Machine Learning for Detection of Fake News

By

Md. Aleem Nawaz Akhter

Submitted To Project Leader of AnalyticsJobs

Acknowledgment

I express my gratitude to the AnalyticsJobs and Project Team leader, for giving me an opportunity to work with them and make the best outcomes on this project (Fake news Project).

I specially thank my institute, trainers, for constantly guiding and supporting me throughout the project period. My heartfelt gratitude also goes out to the staff and employees and specifically to Mrs. Deepika Sharma, training head of DataTarined with me and guiding me.

In this project I am very much inspired from Mr. Krish Naik,

In this project the data is given by AnalyticsJobs.

Introduction

- ## Business Problem Framing

The authenticity of Information has become a longstanding issue affecting businesses and society, both for printed and digital media. On social networks, the reach and effects of information spread occur at such a fast pace and so amplified that distorted, inaccurate, or false information acquires a tremendous potential to cause real-world impacts, within minutes, for millions of users. Recently, several public concerns about this problem and some approaches to mitigate the problem were expressed.

- ## Conceptual Background of the Domain Problem

In this paper I experiment the possibility to detect fake news based only on textual information by applying traditional machine learning techniques on one different dataset that contain different kinds of news.

In order to work on fake news detection, it is important to understand what is fake news and how they are characterized. .

The first is characterization or what is fake news and the second is detection. In order to build detection models, it is need to start by characterization, indeed, it is need to understand what is fake news before trying to detect them.

- ## Review of Literature

In this chapter I have investigated how state-of-the-art **Natural language processing** (**NLP**) models work on fake news detection, and it shows that for the particular case of fake news detection it does not outperform traditional machine learning methods. I have also made some addition to the original model that improves the performances by a few percent by replacing the tuneable word embedding by constant one using **CountVectorizer**. It shows out that it helps reduce overfitting and increase result on the testing set.

- Motivation for the Problem Undertaken

News is important in daily life as it informs our view to the world and in response, we take actions and make choices in various aspects. Gradually, the tendency towards online news is increasing as it is more concise and is available at the finger tips. There has been large generation of deceptive content worldwide that has an effect on the formation of opinions, making decisions and voting trends. Most of the “fake news” is initially circulated through social media networks such as Twitter, Facebook, and then makes the way into mainstream media outlets such as Radio and TV. The fake news articles share linguistic features such as heavy use of quoted material. In this use case, the results of fake news detection test and the performance of fake news classifiers is discussed. The aim is to build a new fake news detector using classifiers like Logistic Regression, Decision tree classification, Multinomial Naive Bayes classification.

Analytical Problem Framing

- Mathematical/ Analytical Modelling of the Problem

Text pre-processing is an important activity in language processing tasks. It transforms text into a digestible form to enhance the working of machine learning algorithms. There are three main components of text pre-processing

a. Normalization
b. Noise removal

c. Tokenization
d. Stemming

- ## Data Sources and their formats

The source of Data set is given by AnalyticsJobs. This Data was in CSV(comma separated value) format. Data Set contain 6 columns and 20800 rows.

The columns names are as follows:

1. Unnamed :0 => This contains the serial no. Of the rows.
2. Id => this contains unique identification of tat news
3. Headlines => This contains the headlines of particular news
4. Written by => The auther of news
5. News => This contains the news
6. Label => this contains the value of news is it fake or not. 0 means “Real” and 1 means “Fake”.

- ## Data Pre-processing Done

1. Normalization :-Normalization aims to keep the data relatively in smaller range. In text pre-processing, Normalization attempts to place all texts on an equal footing. It converts all characters into either lowercase or uppercase.
2. Noise removal :- Noise removal is elimination of extra spaces, special characters, numbers from text and also removal of rows with null values.
3. Tokenization:- Tokenization is splitting paragraphs into sentences and splitting sentences into words.
4. Stemming:- The method of creating morphological variants of basic/root word is known as Stemming. The Stemming algorithm will reduce the words

words such as “chocolates”, “chocolatey”, “choco” to the base word, “ chocolate” and “retrieval”, “retrieved”, “retrieves” to the base “retrieve”

- Hardware and Software Requirements and Tools Used

Hardware: Dell inspiron 5000. 1TB HDD, 160GB SSD, 4GB RAM

Software: windows 10, Anaconda3.8, Jupiter Notebook

Libraries: Pandas, Nampy, matplotlib.pyplot, CountVectorizer, Tfidf,PorterStemer, Stopword, train_test_split, Confusion matrix, MultinomialNB, ClassificationReport, sklearn.svm, DecisionTreeClassifier, KNeighborsClassifier

Model/s Development and Evaluation

- Identification of possible problem-solving approaches (methods)

As i got the Data Set it was balanced data set and it was label based problem data set means there was only two types of results output (0: Real,1: Fake). So basically. The News contains a lot of information so for this i was gone for Contvectorizer(bag of words). Then tokenize, lemmatize and stemming with porterStemmer.

• Run and Evaluate selected models

The process of categorizing texts into organized groups is known as text classification or text tagging. Some of the classifiers that are used here are

1.Support vector Classifier

2. Decision Tree

3. Naïve Bayes

4. KNeighborsClassifier

A. Multinomial Naïve Bayes Classifier

Multinomial Naive Bayes Multinomial Naive Bayes is used when, in essence, data is discrete. The features / predictors the classifier uses are the frequency of the terms that appear in the text.

E.g.: Number of occurrences.

The mean of every word for a given class is determined to determine our likelihood.

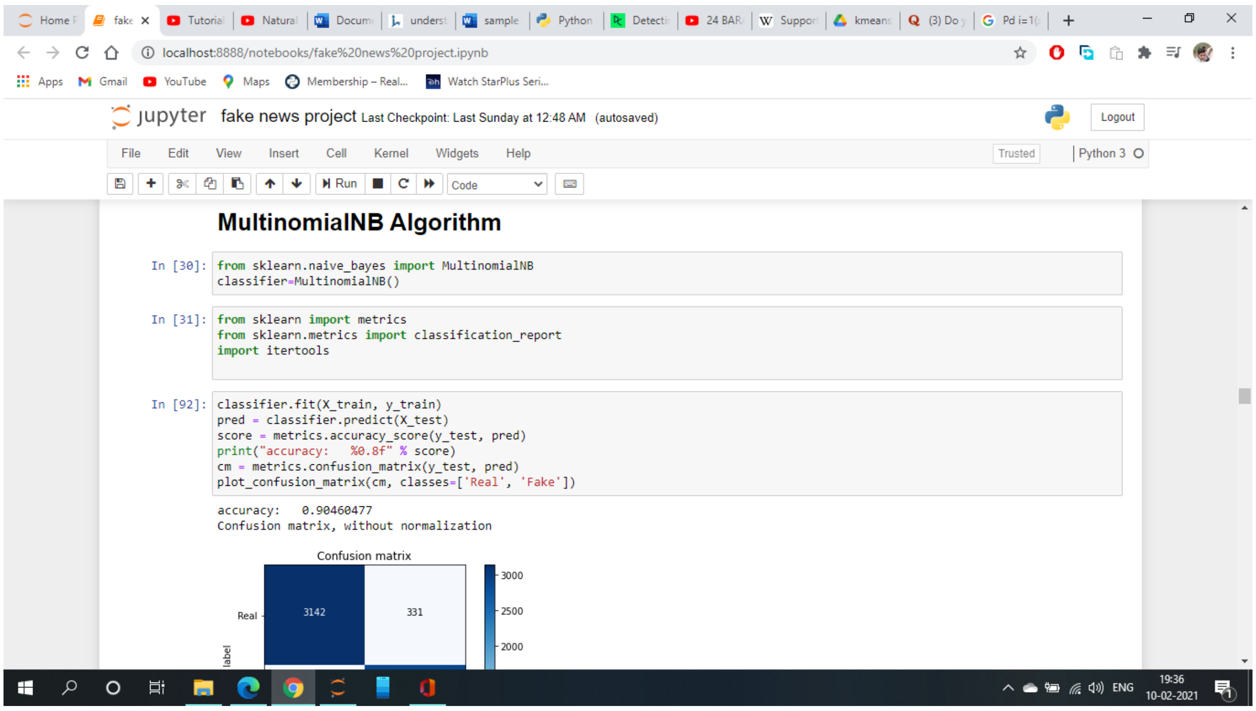
The average listed for word i and class j is:

$P(i \mid j) = word i\ j \,/\, word j$

Since there are 0 terms, however, Laplace Smoothing is done with a low α:

$P(i \mid j) = word i\ j + \alpha \,/\, word j + |V| + 1$

```
accuracy score =  0.9046047697615119

classification_report is

                    precision         recall    f1-score       support 0
0.91                0.90         0.91          34731                    0.90
         0.90          0.90          3194accuracy
0.90           6667

macro avg           0.90                0.90           0.90           6667

weighted avg        0.90                0.90           0.90           6667Metrices:

array([[3142, 331],        [ 305, 2889]], dtype=int64)
```



MultinomialNB Algorithm

```
In [30]: from sklearn.naive_bayes import MultinomialNB
         classifier=MultinomialNB()

In [31]: from sklearn import metrics
         from sklearn.metrics import classification_report
         import itertools

In [92]: classifier.fit(X_train, y_train)
         pred = classifier.predict(X_test)
         score = metrics.accuracy_score(y_test, pred)
         print("accuracy:   %0.8f" % score)
         cm = metrics.confusion_matrix(y_test, pred)
         plot_confusion_matrix(cm, classes=['Real', 'Fake'])

         accuracy:   0.90460477
         Confusion matrix, without normalization
```

B. Decision Tree Classification

Continuous splitting is known as the grouping of Decision tree according to certain parameter.

Decision Tree includes:

1. Nodes: Checking condition for a given attribute value.
2. Edges / branch: Edges match the result of a check and bind to the following node or leaf.
3. Leaf nodes: These are terminal nodes which forecast the result (represent class distribution).

Heuristic partitioning is used to construct the Decision Tree and the process is called recursive partitioning. This method is also described as dividing and conquering as it divides the data into sub-sets, that are instead divided recursively into smaller sub-sets, and so on and so forth until the cycle halts whenever the algorithm decides that the data inside the sub-sets is homogeneous enough or that any other stop requirement has been satisfied

Algorithm

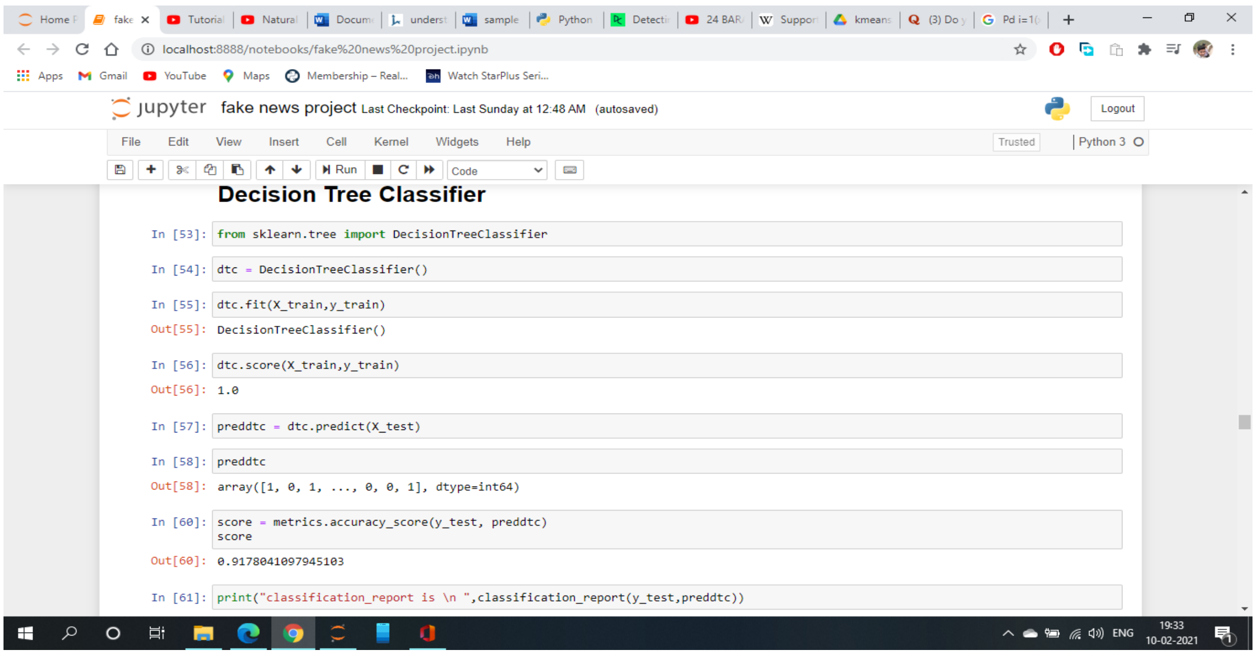
1. Decision tree algorithm begins with the root of the tree and breaks the data on the feature resulting in the greatest Information Gain (IG) (decrease in uncertainties towards final choice).

2. We can then perform this splitting process at every child node in an iterative cycle, until the leaves are not further divisible. It implies that the samples at each node of the leaf are all of the same class.

3. In implementation, we should set a cap on tree depth to prevent overfitting. We rely on pureness somewhat here since the final leaves may still be unclean

1

```
classification report is

precision    recall   f1-score    support 0                0.93     0.91
0.9        34731                0.90       0.93       0.92       3194accuracy
0.92       6667

macro avg       0.92     0.92           0.92       6667weighted avg   0.92
0.92            0.92       6667array ([[3158, 315],        [ 233, 2961]],
dtype=int64)
```


jupyter fake news project Last Checkpoint: Last Sunday at 12:48 AM (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Logout

Decision Tree Classifier

```
In [53]: from sklearn.tree import DecisionTreeClassifier

In [54]: dtc = DecisionTreeClassifier()

In [55]: dtc.fit(X_train,y_train)
Out[55]: DecisionTreeClassifier()

In [56]: dtc.score(X_train,y_train)
Out[56]: 1.0

In [57]: preddtc = dtc.predict(X_test)

In [58]: preddtc
Out[58]: array([1, 0, 1, ..., 0, 0, 1], dtype=int64)

In [60]: score = metrics.accuracy_score(y_test, preddtc)
         score
Out[60]: 0.9178041097945103

In [61]: print("classification_report is \n ",classification_report(y_test,preddtc))
```

Support Vector Classification (SVC)

Consider a binary classification task, where we are given a training set {(x1, y1), . . . ,(xm, ym)} with xi ∈ H and yi ∈ {±1}. Our aim is to find a linear decision boundary parameterized by (w, b) such that hw, xii + b ≥ 0

whenever yi = +1 and hw, xii+b < 0 whenever yi = −1. Furthermore, as discussed above, we fix the scaling of w by requiring mini=1,...m | hw, xii+b | =

1. A compact way to write our desiderata is to require yi(hw, xii + b) ≥ 1 for all i (also see Figure 7.1). The problem of maximizing the margin therefore reduces to

max (w,b) = 1 /|w|

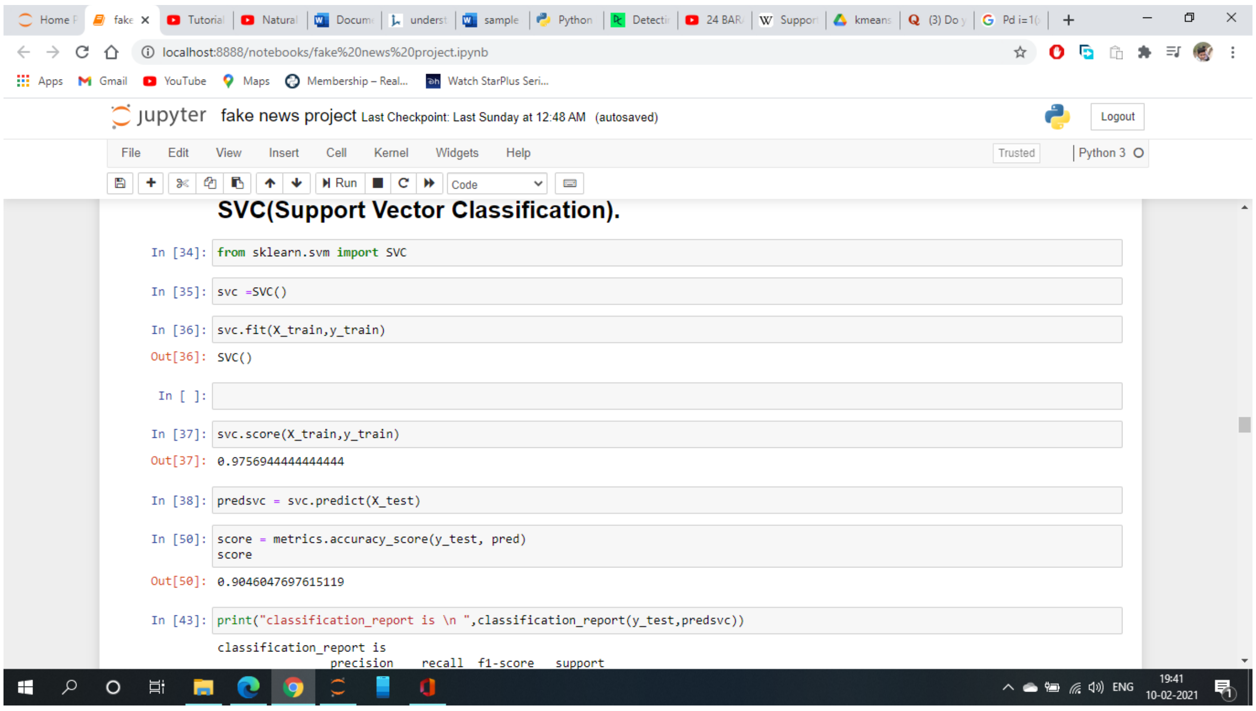
s.t. yi((w, xi) + b) ≥ 1 for all i,

or equivalently

min (w,b) 1/ 2 |w|^2

s.t. yi((w, xi) + b) ≥ 1 for all i.

```
classification_report is
                   precision    recall  f1-score   support
            0           0.99   0.87      0.93    3473
            1           0.87   0.99      0.93    3194
     accuracy                            0.93    6667
     macro avg          0.93   0.93      0.93    6667
     weighted avg       0.94   0.93      0.93    6667
Array([[3018, 455],
        [  26, 3168]], dtype=int64)
```



SVC(Support Vector Classification).

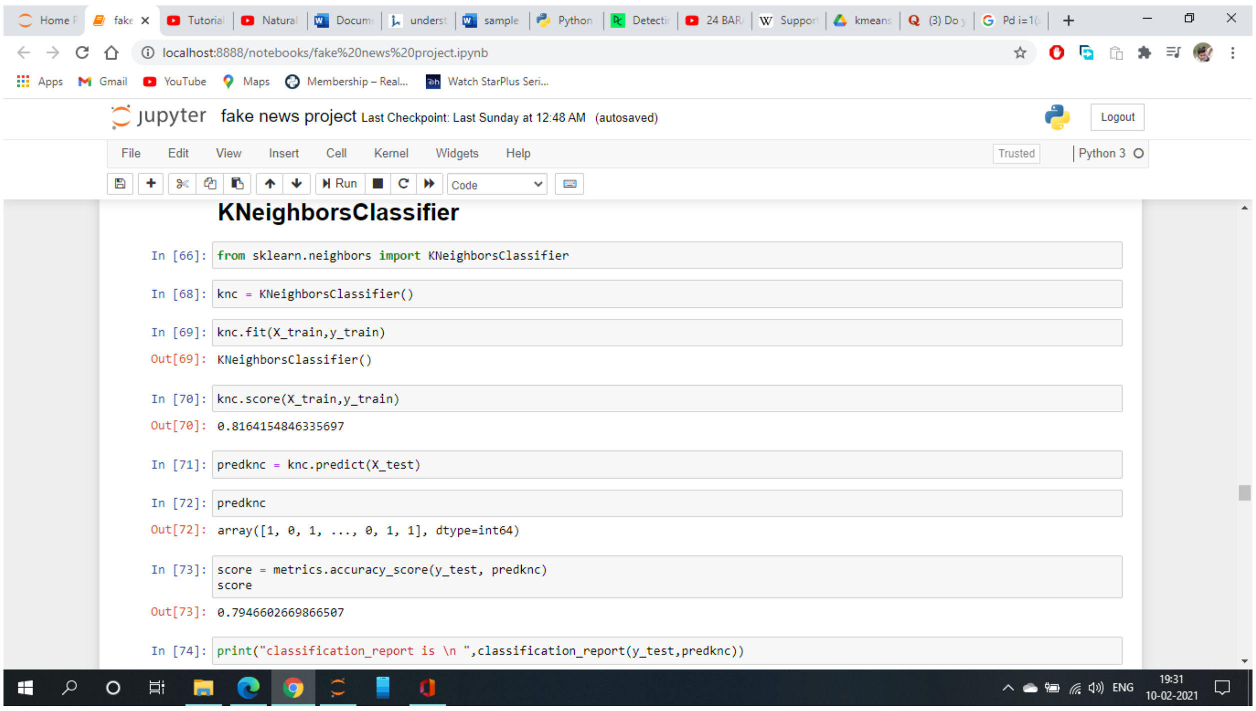
```
In [34]: from sklearn.svm import SVC

In [35]: svc =SVC()

In [36]: svc.fit(X_train,y_train)

Out[36]: SVC()

In [ ]:

In [37]: svc.score(X_train,y_train)

Out[37]: 0.9756944444444444

In [38]: predsvc = svc.predict(X_test)

In [50]: score = metrics.accuracy_score(y_test, pred)
         score

Out[50]: 0.9046047697615119

In [43]: print("classification_report is \n ",classification_report(y_test,predsvc))

         classification_report is
                       precision    recall  f1-score   support
```

k Nearest Neighbors

Throughout the entire chapter we assume that our instance domain, X , is endowed with a metric function ρ. That is, ρ : X ×X → R is a function that returns the distance between any two elements of X . For example, if X = R d then ρ can be the Euclidean distance, ρ(x, x') = ||x − x'|| = sigma(i=1)[(xi − x 0 i)^ 2]^(-2). Let S = (x1, y1), . . . ,(xm, ym) be a sequence of training examples. For each x ∈ X , let π1(x), . . . , πm(x) be a reordering of {1, . . . , m} according to their distance to x, ρ(x, xi). That is, for all i < m, ρ(x, xπi(x)) ≤ ρ(x, xπi+1(x)). For a number k, the k-NN rule for binary classification is defined as follows:

k-NN
input: a training sample S = (x1, y1), . . . ,(xm, ym)
output: for every point x ∈ X , return the majority label among {yπi(x) : i ≤ k}

```
classification_report is

                     precision   recall   f1-score      support           0
1.00     0.61       0.76         3473      1                        0.70
1.00       0.82         3194

accuracy                                      0.79         6667

macro avg                  0.85     0.80       0.79         6667weighted avg
0.86     0.79       0.79         6667
```

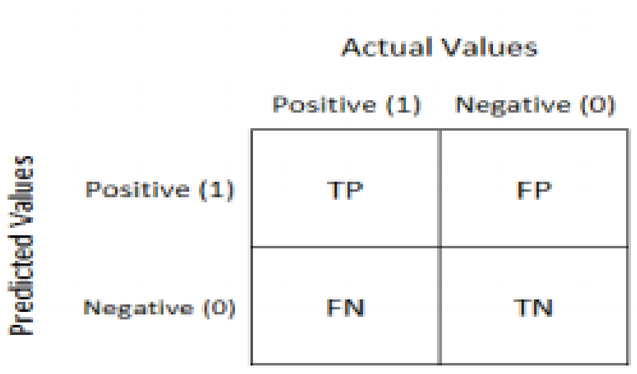


KNeighborsClassifier

```
In [66]: from sklearn.neighbors import KNeighborsClassifier

In [68]: knc = KNeighborsClassifier()

In [69]: knc.fit(X_train,y_train)
Out[69]: KNeighborsClassifier()

In [70]: knc.score(X_train,y_train)
Out[70]: 0.8164154846335697

In [71]: predknc = knc.predict(X_test)

In [72]: predknc
Out[72]: array([1, 0, 1, ..., 0, 1, 1], dtype=int64)

In [73]: score = metrics.accuracy_score(y_test, predknc)
         score
Out[73]: 0.7946602669866507

In [74]: print("classification_report is \n ",classification_report(y_test,predknc))
```

Confusion Matrix

Confusion Matrix, also known as an error matrix, is a table that is frequently used to define the performance of a classification model (or "classifier") on a collection of test data that knows the exact true values for. It enables the output of an algorithm to be visualized. It is a description of the results of prediction on a classification problem. The number of false and true predictions is articulated and divided by each class with count values. It also concentrates on the type of errors made by the classifier. It is a table of predicted and actual values.



Predicted Values		Actual Values	
		Positive (1)	Negative (0)
	Positive (1)	TP	FP
	Negative (0)	FN	TN

Fig 5. Confusion Matrix

Confusion Matrix For measuring Recall, Speed, Specificity and Accuracy, Confusion matrix is extremely useful.

True Positive (TP): Positive expected by the classifier, and that is true.

True Negative (TN): Negative expected by the classifier, and that is true.

False Positive (FP): Positive expected by the classifier, and false.

False Negative (FN): Negative expected by the classifier, and it is false.

B. Accuracy How much is the classifier right overall?

$Accuracy$ =($TP + TN$) / ($TP + TN + FP + FN$)

Conclusion

The work presented in this topic suggests that Multinomial Naive Bayes classifier is better in performing natural language processing tasks. Naive Bayes also has other variants but Multinomial is chosen because it operates better on discrete nature of data for example, number of occurrences if a term in the text. The work done in this paper is also encouraging, as it explains a fairly successful level of machine learning classification with only one extraction feature for huge fake news materials. Ultimately, further study and analysis is due to begin to define and create additional fake news classification grammars which will produce a more detailed classification strategy for both fake news and official statements.