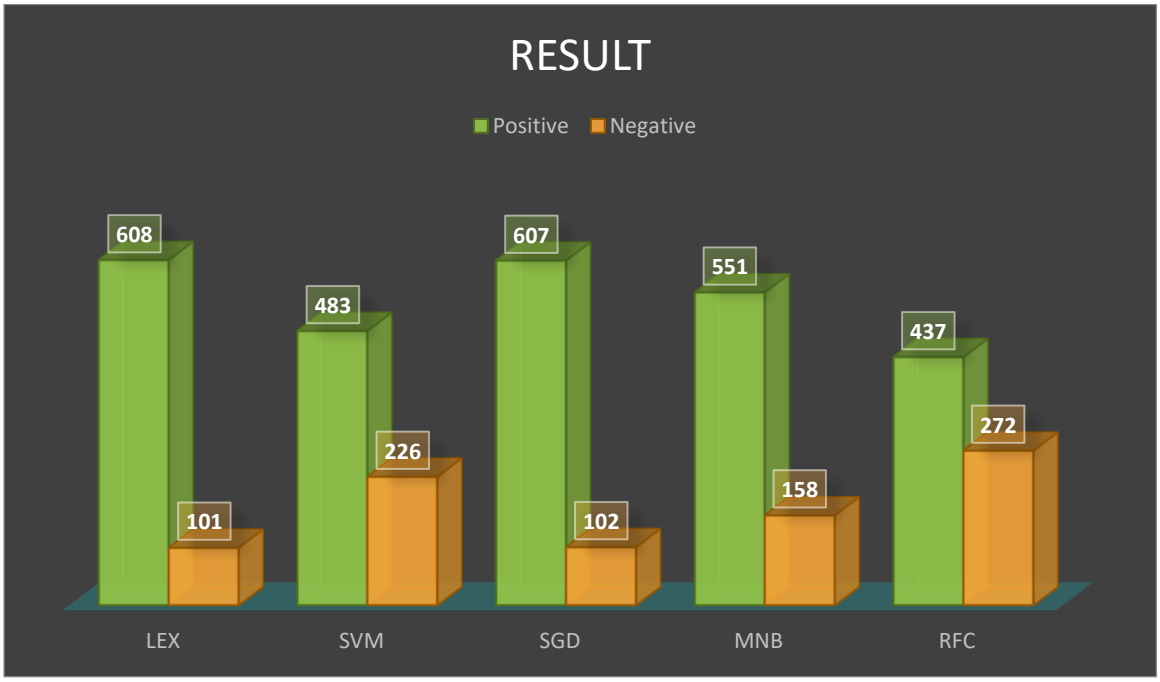
RESULT:

Our final aim was to determine the public opinion of this year's union budget through the most used public forum twitter and doing sentiment analysis on it. We have done it through two processes:

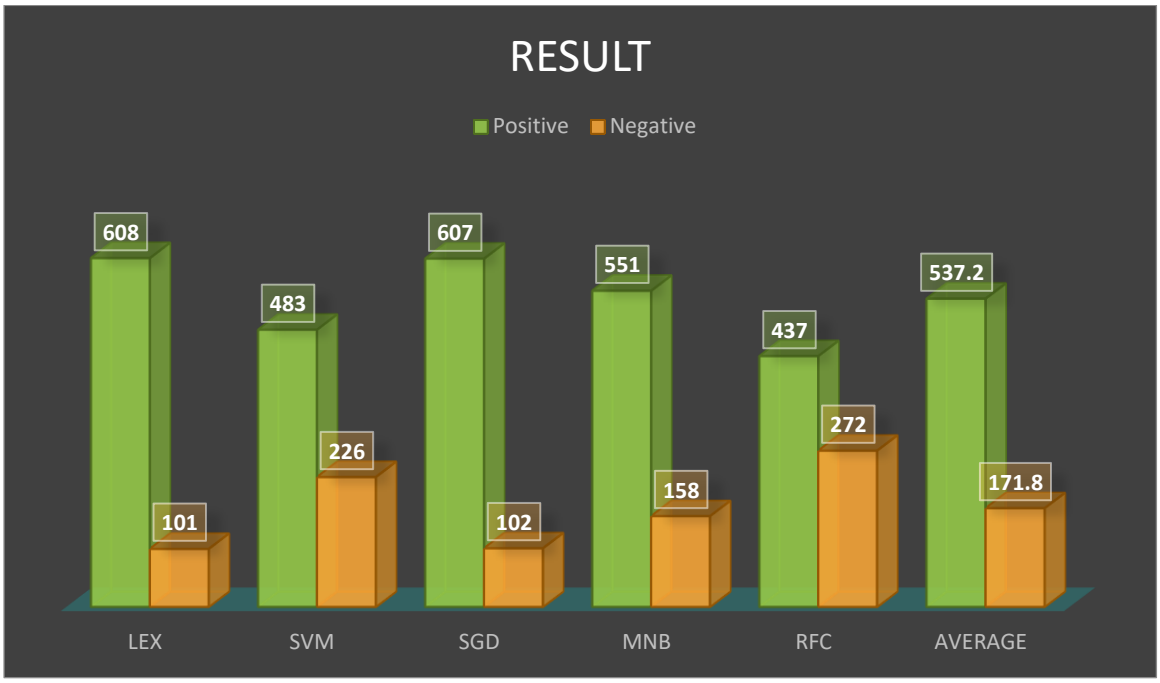
1. Lexicon Based Method
2. Machine Learning Method

So the no. of positive and negative tweets through this processes are given as follows in this chart:

	LEX (LEXICON METHOD)	*SVM (SUPPORT VECTOR MACHINES)*	*SGD (STOCHASTIC GRADIENT DESCENT)*	*MNB (MULTINOMIAL NAÏVE BAYES)*	*RFC (RANDOM FOREST CLASSIFIER)*
Positive	608	483	607	551	437
Negative	101	226	102	158	272



CONCLUSION & FUTURE SCOPE OF IMPROVEMENT:



- We have presented a lexicon based method and a machine learning method.
- SGD (STOCHASTIC GRADIENT DESCENT) seems to give close enough result as lexicon based method
- According to the accuracy the best result is given by SVM (SUPPORT VECTOR MACHINES) which is also close enough to the average value
- The outcomes seems to be inclined to towards positivity
- The models performance can be increased and speed up by using Deep Learning.
- Training Dataset with 3 classes of labels: Positive, Negative and Neutral sentiments can give much wider sentiment prediction on the Test Data

BIBLIOGRAPHY:

- https://www.geeksforgeeks.org/text-preprocessing-in-python-set-1/
- https://www.youtube.com/playlist?list=PLQVvvaa0QuDf2JswnfiGkliBInZnIC4HL
- https://stackoverflow.com/
- https://www.youtube.com/watch?v=4jRBRDbJemM&feature=youtu.be
- https://towardsdatascience.com/gentle-start-to-natural-language-processing-using-python-6e46c07addf3
- This Is How Twitter Sees The World : Sentiment Analysis Part One (https://towardsdatascience.com/the-real-world-as-seen-on-twitter-sentiment-analysispart-one-5ac2d06b63fb)
- Creating The Twitter Sentiment Analysis Program in Python with Naive Bayes Classification (https://towardsdatascience.com/creating-the-twitter-sentimentanalysis-program-in-python-with-naive-bayes-classification-672e5589a7ed)
- Converting words to Features with NLTK (https://pythonprogramming.net/words-asfeatures-nltk-tutorial/)
- Receiver Operating Characteristic (ROC) metric sss(https://scikitlearn.org/stable/auto_examples/model_selection/plot_roc.html)sss

CERTIFICATE

This is to certify that Mr. Riddhinath Ganguly of MCKV Institute of Engineering, registration number 171160110038 of 2017-2018, has successfully completed a project on Sentiment Analysis on Union Budget of 2019 using Machine Learning with Python under the guidance of Prof. Arnab Chakraborty

_____

Prof. Arnab Chakraborty

Globsyn Finishing School

CERTIFICATE

This is to certify that Mr. Sachin Kumar Roy of MCKV Institute of Engineering, registration number 171160110039 of 2017-2018, has successfully completed a project on Sentiment Analysis on Union Budget of 2019 using Machine Learning with Python under the guidance of Prof. Arnab Chakraborty

_____

Prof. Arnab Chakraborty

Globsyn Finishing School

CERTIFICATE

This is to certify that Mr. Dipak Gupta of MCKV Institute of Engineering, registration number 171160110021 of 2017-2018, has successfully completed a project on Sentiment Analysis on Union Budget of 2019 using Machine Learning with Python under the guidance of Prof. Arnab Chakraborty

_____

Prof. Arnab Chakraborty

Globsyn Finishing School

CERTIFICATE

This is to certify that Mr. Arijit Ray of The Heritage Academy, registration number 172131010016 of 2017-2018, has successfully completed a project on Sentiment Analysis on Union Budget of 2019 using Machine Learning with Python under the guidance of Prof. Arnab Chakraborty

_____

Prof. Arnab Chakraborty

Globsyn Finishing School

CERTIFICATE

This is to certify that Mr. Sankha Misra of The Heritage Academy, registration number 172131010080 of 2017-2018, has successfully completed a project on Sentiment Analysis on Union Budget of 2019 using Machine Learning with Python under the guidance of Prof. Arnab Chakraborty

_____

Prof. Arnab Chakraborty

Globsyn Finishing School