CHAPTER 1
INTRODUCTION

1.11 INTRODUCTION TO PROJECT:

Share Market Trends Analysis. The markets are forward-looking: the price one see is a reflection of what the market thinks the price will be six to 12 months in the future rather than in the present day. If we can identify the leading indicators of GDP, then we can accurately predict future moves in GDP and therefore the stock market. Stock Market prediction and analysis is the act of trying to determine the future value of a company stock or other financial instrument traded on an exchange. Stock market is the important part of economy of the country and plays a vital role in the growth of the industry and commerce of the country that eventually affects the economy of the country. Both investors and industry are involved in stock market and wants to know whether some stock will rise or fall over certain period of time. The stock market is the primary source for any company to raise funds for business expansions. It is based on the concept of demand and supply. If the demand for a company's stock is higher, then the company share price increases and if the demand for company's stock is low then the company share price decrease.

NLP is a technique is a technique used by a computer to understand and manipulate natural languages. By natural languages, we mean all human derived languages. Natural language processing is used to analyze text and let machine derive meaning from the input. This HCI allows us to come up with many different applications to bring man and machine as one. For example, on Google, if we use Google translation, that is NLP and so is speech recognition. In this project, I make use of some established NLP technique to evaluate past data pertaining to the stock market and world affairs of the corresponding time period, in order to make predictions in stock trends.

In order to proceed with this objective, there is needed to understand what Sentimental Analysis is. Sentimental analysis is an analytical method that the computer uses to understand a natural language and deduce if the message is positive, neutral or negative. In this case, Sentimental analysis refers to the deduction of the news headlines if they increase the stock or reduce it. By doing so, it end up with the "emotional" status of the data which is what sentimental analysis gives its user.

1.11.1 Generating Model

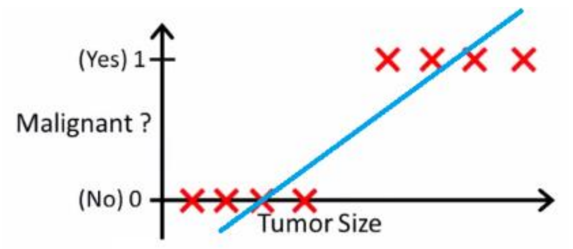
The data available is in textual format, and gives the output as raise, remain same: '1' and fall down as '0'. Model can be applied are as classification model are:

1.11.2 What is a Classification Problem?

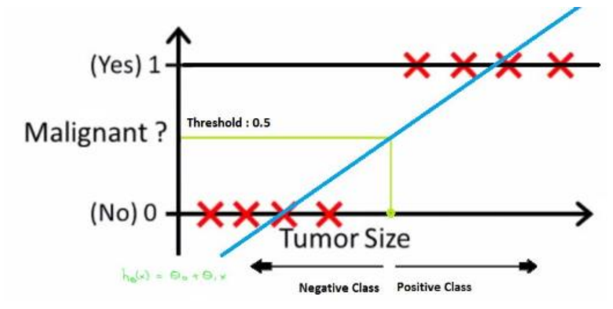
We identify problem as classification problem when independent variables are continuous in nature and dependent variable is in categorical form i.e. in classes like positive class and negative class. The real life example of classification example would be, to categorize the mail as spam or not spam, to categorize the tumour as malignant or benign and to categorize the transaction as fraudulent or genuine. All these problem's answers are in categorical form i.e. Yes or No. and that is why they are two class classification problems.

1.11.3 Why not use Linear Regression?

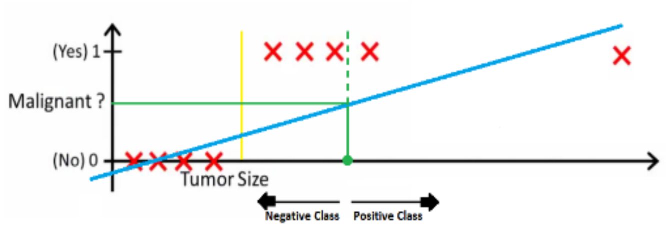
Suppose a data of tumour size v/s its malignancy. As it is a classification problem, if we plot, we can see, all the values will lie on 0 and 1. And if we fit best found regression line, by assuming the threshold at 0.5, we can do line pretty reasonable job.



We can decide the point on the x axis from where all the values lie to its left side are considered as negative class and all the values lie to its right side are positive class.



But what if there is an outlier in the data. Things would get pretty messy. For example, for 0.5 threshold,



If we fit best found regression line, it still won't be enough to decide any point by which we can differentiate classes. It will put some positive class examples into negative class. The green dotted line (Decision Boundary) is dividing malignant tumours from benign tumours but the line should have been at a yellow line which is clearly dividing the positive and negative examples. So just a single outlier is disturbing the whole linear regression predictions. And that is where logistic regression comes into a picture.

Model can be applied are as classification model are:

1.11.4 Logistic Regression

Logistic Regression is one of the basic and popular algorithm to solve a classification problem. It is named as 'Logistic Regression', because it is underlying technique is quite the same as Linear Regression.

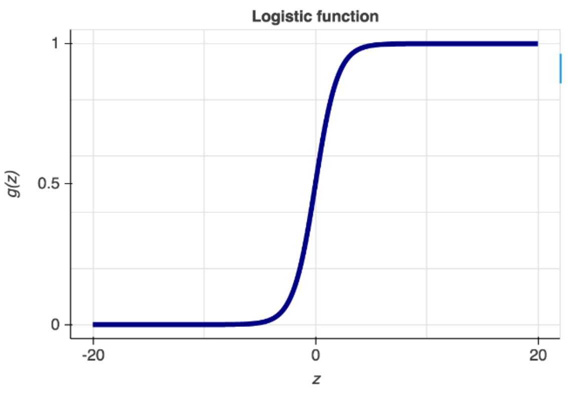


Figure 1 Sigmoid Function

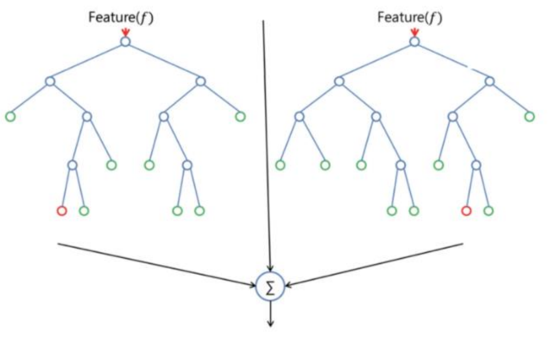
1.11.5 Naive Bayes

Principle on which it works is “every pair of features being classified is independent of each other”. In Gaussian Naive Bayes, continuous values associated with each feature are assumed to be distributed according to a Gaussian distribution.

A Gaussian distribution is also called Normal distribution. When plotted, it gives a bell shaped curve which is symmetric about the mean of the feature values as shown above.

1.11.6 Random Forest Tree

Random Forest is a supervised learning algorithm. Like its name, it creates a forest and makes it somehow random. The “forest” it builds, is an ensemble of Decision Trees, most of the time trained with the “bagging” method. The general idea of the bagging method is that a combination of learning models increases the overall result.



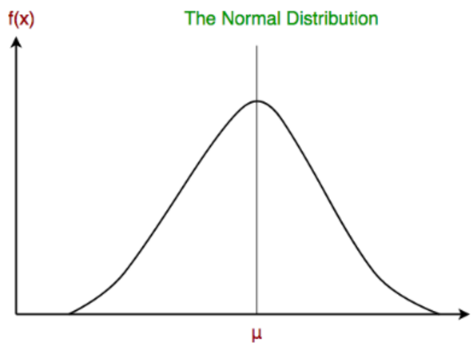


Figure 2 Naive Bayes function

Figure 3 Random Forest Tree

1.11.7 Support Vector Machine

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 two dimensional space this hyperplane is a line dividing a plane in two parts where in each class lay in either side.

1.12 MOTIVATION

Motivation for research in this field is that it possesses many theoretical and experimental challenges. Stock Market prediction and analysis is the act of trying to determine the future value of a company stock or other financial instrument traded on an exchange. Stock market is the important part of economy of the country and plays a vital role in the growth of the industry and commerce of the country that eventually affects the economy of the country. Both investors and industry are involved in stock market and wants to know whether some stock will rise or fall over certain period of time. The stock market is the primary source for any company to raise funds for business expansions. It is based on the concept of demand and supply. If the demand for a company's stock is higher, then the company share price increases and if the demand for company's stock is low then the company share price decrease

CHAPTER 2
LITERATURE SURVEY

2.1 EXISTING SYSTEM

In the last few decades forecasting of stock returns has become an important field of research. In most of the cases the researchers had attempted to establish a linear relationship between the input macroeconomic variables and the stock returns. After the discovery of nonlinearity in the stock market index returns, many literatures have come up in nonlinear statistical modeling of the stock returns, most of them required that the nonlinear model be specified before the estimation is done. But since stock market return is noisy, uncertain, chaotic and nonlinear in nature, different ML algorithm has evolved out to be better technique in capturing the structural relationship between a stock's performance and its determinant factors more accurately than many other statistical techniques. In literature, different sets of input variables are used to predict stock returns. In fact, different input variables are used to predict the same set of stock return data. Some researchers used input data from a single time series where others considered the inclusion of heterogeneous market information and macro-economic variables. Some researchers even pre-processed these input data sets before feeding it to the ML algorithm for forecasting.

2.2 PROBLEM WITH CURRENT SCENARIO

Statement of the problem Stock market is very vast and difficult to understand. It is considered too uncertain to be predictable due to huge fluctuation of the market. Stock market prediction task is interesting as well as divides researchers and academics into two groups, those who believe that we can devise mechanisms to predict the market and those who believe that the market is efficient and whenever new information comes up the market absorbs it by correcting itself, thus there is no space for prediction. Investing in a good stock but at a bad time can have disastrous result, while investing in a stock at the right time can bear profits. Financial investors of today are facing this problem of trading as they do not properly understand as to which stocks to buy or which stocks to sell in order to get optimum result. So, the purposed project will reduce the problem with suitable accuracy faced in such real time scenario.

CHAPTER 3
PROPOSED PROJECT WORK

3.1 OBJECTIVE

Model will be able to buy and sell stock based on profitable prediction, without any human interactions. The model uses NLP to make smart “decisions” based on current affairs, article, etc, With NLP and the basic rule of probability, our goal is to increase the accuracy of the stock predictions. We make use of some established NLP techniques to evaluate past data pertaining to the stock market and world affairs of the corresponding time period, in order to make predictions in stock trends.

3.2 METHODOLOGY

The purposed method for developing the system consists of mainly three main steps. Firstly, data is collected and sorted for relevancy from various sources. Secondly, analysis is carried out on the collected data by examining the current market direction, tracking the industry group and specific companies after which the data is represented and scored accordingly. At last, some model are designed (Classification based) and a suitable algorithm yielding best accuracy is chosen to predict the stock value.

3.3 DATA COLLECTION AND WRANGLING

3.3.1 Data Sources

This project attempts to predict the stock value with respect to the stock’s previous value and trends. It requires historic data of stock market as the project also emphasizes on data mining techniques. So, it is necessary to have a trusted source having relevant and necessary data required for the prediction.

Data used the Combined_News_DJIA.csv dataset (courtesy Aaron7sun, Kaggle.com). The Combined_News_DJIA.csv dataset spans from 2008 to 2016. I extended the dataset to include additional data. This additional data is collected from the Guardian's Restful API for the 2000 to 2008 period. I took the 25 most popular headlines for each given day in this period. In addition, I also pull the Dow Jone s Index(DJI) of Yahoo. Finance's website for the 2000-2008 period to compare the influence of the data. There are two channels of data provided in this dataset:

1. News data that has historical news headlines from Reddit World News Channel. Only the top 25 headlines are considered for a single date.
2. Stock data for DJIA over the corresponding time range is used to label the data. The stock data is compiled from Yahoo Finance.
   Note: The headlines for each data acts as the explanatory data (or Feature variable or Independent data or Predictor) that causes the stock price to either rise (labelled 1) or to fall (labelled 0). We have the top 25 headlines for one single date arranged as one row of the extracted data set.

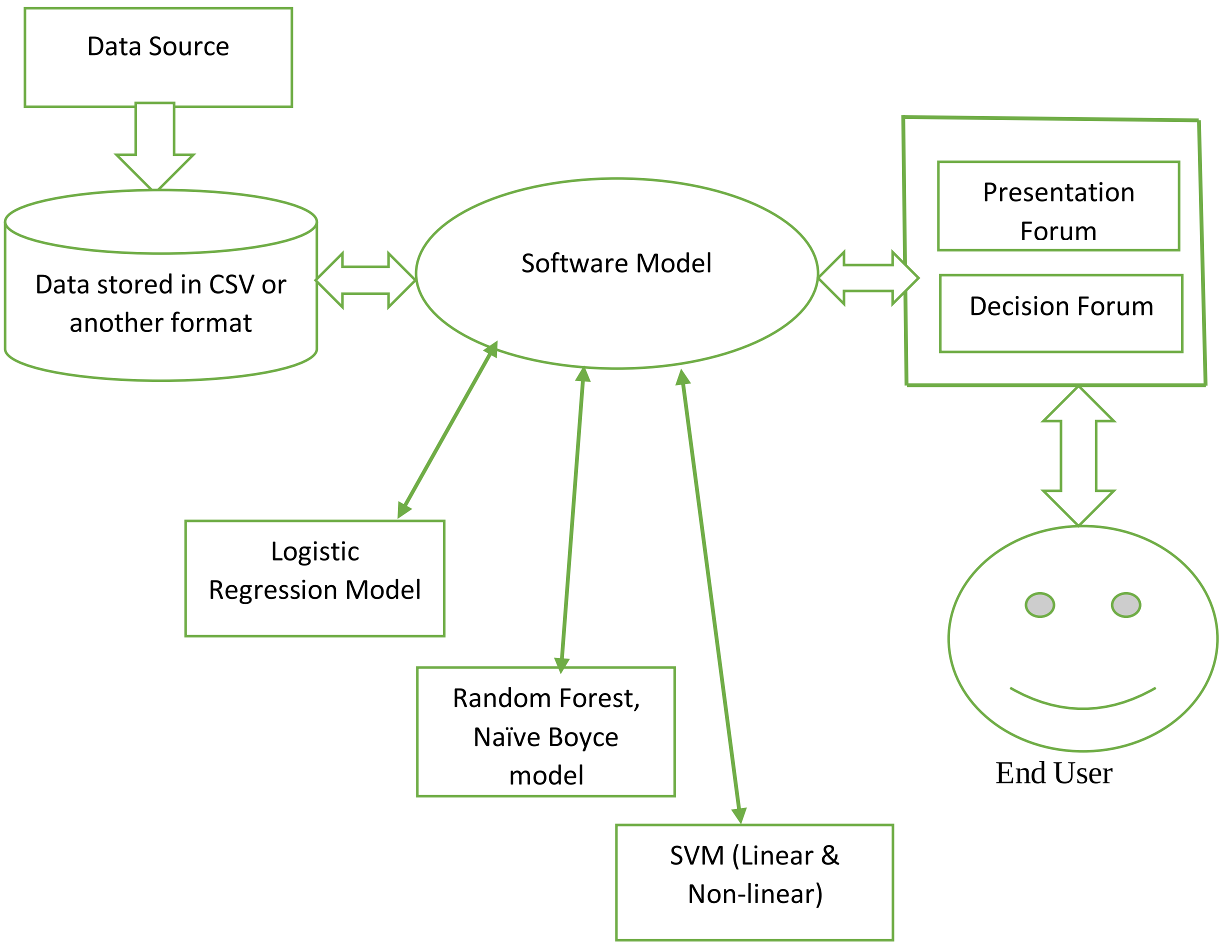
Since goal is to predict the tendency of the stock of a specific company, the data that lead the stock's price of the next day to decline or stay the same are labelled "0", while the data that lead the price of the next day to rise are labelled "1". We compare the data between the two pulled data set and then together to get the more accurate prediction.

With the raw data, proceeding much further is not possible until there is some manipulation done on the data to suit our analysis and convert the data into vectors that are much easier to work on. For this, we use Word2Vec. This is a group of related models used to create word embedding. Word embedding are sets of language modeling and feature learning techniques in NLP where words or phrases from the vocabulary are mapped to vectors of real numbers. These vectors make up the training and test sets. English is really easy- see all those spaces? That make it really easy to tokenize- in other words, to determine what's a word. So we just use a simple set of rules for English tokenization.

This raw data is manipulated using python. We first split data into lists of words but these lists are flooded with HTML tags and punctuations. We cleaned up the data and removed all HTML tags and punctuations. Then we moved forward with removing stop words. Stop words that do not contribute to the meaning or sentiment of the data such as 'the', 'is', 'and', etc. We have also converted all the letters to lowercase to make a more even data set to play with.

3.4 WORKFLOW

With the training data set, we got to convert them into numeric representation for machine learning. For this, we use 'Bag of Words' model. The Bag of Words model learns a vocabulary from all of the documents, then models each document by counting the number of times each word appears. These values are the feature vectors that are derived from the model.


Data Source
Data stored in CSV or another format
Software Model
Presentation Forum
Decision Forum
Logistic Regression Model
Random Forest, Naïve Boyce model
SVM (Linear & Non-linear)
End User

The thing is we cannot stop at just using the Bag of Words model as this generates feature vectors that only give importance to the number of occurrences of words rather than where they occur and with what words they accompany. To get past this, we use the n-gram model or the skip gram model. Now, with this model, we can store the order of words in the way they occur in the data. The number of words stored in a single order depends on the value on n.

Say n=2, calls for a bigram model which stores sets of 2 words in order.

We use NLP to interpret and construct the data set. The data are composed of a row of sentences. In order to reduce the complexity, the stop words such as “a”, “and”, “the” have been cleaned. In addition, we came up across the N-gram model which helps predict the next set of words in an n-worked text or speech. The Google’s Word2Vec deep learning method are also provided to focus on the sentiment of the words by means of the bag of words concept. This method is suitable for us because it does not need labels in order to create meaningful representations. If there are enough training data, it would produces word vector with intriguing characteristics so that we could analyse the relationship between the words that have similar meanings

With all this done, we’ve our manipulated data vectors ready to be trained and tested. We have split the extracted dataset in the ratio of 4:1.

80% of the extracted data will be the training data and 20% of the extracted data will be the test data. We work with 4 models in this project to train this data.

- Naïve Bayes
- Random Forest Tree
- Logistic Regression
- Support Vector Machine

3.5 HARDWARE AND SOFTWARE REQUIREMENTS

Table 1: Software requirements

SOFTWARE REQUIREMENTS	
OS	Windows/ Linux/ Mac OS
OCR	Tesseract OCR
PYTHON-DISTRIBUTION	Anaconda

Table 2: Hardware Requirements

HARDWARE REQUIREMENTS	
PROCESSOR	Intel
RAM	8GB
SECONDARY SPACE	512GB

CHAPTER 4
PROJECT IMPLEMENTATION

4.1 ENVIRONMENT SETUP

4.1.1 Python

Python is developed by Guido van Rossum. Guido van Rossum started implementing Python in 1989. Python is a very simple programming language. What can one do with python?

There are so many applications of Python, here are some of the them: **Web development** – Web framework like Django and Flask are based on Python. They help to write server side code which helps managing the database, writing backend programming logic, mapping urls etc.

Machine learning – There are many machine learning applications written in Python. Machine learning is a way to write a logic so that a machine can learn and solve a particular problem on its own. For example, products recommendation in websites like Amazon, Flipkart, eBay etc. is a machine learning algorithm that recognises user's interest. Face recognition and Voice recognition in one's phone is another example of machine learning.

Data Analysis – Data analysis and data visualisation in form of charts can also be developed using Python.

Scripting – Scripting is writing small programs to automate simple tasks such as sending automated response emails etc. Such type of applications can also be written in Python programming language.

Game development – One can develop games using Python.

One can develop **Embedded applications** in Python.

Desktop applications – One can develop desktop application in Python using library like TKinter or QT.

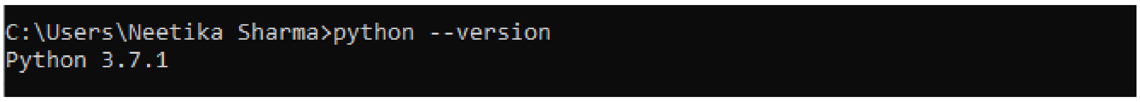
Download and install Python

The Python download requires about 25 Mb of disk space. Enter the given URL for downloading the python setup https://www.python.org/

Install the python 34/64 bit. Follow the instruction given in the below link

https://www.ics.uci.edu/~pattis/common/handouts/pythone clipsejava/python.html

Now verify the python installation. Open CMD and enter such as



```
C:\Users\Neetika Sharma>python --version
Python 3.7.1
```

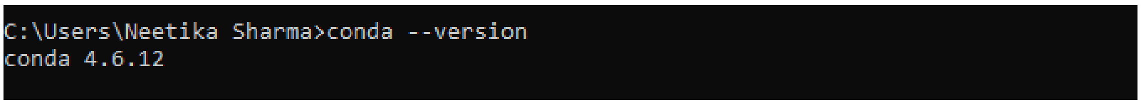
Python installed successfully.

4.1.2 Anaconda

Now comes to the Python distribution for scientific properties for Data Science and Machine Learning purpose. **Anaconda** is a free and open-source distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.), that aims to simplify package management and deployment. Package versions are managed by the package management system *conda* Install Anaconda

Download Anaconda Distribution from the URL given below:

https://www.anaconda.com/distribution/ according to the python version. Check anaconda installation version from cmd.



```
C:\Users\Neetika Sharma>conda --version
conda 4.6.12
```

Anaconda installed successfully.

4.2 SHARE MARKET TREND ANALYSIS WORK FLOW

Steps in the project are shown in figure below are implemented in further steps.

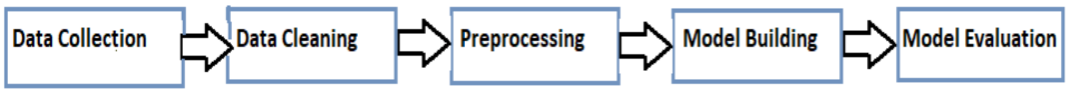


Figure 4 Work Flow of Project Work

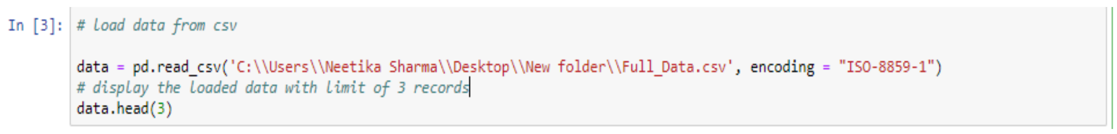
4.2.1 Data Collection

Data set in Deep learning and NLP prospective. Data used the Combined_News_DJIA.csv dataset (courtesy Aaron7sun, Kaggle.com).

The Combined_News_DJIA.csv dataset spans from 2008 to 2016. Available at this link:

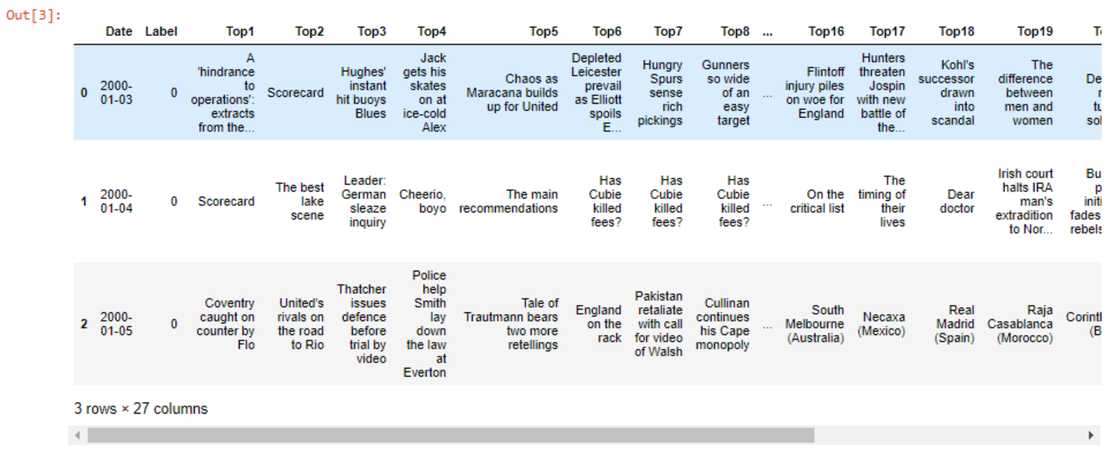
https://www.kaggle.com/aaron7sun/stocknews

After collecting the data, now let’s come to the implementation part. Load the data into the Jupyter notebook for processing.



```
In [3]: # load data from csv

        data = pd.read_csv('C:\\Users\\Neetika Sharma\\Desktop\\New folder\\Full_Data.csv', encoding = "ISO-8859-1")
        # display the loaded data with limit of 3 records
        data.head(3)
```

Output of ‘data.head (3)’ is



Out[3]:

	Date	Label	Top1	Top2	Top3	Top4	Top5	Top6	Top7	Top8	...	Top16	Top17	Top18	Top19	T
0	2000-01-03	0	A 'hindrance to operations': extracts from the...	Scorecard	Hughes' instant hit buoys Blues	Jack gets his skates on at ice-cold Alex	Chaos as Maracana builds up for United	Depleted Leicester prevail as Elliott spoils E...	Hungry Spurs sense rich pickings	Gunners so wide of an easy target	...	Flintoff injury piles on woe for England	Hunters threaten Jospin with new battle of the...	Kohl's successor drawn into scandal	The difference between men and women	De r tu sol
1	2000-01-04	0	Scorecard	The best lake scene	Leader: German sleaze inquiry	Cheerio, boyo	The main recommendations	Has Cubie killed fees?	Has Cubie killed fees?	Has Cubie killed fees?	...	On the critical list	The timing of their lives	Dear doctor	Irish court halts IRA man's extradition to Nor...	Bu p initi fades rebels
2	2000-01-05	0	Coventry caught on counter by Flo	United's rivals on the road to Rio	Thatcher issues defence before trial by video	Police help Smith lay down the law at Everton	Tale of Trautmann bears two more retellings	England on the rack	Pakistan retaliate with call for video of Walsh	Cullinan continues his Cape monopoly	...	South Melbourne (Australia)	Necaxa (Mexico)	Real Madrid (Spain)	Raja Casablanca (Morocco)	Corintl (B

3 rows × 27 columns

4.2.2 Data Cleaning

Data cleaning is the process of detecting and correcting (or removing) corrupt or inaccurate records from a record set, table, or database and refers to identifying incomplete, incorrect, inaccurate or irrelevant parts of the data and then replacing, modifying, or deleting the dirty or coarse data.

Better Data Fancier Algorithm. So steps to clean the data are given below:

Remove unwanted observations

The first step to data cleaning is removing unwanted observations from one's dataset.

This includes **duplicate** or **irrelevant** observations.

Duplicate observations

Duplicate observations most frequently arise during **data collection**, such as when one:

- Combine datasets from multiple places
- Scrape data
- Receive data from clients/other departments

Irrelevant observation

Irrelevant observations are those that don't actually fit the specific problem that one is trying to solve

- For example, if one was building a model for Single-Family homes only, one wouldn't want observations for Apartment in there.
- This is also a great time to review one's charts from Exploratory Analysis. One can look at the distribution charts for categorical features to see if there are any classes that shouldn't be there.
- Checking for irrelevant observations **before engineering features** can save one many problems down the road.

Fix Structural Errors

Structural errors are those that arise during measurement, data transfer, or other types of **"poor housekeeping."**

Finally, check for **mislabelled classes**, i.e. separate classes that should really be the same.

- e.g. If ’N/A’ and ’Not Applicable’ appear as two separate classes, one should combine them.
- e.g. If ’N/A’ and ’Not Applicable’ appear as two separate classes, one should combine them.

Handle Missing Data

Missing data is a deceptively tricky issue in applied machine learning.

First, just to be clear, one cannot simply ignore missing values in one’s dataset. One must handle them in some way for the very practical reason that most algorithms do not accept missing values. "Common sense" is not sensible here. Unfortunately, from our experience, the 2 most commonly recommended ways of dealing with missing data actually suck. They are:

- **Dropping** observations that have missing values
- **Imputing** the missing values based on other observations

Dropping missing values is sub-optimal because when one drop observations, one drop information. The fact that the value was missing may be informative in itself. Plus, in the real world, one often need to make predictions on new data even if some of the features are missing!

Imputing missing values is sub-optimal because the value was originally missing but one filled it in, which always leads to a loss in information, no matter how sophisticated one ’s imputation method is. Again, "missingness" is almost always informative in itself, and one should tell one’s algorithm if a value was missing.

Even if one build a model to impute one’s values, one is not adding any real information. One is just reinforcing the patterns already provided by other features.

- **Missing categorical data**

The best way to handle missing data for categorical features is to simply label them as ’Missing’! Essentially adding a new class for the feature.

This tells the algorithm that the value was missing. This also gets around the technical requirement for no missing values.

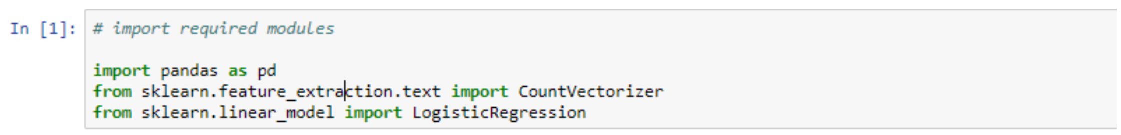
- **Missing numeric data**

  For missing numeric data, one should flag and fill the values. Flag the observation with an indicator variable of missingness. Then, fill the original missing value with 0 just to meet the technical requirement of no missing values.

  By using this technique of flagging and filling, one is essentially allowing the algorithm to estimate the optimal constant for missingness, instead of just filling it in with the mean.

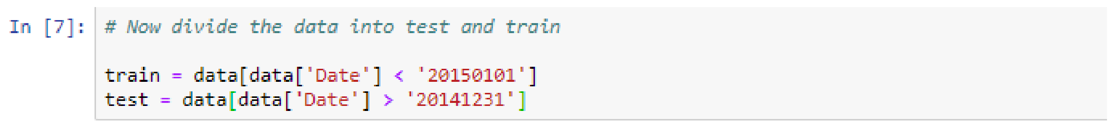
4.2.3 Pre-processing

Import required modules for the project



```
In [1]: # import required modules

        import pandas as pd
        from sklearn.feature_extraction.text import CountVectorizer
        from sklearn.linear_model import LogisticRegression
```

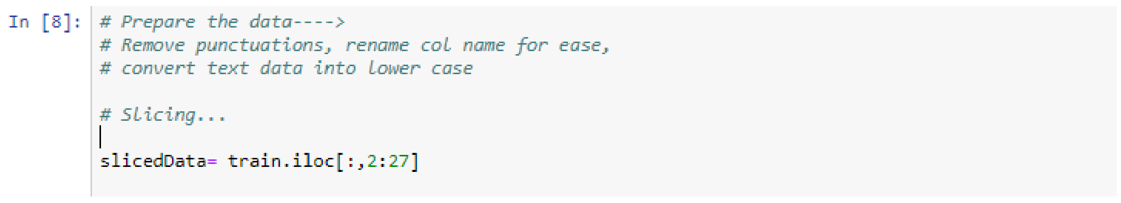
Now divide the data into test and train for further processing.



```
In [7]: # Now divide the data into test and train

        train = data[data['Date'] < '20150101']
        test = data[data['Date'] > '20141231']
```

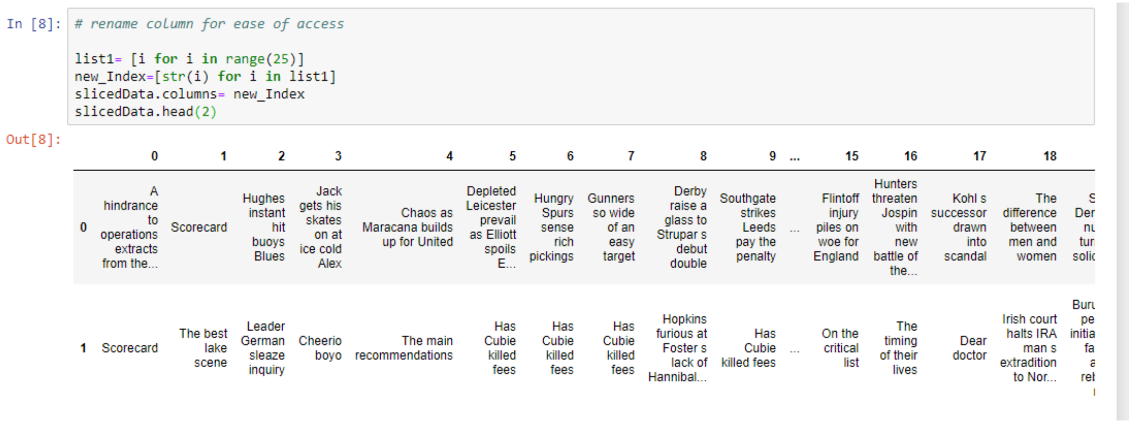
Prepare the data:

Remove punctuations, rename the column name for ease of use and finally convert text date into the lower case letters.



```
In [8]: # Prepare the data---->
        # Remove punctuations, rename col name for ease,
        # convert text data into lower case

        # Slicing...

        slicedData= train.iloc[:,2:27]
```

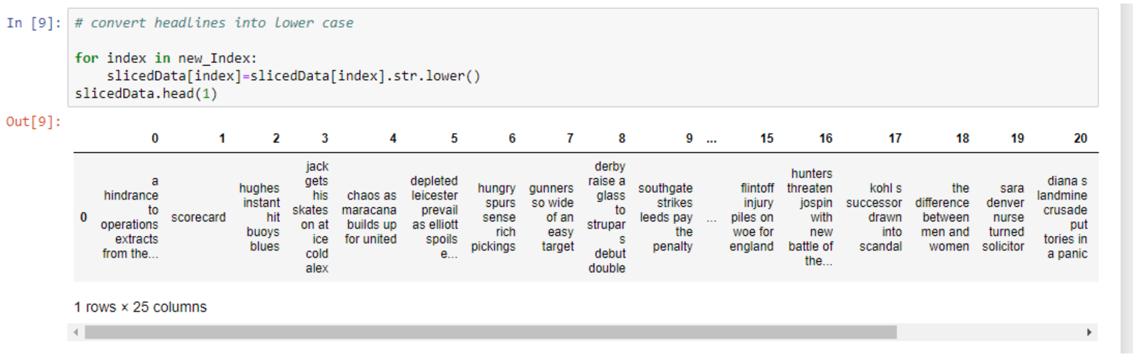
Now rename the column name for the ease of access and the output is given below:



```
In [8]: # rename column for ease of access

        list1= [i for i in range(25)]
        new_Index=[str(i) for i in list1]
        slicedData.columns= new_Index
        slicedData.head(2)
Out[8]:
```

	0	1	2	3	4	5	6	7	8	9	...	15	16	17	18	
0	A hindrance to operations extracts from the...	Scorecard	Hughes instant hit buoys Blues	Jack gets his skates on at ice cold Alex	Chaos as Maracana builds up for United	Depleted Leicester prevail as Elliott spoils E...	Hungry Spurs sense rich pickings	Gunners so wide of an easy target	Derby raise a glass to Strupar s debut double	Southgate strikes Leeds pay the penalty	...	Flintoff injury piles on woe for England	Hunters threaten Jospin with new battle of the...	Kohl s successor drawn into scandal	The difference between men and women	S Der nu tur solic
1	Scorecard	The best lake scene	Leader German sleaze inquiry	Cheerio boyo	The main recommendations	Has Cubie killed fees	Has Cubie killed fees	Has Cubie killed fees	Hopkins furious at Foster s lack of Hannibal...	Has Cubie killed fees	...	On the critical list	The timing of their lives	Dear doctor	Irish court halts IRA man s extradition to Nor...	Buru pe initia fa a ret

Next step is to convert the alphabets pattern into same pattern i.e. lower case.

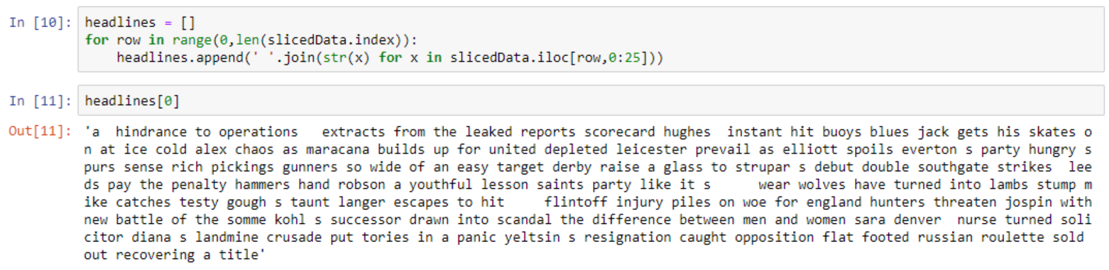


```
In [9]: # convert headlines into lower case

        for index in new_Index:
            slicedData[index]=slicedData[index].str.lower()
        slicedData.head(1)
Out[9]:
```

	0	1	2	3	4	5	6	7	8	9	...	15	16	17	18	19	20
0	a hindrance to operations extracts from the...	scorecard	hughes instant hit buoys blues	jack gets his skates on at ice cold alex	chaos as maracana builds up for united	depleted leicester prevail as elliott spoils e...	hungry spurs sense rich pickings	gunners so wide of an easy target	derby raise a glass to strupar s debut double	southgate strikes leeds pay the penalty	...	flintoff injury piles on woe for england	hunters threaten jospin with new battle of the...	kohl s successor drawn into scandal	the difference between men and women	sara denver nurse turned solicitor	diana s landmine crusade put tories in a panic

1 rows × 25 columns

For the ease of further access change the pattern of data. Change it from the Dataframe (Pandas) into List. And the output is given below. Data is stored in the form of String in list.



```
In [10]: headlines = []
         for row in range(0,len(slicedData.index)):
             headlines.append(' '.join(str(x) for x in slicedData.iloc[row,0:25]))

In [11]: headlines[0]

Out[11]: 'a  hindrance to operations   extracts from the leaked reports scorecard hughes  instant hit buoys blues jack gets his skates o
         n at ice cold alex chaos as maracana builds up for united depleted leicester prevail as elliott spoils everton s party hungry s
         purs sense rich pickings gunners so wide of an easy target derby raise a glass to strupar s debut double southgate strikes  lee
         ds pay the penalty hammers hand robson a youthful lesson saints party like it s      wear wolves have turned into lambs stump m
         ike catches testy gough s taunt langer escapes to hit     flintoff injury piles on woe for england hunters threaten jospin with
         new battle of the somme kohl s successor drawn into scandal the difference between men and women sara denver  nurse turned soli
         citor diana s landmine crusade put tories in a panic yeltsin s resignation caught opposition flat footed russian roulette sold
         out recovering a title'
```

4.2.4 Model Building

Logistic Regression

Logistic Regression is one of the basic and popular algorithm to solve a classification problem. It is named as ‘Logistic Regression’, because it is underlying technique is quite the same as Linear Regression.

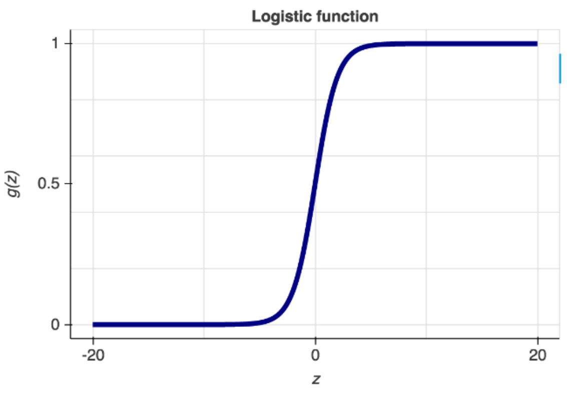


Figure 5 Sigmoid Function

The logistic curve relates the independent variable, X, to the rolling mean of the DV, P (Y'). The formula to do so may be written either

$$P = \frac{e^{a+bX}}{1+e^{a+bX}} \quad \text{or} \quad P = \frac{1}{1+e^{-(a+bX)}}$$

Since the data available is Natural Language Data and the problem is of classification therefore four model selected are: Logistic Regression, Naïve Bayes, Random Forest Tree, and Support Vector Machine (Linear and Non-linear).

It is used to convert a collection of text documents to a matrix of token counts and it is the concept of **Deep Learning.** Target feature is in the form of 0 and 1 so, vectorizer can be applied on the data.

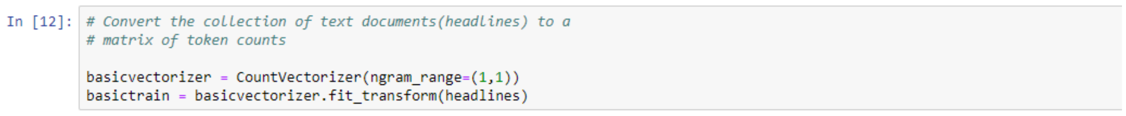
ngram_range: tuple (min_n, max_n)

The lower and upper boundary of the range of n-values for different n-grams to be extracted. All values of n such that **min_n <= n <= max_n** will be used.

Detailed documentation of Feature Extraction (vectorizer) is given at this link

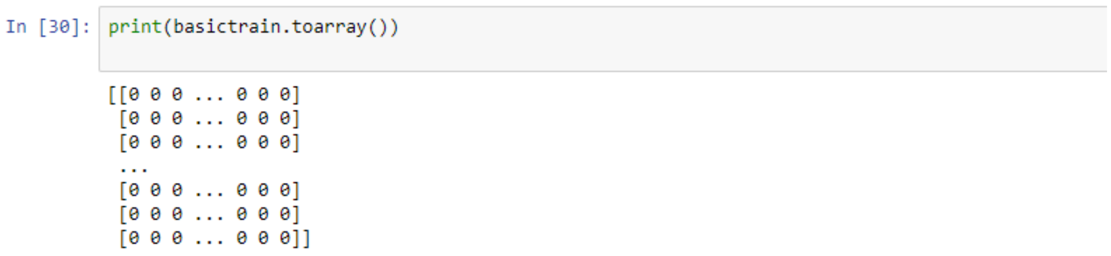
https://scikit-learn.org/stable/modules/feature_extraction.html

1-Gram model



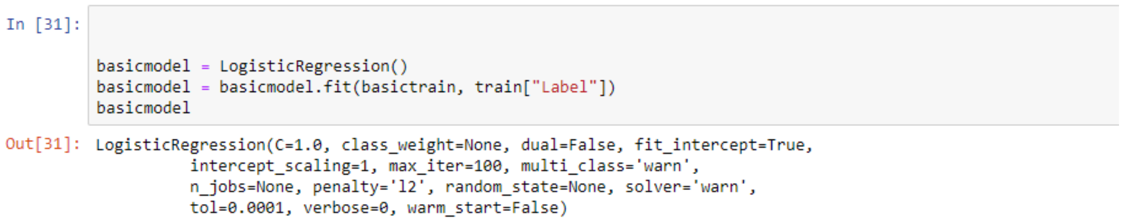
```
In [12]: # Convert the collection of text documents(headlines) to a
         # matrix of token counts

         basicvectorizer = CountVectorizer(ngram_range=(1,1))
         basictrain = basicvectorizer.fit_transform(headlines)
```

Output: Textual data (headlines) is converted into the matrix shown below



```
In [30]: print(basictrain.toarray())

         [[0 0 0 ... 0 0 0]
          [0 0 0 ... 0 0 0]
          [0 0 0 ... 0 0 0]
          ...
          [0 0 0 ... 0 0 0]
          [0 0 0 ... 0 0 0]
          [0 0 0 ... 0 0 0]]
```

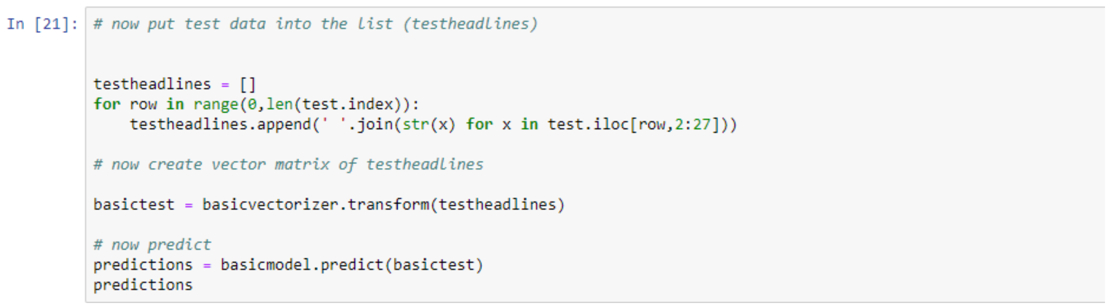
Creating the model and then fitting it to the data. Here ‘**basictrain’** is the feature variable and ‘**train[‘label’]**’ is the Target variable.



```
In [31]:

         basicmodel = LogisticRegression()
         basicmodel = basicmodel.fit(basictrain, train["Label"])
         basicmodel

Out[31]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, max_iter=100, multi_class='warn',
                   n_jobs=None, penalty='l2', random_state=None, solver='warn',
                   tol=0.0001, verbose=0, warm_start=False)
```

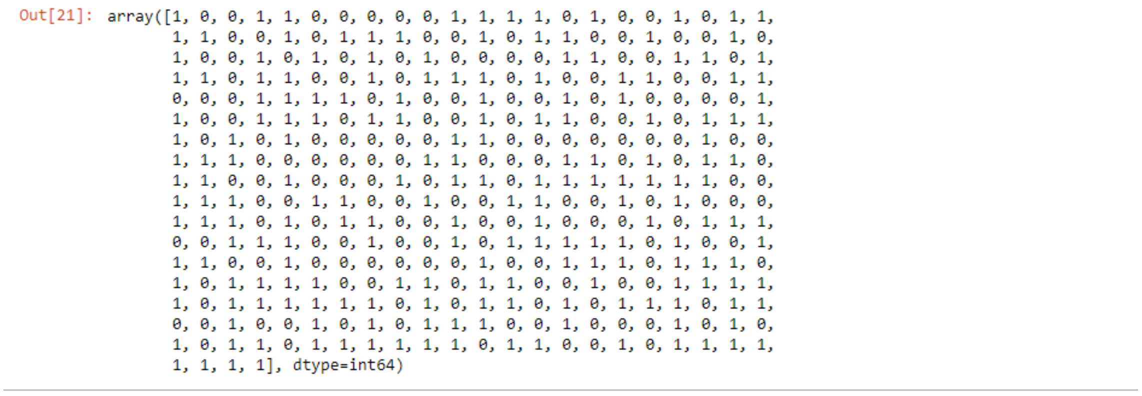
Testing

Predictions predicted from the model can be seen in the output screenshot from the 1-gram model for Logistic Regression. Create an empty list (testheadlines) and put content of Dataframe (Test) into it and apply vectorizer. Finally predict the predictions.



```
In [21]: # now put test data into the list (testheadlines)


         testheadlines = []
         for row in range(0,len(test.index)):
             testheadlines.append(' '.join(str(x) for x in test.iloc[row,2:27]))

         # now create vector matrix of testheadlines

         basictest = basicvectorizer.transform(testheadlines)

         # now predict
         predictions = basicmodel.predict(basictest)
         predictions
```

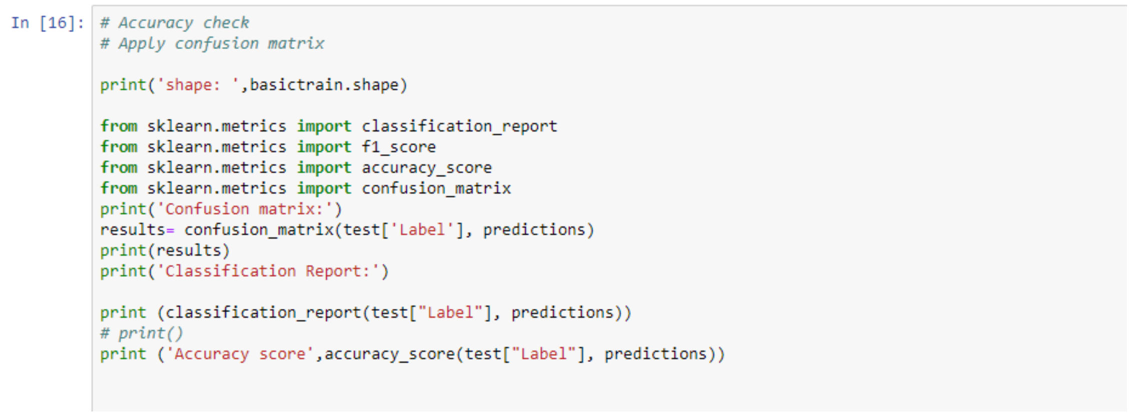
Output:



```
Out[21]: array([1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1,
                1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0,
                1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1,
                1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1,
                0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1,
                1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1,
                1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
                1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0,
                1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0,
                1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0,
                1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1,
                0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1,
                1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0,
                1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1,
                1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1,
                0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0,
                1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1,
                1, 1, 1, 1], dtype=int64)
```

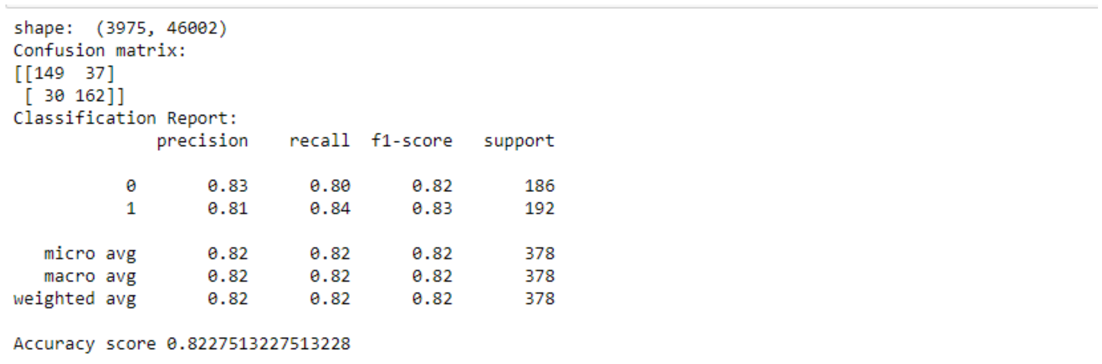
Accuracy for 1-Gram model is:

Confusion matrix and accuracy_score are used to check the rights/wrongs and accuracy of the model.



```
In [16]: # Accuracy check
         # Apply confusion matrix

         print('shape: ',basictrain.shape)

         from sklearn.metrics import classification_report
         from sklearn.metrics import f1_score
         from sklearn.metrics import accuracy_score
         from sklearn.metrics import confusion_matrix
         print('Confusion matrix:')
         results= confusion_matrix(test['Label'], predictions)
         print(results)
         print('Classification Report:')

         print (classification_report(test["Label"], predictions))
         # print()
         print ('Accuracy score',accuracy_score(test["Label"], predictions))
```

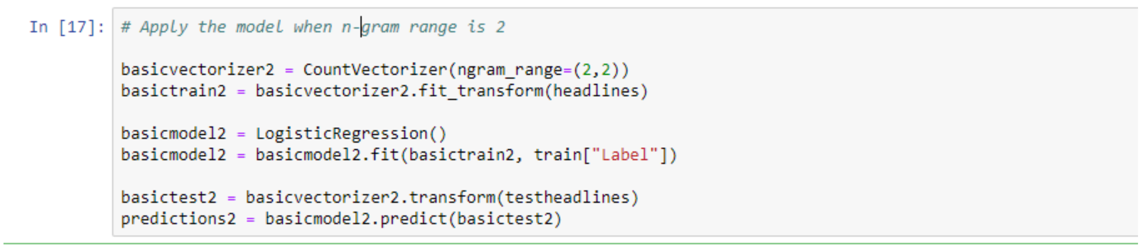
Output:



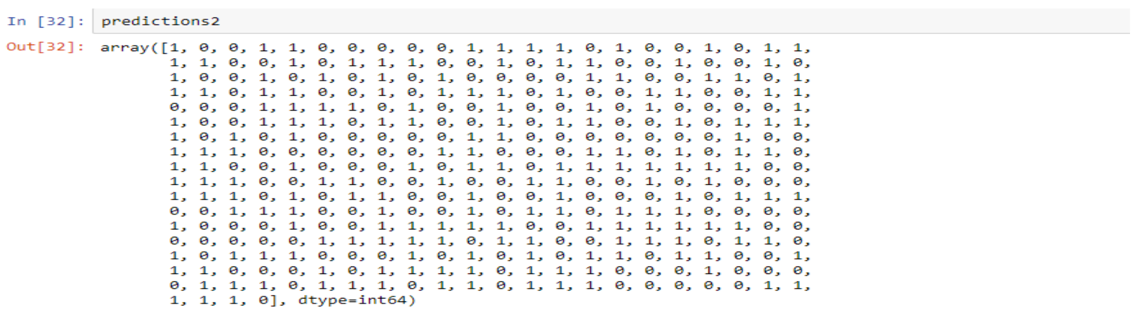
```
shape:  (3975, 46002)
Confusion matrix:
[[149  37]
 [ 30 162]]
Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.80      0.82       186
           1       0.81      0.84      0.83       192

   micro avg       0.82      0.82      0.82       378
   macro avg       0.82      0.82      0.82       378
weighted avg       0.82      0.82      0.82       378

Accuracy score 0.8227513227513228
```

Bi-Gram model

Matrix formed with 2-gram model

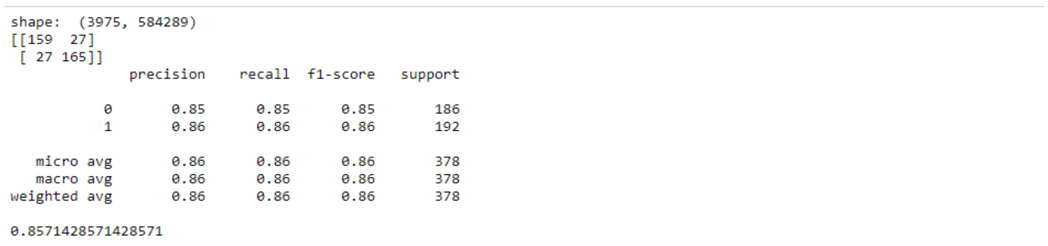


```
In [17]: # Apply the model when n-gram range is 2

         basicvectorizer2 = CountVectorizer(ngram_range=(2,2))
         basictrain2 = basicvectorizer2.fit_transform(headlines)

         basicmodel2 = LogisticRegression()
         basicmodel2 = basicmodel2.fit(basictrain2, train["Label"])

         basictest2 = basicvectorizer2.transform(testheadlines)
         predictions2 = basicmodel2.predict(basictest2)
```



```
In [32]: predictions2
Out[32]: array([1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1,
                1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0,
                1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1,
                1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1,
                0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1,
                1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1,
                1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
                1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0,
                1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0,
                1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0,
                1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1,
                0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0,
                1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0,
                0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0,
                1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1,
                1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0,
                0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1,
                1, 1, 1, 0], dtype=int64)
```

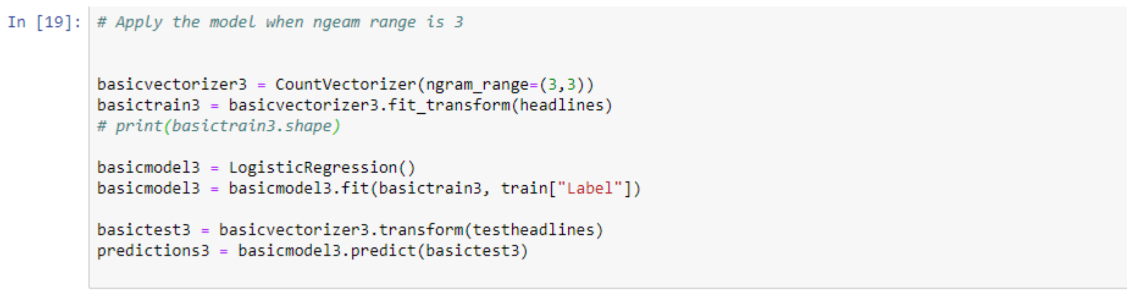
Accuracy for bi-Gram model is:



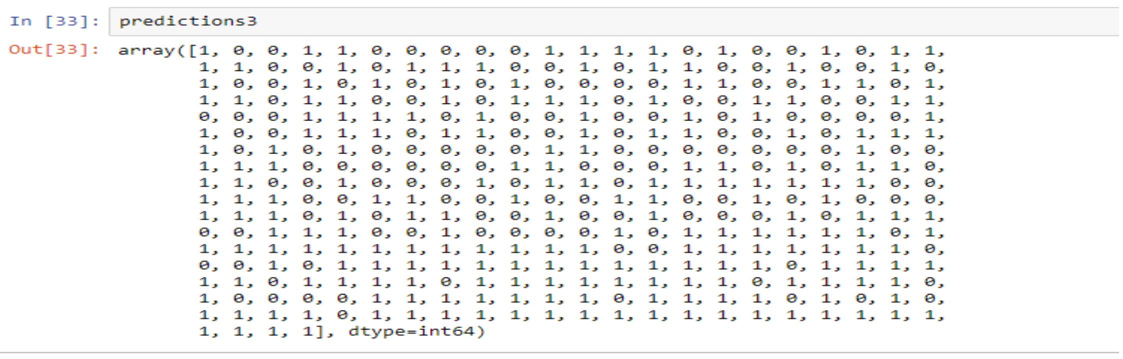
```
shape:  (3975, 584289)
[[159  27]
 [ 27 165]]
              precision    recall  f1-score   support

           0       0.85      0.85      0.85       186
           1       0.86      0.86      0.86       192

   micro avg       0.86      0.86      0.86       378
   macro avg       0.86      0.86      0.86       378
weighted avg       0.86      0.86      0.86       378

0.8571428571428571
```

Tri-gram model

Matrix formed with tri-gram model



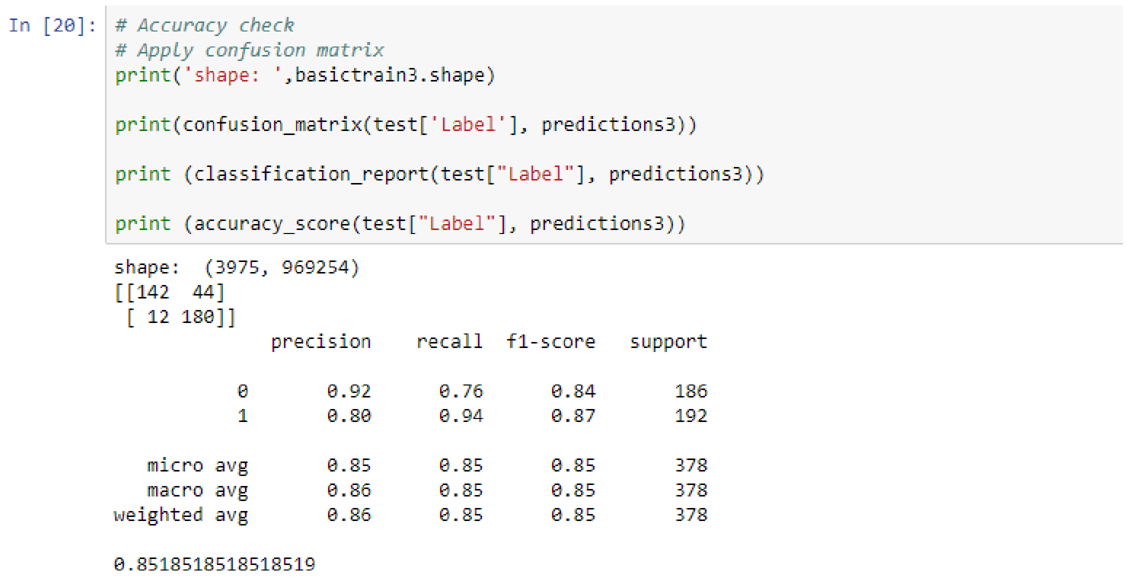
```
In [19]: # Apply the model when ngeam range is 3


         basicvectorizer3 = CountVectorizer(ngram_range=(3,3))
         basictrain3 = basicvectorizer3.fit_transform(headlines)
         # print(basictrain3.shape)

         basicmodel3 = LogisticRegression()
         basicmodel3 = basicmodel3.fit(basictrain3, train["Label"])

         basictest3 = basicvectorizer3.transform(testheadlines)
         predictions3 = basicmodel3.predict(basictest3)
```



```
In [33]: predictions3

Out[33]: array([1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1,
                1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0,
                1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1,
                1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1,
                0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1,
                1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1,
                1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
                1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0,
                1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0,
                1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0,
                1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1,
                0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0,
                0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
                1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0,
                1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0,
                1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1], dtype=int64)
```

Accuracy check for tri-gram model:

Shape of the matrix and accuracy with score are produced as in the output:



```
In [20]: # Accuracy check
         # Apply confusion matrix
         print('shape: ',basictrain3.shape)

         print(confusion_matrix(test['Label'], predictions3))

         print (classification_report(test["Label"], predictions3))

         print (accuracy_score(test["Label"], predictions3))

         shape:  (3975, 969254)
         [[142  44]
          [ 12 180]]
                       precision    recall  f1-score   support

                    0       0.92      0.76      0.84       186
                    1       0.80      0.94      0.87       192

            micro avg       0.85      0.85      0.85       378
            macro avg       0.86      0.85      0.85       378
         weighted avg       0.86      0.85      0.85       378

         0.8518518518518519
```

Naive Bayes

Principle on which it works is “every pair of features being classified is independent of each other”. In Gaussian Naive Bayes, continuous values associated with each feature are assumed to be distributed according to a Gaussian distribution.

A Gaussian distribution is also called Normal distribution. When plotted, it gives a bell shaped curve which is symmetric about the mean of the feature values as shown above.

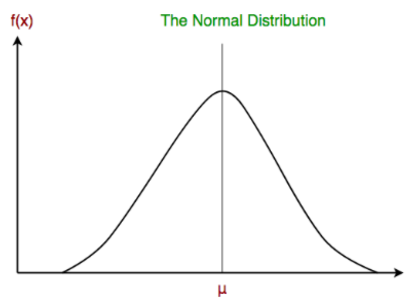


Figure 6 NB Function

Bayes’ Theorem

Explanation: Given a hypothesis H and evidence E, Bayes’ theorem states that the relationship between the P(H) before getting the evidence and P(H) after getting the evidence P(H|E) IS :

$$P(H|E) = \frac{P(H) * P(E|H)}{P(E)}$$

Probability of an event based on the prior knowledge of the conditions that might be related to the event.

P(H|E): Posterior- How probable is our H when observed E is given.(Not directly computable)

P(E|H): Likelihood- How probable is our E? Given that our E is true.

P(H): Prior- How probable was our H before observing the E?

P(E): Marginal/Evidence- How probable is the new E under all possible Hypothesis?

Gaussian Naïve Bayes

A Gaussian Naive Bayes algorithm is a special type of NB algorithm. It’s specifically used when the features have continuous values. It’s also assumed that all the features are following a Gaussian distribution i.e, normal distribution.

$$p(x = v \mid C_k) = \frac{1}{\sqrt{2\pi\sigma_k^2}} e^{-\frac{(v-\mu_k)^2}{2\sigma_k^2}}$$

x = continuous attribute

μ_k = mean of value in x associated with class C_k

σ_k^2 = variance of the values in x associated with class C_k

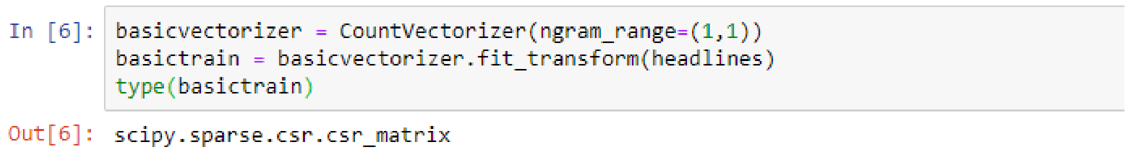
v = observed value

Implementation:

Initially import the module: ‘from sklearn.naive_bayes import GaussianNB’

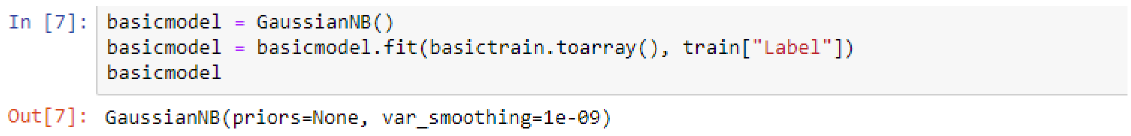
Rest of the steps are same as previous model generation.

Fit the vectorizer to the ‘headlines’



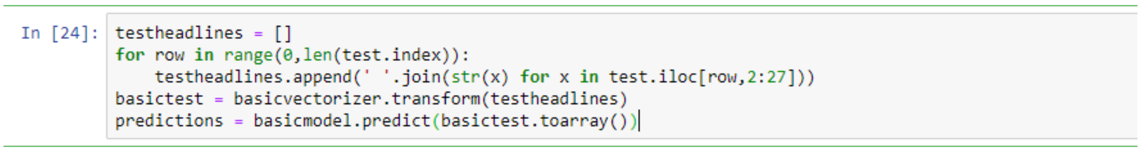
```
In [6]:  basicvectorizer = CountVectorizer(ngram_range=(1,1))
          basictrain = basicvectorizer.fit_transform(headlines)
          type(basictrain)

Out[6]:  scipy.sparse.csr.csr_matrix
```

Create the model of Gaussian Naïve Bayes and fit the model to the feature variable and the target variable.



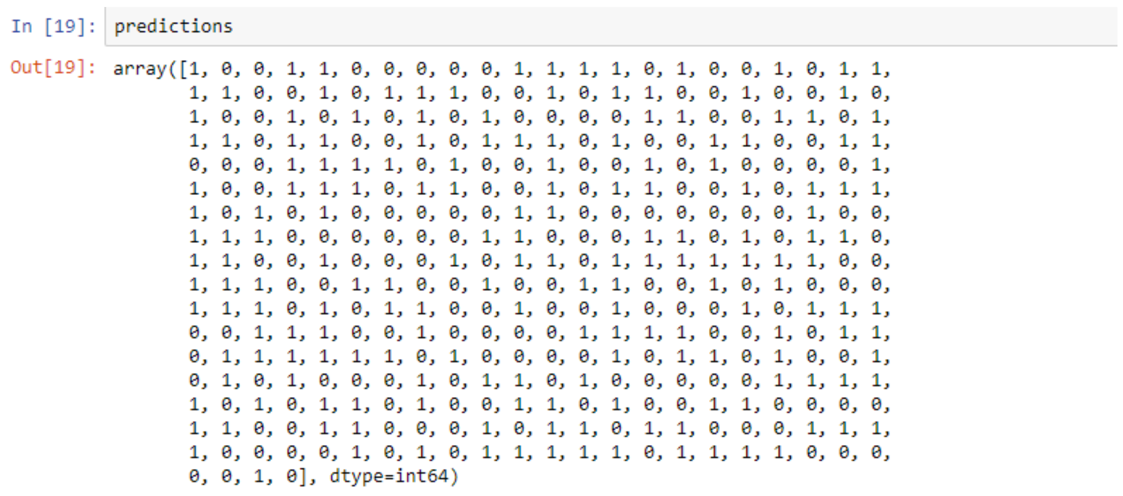
```
In [7]:  basicmodel = GaussianNB()
          basicmodel = basicmodel.fit(basictrain.toarray(), train["Label"])
          basicmodel

Out[7]:  GaussianNB(priors=None, var_smoothing=1e-09)
```

Predict the predictions for test data



```
In [24]:  testheadlines = []
           for row in range(0,len(test.index)):
               testheadlines.append(' '.join(str(x) for x in test.iloc[row,2:27]))
           basictest = basicvectorizer.transform(testheadlines)
           predictions = basicmodel.predict(basictest.toarray())
```

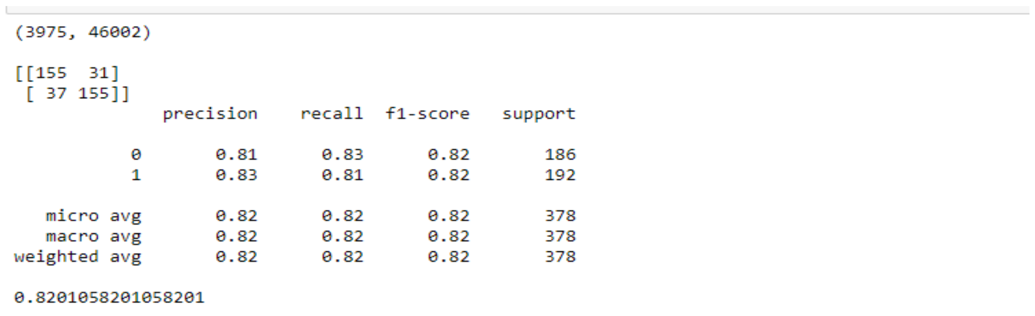
Predictions predicted are given below:



```
In [19]: predictions

Out[19]: array([1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1,
                1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0,
                1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1,
                1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1,
                0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1,
                1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1,
                1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
                1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0,
                1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0,
                1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0,
                1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1,
                0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1,
                0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1,
                0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1,
                1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0,
                1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1,
                1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0,
                0, 0, 1, 0], dtype=int64)
```

Accuracy check:

Use accuracy_score, confusion matrix, classification_report and f1_score for various outputs for accuracy check.



```
(3975, 46002)

[[155  31]
 [ 37 155]]
              precision    recall  f1-score   support

           0       0.81      0.83      0.82       186
           1       0.83      0.81      0.82       192

   micro avg       0.82      0.82      0.82       378
   macro avg       0.82      0.82      0.82       378
weighted avg       0.82      0.82      0.82       378

0.8201058201058201
```

Random Forest Tree

Random Forest is a supervised learning algorithm. Like its name, it creates a forest and makes it somehow random. The “forest” it builds, is an ensemble of Decision Trees, most of the time trained with the “bagging” method. The general idea of the bagging method is that a combination of learning models increases the overall result.

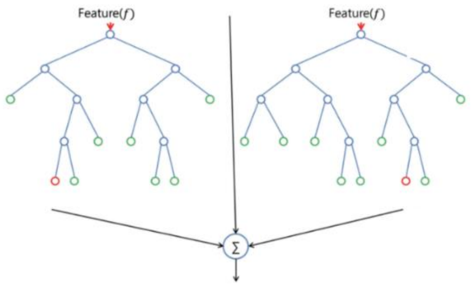


Figure 7 Random Forest Tree

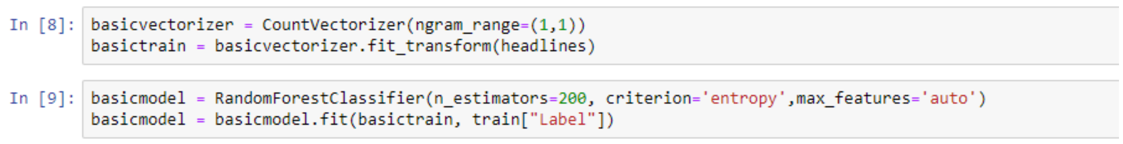
Implementation

Initially import the module: ‘from sklearn.ensemble import RandomForestClassifier’

Rest of the steps are same as previous model generation.

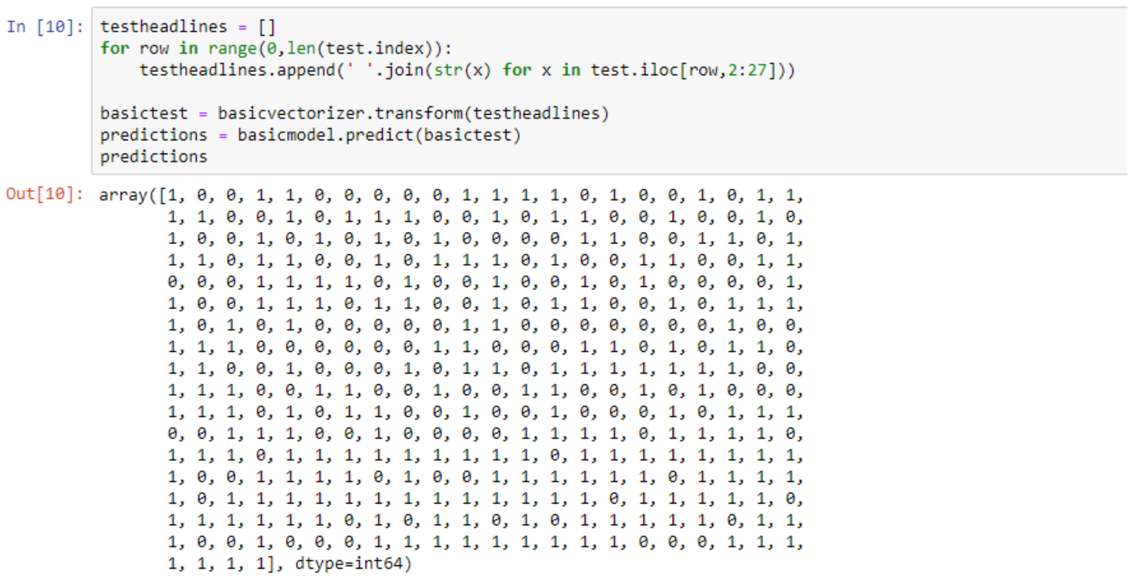
1-Gram Model

Fit the vectorizer to the ‘headlines’



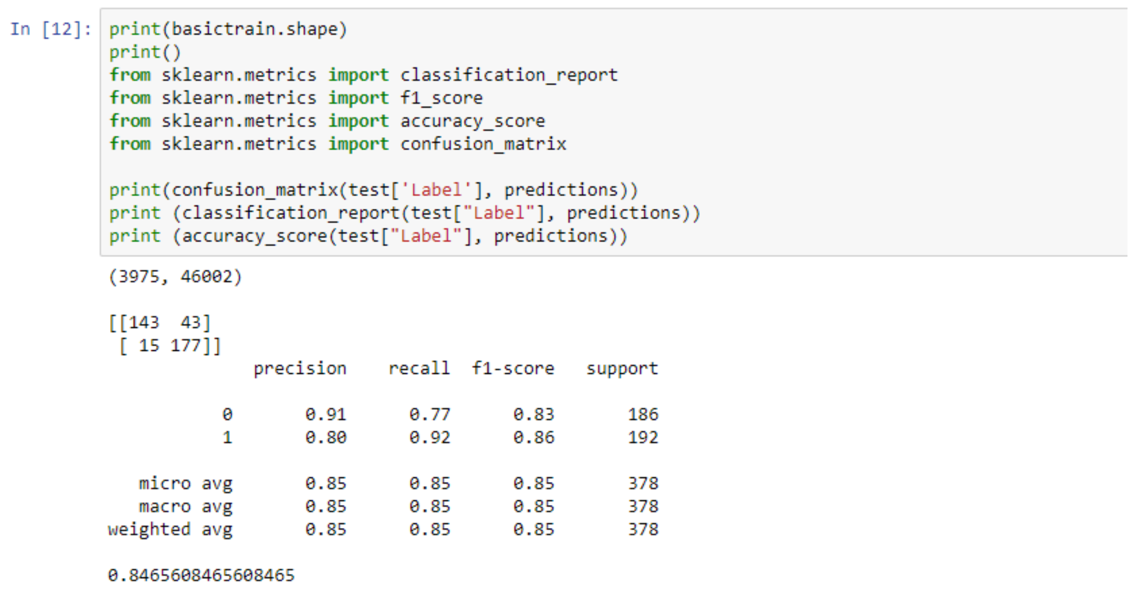
```
In [8]: basicvectorizer = CountVectorizer(ngram_range=(1,1))
        basictrain = basicvectorizer.fit_transform(headlines)

In [9]: basicmodel = RandomForestClassifier(n_estimators=200, criterion='entropy',max_features='auto')
        basicmodel = basicmodel.fit(basictrain, train["Label"])
```

Make the predictions:



```
In [10]: testheadlines = []
         for row in range(0,len(test.index)):
             testheadlines.append(' '.join(str(x) for x in test.iloc[row,2:27]))

         basictest = basicvectorizer.transform(testheadlines)
         predictions = basicmodel.predict(basictest)
         predictions

Out[10]: array([1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1,
                1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0,
                1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1,
                1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1,
                0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1,
                1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1,
                1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
                1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0,
                1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0,
                1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0,
                1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1,
                0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0,
                1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
                1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0,
                1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1,
                1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1,
                1, 1, 1, 1], dtype=int64)
```

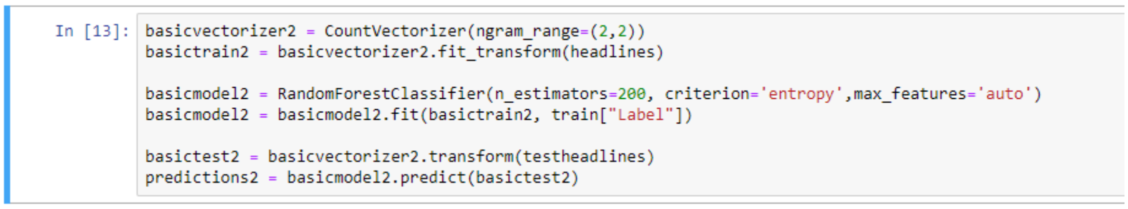
1-Gram Model Prediction Accuracy



```
In [12]: print(basictrain.shape)
         print()
         from sklearn.metrics import classification_report
         from sklearn.metrics import f1_score
         from sklearn.metrics import accuracy_score
         from sklearn.metrics import confusion_matrix

         print(confusion_matrix(test['Label'], predictions))
         print (classification_report(test["Label"], predictions))
         print (accuracy_score(test["Label"], predictions))

(3975, 46002)

[[143  43]
 [ 15 177]]
              precision    recall  f1-score   support

           0       0.91      0.77      0.83       186
           1       0.80      0.92      0.86       192

   micro avg       0.85      0.85      0.85       378
   macro avg       0.85      0.85      0.85       378
weighted avg       0.85      0.85      0.85       378

0.8465608465608465
```

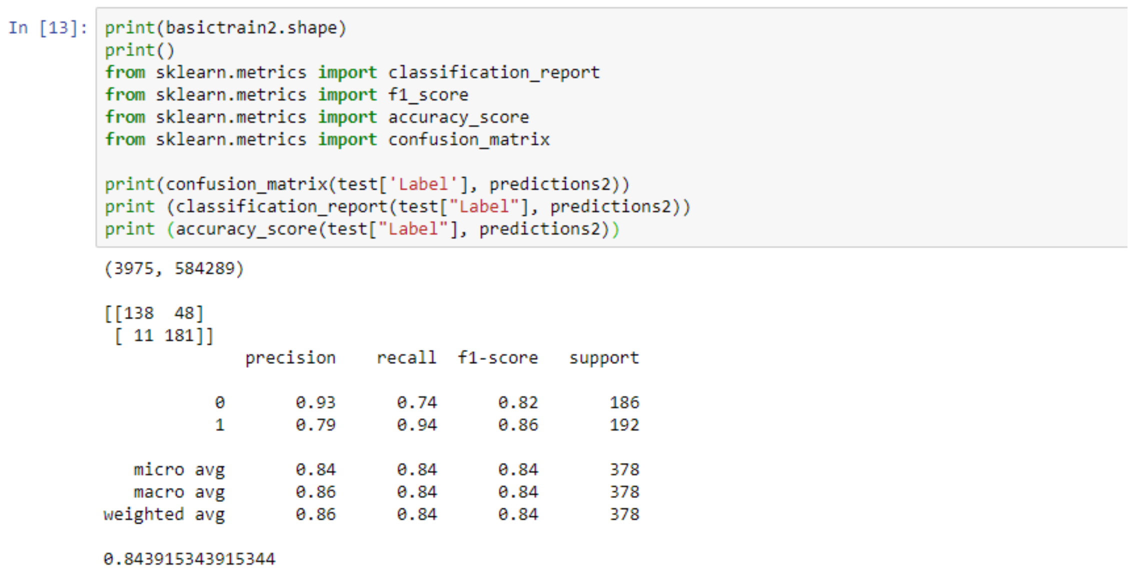
Bi-Gram Model

Fit the vectorizer to the ‘headlines’ and predict the predictions:



```
In [13]: basicvectorizer2 = CountVectorizer(ngram_range=(2,2))
         basictrain2 = basicvectorizer2.fit_transform(headlines)

         basicmodel2 = RandomForestClassifier(n_estimators=200, criterion='entropy',max_features='auto')
         basicmodel2 = basicmodel2.fit(basictrain2, train["Label"])

         basictest2 = basicvectorizer2.transform(testheadlines)
         predictions2 = basicmodel2.predict(basictest2)
```

Bi-Gram Model Prediction Accuracy



```
In [13]: print(basictrain2.shape)
         print()
         from sklearn.metrics import classification_report
         from sklearn.metrics import f1_score
         from sklearn.metrics import accuracy_score
         from sklearn.metrics import confusion_matrix

         print(confusion_matrix(test['Label'], predictions2))
         print (classification_report(test["Label"], predictions2))
         print (accuracy_score(test["Label"], predictions2))

(3975, 584289)

[[138  48]
 [ 11 181]]
              precision    recall  f1-score   support

           0       0.93      0.74      0.82       186
           1       0.79      0.94      0.86       192

   micro avg       0.84      0.84      0.84       378
   macro avg       0.86      0.84      0.84       378
weighted avg       0.86      0.84      0.84       378

0.843915343915344
```

Tri-Gram Model

Fit the vectorizer to the ‘headlines’ and predict the predictions:

```
In [14]: basicvectorizer3 = CountVectorizer(ngram_range=(3,3))
         basictrain3 = basicvectorizer3.fit_transform(headlines)

         basicmodel3 = RandomForestClassifier(n_estimators=200, criterion='entropy',max_features='auto')
         basicmodel3 = basicmodel3.fit(basictrain3, train["Label"])

         basictest3 = basicvectorizer3.transform(testheadlines)
         predictions3 = basicmodel3.predict(basictest3)
```

Tri-Gram Model Prediction Accuracy

```
In [15]: print(basictrain2.shape)
         print()
         from sklearn.metrics import classification_report
         from sklearn.metrics import f1_score
         from sklearn.metrics import accuracy_score
         from sklearn.metrics import confusion_matrix

         print(confusion_matrix(test['Label'], predictions3))
         print (classification_report(test["Label"], predictions3))
         print (accuracy_score(test["Label"], predictions3))
```

```
(3975, 584289)

[[131  55]
 [  0 192]]
              precision    recall  f1-score   support

           0       1.00      0.70      0.83       186
           1       0.78      1.00      0.87       192

   micro avg       0.85      0.85      0.85       378
   macro avg       0.89      0.85      0.85       378
weighted avg       0.89      0.85      0.85       378

0.8544973544973545
```

Support Vector Machine

Support Vectors are simply the co-ordinates of individual observation. Support Vector Machine is a frontier which best segregates the two classes (hyper-plane/ line).

Identify the right hyper-plane (Scenario-1): Here, we have three hyper-planes (A, B and C). Now, identify the right hyper-plane to classify star and circle.

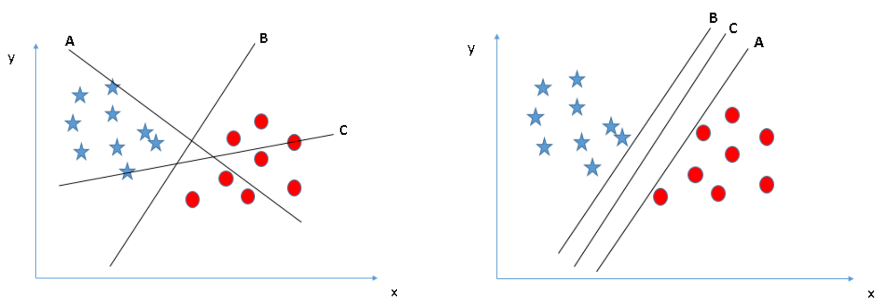


Figure 8 Scenario-1 (a) and Scenario-2 (b)

There is need to remember a thumb rule to identify the right hyper-plane: “Select the hyper-plane which segregates the two classes better”. In this scenario, hyper-plane “B” has excellently performed this job.

Identify the right hyper-plane (Scenario-2): Here, we have three hyper-planes (A, B and C) and all are segregating the classes well. Now, how can we identify the right hyper-plane?

Here, maximizing the distances between nearest data point (either class) and hyper-plane will help us to decide the right hyper-plane. This distance is called as **Margin**. Let’s look at the below snapshot:

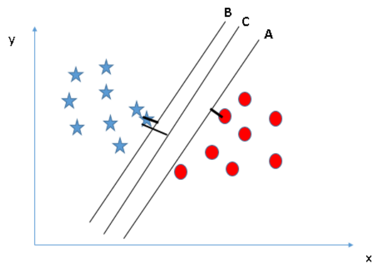


Figure 9 Deep through margin

Above, one can see that the margin for hyper-plane C is high as compared to both A and B. Hence, we name the right hyper-plane as C. Another lightning reason for selecting the hyper-plane with higher margin is robustness. If we select a hyper-plane having low margin then there is high chance of miss-classification.

Identify the right hyper-plane (Scenario-3): Use the rules as discussed in previous section to identify the right hyper-plane

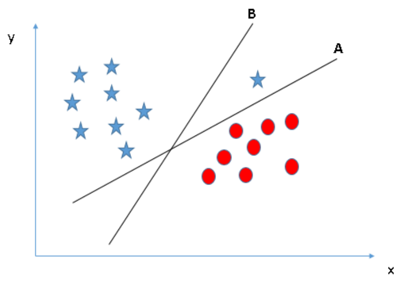


Figure 10 Scenario-3

Some of one may have selected the hyper-plane B as it has higher margin compared to A. But, here is the catch, SVM selects the hyper-plane which classifies the classes accurately prior to maximizing margin. Here, hyper-plane B has a classification error and A has classified all correctly. Therefore, the right hyper-plane is A.

Case of Outlier

Scenario-4: Below, I am unable to segregate the two classes using a straight line, as one of star lies in the territory of other (circle) class as an outlier.

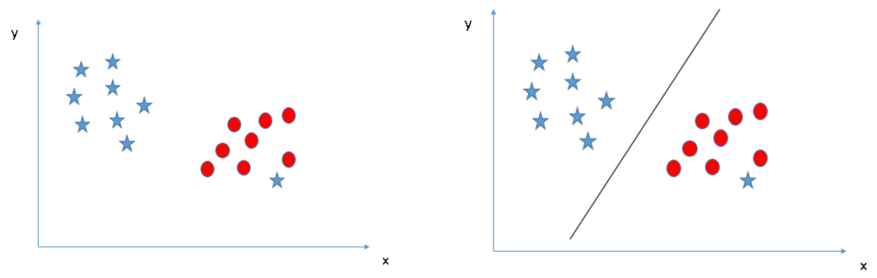


Figure 11 Data contains outlier (a) and Classification of data (b)

As I have already mentioned, one star at other end is like an outlier for star class. SVM has a feature to ignore outliers and find the hyper-plane that has maximum margin. Hence, we can say, SVM is robust to outliers.

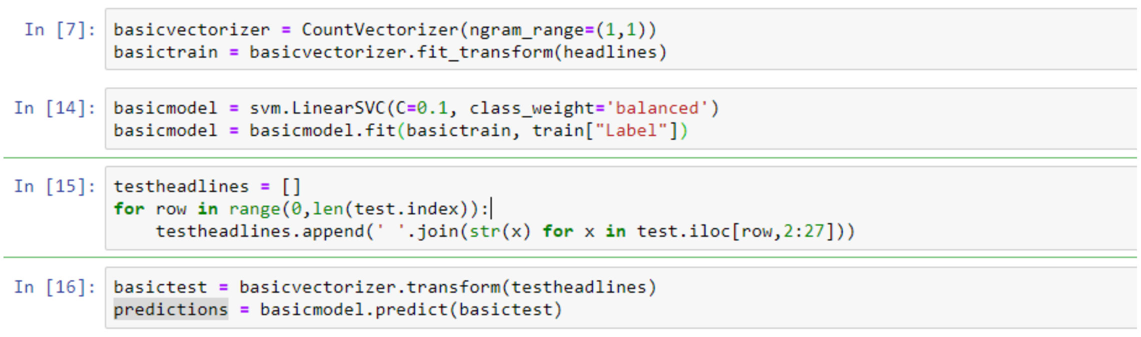
Implementation

Initially import the module: ‘from sklearn.svm import SVC, LinearSVC’

Rest of the steps are same as previous model generation.

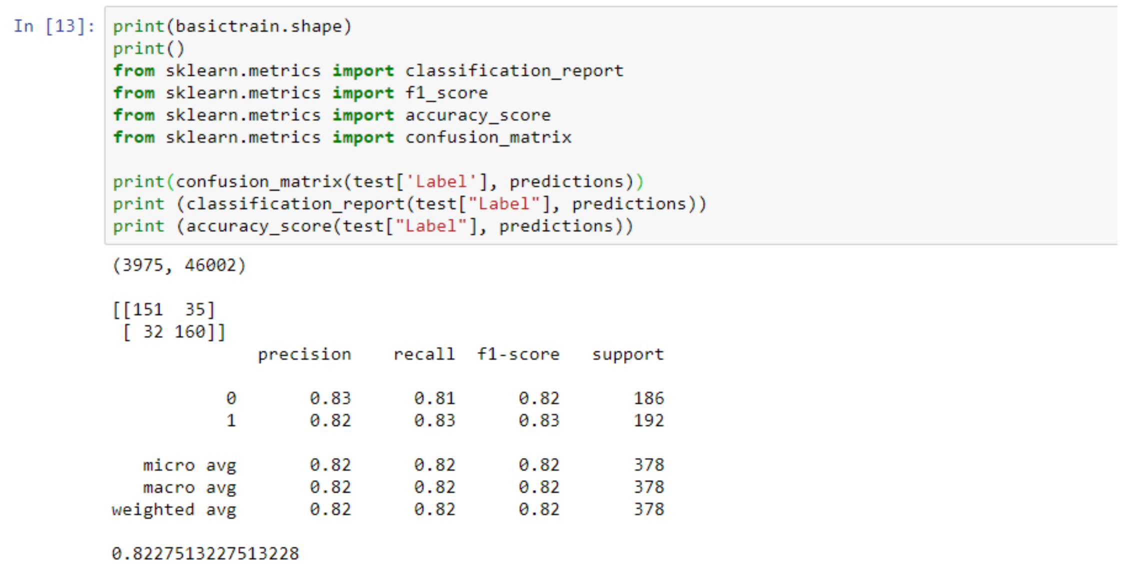
1-Gram Model

Fit the vectorizer to the ‘headlines’ and Predict the predictions:



```
In [7]:  basicvectorizer = CountVectorizer(ngram_range=(1,1))
          basictrain = basicvectorizer.fit_transform(headlines)

In [14]:  basicmodel = svm.LinearSVC(C=0.1, class_weight='balanced')
          basicmodel = basicmodel.fit(basictrain, train["Label"])

In [15]:  testheadlines = []
          for row in range(0,len(test.index)):
              testheadlines.append(' '.join(str(x) for x in test.iloc[row,2:27]))

In [16]:  basictest = basicvectorizer.transform(testheadlines)
          predictions = basicmodel.predict(basictest)
```

1-Gram Model Prediction Accuracy

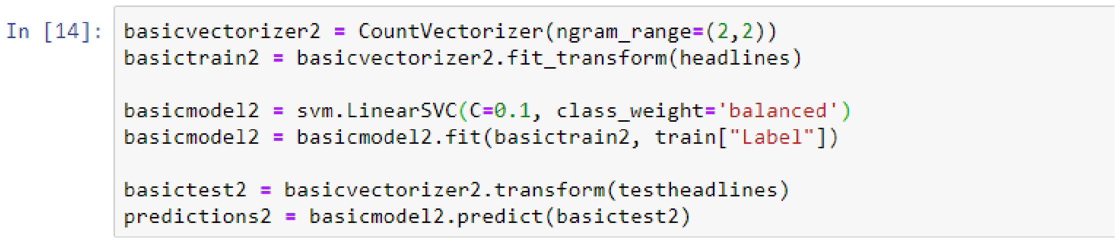


```
In [13]: print(basictrain.shape)
         print()
         from sklearn.metrics import classification_report
         from sklearn.metrics import f1_score
         from sklearn.metrics import accuracy_score
         from sklearn.metrics import confusion_matrix

         print(confusion_matrix(test['Label'], predictions))
         print (classification_report(test["Label"], predictions))
         print (accuracy_score(test["Label"], predictions))
```

```
(3975, 46002)

[[151  35]
 [ 32 160]]
              precision    recall  f1-score   support

           0       0.83      0.81      0.82       186
           1       0.82      0.83      0.83       192

   micro avg       0.82      0.82      0.82       378
   macro avg       0.82      0.82      0.82       378
weighted avg       0.82      0.82      0.82       378

0.8227513227513228
```

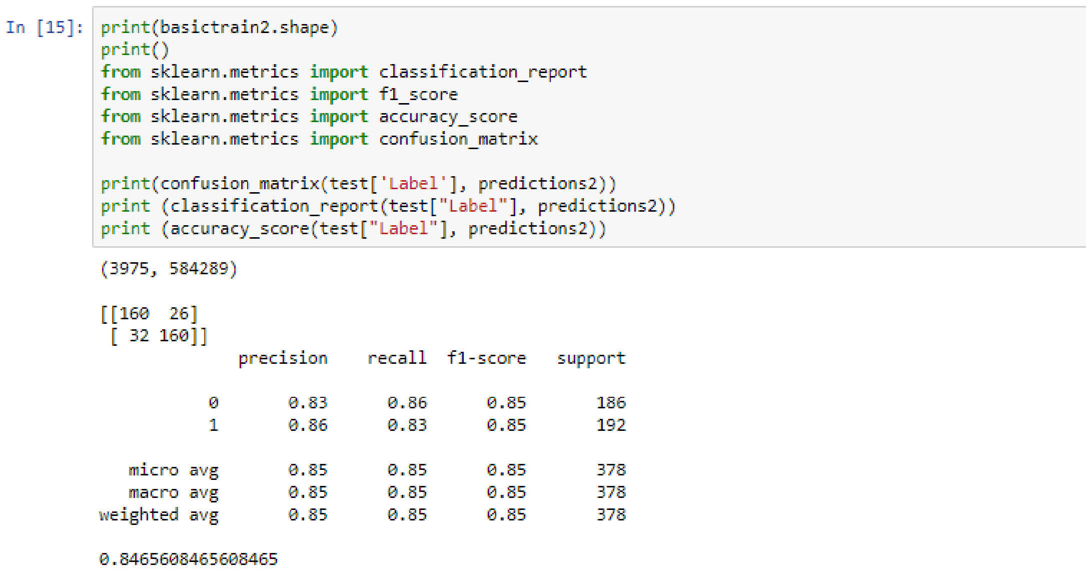
Bi-Gram Model

Fit the vectorizer to the ‘headlines’ and Predict the predictions:



```
In [14]: basicvectorizer2 = CountVectorizer(ngram_range=(2,2))
         basictrain2 = basicvectorizer2.fit_transform(headlines)

         basicmodel2 = svm.LinearSVC(C=0.1, class_weight='balanced')
         basicmodel2 = basicmodel2.fit(basictrain2, train["Label"])

         basictest2 = basicvectorizer2.transform(testheadlines)
         predictions2 = basicmodel2.predict(basictest2)
```

Bi-Gram Model Prediction Accuracy



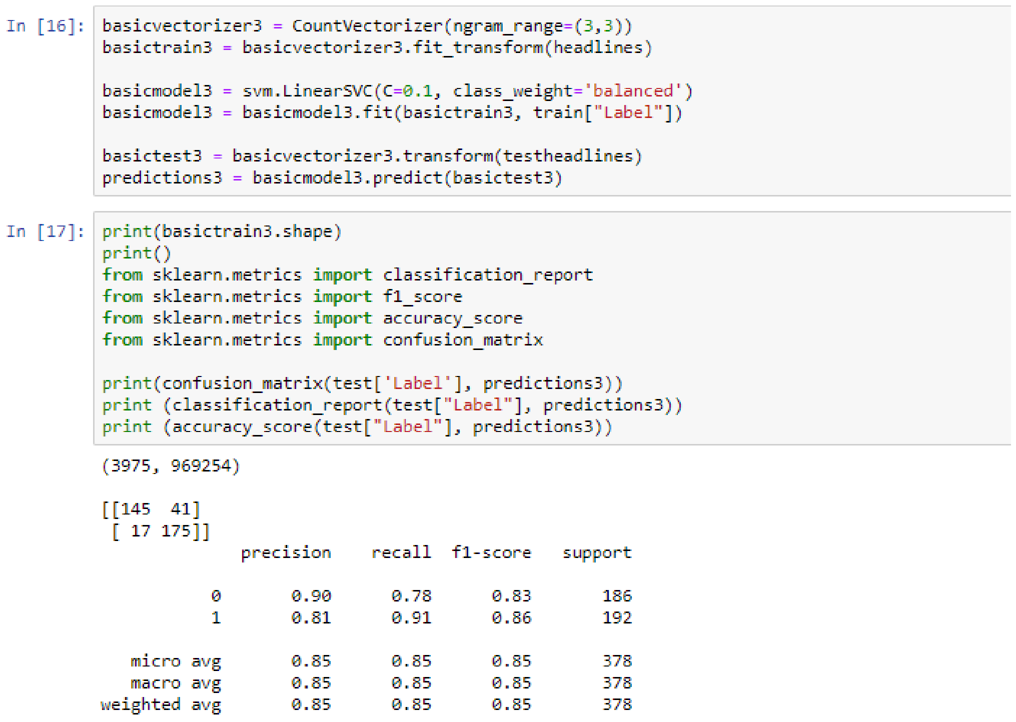
```
In [15]: print(basictrain2.shape)
         print()
         from sklearn.metrics import classification_report
         from sklearn.metrics import f1_score
         from sklearn.metrics import accuracy_score
         from sklearn.metrics import confusion_matrix

         print(confusion_matrix(test['Label'], predictions2))
         print (classification_report(test["Label"], predictions2))
         print (accuracy_score(test["Label"], predictions2))
```

```
(3975, 584289)

[[160  26]
 [ 32 160]]
              precision    recall  f1-score   support

           0       0.83      0.86      0.85       186
           1       0.86      0.83      0.85       192

   micro avg       0.85      0.85      0.85       378
   macro avg       0.85      0.85      0.85       378
weighted avg       0.85      0.85      0.85       378

0.8465608465608465
```

Tri-Gram Model

Fit the vectorizer to the ‘headlines’ and Predict the predictions and finally given is

Tri-Gram Model Prediction Accuracy



```
In [16]: basicvectorizer3 = CountVectorizer(ngram_range=(3,3))
         basictrain3 = basicvectorizer3.fit_transform(headlines)

         basicmodel3 = svm.LinearSVC(C=0.1, class_weight='balanced')
         basicmodel3 = basicmodel3.fit(basictrain3, train["Label"])

         basictest3 = basicvectorizer3.transform(testheadlines)
         predictions3 = basicmodel3.predict(basictest3)
```

```
In [17]: print(basictrain3.shape)
         print()
         from sklearn.metrics import classification_report
         from sklearn.metrics import f1_score
         from sklearn.metrics import accuracy_score
         from sklearn.metrics import confusion_matrix

         print(confusion_matrix(test['Label'], predictions3))
         print (classification_report(test["Label"], predictions3))
         print (accuracy_score(test["Label"], predictions3))
```

```
(3975, 969254)

[[145  41]
 [ 17 175]]
              precision    recall  f1-score   support

           0       0.90      0.78      0.83       186
           1       0.81      0.91      0.86       192

   micro avg       0.85      0.85      0.85       378
   macro avg       0.85      0.85      0.85       378
weighted avg       0.85      0.85      0.85       378
```

Non-linear/rbf SVM

Find the hyper-plane to segregate to classes (Scenario-5): In the scenario below, we can’t have linear hyper-plane between the two classes, so how does SVM classify these two classes? Till now, we have only looked at the linear hyper-plane.

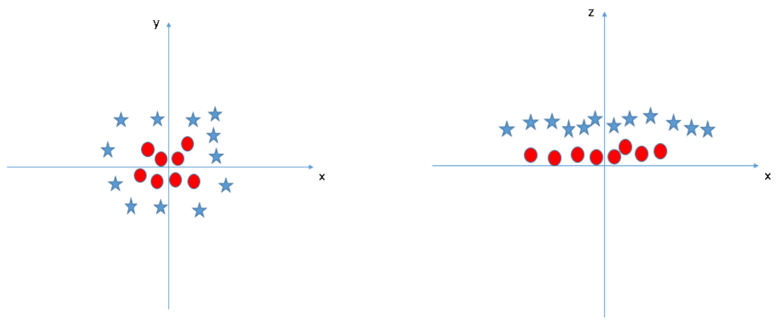


Figure 12 Scenario-5 (a) and Classification (b)

SVM can solve this problem. Easily! It solves this problem by introducing additional feature. Here, we will add a new feature z=x^2+y^2. Now, let’s plot the data points on axis x and z:

In above plot, points to consider are:

All values for z would be positive always because z is the squared sum of both x and y

In the original plot, red circles appear close to the origin of x and y axes, leading to lower value of z and star relatively away from the origin result to higher value of z.

In SVM, it is easy to have a linear hyper-plane between these two classes. But, another burning question which arises is, should we need to add this feature manually to have a hyper-plane. No, SVM has a technique called the kernel trick. These are functions which takes low dimensional input space and transform it to a higher dimensional space i.e. it converts not separable problem to separable problem, these functions are called kernels. It is mostly useful in non-linear separation problem. Simply put, it does some extremely complex data transformations, then find out the process to separate the data based on the labels or outputs is defined.

When we look at the hyper-plane in original input space it looks like a circle:

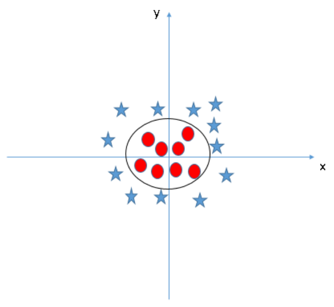


Figure 13 In original input space (Back to transformation)

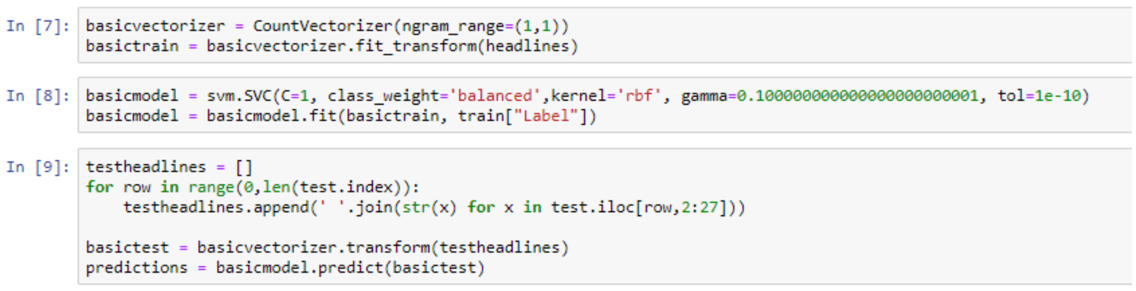
Implementation

Initially import the module: ‘from sklearn.svm import SVC’

Rest of the steps are same as previous model generation.

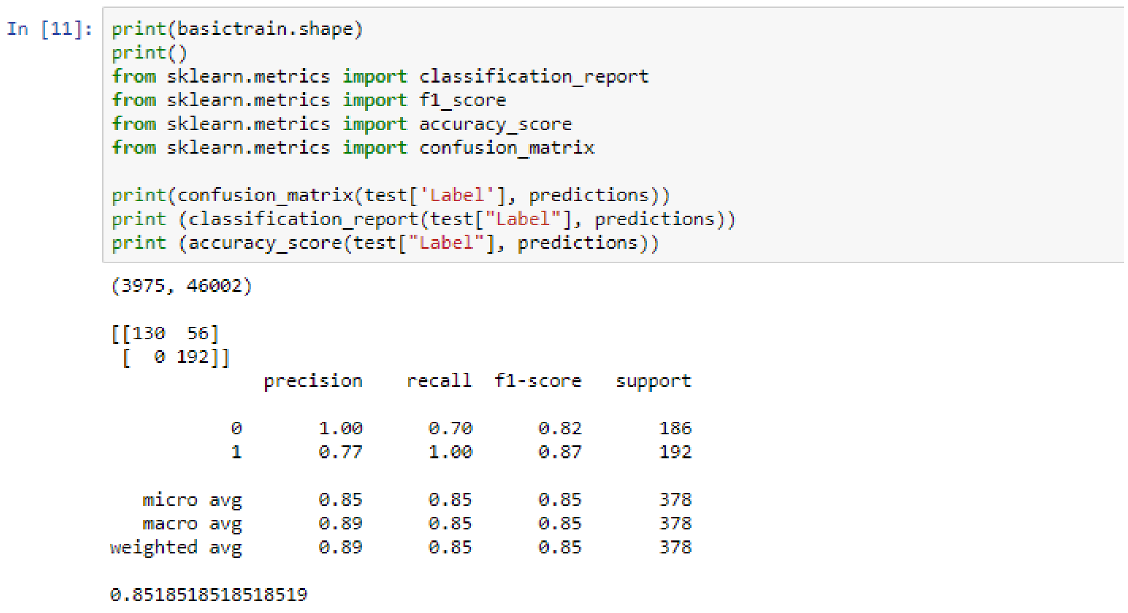
1-Gram Model

Fit the vectorizer to the ‘headlines’ and Predict the predictions:



```
In [7]: basicvectorizer = CountVectorizer(ngram_range=(1,1))
        basictrain = basicvectorizer.fit_transform(headlines)

In [8]: basicmodel = svm.SVC(C=1, class_weight='balanced',kernel='rbf', gamma=0.10000000000000000000001, tol=1e-10)
        basicmodel = basicmodel.fit(basictrain, train["Label"])

In [9]: testheadlines = []
        for row in range(0,len(test.index)):
            testheadlines.append(' '.join(str(x) for x in test.iloc[row,2:27]))

        basictest = basicvectorizer.transform(testheadlines)
        predictions = basicmodel.predict(basictest)
```

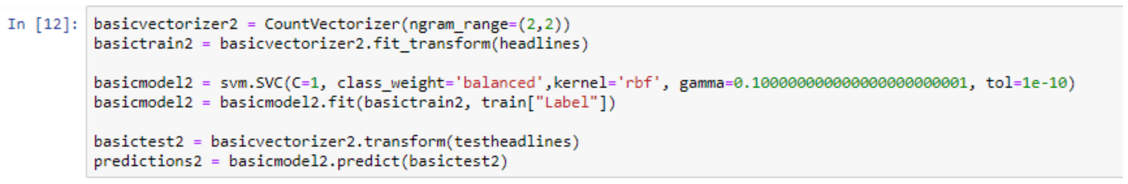
1-Gram Model Prediction Accuracy



```
In [11]: print(basictrain.shape)
         print()
         from sklearn.metrics import classification_report
         from sklearn.metrics import f1_score
         from sklearn.metrics import accuracy_score
         from sklearn.metrics import confusion_matrix

         print(confusion_matrix(test['Label'], predictions))
         print (classification_report(test["Label"], predictions))
         print (accuracy_score(test["Label"], predictions))
```

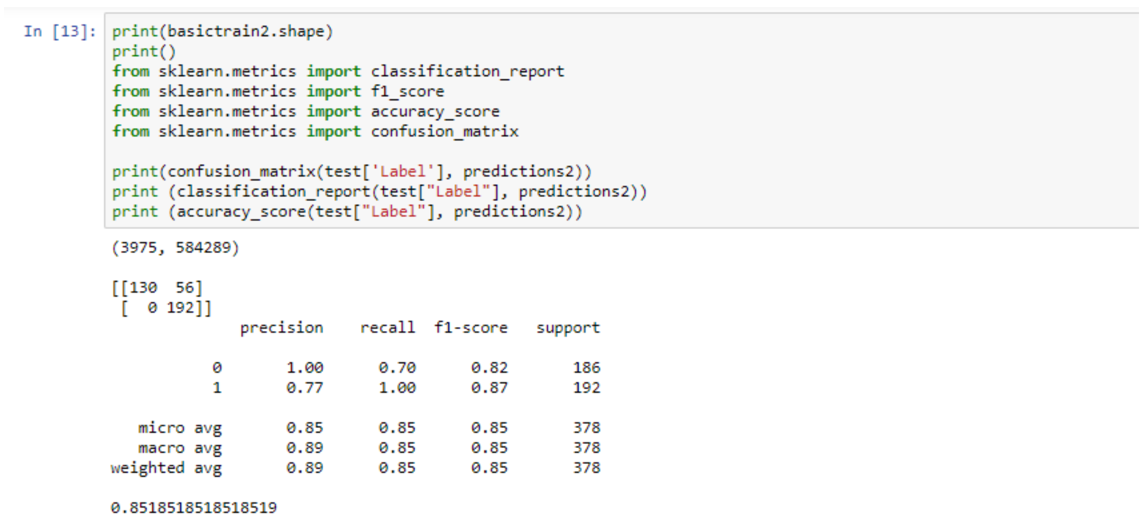
```
(3975, 46002)

[[130  56]
 [  0 192]]
              precision    recall  f1-score   support

           0       1.00      0.70      0.82       186
           1       0.77      1.00      0.87       192

   micro avg       0.85      0.85      0.85       378
   macro avg       0.89      0.85      0.85       378
weighted avg       0.89      0.85      0.85       378

0.8518518518518519
```

Fit the vectorizer to the ‘headlines’ and Predict the predictions:



```
In [12]: basicvectorizer2 = CountVectorizer(ngram_range=(2,2))
         basictrain2 = basicvectorizer2.fit_transform(headlines)

         basicmodel2 = svm.SVC(C=1, class_weight='balanced',kernel='rbf', gamma=0.10000000000000000000001, tol=1e-10)
         basicmodel2 = basicmodel2.fit(basictrain2, train["Label"])

         basictest2 = basicvectorizer2.transform(testheadlines)
         predictions2 = basicmodel2.predict(basictest2)
```

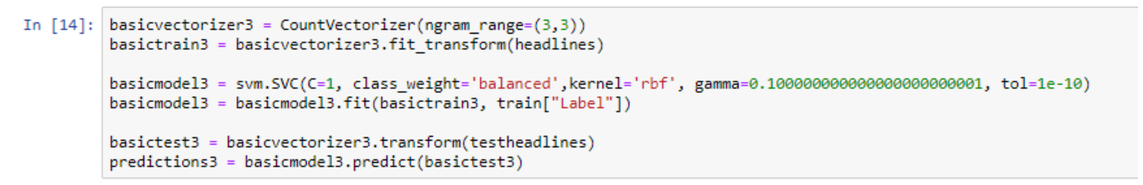
Bi-Gram Model Prediction Accuracy



```
In [13]: print(basictrain2.shape)
         print()
         from sklearn.metrics import classification_report
         from sklearn.metrics import f1_score
         from sklearn.metrics import accuracy_score
         from sklearn.metrics import confusion_matrix

         print(confusion_matrix(test['Label'], predictions2))
         print (classification_report(test["Label"], predictions2))
         print (accuracy_score(test["Label"], predictions2))
```

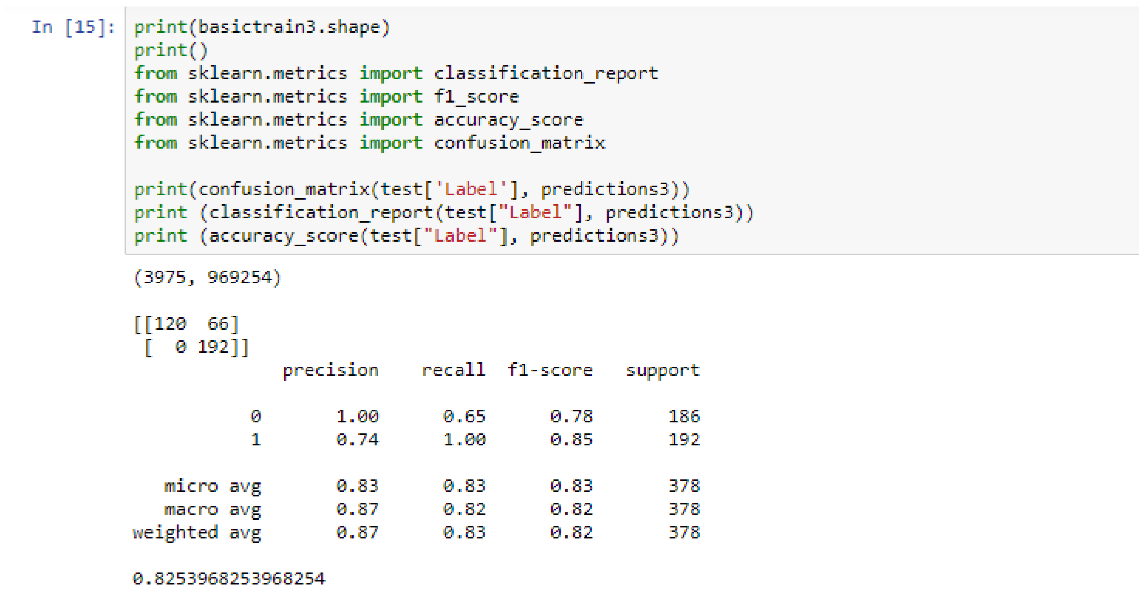
```
(3975, 584289)

[[130  56]
 [  0 192]]
              precision    recall  f1-score   support

           0       1.00      0.70      0.82       186
           1       0.77      1.00      0.87       192

   micro avg       0.85      0.85      0.85       378
   macro avg       0.89      0.85      0.85       378
weighted avg       0.89      0.85      0.85       378

0.8518518518518519
```

Fit the vectorizer to the ‘headlines’ and Predict the predictions:



```
In [14]: basicvectorizer3 = CountVectorizer(ngram_range=(3,3))
         basictrain3 = basicvectorizer3.fit_transform(headlines)

         basicmodel3 = svm.SVC(C=1, class_weight='balanced',kernel='rbf', gamma=0.1000000000000000000000001, tol=1e-10)
         basicmodel3 = basicmodel3.fit(basictrain3, train["Label"])

         basictest3 = basicvectorizer3.transform(testheadlines)
         predictions3 = basicmodel3.predict(basictest3)
```

Tri-Gram Model Prediction Accuracy



```
In [15]: print(basictrain3.shape)
         print()
         from sklearn.metrics import classification_report
         from sklearn.metrics import f1_score
         from sklearn.metrics import accuracy_score
         from sklearn.metrics import confusion_matrix

         print(confusion_matrix(test['Label'], predictions3))
         print (classification_report(test["Label"], predictions3))
         print (accuracy_score(test["Label"], predictions3))
```

```
(3975, 969254)

[[120  66]
 [  0 192]]
              precision    recall  f1-score   support

           0       1.00      0.65      0.78       186
           1       0.74      1.00      0.85       192

   micro avg       0.83      0.83      0.83       378
   macro avg       0.87      0.82      0.82       378
weighted avg       0.87      0.83      0.82       378

0.8253968253968254
```

CHAPTER 5
CONCLUSION & FUTURE SCOPE

5.1 COMPARISON OF MODEL PERFORMANCE USING ACCURACY OF PREDICTION

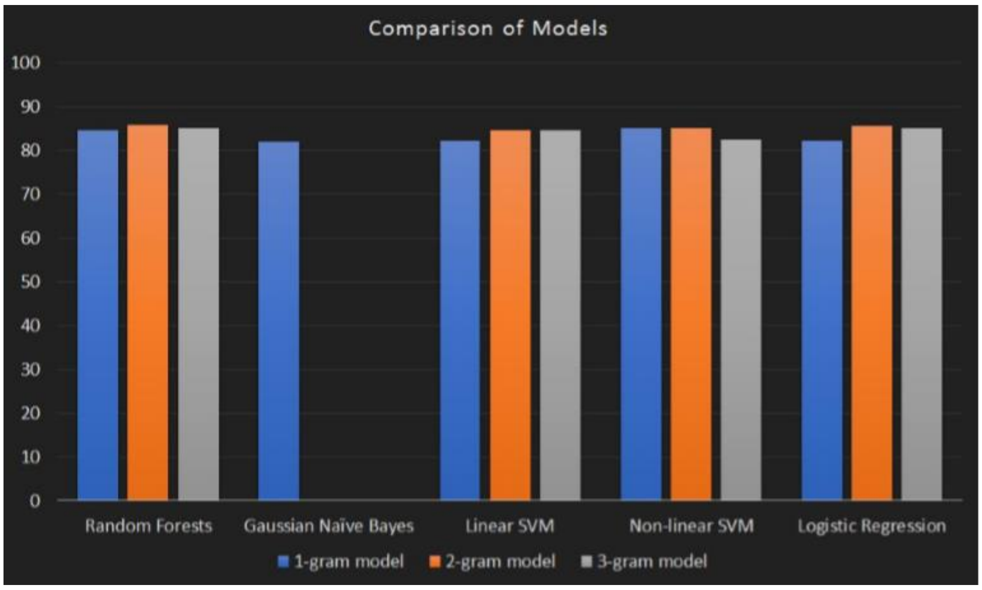


Figure 14 Accuracy on scale (0-100%)

Logistic Regression had highest accuracy on the Bi-gram model as shown in the chart. The prediction accuracy was 85.71%. Using Natural Language Processing techniques, we were able to accurately predict the stock market trends 85% of the time.

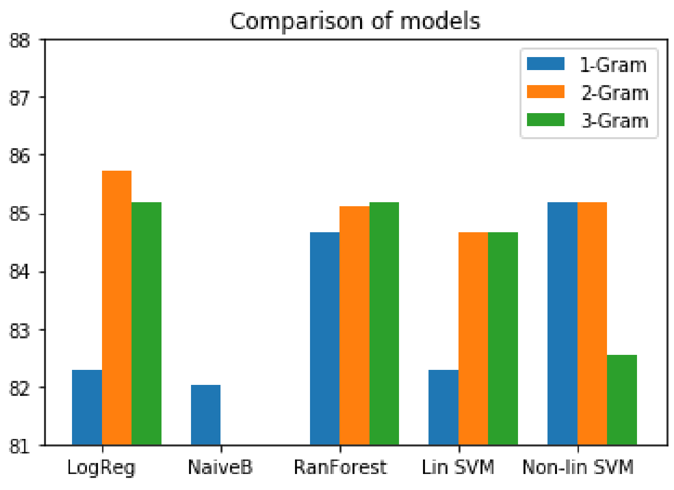


Figure 15 Accuracy on scale (81-88%)

5.2 CONCLUSION

Social Media can sometimes be deceiving when delivering the right frame of speech. Here we have used twitter feed and news articles around the web that has influenced the stock market of a company. Through this project, it helped us understand the basics of Natural Language Processing. Even though one can’t bet one’s money on the stock from this project, this work can be treated as solid understanding of the basics of NLP. Using the same model for different text data is also feasible. It was interesting to know about how to go from text data to vectors of numbers and applying ML techniques that can help to influence the stock market of a company. It helped us a gain wider sense of the power of NLP in various applications. From reading about machine learning models in class to implement them with real data and observe the performance of a model, tuning the parameters, performing exploratory data analysis set a great learning curve for future projects.

We also went from using an available data set to scraping our own data which made this project a little more interesting and challenging. To get more insights on which model to use and how to construct them, we learnt by reading through research papers and usage of **scikitlearn** to build our models. As any project that does not take a straight path towards completion we hit certain roadblocks while implementing this project.

Road block 1

As any machine learning task is concerned the data in consideration was restrictive and had few data cleaning to be done. Firstly the data consisted of a character “b” appended to text in multiple ways and was a little challenging to remove it across the entire dataset.

Road Block 2

It was also challenging to find more data. The dataset available to use was from 2008 to 2016. We had to scrape it from another source (Yahoo News) completely and put in the format that we wanted to work for our Machine learning model. It was a challenging task to scrape it and wrangle it to the data set that we wanted to (25 top headlines and labels associated with them).

Road Block 3

While we put the dataset for our SVM model, there were quite a few errors it kept throwing and one of them being NaN values and the number of cores we used to train our model. We used all the 4 cores to run our algorithm in parallel for faster execution.

5.3 FUTURE WORK

This project leaves room for future work and ways to accomplish them:

1. The number of features used in the dataset can be expanded. Right now we have gathered the top 25 News headlines, it is important to have more features that help the model learn better.
2. We are looking at the stock of one company. We can expand it to work for multiple companies at once and we can also include real time- time series analysis.
3. Perform multi class classification for various parameters of stock trading.

REFERENCES