GROUP MEMBERS:

Aamish Tariq FA19-BCS-087

Muhammad Umair FA19-BCS-139

MACHINE LEARNING PROJECT

TOPIC:

Stock Sentiment Prediction using headlines of a newspaper.

Summary

About Model:

The title of our project is stock sentiment Prediction. We will see that how we can perform stock sentiment analysis using the headlines of a newspaper. We will predict if the stock market goes up or down. This is a simple but very interesting project due to its prediction power.

We choose classification Problem. In this problem we will classify news given a news statement. Two classes will be used 0 and 1. By training this model we will see how stock market varies with newspaper headlines. By this we can already predict when stock goes up and down

News Headlines is the best way to predict stocks. So, if we train a model to predict stock sentiments based on previous dataset, we can take precautions and do business moves according to prediction. This will be very helpful for the people who rely on stock markets. The idea behind this project is to make a Machine Model to classify and perform stock sentiment analysis using the headlines of a newspaper.

First the data is read by using python libraries and then the data is cleaned to perform actions or analysis on it. Now split the data for testing and training and store them in an array. Now at last we train our model using Random Forest Classifier. Fit train data and Take predictions in the basis of data and at last Print its performance.

Working?

Following is sequence for the working of model.

1. Importing libraries required for Stock Sentiment Analysis.
2. Reading input data.
3. Cleaning our data.
4. Initializing Count Vectorizer.
5. Checking our data.
6. Splitting data.
7. Training our model.

1. Importing libraries required for Stock Sentiment Analysis.

First, we will import libraries.

```
In [2]: import pandas as pd

        import pickle
        import joblib
        from sklearn.ensemble import RandomForestClassifier
        from sklearn.feature_extraction.text import CountVectorizer
        from sklearn.metrics import confusion_matrix,accuracy_score
        from sklearn import metrics
        from sklearn.metrics import classification_report
        import numpy as np
        import matplotlib.pyplot as plt
        import plotly.express as px
        from sklearn.metrics import confusion_matrix
        import itertools
```

2. Reading input data.

The data is read using Panda's library in python. Once the data is read,

```
In [6]: df = pd.read_csv('C:/Users/aamish/Desktop/data/Data.csv')
        df.head(3)
```

we will print our data and note out number of rows and attributes. Since it is stock analysis using newspaper headlines it includes 27 total attributes in which 25 are headlines that are strings, 1 is date and one is our label that tells stocks goes up or down, which includes 0 and 1 values.

4101 no of rows and 27 attributes

```
In [10]: cleaned_df.shape
Out[10]: (4101, 27)
```

3. Cleaning our data.

Since headlines are string, we will then clean our data Here, we are using regex here to replace everything that's not a-z or A-Z by a space. Then we are just removing the extra spaces using regex. Finally, we are converting everything to lowercase, joining every headline from column 2 to 27 (because column 0 is timestamp and column 1 is label) and appending it to our headlines array.

```
In [7]: # fitting the count vectorizer on the sentences
        headlines = []

        cleaned_df = df.copy()

        cleaned_df.replace('[^a-zA-Z]', ' ',regex=True,inplace=True)
        cleaned_df.replace('[ ]+', ' ',regex=True,inplace=True)


        for row in range(len(df)):
            headlines.append(' '.join(str(x) for x in cleaned_df.iloc[row,2:]).lower())

In [9]: headlines[0]
Out[9]: 'a hindrance to operations extracts from the leaked reports 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 everton s party 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 hammers hand robson a youthful lesson saints party like it s  wear wolves have turned into lambs stump mike 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 solicitor diana s l
        andmine crusade put tories in a panic yeltsin s resignation caught opposition flat footed russian roulette sold out recovering
        a title'
```

4. Initializing Count Vectorizer.

Now, simply initializing Count Vectorizer to convert headlines to bag of words(vectors). In this Step we take the headlines array that we cleaned and put it into count vector, Countvectorizer is a method to convert text to numerical data.

```
In [25]: cv = CountVectorizer()
         cv.fit(headlines)
```

Once we are done Vectorizing we see there are now 46678 attributes, these are 46678 different words.

5. Checking out Data.

There are now 46678 attributes and 4101 rows.

6. Splitting Data.

There are some phases for the making of Machine Learning model, one of them include splitting data for training and testing. The data set divided into 2, one for the training purpose and one for the testing purpose.

Here we divide dataset based on time (date) attribute. Train data is all before 1 Jan 2015 (excluding it). Test data is all after 31 December 2014 (starting from 1 Jan 2015). Then simply transforming headlines using CountVectorizer we initialized above.

```
In [12]: train_data = cleaned_df[df['Date']<'20150101']
         test_data = cleaned_df[df['Date']>'20141231']

         train_data_len = len(train_data)

         train_headlines = cv.transform(headlines[:train_data_len])
         test_headlines = cv.transform(headlines[3723:])
```

The data is then split through length of dataset.

Total Rows = 4101

For test = 3973

For train = 378

```
In [13]: print(train_data.shape)
         print(test_data.shape)

         (3975, 27)
         (378, 27)
```

Training phases include 3975 row and 46678 attributes whereas testing phase has 378 rows and 46678 attributes.

```
In [25]: print(train_headlines.shape)
         #train_data['Label'].shape

         (3975, 46678)

In [15]: print(test_headlines.shape)

         (378, 46678)
```

7. Training our model

We applied 3 algorithms for the training of our model which include,

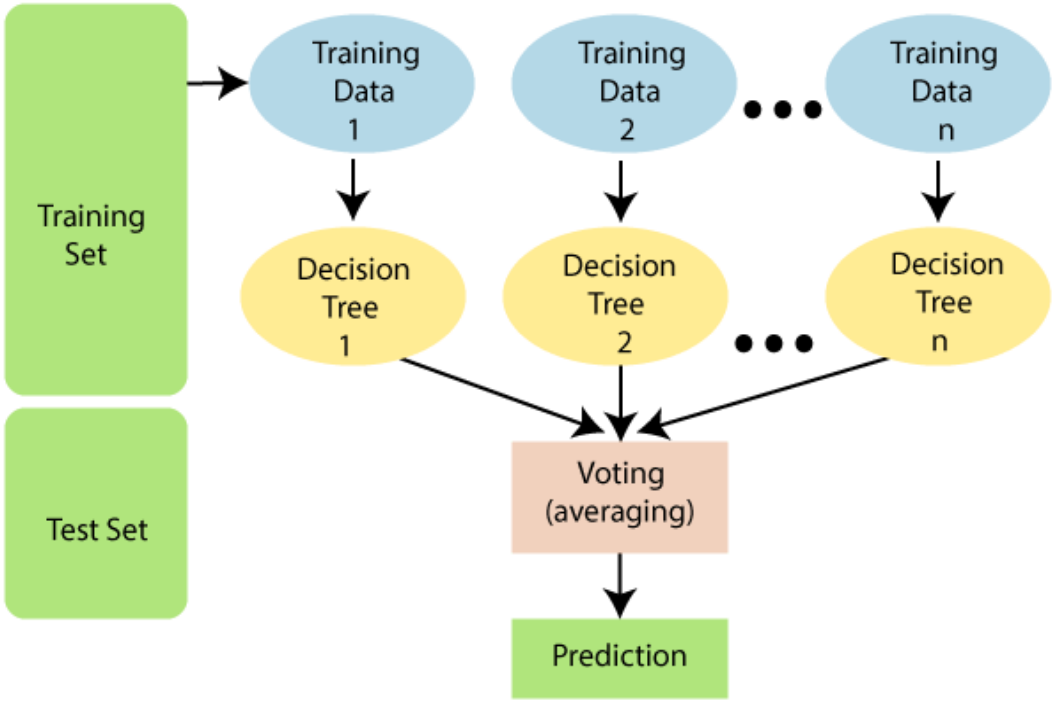
- Random Forest Classifier
- Support Vector Machine
- K Nearest Neighbors Classifier

Random Forest Classifier

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model.

As the name suggests, "Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset." Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.

The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.



Random Forest Classifier includes 2 things estimators and entropy.

- **n_estimators =** The required number of trees in the Random Forest. The default value is 10. We can choose any number but need to take care of the overfitting issue.
- **criterion=** It is a function to analyze the accuracy of the split. Here we have taken "entropy" for the information gain.

```
# lets try with random forest
rfc = RandomForestClassifier(n_estimators=200,criterion='entropy')

# fitting the model
rfc.fit(train_headlines,train_data['Label'])

# making predictions
preds = rfc.predict(test_headlines)
```

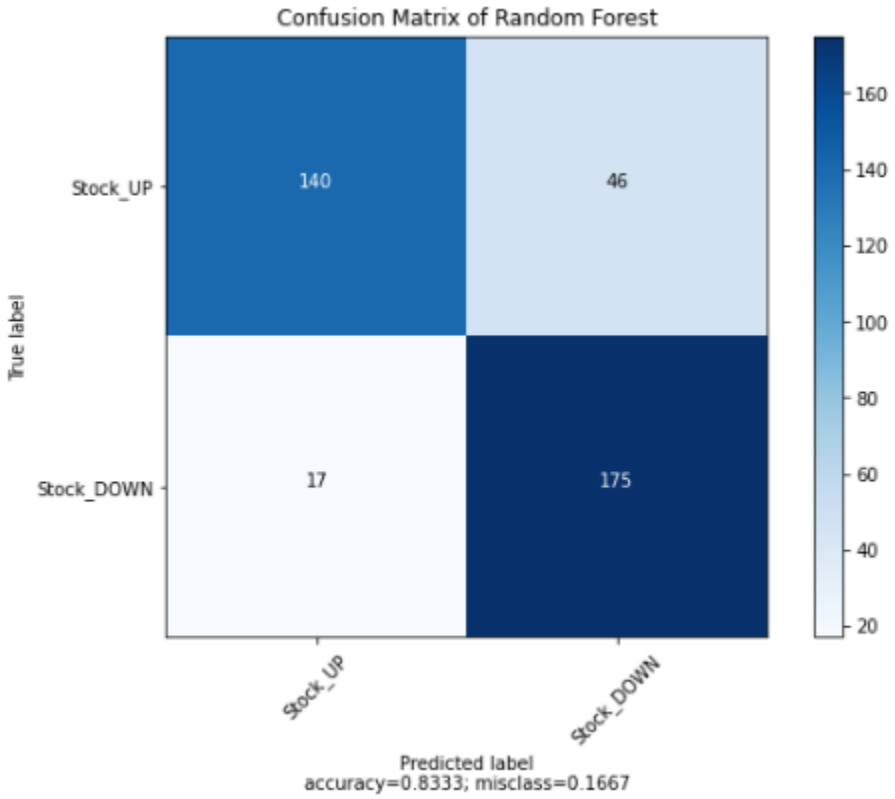
Once the algorithm is applied the data set is made fit, the observed output is then compared with actual output of the dataset.

```
In [15]: print(accuracy_score(test_data['Label'],preds))
         confusion_matrix(test_data['Label'],preds)

         0.8333333333333334

Out[15]: array([[140,  46],
                [ 17, 175]], dtype=int64)
```

The results shows that Random Forest Classification show 83.33333334 % accuracy

When we applied confusion matrix it shows it shows following figures.

- 140 were those that model predicted stocks and that were Stock up.
- 46 model predicted Stock DOWN but was UP.
- 17 model predicted UP but was DOWN.
- 175 model predicted down and was down.



Support Vector Machine

Support Vector Machine (SVM) is a supervised machine learning algorithm used for both classification and regression. Though we say regression problems as well its best suited for classification. The objective of SVM algorithm is to find a hyperplane in an N-dimensional space that distinctly classifies the data points. The dimension of the hyperplane depends upon the number of features. If the number of input features is two, then the hyperplane is just a line. If the number of input features is three, then the hyperplane becomes a 2-D plane. It becomes difficult to imagine when the number of features exceeds three.

We used RBF as a kernel in the algorithm and state is set to 1

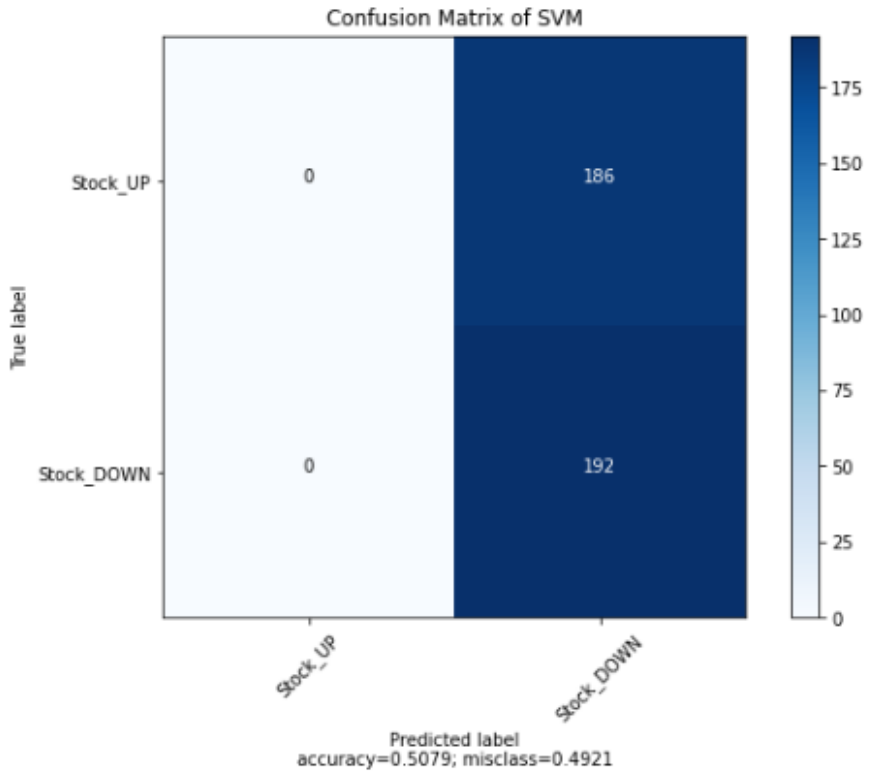
```
In [19]: from sklearn import svm
         clf = svm.SVC(kernel='rbf', random_state=1, gamma='auto', C=1.0)
         #
         # training
         print("********************Training**********************")
         clf.fit(train_headlines, train_data['Label'])

         preds = clf.predict(test_headlines)

         ********************Training**********************
```

After training we compare the predicted value with the actual value, it is found that SVM could not classify the given data set and thus we concluded that we cannot train this data set with SVM.

The accuracy we obtained from SVM is 50%.

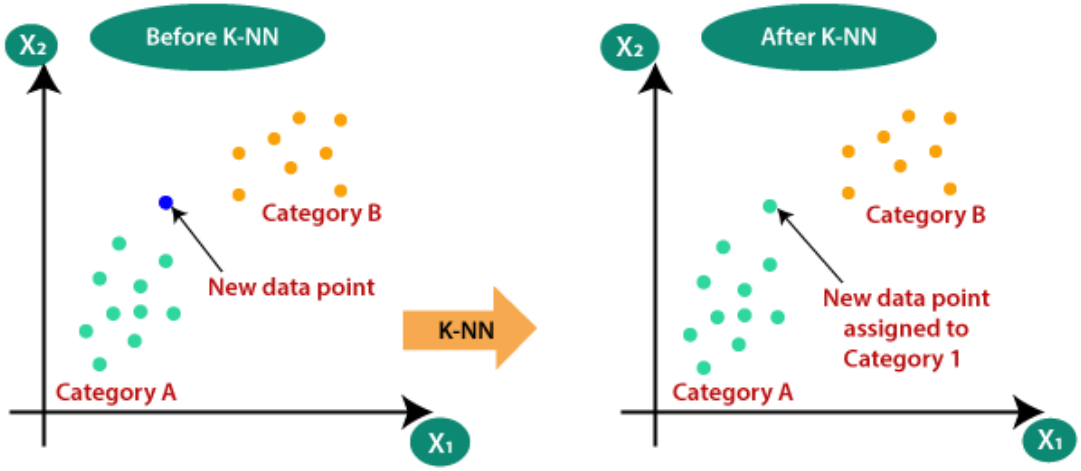
```
In [20]: print(accuracy_score(test_data['Label'],preds))
         confusion_matrix(test_data['Label'],preds)

         0.5079365079365079

Out[20]: array([[  0, 186],
                [  0, 192]], dtype=int64)
```

The Confusion matrix is shown



K-Nearest Neighbor Classifier

K-Nearest Neighbor is one of the simplest Machine Learning algorithms based on Supervised Learning technique. K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most like the available categories. K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm, K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.



Fitting K-NN classifier to the Training data:

To do this we will import the **KNeighborsClassifier** class of **Sklearn Neighbors** library. After importing the class, we will create the **Classifier** object of the class. The Parameter of this class will be

- **n_neighbors:** To define the required neighbors of the algorithm. Usually, it takes 3.

And then we will fit the classifier to the training data. Below is the code for it:

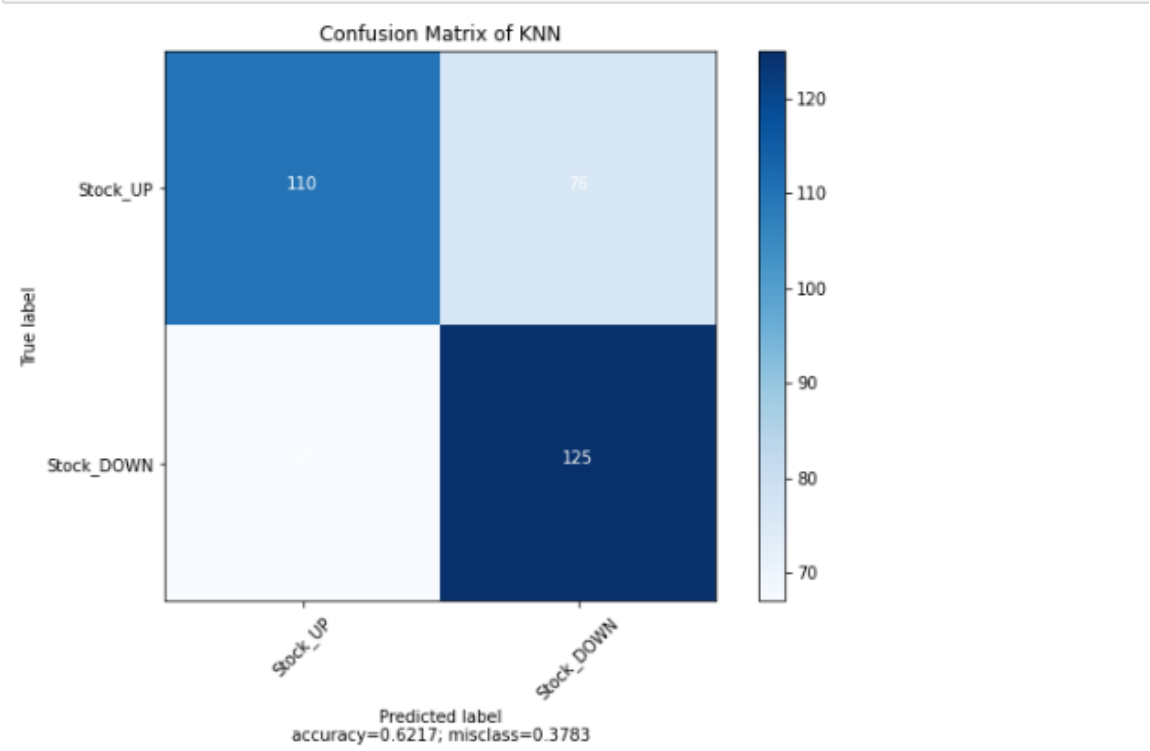
```
In [23]: from sklearn.neighbors import KNeighborsClassifier
         classifier = KNeighborsClassifier(n_neighbors=3)
         classifier.fit(train_headlines, train_data['Label'])
         preds = classifier.predict(test_headlines)
```

One the dataset is fit in the predicted value, it is then compared to actual value.

```
In [24]: print(accuracy_score(test_data['Label'],preds))
         confusion_matrix(test_data['Label'],preds)

         0.6216931216931217

Out[24]: array([[110,  76],
                [ 67, 125]], dtype=int64)
```

The results show that KNN has 62% accuracy for the given dataset.

Its confusion matrix is shown below




```
28]: x = ["RandomForest", "SVM", "KNN"]
```

Comparison

The comparison shows that RFC has highest accuracy and the best fit algorithm for the given data set.

In [28]:
```
x = ["RandomForest", "SVM", "KNN"]
h = result
c = ["red", "green", "orange", "black"]
plt.bar(x,h,width=0.3, color=c)
plt.xlabel("Classifications")
plt.ylabel("Percentage")
plt.show()
```