PREDICTIVE ANALYTICS ON CROSS-DOMAIN DATASETS USING ENSEMBLERS

Prateek Chauhan - 2931465

Griffith College Dublin

089 211 7387

chauhanprateek89@gmail.com

Table of Contents

Table of Figures 5
Disclaimer 8
Abstract 9
Acknowledgements 10
Chapter 1 – Introduction 11
Chapter 2 – Personalized Medicine 16
2.1 Motivation 16
2.2 Data Source 16
2.3 Exploratory Data Analysis 17
2.3.1 The variants data tables 17
2.3.2 The Text Files 23
2.4 Classification 35
2.4.1 Boosting Algorithms 35
2.4.2 LightGBM 36
2.4.3 Linear SVC 43
2.4.4 Keras – Deep Learning Library [19]. 45
2.5 Conclusion and Future Work 47
Chapter 3 – Instagram Like Prediction 48
3.1 Background 48
3.2 Data Source 49
3.2.1 Building the dataset 49
3.3 Data Preprocessing 52
3.4 Feature Extraction 53
3.4.1 Natural Language Processing 56
3.5 Prediction Models 60
3.5.1 ANNs – Keras API 60
3.5.2 Regression – Linear Regression [27] 62
3.5.3 Gradient Boosting Regressor [28] 63
3.5.4 AdaBoost Regression [30] 64
3.5.5 Lasso Regression 65
3.6 Conclusion and Future Work 66
Chapter 4 – Stock market prediction 67

4.1 Introduction 67
4.1.1 History of stocks 67
4.1.2 Function and Purpose of Stock Markets 67
4.1.3 Stock market crash 67
4.1.4 Irrational behaviour of the stock market 68
4.2 Background 68
4.3 Methodology 69
4.3.1 S&P-500 Stock Prediction 69
4.3.2 Stock Predictions of ‘Reliance’ [75] using tweets. 80
4.3.3 Stock Prediction using Facebook’s Prophet 86
4.4 Conclusion and Future Work 89
Chapter 5 – Fake profile detection using Facebook data 90
5.1 Motivation 90
5.2 Background / Similar work 90
5.3 Data Source 91
5.4 Data Preprocessing 91
5.5 Feature Extraction 92
5.6 Model 92
5.6.1 Random Forest Classifier [73] 92
5.7 Conclusion and Future Work 96
Chapter 6 – Intrusion Detection System 98
6.1 Introduction 98
6.2 Background / Similar Work 98
6.3 Motivation 99
6.4 Data Source 100
6.5 Data Processing 100
6.6 Feature Building 102
6.7 Classification 103
6.7.1 Random Forest Classifier 103
6.7.2 Logistic Regression using Principle Component Analysis 103
6.8 Conclusion 104
Appendix 1 105
Keras 105

SVM 106
Appendix 2 107
Part A 107
Part B 107
Part C 108
Part D 108
Part E 108
Part F 109
Lasso Regression 109
Appendix 3 110
Part A 110
Part B 111
Part C 111
Part D 112
Part E 112
Part F 113
Bibliography 114

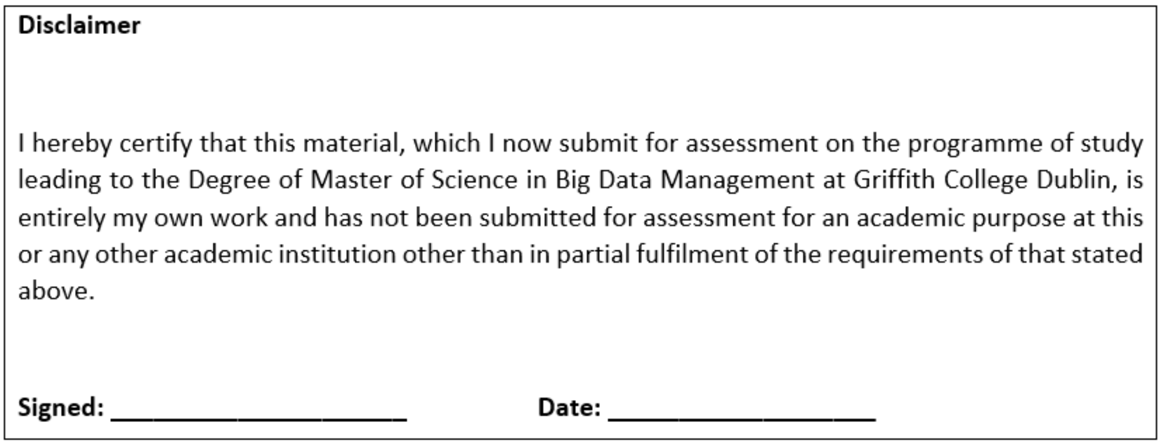
Table of Figures

Figure 1. Predictive analytics 12
Figure 2. Human-in-the-loop ML [108] 13
Figure 3. How to select an algorithm [109] 14
Figure 4. Variant data table 17
Figure 5. Summary of training-variant 17
Figure 6. Distribution of data according to labels 18
Figure 7. Gene and Variation in train and test sets 19
Figure 8. Frequency of most occurred Genes in train set 19
Figure 9. Frequency of most occurred Genes in test set 20
Figure 10. distribution of the Genes and their occurrences in training dataset per class 21
Figure 11.cumulative frequencies of Variations present training and testing data 22
Figure 12. Distribution of Gene vs Mutation in train and test sets 23
Figure 13. Text dump of first instance in train set 23
Figure 14. Text length distribution per class 24
Figure 15. Word clouds - top 25 words per class 27
Figure 16. top twenty counts of most common words among text 29
Figure 17. top twenty counts of most common words among text - after filtering 29
Figure 18. Feature Building - Step 1 32
Figure 19. Feature Building - Step 2 32
Figure 20. Feature Building - Step 3 33
Figure 21. Feature Building - Step 4 33
Figure 22. Feature Building - Step 5 33
Figure 23. Final dataset after feature building 34
Figure 24. . Feature Building - code snippet 34
Figure 25. test train split 36
Figure 26. LightGBM model fitting 37
Figure 27. LightGBM – calculating log loss 37
Figure 28. LightGBM parameters 40
Figure 29. LightGBM – results 41
Figure 30. LightGBM - results 42
Figure 31. LightGBM - feature importance 42
Figure 32. SVM - training and testing datasets 43
Figure 33. SVM – pipeline 44
Figure 34. SVM - model accuracy 44
Figure 35. SVM -results 45
Figure 36. Keras - sequential baseline model 46
Figure 37. Keras- model accuracy and loss 47
Figure 38. Instagram profile - JSON structure 50
Figure 39. Instagram dataframe of 16539 posts 51
Figure 40. Dashboard for EDA on Instagram dataset 51
Figure 41. Dataset structure 53
Figure 42. one-hot encoding: website 54
Figure 43. one-hot encoding: date 54

Figure 44. Dashboard for number of posts per day....54
Figure 45. percentage of posts per day....55
Figure 46. Dashboard for Likes per posr per day....56
Figure 47. Data filtering....57
Figure 48. post description....57
Figure 49. emoji removed....58
Figure 50. Original post description....58
Figure 51. Processed posts description....58
Figure 52. sparse matrix from post description....59
Figure 53. top 37 most used emojis on Instagram....59
Figure 54. Sparse matrix using profile description....59
Figure 55. final dataset to be used for prediction....60
Figure 56. attributes used for prediction....61
Figure 57. performance comparison of prediction models....66
Figure 58. S&P-500 stock prices....71
Figure 59. Adjusted close price of S&P-500....72
Figure 60. Percentage change of S&P-500....73
Figure 61. dimesnitons of different datasets....74
Figure 62. Oil prices....75
Figure 63. Oil prices for the past 7 years....75
Figure 64. percentage change in oil prices....76
Figure 65. dimensions of commoditeis datasets....77
Figure 66. Merging of datasets....77
Figure 67. Merged dataset....78
Figure 68. Final dataset....78
Figure 69. Tweets of 'Reliance' dataset....81
Figure 70. total number of tweets collected per day....82
Figure 71. close score and open score for a given date....83
Figure 72. pre-processed dataset....84
Figure 73. adding temporal shift for open price and close price....84
Figure 74. Performance evaluation of different approaches....85
Figure 75. S&P-500 dataset with just adjusted close price....86
Figure 76. Future dataframe....87
Figure 77. Forecasting results....87
Figure 78. Forecast plot....88
Figure 79. concatenating dataframes and calculation of labels....91
Figure 80. code snippet for sec machine....92
Figure 81. Random Forest scores....93
Figure 82. Random forest - learning curve....94
Figure 83. Random Forest - confusion matrix....94
Figure 84. Random Forest - ROC curve....95
Figure 85. Random Forest - feature ranking....96
Figure 86. Random Forest - feature importance plot....96
Figure 87. 24 labels for types of attacks....101

Figure 88. Protocol type - categories 101
Figure 89. Service - categories 102
Figure 90. Flag - categories 102
Figure 91. label - categories 102
Figure 92. Random Forest - prediction accuracy 103
Figure 93. Logistic regression - prediction accuracy 104
Figure 94. Keras - Part 1 105
Figure 95. Keras - Part 2 105
Figure 96. linear Support Vector Machine 106
Figure 97. JSON to CSV 107
Figure 98. Instagram data split into train and test 108
Figure 99. Linear regression 108
Figure 100. Gradient Boosting 108
Figure 101. AdaBoost regression 109
Figure 102. S&P-500 prices 110
Figure 103. Oil prices 110
Figure 104. prediction models 111
Figure 105. tweets scrapping 111
Figure 106. Vader Sentiment 112
Figure 107. open and close sentiment score calculation 112
Figure 108. Prophet implementation 113

Disclaimer



Disclaimer

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the Degree of Master of Science in Big Data Management at Griffith College Dublin, is entirely my own work and has not been submitted for assessment for an academic purpose at this or any other academic institution other than in partial fulfilment of the requirements of that stated above.

Signed: ______________________ **Date:** ____________________

Abstract

The focus of this thesis is to prove a hypothesis that predictive analytics using ensemble models by building feature extraction can outperform neural networks with n number of epochs and low learning rates. For this purpose, five different datasets from five different domains are taken for processing. Each of the datasets is different from one another having unrelatable characteristics with each other. The approaches implemented in this thesis range from text processing, dimensionality reduction, singular value decomposition, regression, additive boosting, ensemble classification on binary as well as multivariate models to sentiments calculations, scrapping of data from Twitter and Instagram to create datasets, using Yahoo Financial and Quandl APIs for financial data. For the implementation of every algorithm, why was it chosen for the dataset, what are the strength of this approach over other algorithms and how can the results be optimized, questions like this have been raised and answers for them have been explained.

Acknowledgements

I would like to thank my thesis supervisor Mr. Osama Abusham at Griffith College Dublin. The door to Prof. Abusham's office was always open whenever I ran into a trouble spot or had a question about my research or writing. He constantly allowed this paper to be my own work but steered me in the right direction whenever he thought I needed it.

I would also like to thank Dr. Parul Tomar, a researcher at Max Plank's Institute of Immunobiology and Epigenetics, without whom one of the chapters i.e. Personalized Medicine for Cancer wouldn't even have existed.

I would also like to thank Mr. Sumanth Mattapalli an alumnus of Griffith College Dublin, who helped me in a very crucial time when I was struggling to scrape data. He automated the entire process using Selenium and because of him, I could include Chapter 2 in this dissertation.

Last but not the least I would like to thank the genius scientist Rick Sanchez who has always inspired me to give my best to whatever task I am up to and has taught me to never give up.

Chapter 1 – Introduction

The core focus of this entire dissertation is the implementation of predictive analytics using Ensemble Models via feature extraction and how this implementation is better than other approaches. To verify this initial hypothesis datasets from 5 different domains are explored and predictive analytics is applied using methods that are best for the type of datasets. The domains include:

- Healthcare – Cancer research
- Social Media and Digital Marketing
- Stock Markets
- Social Network Analysis
- Intrusion detection in networks

Predictive analytics is the branch of data mining that helps predict probabilities and trends about unknown future events. Predictive analytics uses many techniques from data mining, statistics, modeling, machine learning, and artificial intelligence to analyze current data to make predictions about future and to bring together the management, information technology, and modeling business process to make predictions about the future. The patterns found in historical and transactional data can be used to identify risks and opportunities for future. These models capture relationships among many factors to assess risk with a particular set of conditions to assign a score or weight. By successfully applying predictive analytics the business can effectively interpret big data for their benefit.

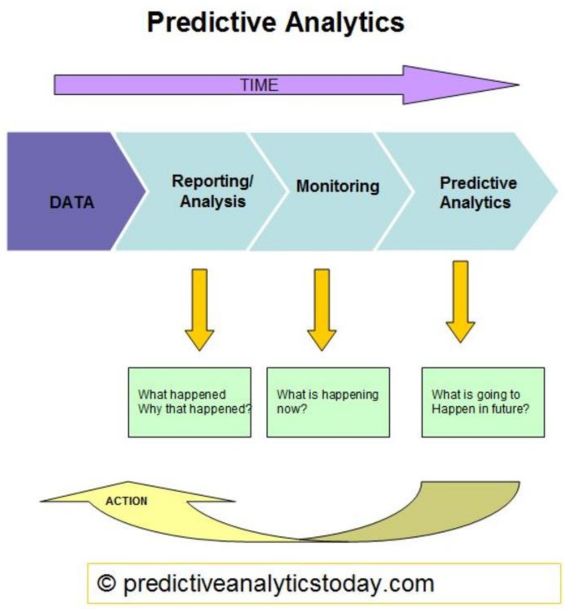


Figure 1. Predictive analytics

Though huge advances have been seen in the quality and accuracy of pure machine-driven systems, these tend to fall short of acceptable accuracy rates. The combination of machine-driven classification enhanced by human correction, on the other hand, provides a clear path forward in acceptable accuracy.

The approach taken to apply predictive analytics is based roughly on Human-in-the-loop Machine Learning [107]. Evidence suggests that a variant of Pareto's famous 80:20 leads to some of the most accurate machine learning systems to date, with 80% computer driven, 19% human input and 1% unknown randomness to balance things out.

Human input can come into two different forms.

- Helping label the original dataset that will be fed into a machine learning model
- Helping correct inaccurate predictions that arise as the system goes live.

The image given below describes the iterative process of human-in-the-loop machine learning, and this is repeated until a satisfactory result is achieved.

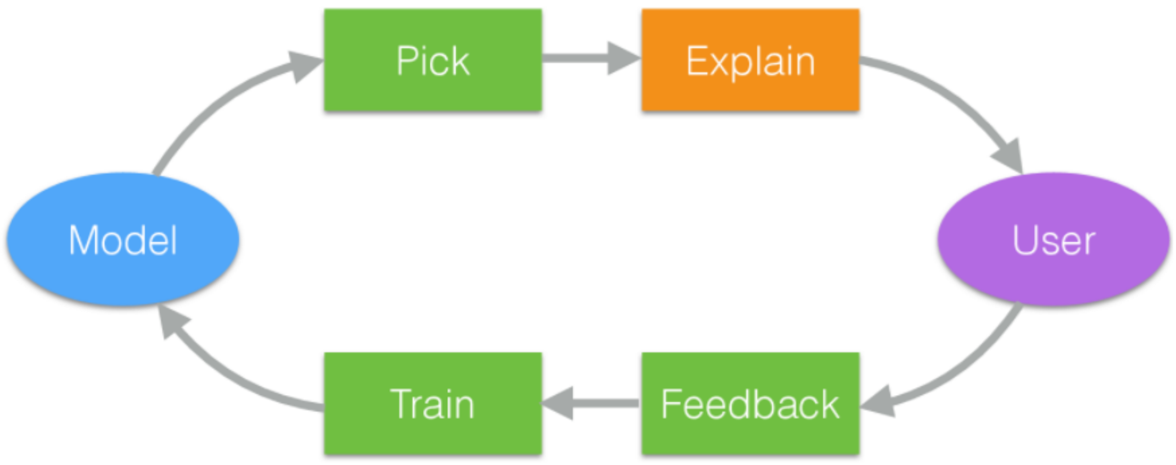


Figure 2. Human-in-the-loop ML [108]

Though this approach might not be shown how it is being implemented this is the iterative process that has been used to implement predictive analytics across the domains being used in this dissertation.

The datasets used, all of them have different dimensions, shapes, characteristics, and behavior. No two datasets are similar. Therefore, different approaches have been taken to predict or classify the data. The decision-making step i.e. which approach is to be used when follows a flow chart which is given below. Using this flowchart decision were made on what methodology is to be used and why it is supposed to be used.

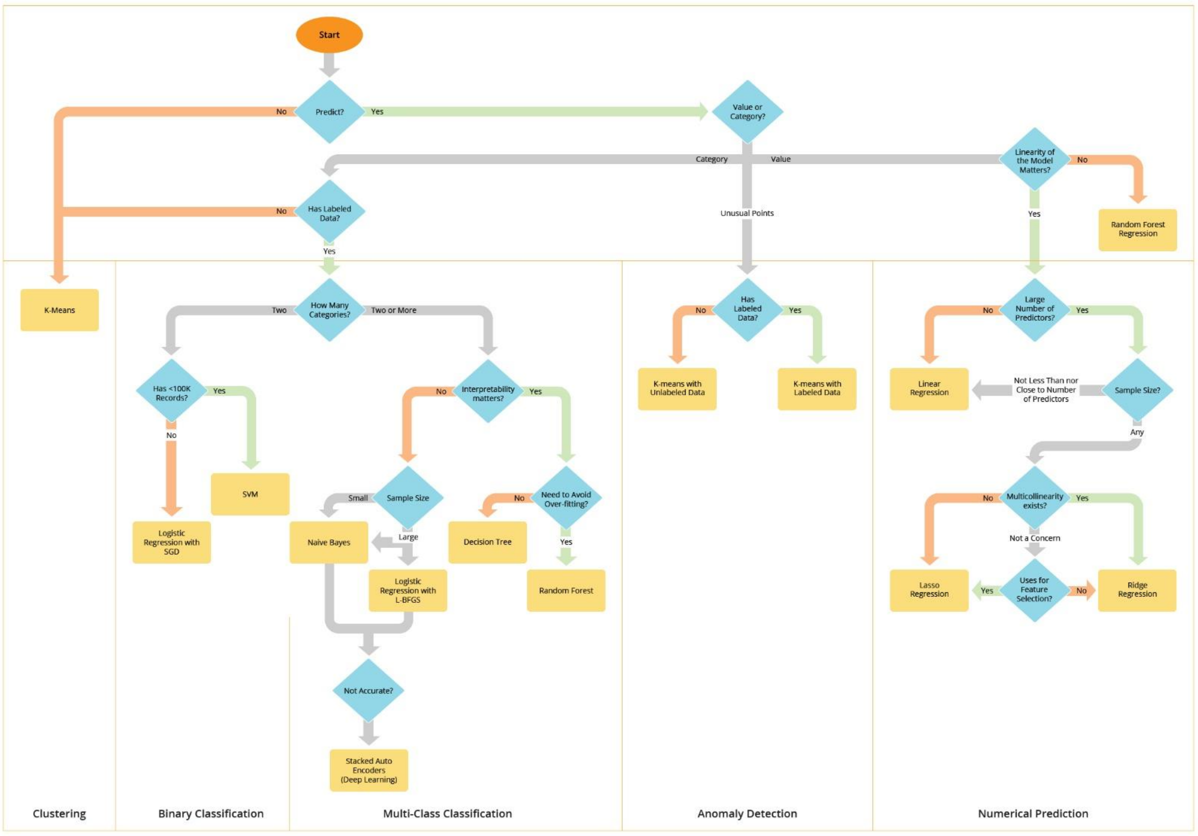


Figure 3. How to select an algorithm [109]

Chapter 2 covers the exploratory analysis of the dataset being used for Healthcare – Cancer research domain. The data has integers, string, and textual characteristics and has 9 labels to classify any instance into thus making this a multivariate problem for classification. The issues faced in this chapter are during handling of textual data and using a sparse matrix instead of textual data to increase the efficiency of the model. The algorithms used are LightGBM, Support Vector Machine and Artificial Neural Networks.

Chapter 3 covers an interesting topic of how social media can be used by digital marketers to promote their commodities and due to this how social media influencers are earning big amount of money. The dataset used here has been scrapped from around 1000 Instagram profile and a heavy amount of data cleaning is involved to come up with a dataset that can be used for prediction. As there are no labels and a numerical value has to be predicted, this makes it a Regression problem. Five different approaches have been used to come up with prediction values. Their comparison and limitations are also discussed.

Chapter 4 is an age-old problem of stock prediction. Three different approaches are proposed to predict the prices of stock, the first one being by just using the prices of stocks across the world markets. The second approach is implemented by using sentiment scores from tweets for a particular corporation to see the if there is a trend in the sentiment scores of tweets in a day and the stock prices for that day. The last approach is implemented using Facebook's latest forecasting package 'Prophet'. All these approaches have some limitations which have been discussed in the chapter.

Chapter 5 focuses on fake profiles on Social Media and its adverse effects for other people. As the data being used has just two labels, it makes this problem into a binary classification model. Random forest classifier has been used to solve this problem. This chapter also discusses how a profile can be labeled as a genuine profile or a fake profile using different features.

Chapter 6 aims to build an Intrusion Detection System using the KDD 1999 Cup dataset which is a widely renowned network security dataset. This is also a multivariate problem but the approach to solve this is different. This chapter also focuses on previous works and their implementation on the same dataset and how these implementations can be out performed just by using an ensemble model with features.

The Appendix consists important parts of the code used in Chapters 2-6.

Chapter 2 – Personalized Medicine

2.1 Motivation

The role of genetic testing in advancing the understanding of cancer and designing of more precise and effective treatments has shown huge potential, the progress in this area has been significantly slow due to a significant amount of manual work still required to understand genomics. Once sequenced, a cancer tumor can have thousands of genetic mutations, but the challenge is distinguishing the mutations that contribute to tumor growth (drivers) from the neutral mutations (passengers). This is a very time-consuming task where a clinical pathologist has to manually review and classify every single genetic mutation based on evidence from the text-based clinical literature. For the past several years, researchers at Memorial Sloan Kettering Cancer Centre have been working to create an expert-annotated precision oncology knowledge base. The dataset is designed in such a way that it can be used to train machine learning models to help experts significantly speed up the research on targeted medicines.

2.2 Data Source

The data comes in 4 different files. Two CSV and two text files:

- Training/test variants – these are the CSV files of the gene mutations together with the target value Class, which is (manually) classified assessment of the mutation. The feature variables are Gene, the specific gene where the mutation took place and the Variation, the nature of the mutation. The test data, of course, doesn't have the Class value. This is what has to be predicted. These two files are linked through an ID variable to another file, namely:
- Training/test text – these files contain an extensive description of the evidence that was used (by experts) to manually label the mutation classes.

An initial exploration of the dataset, it is observed that the total number of instances in training variants is 3321 and in the test variants is 5668. Which means that there is 70% more test data than training data. The data description mentions: 'Some of the test data is machine generates to prevent hand labeling', which should explain this otherwise curious imbalance.

2.3 Exploratory Data Analysis

2.3.1 The variants data tables

```
## Observations: 3,321
## Variables: 4
## $ ID        <int> 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15...
## $ Gene      <fctr> FAM58A, CBL, CBL, CBL, CBL, CBL, CBL, CBL, CBL, CBL...
## $ Variation <fctr> Truncating Mutations, W802*, Q249E, N454D, L399V, V...
## $ Class     <fctr> 1, 2, 2, 3, 4, 4, 5, 1, 4, 4, 4, 4, 4, 4, 5, 4, 1, ...
```

Figure 4. Variant data table

Fig above gives us a glimpse of the training variants file, it has 4 attributes and 3321 instances. Fig below gives a somewhat detailed view of what the attributes contain. Concentrating on the 'Class' attribute which also acts the label for our dataset there are 9 classes in total which the researchers have manually encoded the dataset with, which can mean that the provided dataset has 3321 instances of 9 different types of cancer. This, obviously is concluded from initial exploration of the dataset.

```
##        ID              Gene                       Variation     Class
##  Min.   :   0   BRCA1   : 264   Truncating Mutations:  93    1:568
##  1st Qu.: 830   TP53    : 163   Deletion            :  74    2:452
##  Median :1660   EGFR    : 141   Amplification       :  71    3: 89
##  Mean   :1660   PTEN    : 126   Fusions             :  34    4:686
##  3rd Qu.:2490   BRCA2   : 125   Overexpression      :   6    5:242
##  Max.   :3320   KIT     :  99   G12V                :   4    6:275
##                 BRAF    :  93   E17K                :   3    7:953
##                 ALK     :  69   Q61H                :   3    8: 19
##                 (Other):2241    (Other)             :3033    9: 37
```

Figure 5. Summary of training-variant

'Class' 3, 8 and 9 have relatively less number of instances and 'Class' 7 has the maximum number of instances as compared to other classes. This distribution, i.e. uneven spread of instances in the training set can affect the outcome of the predictions that should be performed, but this is again an initial hypothesis. More can be learned about this effect once the dataset is investigated in more depth.

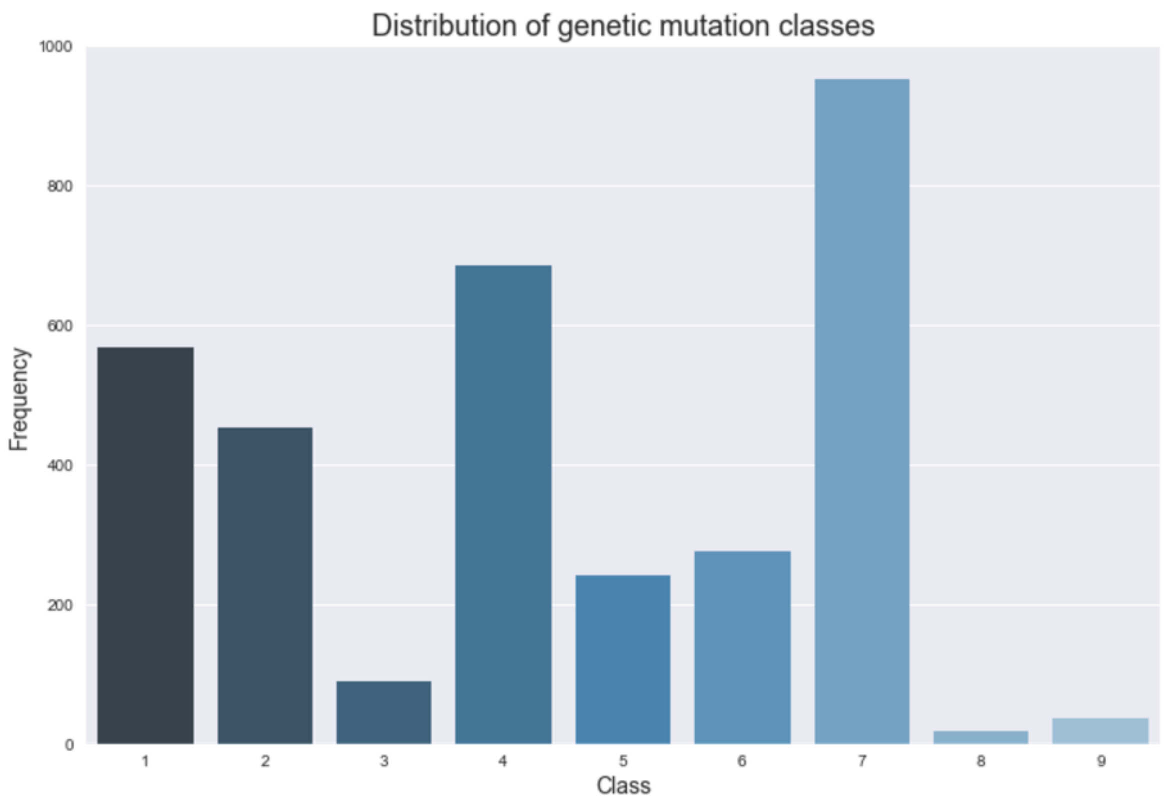
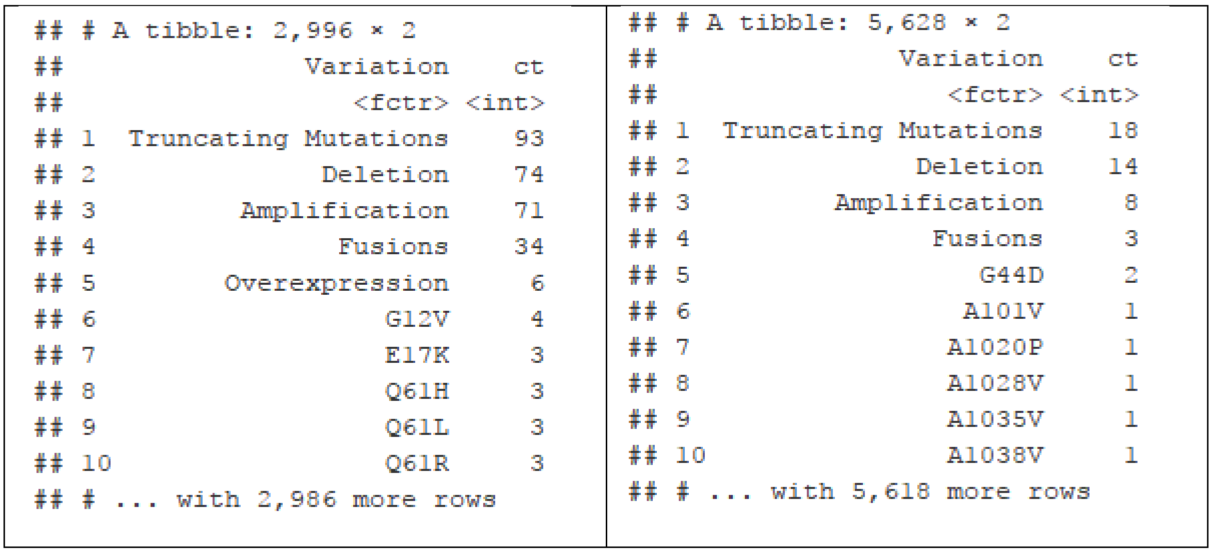


Figure 6. Distribution of data according to labels

Now, observing the other two attributes i.e. ‘Gene’ and ‘Mutation’ for training and testing datasets, the following things were observed:

Train	Test
`## # A tibble: 264 × 2`	`## # A tibble: 1,397 × 2`
`##       Gene    ct`	`##       Gene    ct`
`##     <fctr> <int>`	`##     <fctr> <int>`
`## 1    BRCA1   264`	`## 1       F8   134`
`## 2     TP53   163`	`## 2     CFTR    57`
`## 3     EGFR   141`	`## 3       F9    54`
`## 4     PTEN   126`	`## 4     G6PD    46`
`## 5    BRCA2   125`	`## 5      GBA    39`
`## 6      KIT    99`	`## 6       AR    38`
`## 7     BRAF    93`	`## 7      PAH    38`
`## 8      ALK    69`	`## 8     CASR    37`
`## 9    ERBB2    69`	`## 9     ARSA    30`
`## 10  PDGFRA    60`	`## 10   BRCA1    29`
`## # ... with 254 more rows`	`## # ... with 1,387 more rows`



```
## # A tibble: 2,996 × 2
##                 Variation    ct
##                    <fctr> <int>
## 1  Truncating Mutations    93
## 2              Deletion    74
## 3         Amplification    71
## 4               Fusions    34
## 5        Overexpression     6
## 6                  G12V     4
## 7                  E17K     3
## 8                  Q61H     3
## 9                  Q61L     3
## 10                 Q61R     3
## # ... with 2,986 more rows
```

```
## # A tibble: 5,628 × 2
##                 Variation    ct
##                    <fctr> <int>
## 1  Truncating Mutations    18
## 2              Deletion    14
## 3         Amplification     8
## 4               Fusions     3
## 5                  G44D     2
## 6                 A101V     1
## 7                A1020P     1
## 8                A1028V     1
## 9                A1035V     1
## 10               A1038V     1
## # ... with 5,618 more rows
```

Figure 7. Gene and Variation in train and test sets

- The Gene and Variation features contain character strings of various lengths.
- The most frequent Genes in the train vs test data are completely different.
- The test data seems to contain significantly more different Genes and fewer high-frequency Genes than the train data.
- The difference in frequency might mirror the true fraction of effective test data over train data.
- In contrast, the most frequent 'Variations' in train vs test are largely identical. Although, the corresponding frequencies are lower in the test data.

2.3.1.1 Individual feature visualization – Genes

Figure 8 and Figure 9 shows the frequencies of Genes in the train and test variants datasets, respectively.

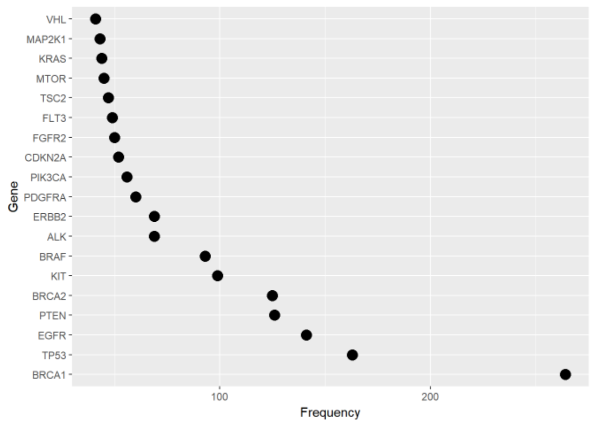


Figure 8. Frequency of most occurred Genes in train set

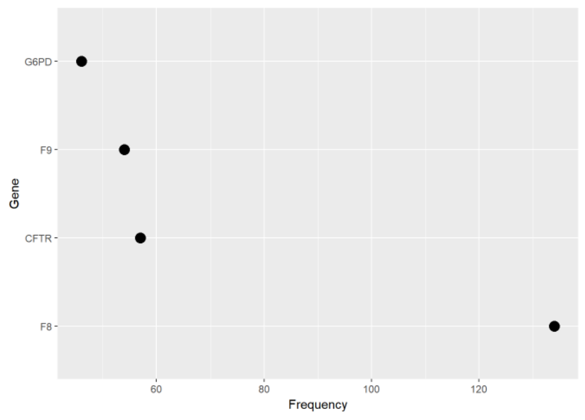


Figure 9. Frequency of most occurred Genes in test set

- A relatively small group of Gene levels make up a sizeable part of the feature values in both train and test data.
- The test data has fewer high-frequency Genes.

In the figure below, distribution of the Genes and their occurrences in training dataset is shown. It is worth noting that:

- BRCA1 is highly dominating Class 5
- SF3B1 is highly dominating Class 9
- BRCA1 and BRCA2 are dominating Class 6
- PTEN is predominately present in just Class 4.
- TP53 are mainly shared between Class 1 and Class 4.

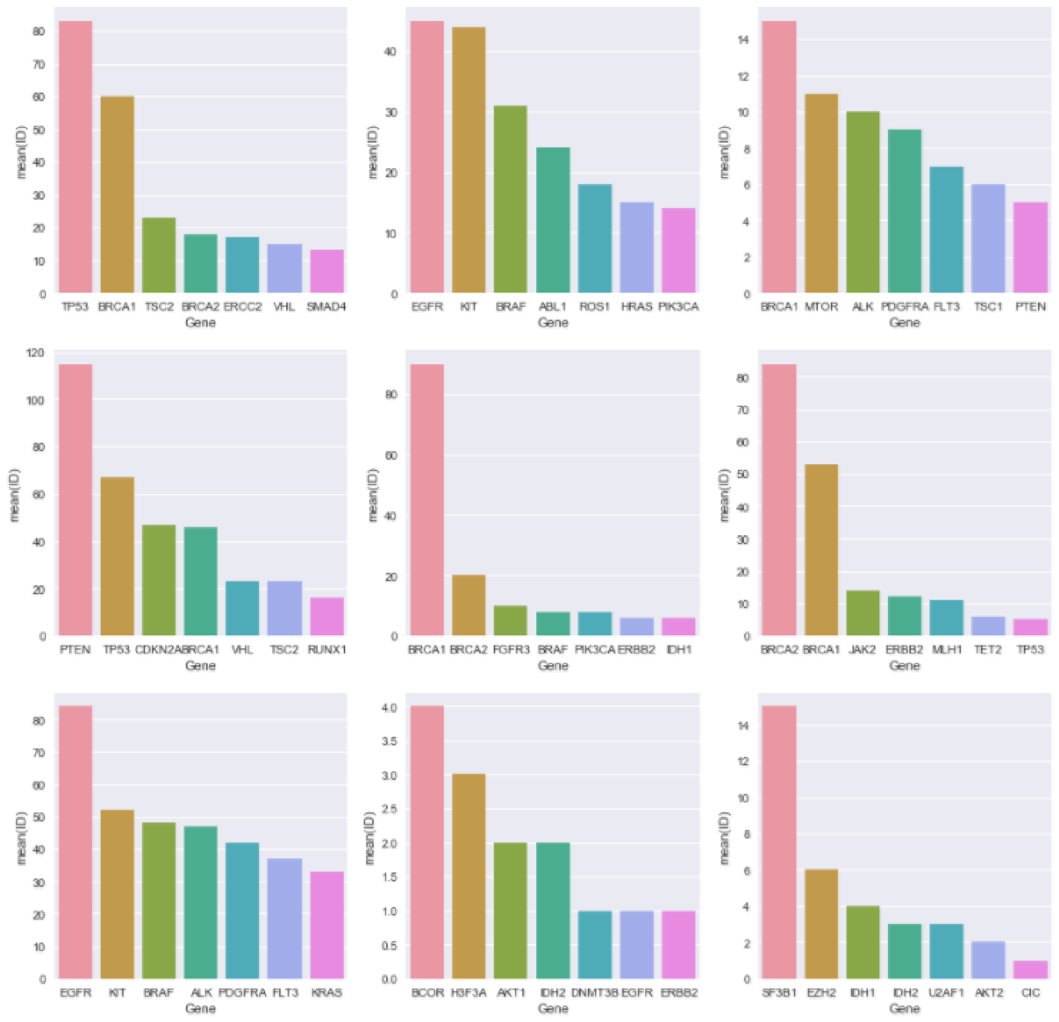


Figure 10. distribution of the Genes and their occurrences in training dataset per class

Just a small background check on these genes reveal that BRCA1 and BRCA2 are mostly responsible for Breast Cancer, whereas SF3B1 is SF3B1 is involved in 3'-splice site recognition during RNA splicing. Alterations in SF3B1 were initially discovered in myelodysplastic syndromes (MDS) and chronic lymphocytic leukemia (CLL), together with other mutations of splicing factors, such as U2AF1, SRSF2, and ZRSR2. It has been shown that SF3B1 is mutated in a significant proportion (~20%) of uveal melanoma (UM), a rare malignant entity deriving from melanocytes from the uveal tract and in other solid tumors at lesser frequencies). The phosphatase and tensin homolog deleted on chromosome 10 (PTEN) [106] tumor suppressor is a phosphatase that antagonizes the phosphoinositol-3-kinase/AKT signaling pathway and suppresses cell survival as well as cell proliferation. PTEN is the second most frequently mutated gene in human cancer after p53. Mutations in the PTEN gene are documented in cancers of the breast, prostate, endometrium, ovary, colon, melanoma, glioblastoma. and lymphoma. Animal models have shown that the loss of just one copy of the PTEN gene is enough to interrupt cell signaling and begin the process of uncontrolled cell growth. TP53 [105] is the most commonly mutated gene in cancers and is found to be mutated in the majority of cancers. This gives more insight on what the labels associated with each instance in the 'Class' feature represent. In the training dataset, each of the 2996 mutations is associated with 264 different Genes.

That means each mutation affects a Genome in certain kind and causes a different type of cancer which are categorized into 9 different classes. Some more research on these findings led to the information that a certain Gene might be the subject of several mutations. However, some mutations don't have any noticeable effect on the phenotype of an organism. This can happen in many situations: perhaps the mutation occurs in a stretch of DNA with no function, or perhaps the mutation occurs in a protein-coding region but ends up not affecting the amino acid sequence of the protein. On the other hand, if the mutation has caused an alteration in the amino acid sequence, most of the times it affects the phenotype.

2.3.1.2 Individual feature visualization – Mutation / Variation

The figure below illustrates the cumulative frequencies of the most important Variations present in both training and testing data.

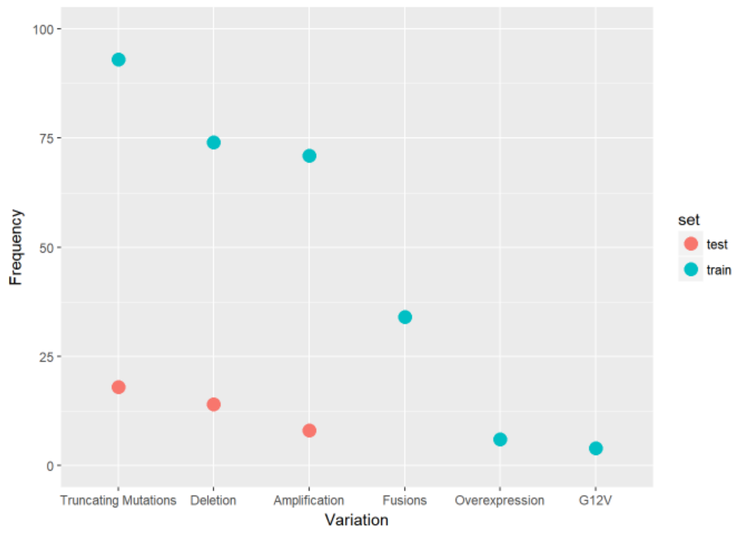


Figure 11.cumulative frequencies of Variations present training and testing data

Unlike the Gene feature, variations are common in both the training and testing sets, but the number of occurrences of this variation is higher in training set as compared to the testing set.

And finally, as it is already mentioned that each mutation is associated with certain Gene, this relation can be visualized by repurposing a count plot and the distribution of variations for the most frequent Genes can be observed. Since there are too many different variations, the labels of y-axis are dropped to have a better visual of how Gene-Variation combination exists in the data.

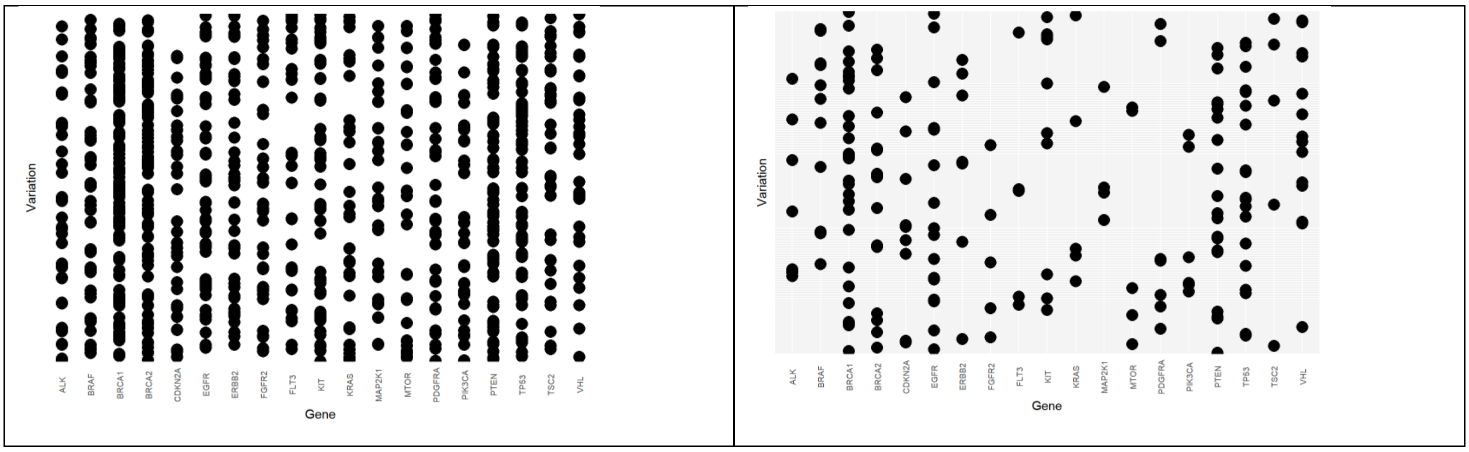
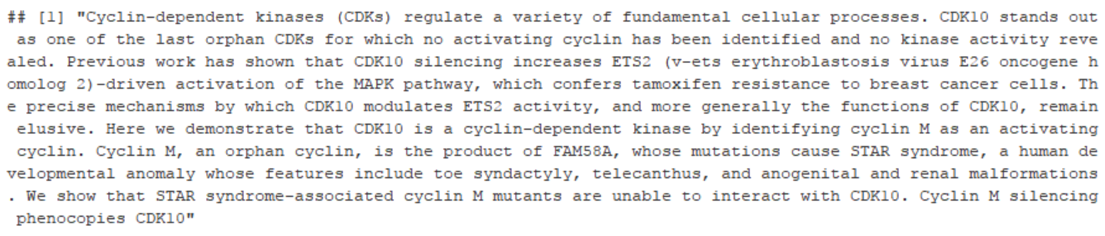


Figure 12. Distribution of Gene vs Mutation in train and test sets

2.3.2 The Text Files

2.3.2.1 Overview

The second kind of data files contains a whole lot of text from what looks like scientific papers or proceedings. The fig below shows the beginning of the first entry:



```
## [1] "Cyclin-dependent kinases (CDKs) regulate a variety of fundamental cellular processes. CDK10 stands out
 as one of the last orphan CDKs for which no activating cyclin has been identified and no kinase activity reve
aled. Previous work has shown that CDK10 silencing increases ETS2 (v-ets erythroblastosis virus E26 oncogene h
omolog 2)-driven activation of the MAPK pathway, which confers tamoxifen resistance to breast cancer cells. Th
e precise mechanisms by which CDK10 modulates ETS2 activity, and more generally the functions of CDK10, remain
 elusive. Here we demonstrate that CDK10 is a cyclin-dependent kinase by identifying cyclin M as an activating
 cyclin. Cyclin M, an orphan cyclin, is the product of FAM58A, whose mutations cause STAR syndrome, a human de
velopmental anomaly whose features include toe syndactyly, telecanthus, and anogenital and renal malformations
. We show that STAR syndrome-associated cyclin M mutants are unable to interact with CDK10. Cyclin M silencing
 phenocopies CDK10"
```

Figure 13. Text dump of the first instance in train set

Sure enough, it can be easily confirmed that the first part of the complete entry corresponds to this paper [103] and then later switches to this one [104]. Therefore, the data file appears to be a data dump of the complete publication texts for the papers that the classification was based on (including figure captions, manuscript structure, and sometimes affiliations).

Fig below shows the text length distribution of each class in training set. This was achieved by joining the variants file and the text file for training and testing, both the datasets, on 'ID' attribute and calculating the text length for each instance and categorizing them according to the 'Class' attribute.

The effects of the distribution of data per class as shown previously can also be seen here with Class 7, 2 1, and 4 having the highest text count in the mentioned order.

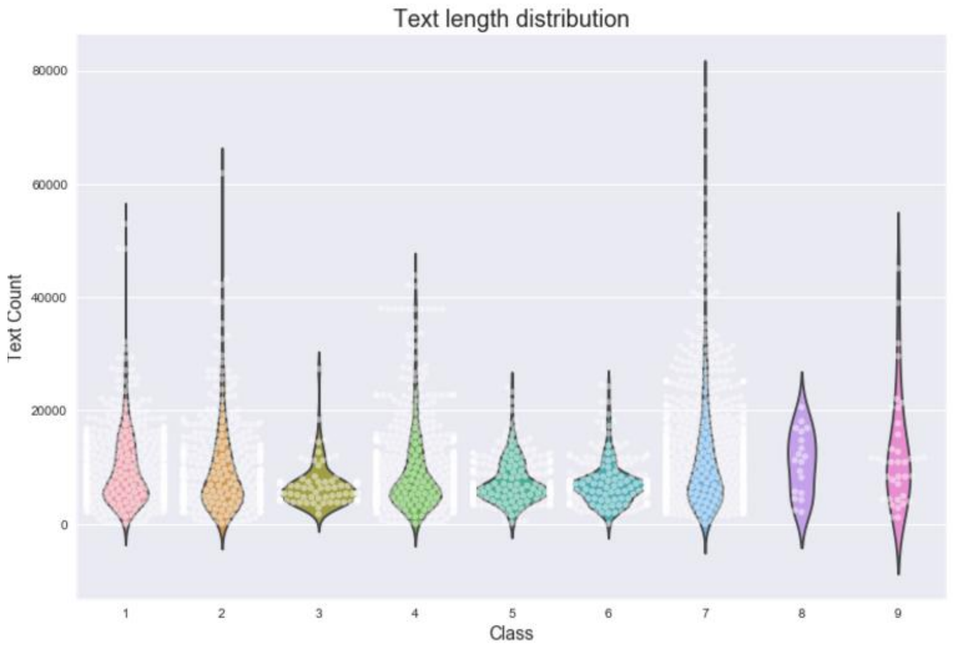
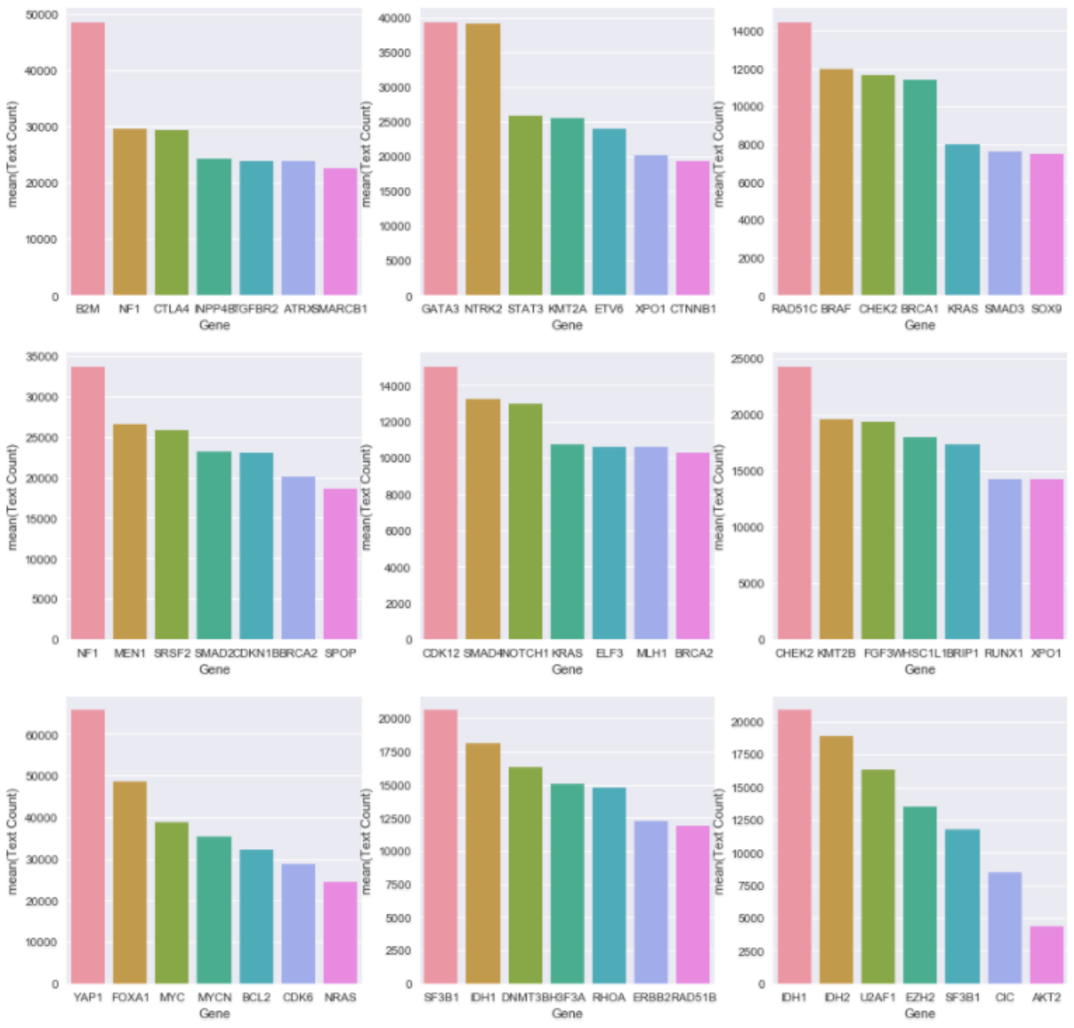


Figure 14. Text length distribution per class

The above-mentioned plot is a combination of violin plot and swarm plot from the seaborn package in python. It plays a similar role as a box and whisker plot. It shows the distribution of quantitative data across several levels of one (or more) categorical variables such that those distributions can be compared. Unlike a box plot, in which all the plot components correspond to actual data points, the violin plot features the kernel density estimations of the underlying distribution. A swarm plot draws a scatterplot but the points are adjusted (only along the categorical axis) so that they don't overlap. This gives a better representation of the distribution of values.

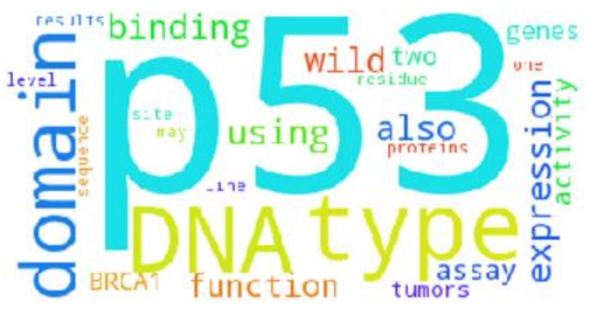
Fig below shows bar plots between the top 7 Genes in each class that has the highest word count. This can mean that to classify the Genes in their respective classes based on the mutations that they have, the most amounts of study was done by researchers as the result of their research has the most number of words. Although these Genes have the highest word counts in their respective classes it doesn't necessarily mean that these are the most important genes in their class because correlation does not mean causation.



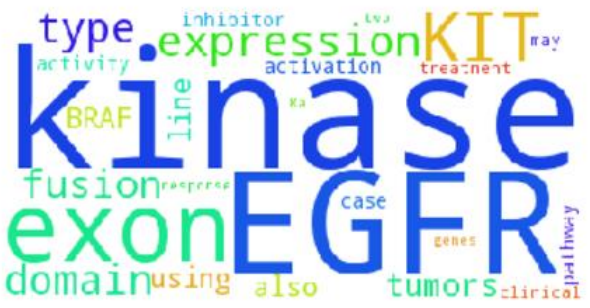
2.3.2.2 Data cleaning and preprocessing

The first part of preprocessing involves creating a list of stop-words that is customized according to medical research data i.e. that includes words that may be common in medical terms but cannot be found in any packages made available for stop-words by Python. While creating this list, a balance had to be maintained between the words being included in the stop-words without any information loss from the actual corpus of the data. So, finally doing the previous steps, the 9 words clouds are given below which represent the most common 25 words in each class.

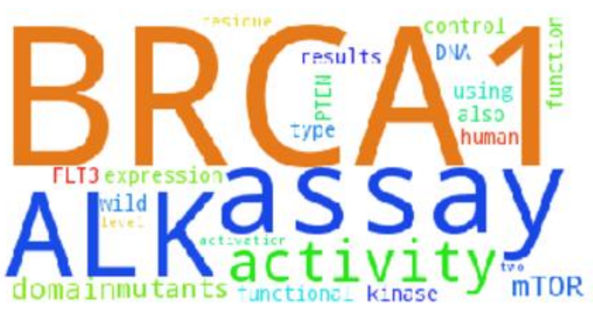
Class 1



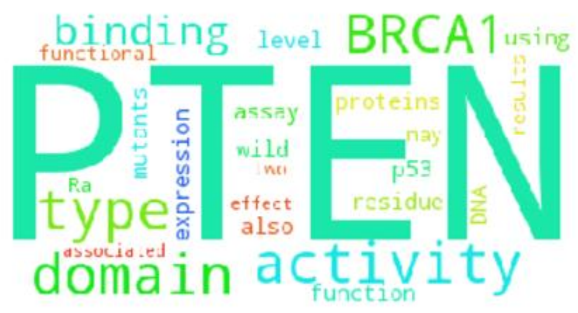
Class 2



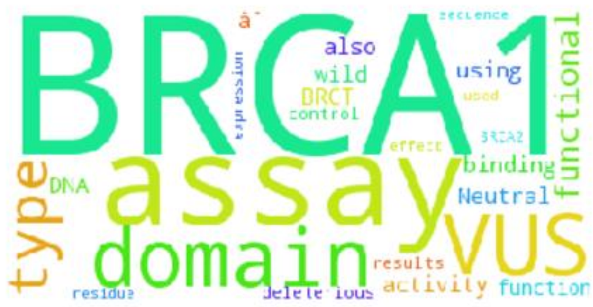
Class 3



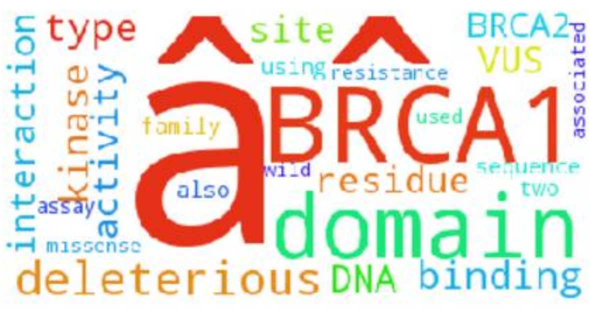
Class 4



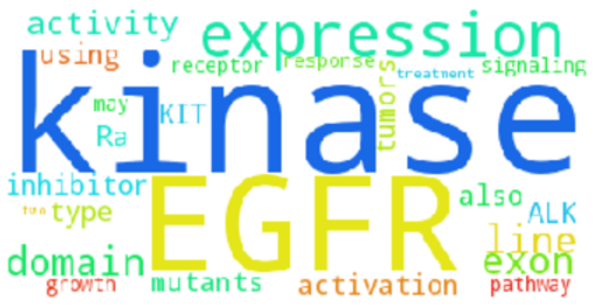
Class 5



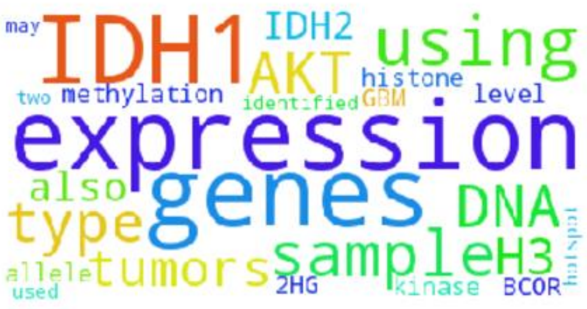
Class 6



Class 7



Class 8



Class 9

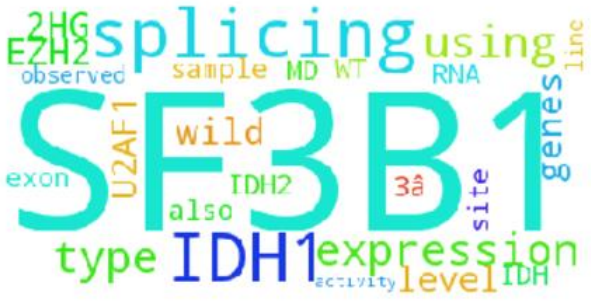


Figure 15. Word clouds - top 25 words per class

Even though text data has been presented in the form of word clouds, it still makes no sense if it must be used in prediction of test data. To make use of the text feature given in the dataset, it has to be processed via the concept of Natural Language Processing, which in brief means building a system that can 'understand' language. When NLP is applied with machine learning it builds systems that can learn how to understand language. In the current context, using NLP features with machine learning will help the prediction models in understanding why the classification of Genes having certain mutations was done in 9 different classes. The more features that are extracted using NLP, the better the understanding becomes the model which ultimately helps in predicting accurately the new data into the desired classes.

The focus here is to build a classification system which utilizes features derived from information retrieval, information extraction and NLP techniques [1]. The model then combines these features and learns them using a machine learning algorithm. To perform these steps there is a pattern for text processing which includes:

- Creating a text corpus

- Removing punctuations
- Removing Stopwords
- Stemming of text corpus
- Remove numbers
- Transform text to lower space and stripping white space (optional)

In this case, each entry in the 'Text' feature associated with a single instance can serve as a single document as it has already been previously discussed that each text entry is a data dump of research papers. So, the steps included here to perform text processing is to remove any character other than A-Z, a-z, 0-9 and certain special symbol. Each document in the text corpus is then converted to lowercase and split. On most occasions, a set of predefined stop words can be used eliminate words that do not cause any kind of information loss, but as this data belongs to medical research, a set of stop words was created which was then used to check whether each word in the text corpus belonged to this set or not, if yes, it was eliminated from the corpus. All the words were then again joined to return a list of words that had been now cleaned and ready to be used.

The figure below displays the top twenty counts of most common words among text. The plot obtained is left skewed and more filtering was done to balance the plot.

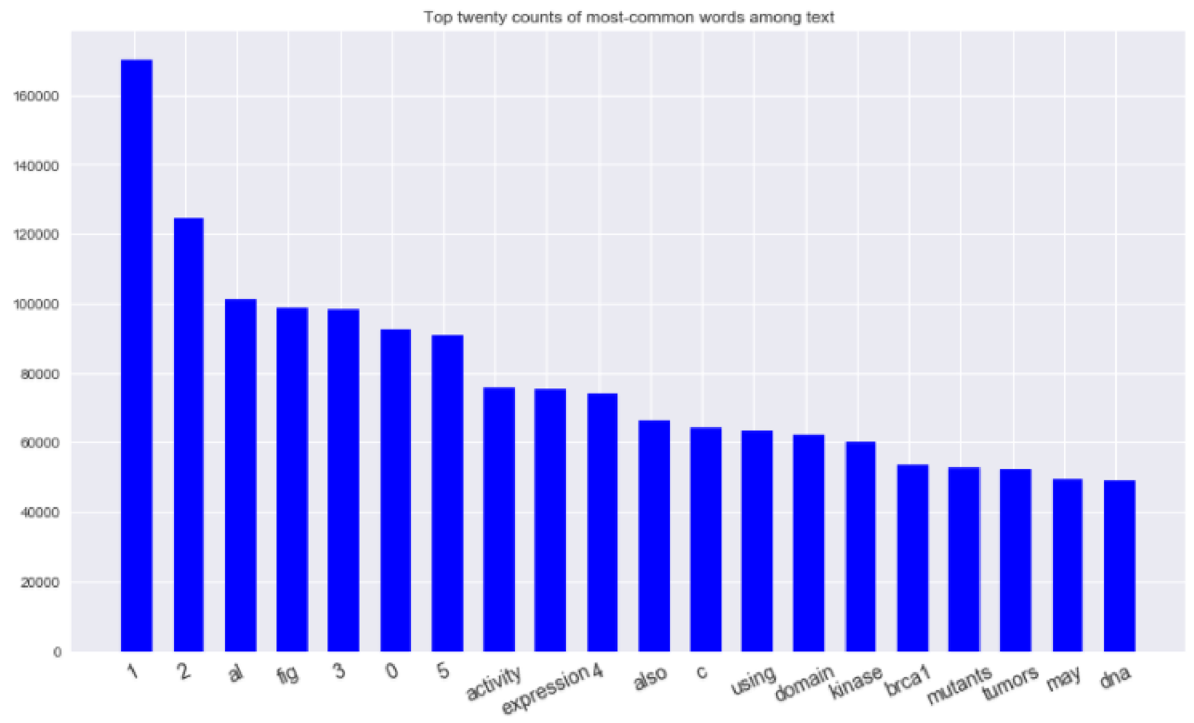


Figure 16. top twenty counts of most common words among text

Fig below shows the plot obtained when filtering was done again on the text corpus. Here the plot seems to be little normalized than the previous one.

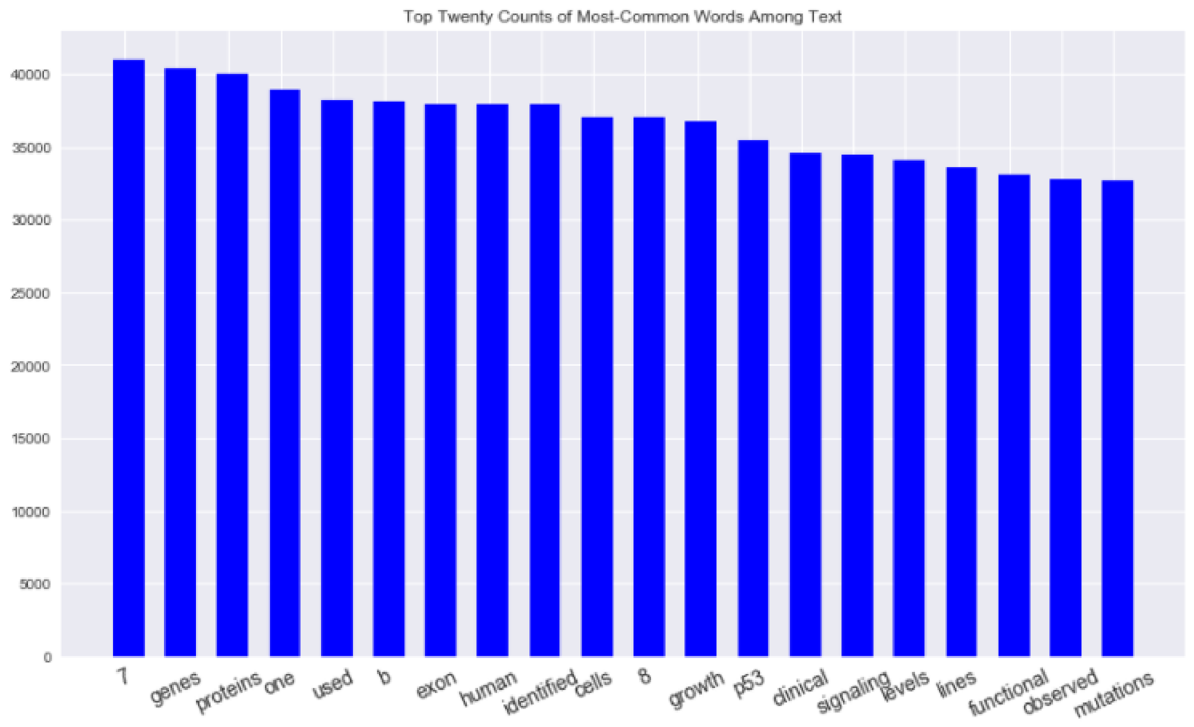


Figure 17. top twenty counts of most common words among text - after filtering

The most common approach to extract features from text corpus is to calculate the tf-idf scores of the words in the corpus and take the top-n features by concatenating them to the existing features. Here this limit of maximum features is set to 500.

Python's sk*learn* package contains TfidfVectorizer [2] and CountVectorizer[3] which can be used in the program by importing these functionalities from *sklearn.feature_extraction.text* package. TfidfVectorizer transforms text to feature vectors that can be used to input to the estimator, e.g.: u'me':8 represents that token 'me' is represented as feature number 8 in the output matrix. CountVectorizer converts a collection of text documents to a matrix of token counts. The implementation produces a sparse representation of the counts using *scipy.sparse.csr_matrix*.

The most common issue that arises whenever one or more dimensions are introduced into the dataset as features are 'the curse of dimensionality'. This issue is explained in detail in [4] paragraph 6, 'Intuition fails in High Dimensions'. Imagine a child has only 3 toys –

- a blue soccer ball
- a blue freesbe
- and a green cube.

The initial hypothesis regarding how a toy can be made:

- Possible colors are: red, green, blue
- Possible shapes are: circle, square, triangle

Num_colors * num_shapes = 3 * 3 = 9 possible ways of creating a toy. The child would group (cluster) the toys as:

- Cluster A – contains the blue ball and the blue freesbe, because they have the same color and shape
- Cluster B – contains the green cube.

Using only these 2 dimensions, 2 non-empty clusters are created: so, in this case, 7/9 ~ 77% of the space is empty. If the dimensions that the child should consider are increased, the initial hypothesis will also change into:

- Size of the toy can vary between few centimeters to 1 meter, in step of ten centimeters: 0-10cm, 11-20cm and so on.
- The weight of the toy can vary in a similar manner up to 1 kilogram, with steps of 100grams: 0-100g, 101-200g and so on.

Now, if the clusters are to be created on all the possible ways of creating a toy, it becomes (num_colors * num_shapes * num_sizes * num_weights) = 3 * 3 * 10 * 10 = 900 possible clusters. The child will now have to cluster the toys as follows:

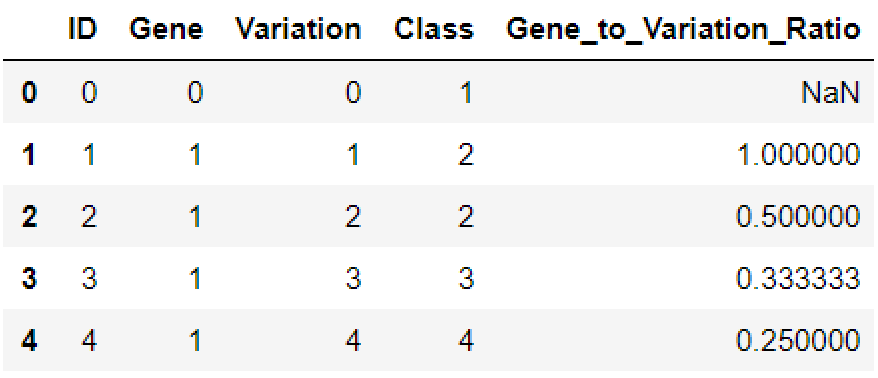
- Cluster A – contains the blue soccer ball because it is blue and heavy
- Cluster B – contains the blue freesbe because it is blue and light
- Cluster C – contains the green cube

Using the current 4 dimensions (shape, color, size, weight) only 3 clusters are non-empty: so, in this case, 897/900 ~ 99.7% of the space is empty. Therefore, when the dimensionality increases, the volume of the space increases so fast that the available data becomes sparse [5].

The solutions to overcome these problems are increasing the amount of data by using Synthetic Minority Oversampling Technique (SMOTE) [6], Principle Component Analysis (PCA) [7], or the third technique that has been used here, Truncated SVD [8]. This transformer performs linear dimensionality reduction by means of truncated singular value decomposition (SVD). Contrary to PCA, this estimator does not center the data before computing the singular value decomposition. This means it can work with scipy.sparse matrices efficiently. In particular, truncated SVD works on term count/tf-idf matrices as returned by the vectorizers in sklearn.feature_extraction.text. In that context, it is known as latent semantic analysis. The number of components that have been set to 390 means that truncated SVD will create a dimension space of 390 features. This is brute-force approach and the value of 390 was thought to be the best after going through several iterations and looking at the accuracy of the predictive model.

2.3.2.3 Feature Building

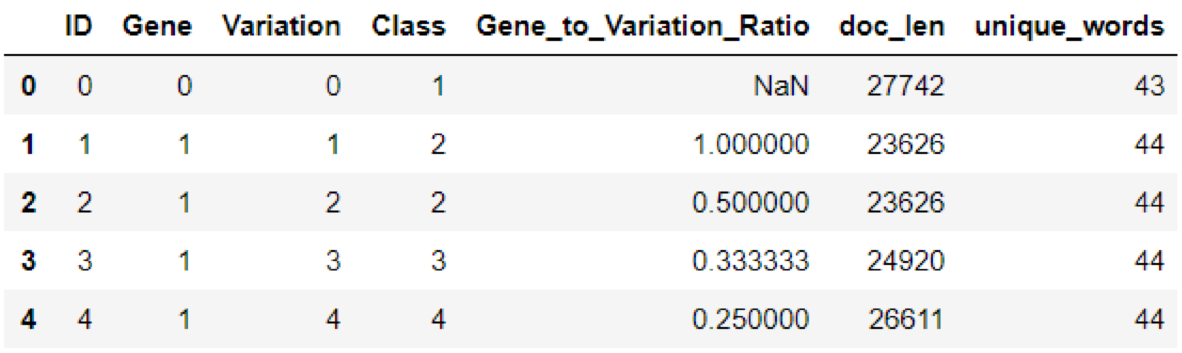
After all, the previously mentioned steps are performed, the next step involves the building of features in such a way that it becomes compatible with our classification algorithm. So, the first thing to do is to encode the Gene and Variation attribute from strings to integers. Also, a new feature 'Gene to Variation Ratio' is added in the dataframe, which then becomes:



	ID	Gene	Variation	Class	Gene_to_Variation_Ratio
0	0	0	0	1	NaN
1	1	1	1	2	1.000000
2	2	1	2	2	0.500000
3	3	1	3	3	0.333333
4	4	1	4	4	0.250000

Figure 18. Feature Building - Step 1

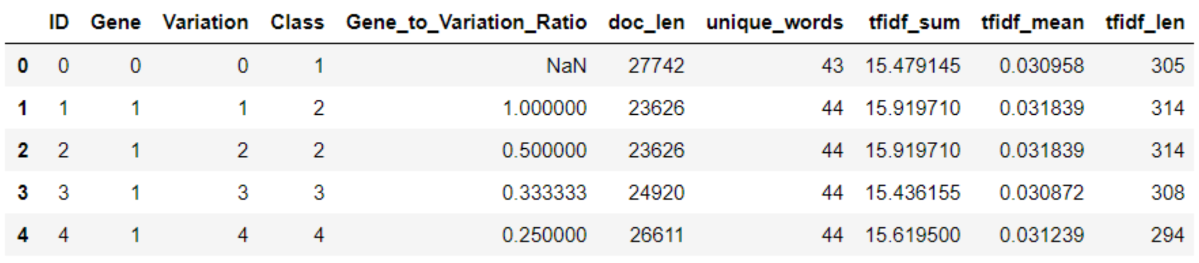
The Genes and variation in the dataframe have been replaced by a unique label associated with each Gene and its variation. The next step involves adding two more features in the dataframe which is, 'document length' and 'unique words' in each document. As previously the length of each entry in the Text attribute is so large, that it can be considered as a document itself. Thus, building upon the same concept, these two features are added making the dataframe as:



	ID	Gene	Variation	Class	Gene_to_Variation_Ratio	doc_len	unique_words
0	0	0	0	1	NaN	27742	43
1	1	1	1	2	1.000000	23626	44
2	2	1	2	2	0.500000	23626	44
3	3	1	3	3	0.333333	24920	44
4	4	1	4	4	0.250000	26611	44

Figure 19. Feature Building - Step 2

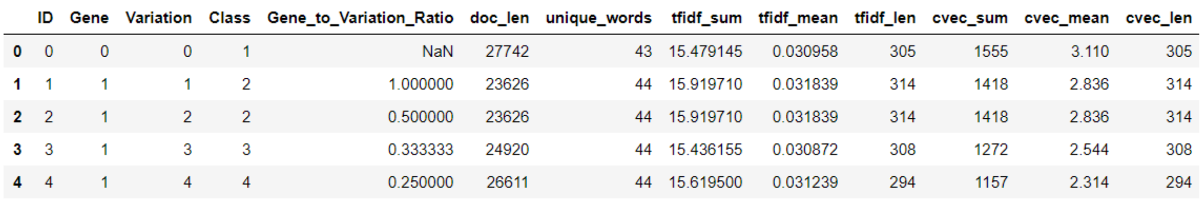
The next part involves using some information by calculating the TF-IDF scores and using the sum, mean and length produced by transformation done by TF-IDF on the list of documents as features in the dataset. After the transformation is applied, a scipy.sparse matrix is received having a shape of 3321 rows and 500 columns. The three attributes are then added and the dataframe becomes:



	ID	Gene	Variation	Class	Gene_to_Variation_Ratio	doc_len	unique_words	tfidf_sum	tfidf_mean	tfidf_len
0	0	0	0	1	NaN	27742	43	15.479145	0.030958	305
1	1	1	1	2	1.000000	23626	44	15.919710	0.031839	314
2	2	1	2	2	0.500000	23626	44	15.919710	0.031839	314
3	3	1	3	3	0.333333	24920	44	15.436155	0.030872	308
4	4	1	4	4	0.250000	26611	44	15.619500	0.031239	294

Figure 20. Feature Building - Step 3

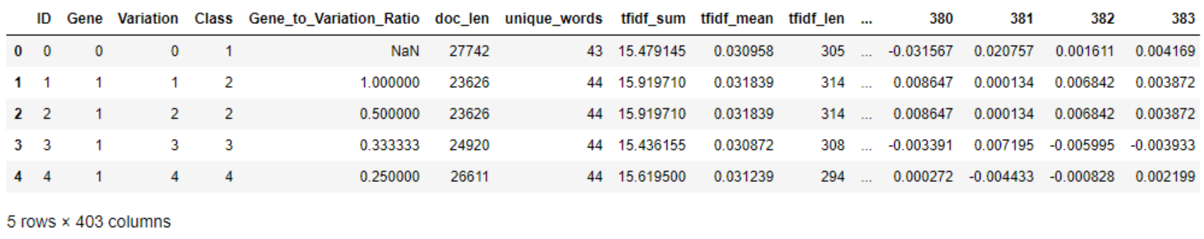
The Same process is repeated with Count Vectorizer thus changing the dataframe as:



	ID	Gene	Variation	Class	Gene_to_Variation_Ratio	doc_len	unique_words	tfidf_sum	tfidf_mean	tfidf_len	cvec_sum	cvec_mean	cvec_len
0	0	0	0	1	NaN	27742	43	15.479145	0.030958	305	1555	3.110	305
1	1	1	1	2	1.000000	23626	44	15.919710	0.031839	314	1418	2.836	314
2	2	1	2	2	0.500000	23626	44	15.919710	0.031839	314	1418	2.836	314
3	3	1	3	3	0.333333	24920	44	15.436155	0.030872	308	1272	2.544	308
4	4	1	4	4	0.250000	26611	44	15.619500	0.031239	294	1157	2.314	294

Figure 21. Feature Building - Step 4

The last step involves performing truncated SVD or better known as latent semantic analysis. Singular value decomposition transformation is applied on the scipy.sparse TF-IDF matrix which gives out a numpy.ndarray. This numpy.ndarray is then converted into a pandas dataframe and is concatenated with the original dataframe, which gives out this:

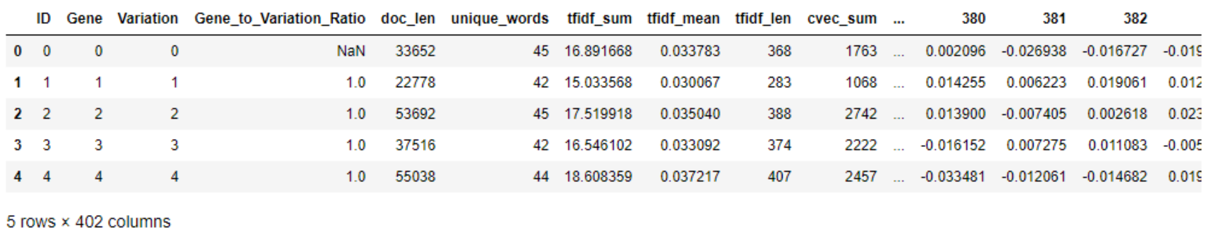


	ID	Gene	Variation	Class	Gene_to_Variation_Ratio	doc_len	unique_words	tfidf_sum	tfidf_mean	tfidf_len	...	380	381	382	383
0	0	0	0	1	NaN	27742	43	15.479145	0.030958	305	...	-0.031567	0.020757	0.001611	0.004169
1	1	1	1	2	1.000000	23626	44	15.919710	0.031839	314	...	0.008647	0.000134	0.006842	0.003872
2	2	1	2	2	0.500000	23626	44	15.919710	0.031839	314	...	0.008647	0.000134	0.006842	0.003872
3	3	1	3	3	0.333333	24920	44	15.436155	0.030872	308	...	-0.003391	0.007195	-0.005995	-0.003933
4	4	1	4	4	0.250000	26611	44	15.619500	0.031239	294	...	0.000272	-0.004433	-0.000828	0.002199

5 rows × 403 columns

Figure 22. Feature Building - Step 5

The same steps are followed with the testing dataset, and the final dataset comes out to be:

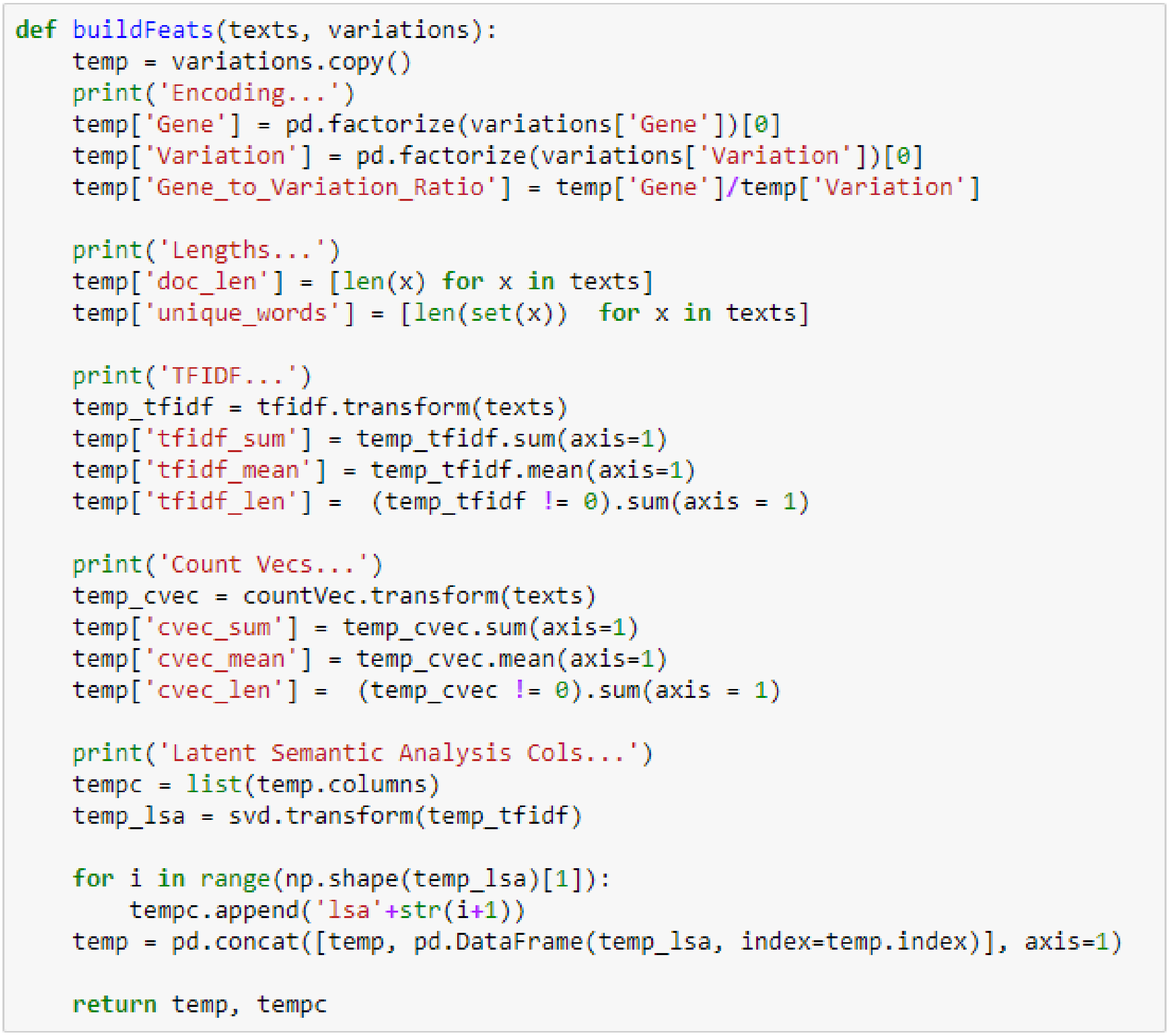


	ID	Gene	Variation	Gene_to_Variation_Ratio	doc_len	unique_words	tfidf_sum	tfidf_mean	tfidf_len	cvec_sum	...	380	381	382	
0	0	0	0	NaN	33652	45	16.891668	0.033783	368	1763	...	0.002096	-0.026938	-0.016727	-0.01
1	1	1	1	1.0	22778	42	15.033568	0.030067	283	1068	...	0.014255	0.006223	0.019061	0.01
2	2	2	2	1.0	53692	45	17.519918	0.035040	388	2742	...	0.013900	-0.007405	0.002618	0.02
3	3	3	3	1.0	37516	42	16.546102	0.033092	374	2222	...	-0.016152	0.007275	0.011083	-0.00
4	4	4	4	1.0	55038	44	18.608359	0.037217	407	2457	...	-0.033481	-0.012061	-0.014682	0.01

5 rows × 402 columns

Figure 23. Final dataset after feature building

Obviously, on comparison, the test dataset created has one less feature, which is the 'Class' feature because it is the attribute that needs to be predicted. The implementation of the entire process described above is given below.



```
def buildFeats(texts, variations):
    temp = variations.copy()
    print('Encoding...')
    temp['Gene'] = pd.factorize(variations['Gene'])[0]
    temp['Variation'] = pd.factorize(variations['Variation'])[0]
    temp['Gene_to_Variation_Ratio'] = temp['Gene']/temp['Variation']

    print('Lengths...')
    temp['doc_len'] = [len(x) for x in texts]
    temp['unique_words'] = [len(set(x))  for x in texts]

    print('TFIDF...')
    temp_tfidf = tfidf.transform(texts)
    temp['tfidf_sum'] = temp_tfidf.sum(axis=1)
    temp['tfidf_mean'] = temp_tfidf.mean(axis=1)
    temp['tfidf_len'] =  (temp_tfidf != 0).sum(axis = 1)

    print('Count Vecs...')
    temp_cvec = countVec.transform(texts)
    temp['cvec_sum'] = temp_cvec.sum(axis=1)
    temp['cvec_mean'] = temp_cvec.mean(axis=1)
    temp['cvec_len'] =  (temp_cvec != 0).sum(axis = 1)

    print('Latent Semantic Analysis Cols...')
    tempc = list(temp.columns)
    temp_lsa = svd.transform(temp_tfidf)

    for i in range(np.shape(temp_lsa)[1]):
        tempc.append('lsa'+str(i+1))
    temp = pd.concat([temp, pd.DataFrame(temp_lsa, index=temp.index)], axis=1)

    return temp, tempc
```

Figure 24. Feature Building - code snippet

Apart from the dataframe, the above-mentioned function also returns a list of column names where the features obtained after performing truncated SVD are appended with string 'lsa' just to have an idea that these are derived by applying latent semantic analysis the list of text data obtained previously. As this process is repeated for training and testing data, the column names are assigned to the columns of the dataframes

2.4 Classification

2.4.1 Boosting Algorithms

The term 'Boosting' [9] refers to a family of algorithms which convert weak learner to strong learner [10]. When the rules created by an algorithm are not powerful enough to classify an instance correctly, such type of rules is called weak learners. To convert weak learners to the strong learner, the prediction of each weak learner is combined using methods like:

- Using average / weighted average
- Considering which prediction has higher vote

There are several types of boosting algorithms and the underlying engine used for boosting algorithms can be anything. It can be a decision stump, margin-maximising classification algorithm, etc. There are many boosting algorithms which use other types of engine such as:

- AdaBoost (Adaptive Boosting) – it was the first boosting techniques that could correctly classify a problem pertaining to binary classification. But the downside of this is its overfitting behavior [11]. Due to this reason and the classification in question is a multinomial model, this was discarded.
- Gradient Tree Boosting – it trains many models sequentially. Each new model gradually minimizes the loss function ($y = ax + b + e$, 'e' needs special attention as it is an error term) of the whole system using Gradient Descent [12] method. The learning procedure consecutively fits new models to provide a more accurate estimate of the response variable. But this model was not chosen due to memory constraints of the system being used.
- XGBoost – it is short for 'Extreme Gradient Boosting' where the term "Gradient Boosting is proposed in the paper *Greedy Function Approximation: A Gradient Boosting Machine,* by Friedman [13]. XGBoost has become the de-facto algorithm for

winning competitions at Kaggle and other such platforms, simply because it is extremely powerful. But given lots of data, even XGBoost takes a long time to train. To improve upon this shortcoming LightGBM is introduced.

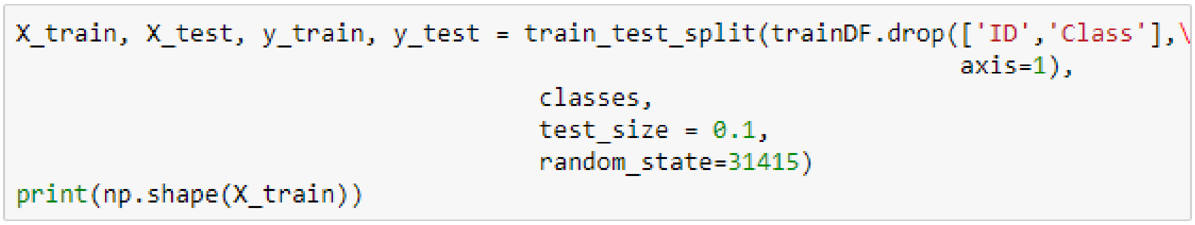
2.4.2 LightGBM

Light GBM is a fast, distributed, high-performance gradient boosting framework based on decision tree algorithm, used for classification. It has:

- faster training speed and higher efficiency
- lower memory usage
- better accuracy than any other boosting algorithm
- compatibility with large datasets
- parallel learning supported

For the given prediction task, LightGBM is used so that it consumes less memory and time. First, the training dataset is split into four sets:

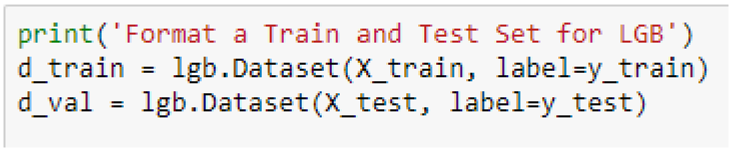
- X_train: the training data that will help the classifier learn the rules
- y_train: the labels associated with the instances in X_train
- X_test: the testing data using which the prediction for the labels will be done utilizing the rules learned while performing training phase of the model
- y_test: the correct labels for the test set, which will be used to calculate the accuracy score of the model, so that it can be used to classify the actual test dataset.



```
X_train, X_test, y_train, y_test = train_test_split(trainDF.drop(['ID','Class'],\
                                                                  axis=1),
                                                    classes,
                                                    test_size = 0.1,
                                                    random_state=31415)
print(np.shape(X_train))
```

Figure 25. test train split

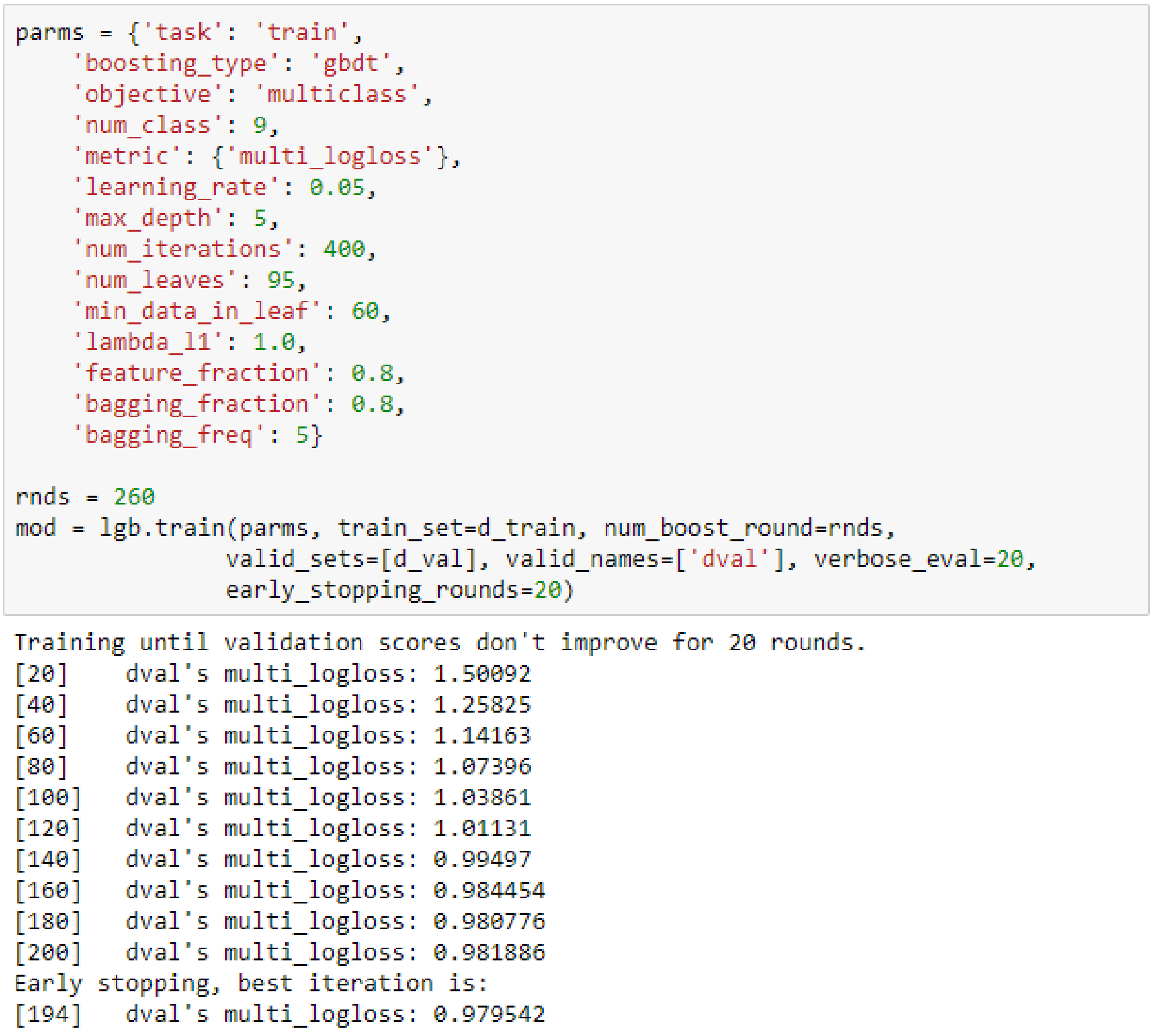
The approach for implementing this algorithm is different from other traditional machine learning approaches. Before allowing the model to learn the rules of classification, a training and testing set should be formatted for LGB. The implementation of this step is given in the figure below:



```
print('Format a Train and Test Set for LGB')
d_train = lgb.Dataset(X_train, label=y_train)
d_val = lgb.Dataset(X_test, label=y_test)
```

Figure 26. LightGBM model fitting

After obtaining the training set it is used to train the classifier with a number of other parameters which have been discussed below:



```
parms = {'task': 'train',
    'boosting_type': 'gbdt',
    'objective': 'multiclass',
    'num_class': 9,
    'metric': {'multi_logloss'},
    'learning_rate': 0.05,
    'max_depth': 5,
    'num_iterations': 400,
    'num_leaves': 95,
    'min_data_in_leaf': 60,
    'lambda_l1': 1.0,
    'feature_fraction': 0.8,
    'bagging_fraction': 0.8,
    'bagging_freq': 5}

rnds = 260
mod = lgb.train(parms, train_set=d_train, num_boost_round=rnds,
               valid_sets=[d_val], valid_names=['dval'], verbose_eval=20,
               early_stopping_rounds=20)
```

```
Training until validation scores don't improve for 20 rounds.
[20]    dval's multi_logloss: 1.50092
[40]    dval's multi_logloss: 1.25825
[60]    dval's multi_logloss: 1.14163
[80]    dval's multi_logloss: 1.07396
[100]   dval's multi_logloss: 1.03861
[120]   dval's multi_logloss: 1.01131
[140]   dval's multi_logloss: 0.99497
[160]   dval's multi_logloss: 0.984454
[180]   dval's multi_logloss: 0.980776
[200]   dval's multi_logloss: 0.981886
Early stopping, best iteration is:
[194]   dval's multi_logloss: 0.979542
```

Figure 27. LightGBM – calculating log loss

A dictionary is supplied to train the model consisting of core parameters having the format as ‘key1=value1 key2=value2 . .’ [14]:

Parameter	Default	Type	Options
task	train	enum	train prediction
boosting (boost, boosting_type)	gbdt	enum	gbdt rf dart goss
application (objective, app)	regression	enum	regression regression_l1 huber fair poisson lambdarank multiclass
num_class	1	int	
metric	{l2 for regression}, {binary_logloss for binary classification}, {ndcg for lambdarank}	multi-enum	l1 l2 ndcg auc binary_logloss binary_error multi_logloss multi_error
learning_rate	0.1	double	
max_depth	-1	Int	Limit the max depth for tree model
num_iterations (num_iteration, num_tree, num_trees,	100	Int	Number of boosting iterations

num_rounds, num_round)			
num_leaves (num_leaf)	31	Int	Number of leaves in one tree
min_data_in_leaf (min_data_per_leaf, min_data)	20	Int	Minimal number of data in one leaf can be used to deal with over-fit
lambda_l1	0	Double	L1 regularization
feature_fraction (sub_feature)	1.0	Double	0.0 < feature_fraction < 1.0 LightGBM will random select part of features on each iteration if feature_fraction smaller than 1.0. e.g.: if set to 0.8, will select 80% features before training each tree. Can be used to speed up training and to deal with overfitting.
bagging_fraction	1.0	double	0.0 < bagging_fraction < 1.0 Like feature_fraction, but this will random select part of data.
bagging_freq	0	Int	The frequency for bagging, 0 means

			disable bagging, k means will perform bagging at every k iteration.
early_stopping_round (early_stopping_rounds, early_stopping)	0	Int	Will stop training of one metric of one validation data doesn't improve in last early_stopping_round rounds.

Figure 28. LightGBM parameters

2.4.2.1 Evaluation Metric – multi-class Logarithmic Loss (Log loss)

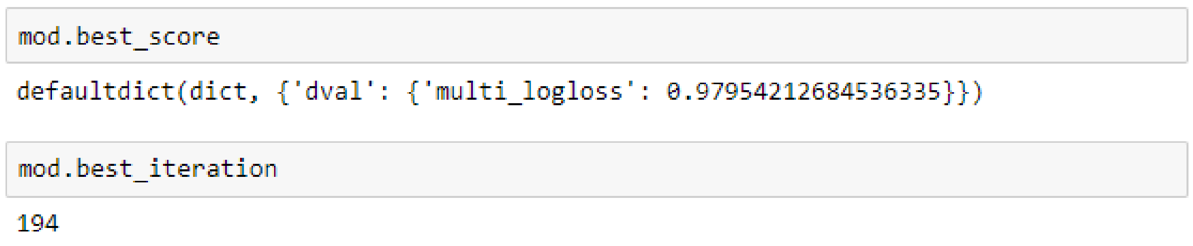
It is a classification loss function [15] that quantifies the accuracy of a classifier by penalizing false classification. Minimising the log loss is basically equivalent to maximising the accuracy of the classifier. Here is the equation to calculate logloss:

$$logloss = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{i,j}\log(p_{i,j}) \qquad \text{...(1)}$$

- N is the number of observations
- M is the number of class labels
- log is the natural logarithm
- $y_{i,j}$ is 1 if observation I is in class j and otherwise 0
- $p_{i,j}$ is the predicted probability that the observation I is in class j

Log Loss heavily penalizes classifiers that are confident about and incorrect classification. For example, if for an observation, the classifier assigns a very small probability to the correct class then the corresponding contribution to the Log Loss will be very large indeed.

The parameters discussed above will be used in training the model in each iteration. The total number of iterations that will occur is set to 260, but LightGBM will stop the iterations early if the evaluation metric that is log loss is not being minimised in the next iteration. So, in the results obtained LightGBM has concluded iteration number 194 to be the best iteration with the minimal log loss of 0.979542. It means that log loss cannot be minimised beyond this number. Also, it doesn't complete its 260 iterations and stops after 200 iterations as the results are not improving. These statistics can be seen in the figure below:



```
mod.best_score
```

```
defaultdict(dict, {'dval': {'multi_logloss': 0.97954212684536335}})
```

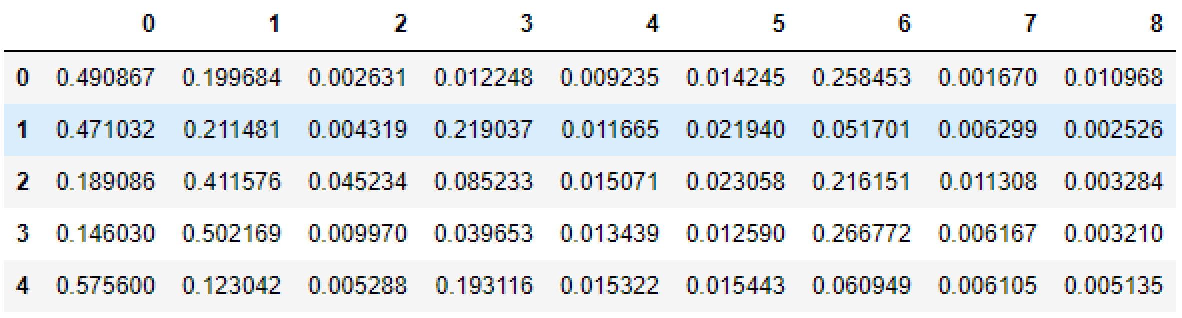
```
mod.best_iteration
```

```
194
```

Figure 29. LightGBM – results

2.4.2.2 Prediction

Using the model that has been trained, predictions are made on the test data. The results obtained on the test dataset does not specifically mention to which Class the instance belongs to, but predicts the probability of that instance to which Class it belongs. This approach is far more better than associating an instance to a class label i.e. instance 1 belongs to Class 5 or instance 100 belongs to Class 8 because by calculating the probabilities the results can show that a mutation in a particular gene can be classified into a Class with some specific probability and to another class with some another probability which can give researchers an idea of where to start in treatment of this Cancer. The figure below shows the probabilities calculated by the model on test dataset for the first 5 instances.



	0	1	2	3	4	5	6	7	8
0	0.490867	0.199684	0.002631	0.012248	0.009235	0.014245	0.258453	0.001670	0.010968
1	0.471032	0.211481	0.004319	0.219037	0.011665	0.021940	0.051701	0.006299	0.002526
2	0.189086	0.411576	0.045234	0.085233	0.015071	0.023058	0.216151	0.011308	0.003284
3	0.146030	0.502169	0.009970	0.039653	0.013439	0.012590	0.266772	0.006167	0.003210
4	0.575600	0.123042	0.005288	0.193116	0.015322	0.015443	0.060949	0.006105	0.005135

Figure 30. LightGBM - results

As it can be seen, instance 0 has a probability of 49% to be labeled as Class 1 cancer. Instance 1 has 47 % probability to be labeled as Class 1. Instance 2 has 41% probability to be labeled as Class 2 cancer. Instance 3 has 50% probability of being a Class 2 cancer instance 4 has 57% probability of being a Class 1 cancer.

Finally, the LightGBM classifier gives a feature importance plot that shows which feature had played the most important part in classifying the instances and calculating the minimum log loss that was obtained in the results. This plot can be seen below:

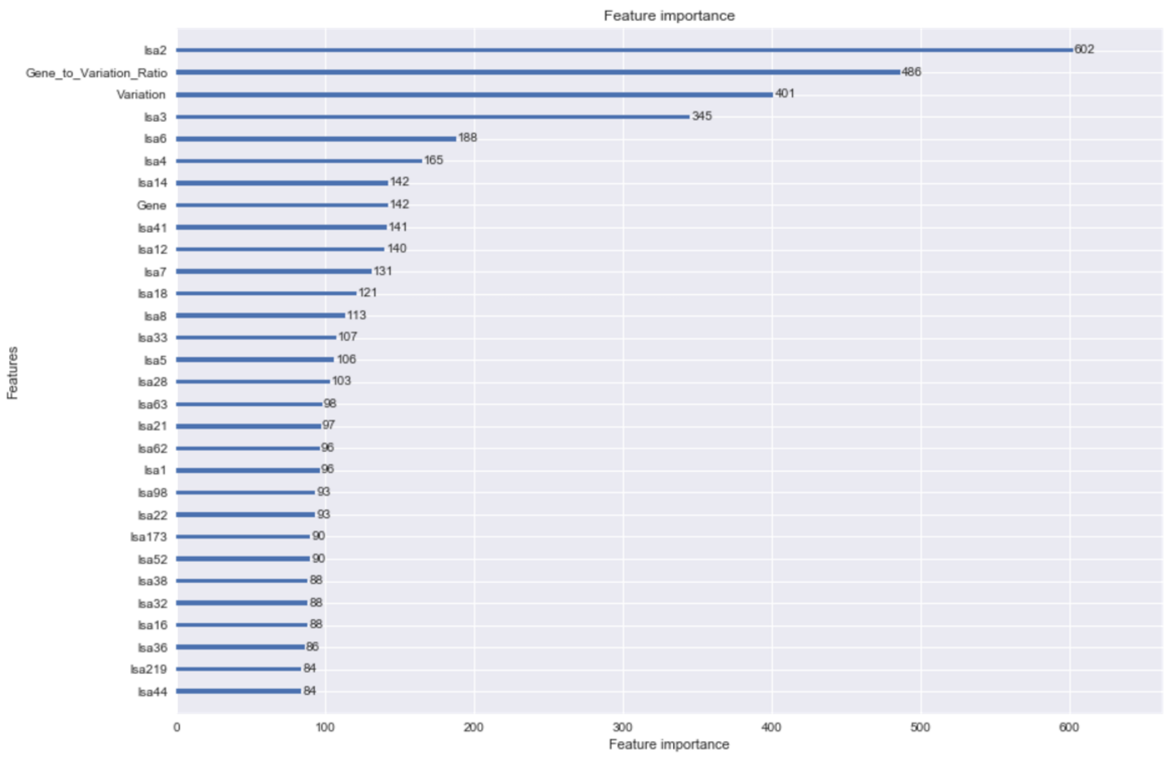


Figure 31. LightGBM - feature importance

Issue: The decision tree constructed by LightGBM at iteration number 194 cannot be plotted because of an issue in the python package installed for LightGBM. More about this issue is available on GitHub [16].

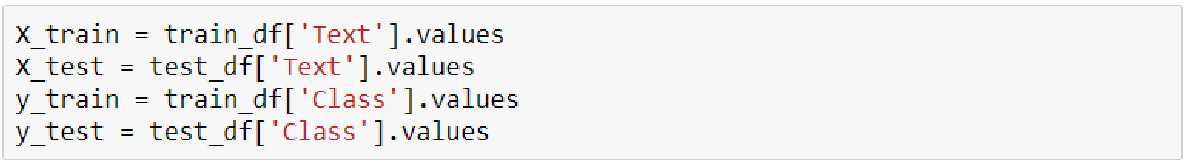
2.4.3 Linear SVC

Support Vector Machines (SVMs) are a set of supervised learning methods used for classification, regression and outliers' detection. The advantages of support vector machines are:

- effective in high dimensional spaces
- still effective in cases where the number of dimensions is greater than the number of samples
- uses a subset of training points in the decision function (called support vectors) so it is also memory efficient
- Versatile: different Kernel functions can be specified for the decision function [17].

Linear Support Vector Classification is like SVC with parameter kernel='linear', but implemented in terms of liblinear rather than libsvm, so it has more flexibility in the choice of penalties and loss functions and scales better to large numbers of samples.

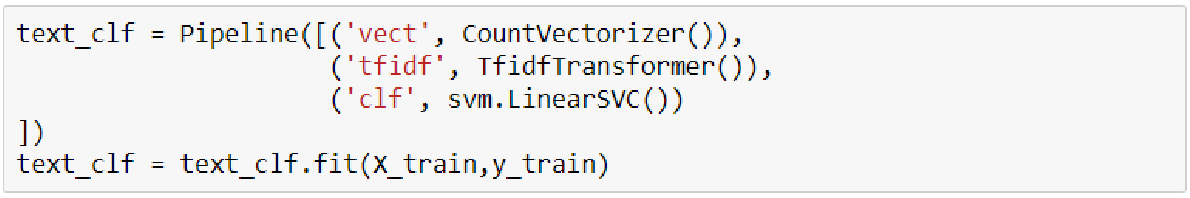
The approach to implement this classifier is different from what was used in the ensemble model. The training dataset was split into 80% training data and 20% testing data and from this split 4 new dataframes were created which are shown below:



```
X_train = train_df['Text'].values
X_test = test_df['Text'].values
y_train = train_df['Class'].values
y_test = test_df['Class'].values
```

Figure 32. SVM - training and testing datasets

Instead of creating new features, only the 'Text' attribute is taken into consideration for predicting 'Class' labels. This is done because support vector machines do not handle sparse data very well. After this, a pipeline is created that is then fitted to the training sets. This pipeline includes Count Vectorizer, TfidfVectorizer and Linear SVC as the classifier. This step is shown below:

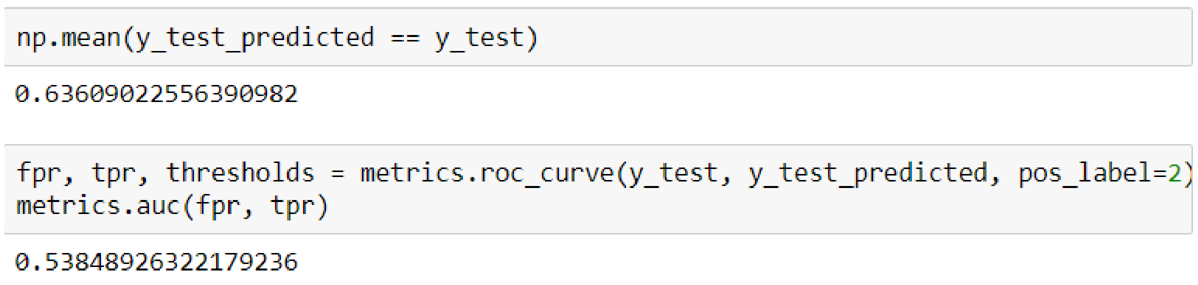


```
text_clf = Pipeline([('vect', CountVectorizer()),
                     ('tfidf', TfidfTransformer()),
                     ('clf', svm.LinearSVC())
])
text_clf = text_clf.fit(X_train,y_train)
```

Figure 33. SVM – pipeline

2.4.3.1 Evaluation metrics – accuracy, Area under Curve (auc)

The accuracy of the model is calculated by comparing the predicted labels with the actual labels of the dataset. Till this step, the predicted labels are one-hot encoded that means probabilities are not calculated as was done in the previous section. The accuracy of the model comes out to be 63.6% and AUC as 53.8%. This can be seen in the figure below:



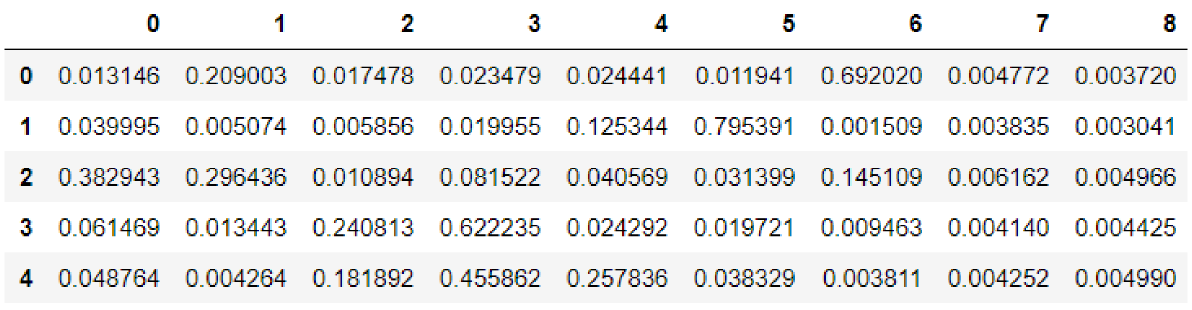
```
np.mean(y_test_predicted == y_test)

0.63609022556390982
```

```
fpr, tpr, thresholds = metrics.roc_curve(y_test, y_test_predicted, pos_label=2)
metrics.auc(fpr, tpr)

0.53848926322179236
```

Figure 34. SVM - model accuracy

2.4.3.2 Prediction

To calculate the probabilities of instances belonging to a ‘Class’ label, the pipeline created in the previous step is tweaked a little and the data is again fitted in the model. The final probabilities are calculated which are shown below for the first 5 instances of the test dataset:



	0	1	2	3	4	5	6	7	8
0	0.013146	0.209003	0.017478	0.023479	0.024441	0.011941	0.692020	0.004772	0.003720
1	0.039995	0.005074	0.005856	0.019955	0.125344	0.795391	0.001509	0.003835	0.003041
2	0.382943	0.296436	0.010894	0.081522	0.040569	0.031399	0.145109	0.006162	0.004966
3	0.061469	0.013443	0.240813	0.622235	0.024292	0.019721	0.009463	0.004140	0.004425
4	0.048764	0.004264	0.181892	0.455862	0.257836	0.038329	0.003811	0.004252	0.004990

Figure 35. SVM -results

2.4.4 Keras – Deep Learning Library [19].

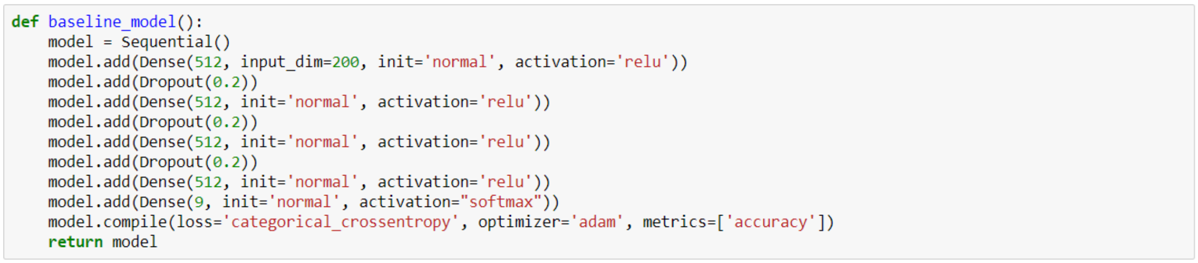
Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation. *Being able to go from idea to result with the least possible delay is key to doing good research*.

For this project, Keras is used on top of TensorFlow [20] and uses the GPU of the system, which is NVIDIA GEFORCE 940M having the 2GB capacity so that the CPU can be used for other purposes while the neural network model is being trained. This capability was used to its utmost, as neural networks models were trained using GPU of the system because it can take 2 hours to 2 days to construct a fully operational model and during this time the CPU was used to train other traditional machine learning models for different datasets included in this dissertation.

The approach for preprocessing of dataset for Keras is same as done for ensemble model. But instead of taking 390 features after applying truncated SVD, only 200 were taken.

2.4.3.1 Classification model:

A sequential baseline model was initially created and layers were added to it. A sequential model is a linear stack of layers. The input shape was set to 200 as the features selected after applying truncated SVD were 200. The model which is given below was then passed to an instance of Keras classifier as the build function and then it was fitted according to the training and testing data created for the task.



```
def baseline_model():
    model = Sequential()
    model.add(Dense(512, input_dim=200, init='normal', activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(512, init='normal', activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(512, init='normal', activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(512, init='normal', activation='relu'))
    model.add(Dense(9, init='normal', activation="softmax"))
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
```

Figure 36. Keras - sequential baseline model

The number of epochs as set to 20 and the model was constructed using various parameters.

2.4.3.2 Evaluation metrics – accuracy and loss of model

With each epoch being made to 'learn' the classification, the model gave out 4 outputs, 2 of them belonging to the accuracy of the model and 2 to the loss suffered by the model. The plots for these and explanation have been discussed below:

- acc: training data accuracy
- val_acc: validation data accuracy
- loss: training data loss
- val_loss: validation data loss

To estimate the ability of the model to generalize to new data, validation accuracy is used, because the validation split contains only the data that the model never sees during training and therefore cannot just memorize. If training data accuracy keeps improving but validation data accuracy is worsened, it's likely an overfitting situation i.e. the model starts to basically just memorize the data.

Each epoch gives out a different metric while running. It is so because each epoch is a training run over all the data. During this run, the parameters of the model are adjusted according to the loss function. The result is a set of parameters which have a certain ability to generalize to new data. That ability is reflected by the validation accuracy. So, each epoch can be thought of as its own model, which can get better or worse if it is trained for another epoch. This change is judged by the change in validation accuracy (better means validation accuracy is increased). Therefore, the key is on decreasing val_loss or increasing val_acc.

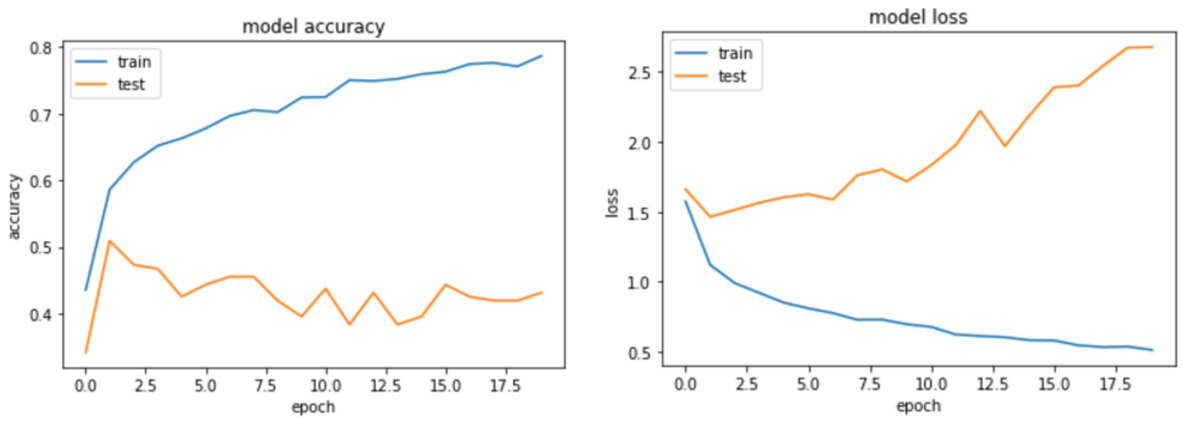


Figure 37. Keras- model accuracy and loss

2.4.3.4 Inference from the plots:

- Even though training accuracy ('acc') increases, validation accuracy ('val_acc') is not improving, which means epochs are not learning anything new.
- The validation loss ('val_loss') keeps on increasing, with a decrease in training loss ('loss), while the opposite should have happened.

This gives a conclusion that the classification done by Keras is not good at all as compared to Ensemble model. Therefore, there is no point in doing prediction doing this model.

The entire approaches for Support Vector Machine and Keras deep learning implementation can be found in Appendix 1.

2.5 Conclusion and Future Work

As observed, the preprocessing for all the three approaches was almost same, but the difference lies in the features that were built for ensemble model's classification. The traditional machine learning algorithm i.e. SVM also couldn't calculate the probabilities with the same efficiency as the ensemble model did as the AUC for both the approaches had a huge difference and Keras model suffered overfitting.

This difference in performance was seen because ensemble models take the shortcomings of traditional machine learning or artificial neural network approaches and build upon them.

ANNs have been known not to perform well with small amount of data [21], and in this case, the data used for training purposes was too small for a neural net. The performance of SVM (a traditional ML approach) was not good, but on the other hand ensemble model took these weak classifiers and created a reasonable predictive model to be worked upon.

The Future work involves several points. First, the shortcomings of ANNs, where the epochs stop to learn after a certain point can be resolved by providing more data, and as this dataset being part of an ongoing Kaggle competition, more data will be released in the second phase of the model to minimize the log loss even further. The second option includes hyper-parameterization of the features created for any of the classification models to boost the efficiency of the classifier. And last, includes optimization of model parameters to obtain relatively better results.

Chapter 3 – Instagram Like Prediction

The goal of this chapter is to predict the likes of a given Instagram post. Instagram is a social networking app made for sharing photos and videos from a smartphone. It is similar to Facebook or Twitter in a sense that if a user has created a profile on Instagram they will have a profile and a news feed and whenever a user posts a photo or video on Instagram, it will be displayed on their profile.

Apart from sharing and 'putting out the photos for the world to see', Instagram is also being used by social media influencers to promote a product or a service which in turn has resulted in the rise of digital marketing on Instagram and for which these personalities are paid huge amounts of money by the respective companies.

3.1 Background

1. Predicting User Likes in Online Media based on Conceptualized Social Network Profiles [22]

   This paper discusses the prediction of user liked in online media and recommendations related products to the user who would bring great profits to certain service providers. It also mentions how data sparsity makes many well-known prediction algorithms perform poorly. The dataset used in this approach is collected from YouTube and Google+. Also, collaborative filtering [23] and content-based methods are discussed.
2. Apply Magic Sauce (AMS) [24]

   This is a service developed by The Psychometrics Centre at University of Cambridge. It predicts users' psycho-demographic traits based on their digital footprints. Using the API predictions are made available for – personality, satisfaction with life, Intelligence, Age, Gender, Interest in given areas, Political views, Religion, and Relationship Status.

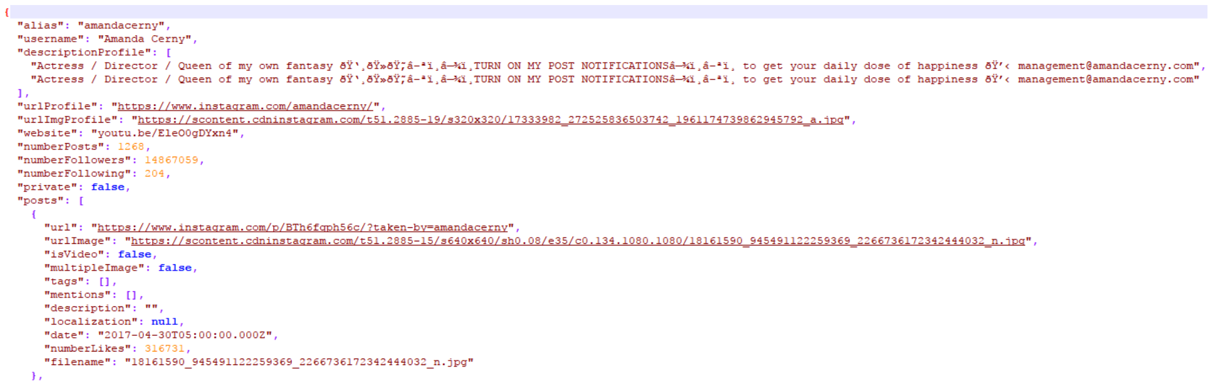
3.2 Data Source

3.2.1 Building the dataset

One of the biggest challenge associated with this chapter of the dissertation was to build the dataset required to perform predictive analytics, there is no open source data for the required task. Therefore, data was scrapped and then was converted into a dataset that can be used for prediction.

The first step in building the dataset was the collection of a list of social media influencers on Instagram i.e. a user, who as part of their occupation, make marketed Instagram posts. This list of users was scrapped from Iconosquare Index Influencers [25]. The list gathered contained at least 1000 users whose Instagram profiles had to be scrapped.

The real task was to scrap the profiles of the users present in the list. This included collecting the metadata from the profile and gathering metadata about the 17 latest images of a user. This number of posts was set to 20 after applying brute-force approach and several technical challenges that were faced while scrapping some profile. The final results were stored in JSON files for each user. The structure of JSON file is given below:



```
{
  "alias": "amandacerny",
  "username": "Amanda Cerny",
  "descriptionProfile": [
    "Actress / Director / Queen of my own fantasy ðŸ'¸ðŸ»ðŸ¦â–ªï¸â—¾ï¸TURN ON MY POST NOTIFICATIONSâ—¾ï¸â–ªï¸ to get your daily dose of happiness ðŸ'‹ management@amandacerny.com",
    "Actress / Director / Queen of my own fantasy ðŸ'¸ðŸ»ðŸ¦â–ªï¸â—¾ï¸TURN ON MY POST NOTIFICATIONSâ—¾ï¸â–ªï¸ to get your daily dose of happiness ðŸ'‹ management@amandacerny.com"
  ],
  "urlProfile": "https://www.instagram.com/amandacerny/",
  "urlImgProfile": "https://scontent.cdninstagram.com/t51.2885-19/s320x320/17333982_272525836503742_1961174739862945792_a.jpg",
  "website": "youtu.be/Ele0OgDYxn4",
  "numberPosts": 1268,
  "numberFollowers": 14867059,
  "numberFollowing": 204,
  "private": false,
  "posts": [
    {
      "url": "https://www.instagram.com/p/BTh6fqph56c/?taken-by=amandacerny",
      "urlImage": "https://scontent.cdninstagram.com/t51.2885-15/s640x640/sh0.08/e35/c0.134.1080.1080/18161590_945491122259369_2266736172342444032_n.jpg",
      "isVideo": false,
      "multipleImage": false,
      "tags": [],
      "mentions": [],
      "description": "",
      "localization": null,
      "date": "2017-04-30T05:00:00.000Z",
      "numberLikes": 316731,
      "filename": "18161590_945491122259369_2266736172342444032_n.jpg"
    },
```

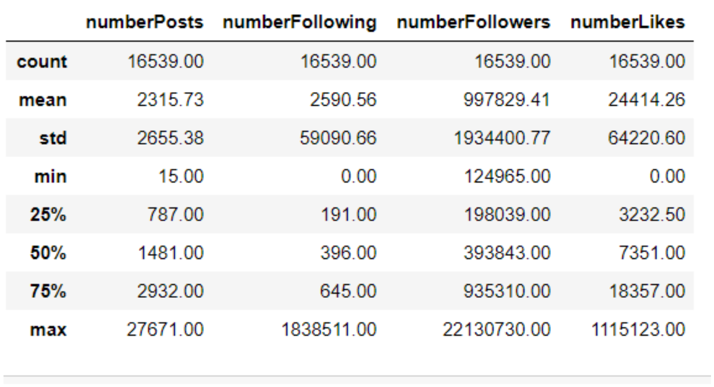
Figure 38. Instagram profile - JSON structure

Instagram's API has a limit of 60 request/hour to their backend servers which makes it useless for any real application or gathering of data. The alternative to official API is crawling each page. The scrapper that was coded using Selenium, a framework that is aimed at building functional tests for web applications but in this context, it is used to crawl web pages and gather data from them. The scrapper initially linearly scans the latest posts from a user, then opens each post to retrieve more information related to each image.

16539 images from 978 Instagram influencers were collected for training and testing purposes. These JSON files were all saved in one directory and then a function was built to parse through all these files and convert each image as a single instance with the profile metadata of the user being redundant for each 17 photos. The code built to perform this task can be seen in Appendix 2 - Part A.

Exploratory Data Analysis

The CSV file generated after parsing of JSON files is read into a pandas dataframe [26]. The initial observations of the collected data were quite peculiar.



	numberPosts	numberFollowing	numberFollowers	numberLikes
count	16539.00	16539.00	16539.00	16539.00
mean	2315.73	2590.56	997829.41	24414.26
std	2655.38	59090.66	1934400.77	64220.60
min	15.00	0.00	124965.00	0.00
25%	787.00	191.00	198039.00	3232.50
50%	1481.00	396.00	393843.00	7351.00
75%	2932.00	645.00	935310.00	18357.00
max	27671.00	1838511.00	22130730.00	1115123.00

Figure 39. Instagram dataframe of 16539 posts

The number of likes has a high standard deviation, 64220.60. Also, the mean value of a number of like is 24414.26 but 75% of the posts have less than 18357 likes, which means there is a high disparity in the data. The dashboard below shows some Exploratory Data Analysis done on the dataset.

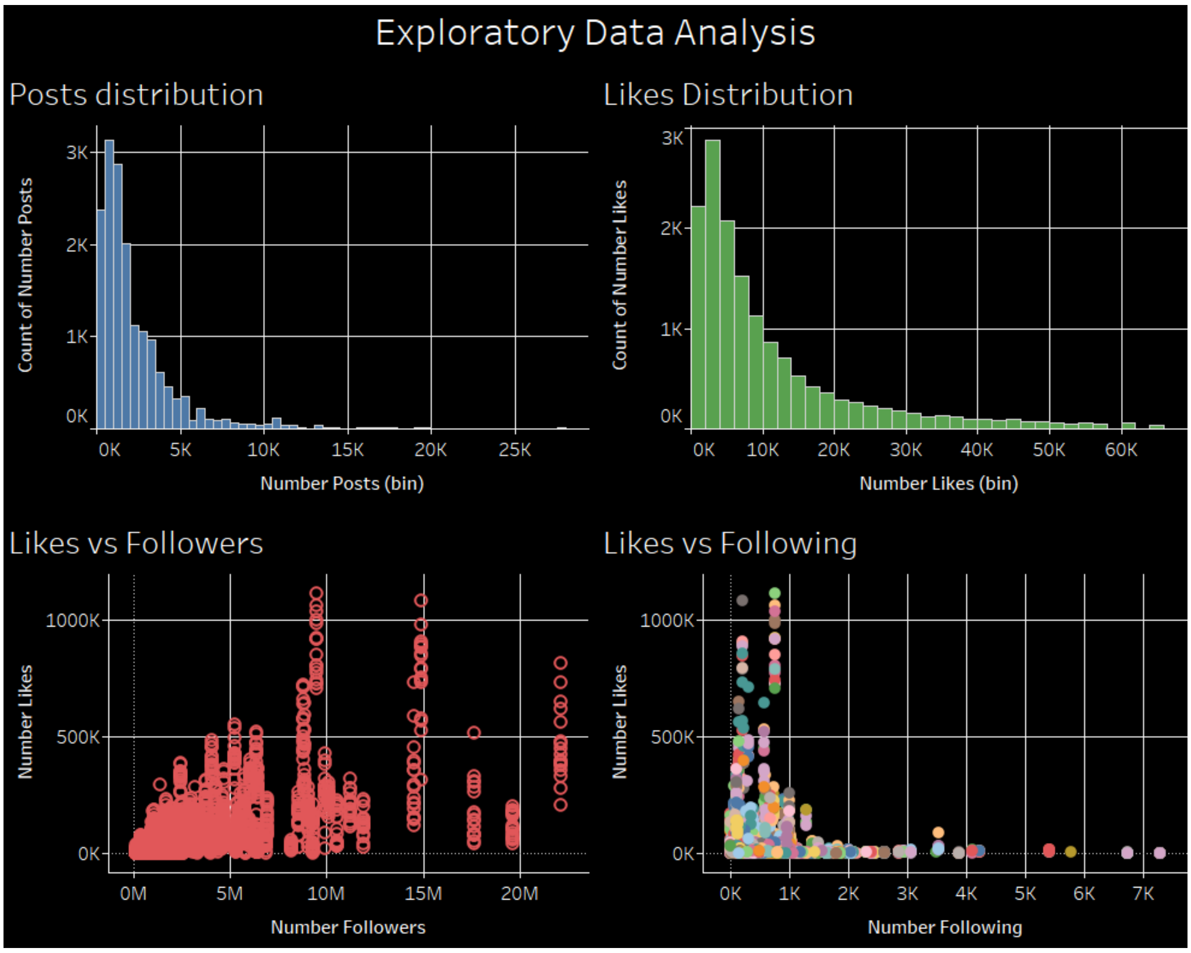


Figure 40. Dashboard for EDA on Instagram dataset

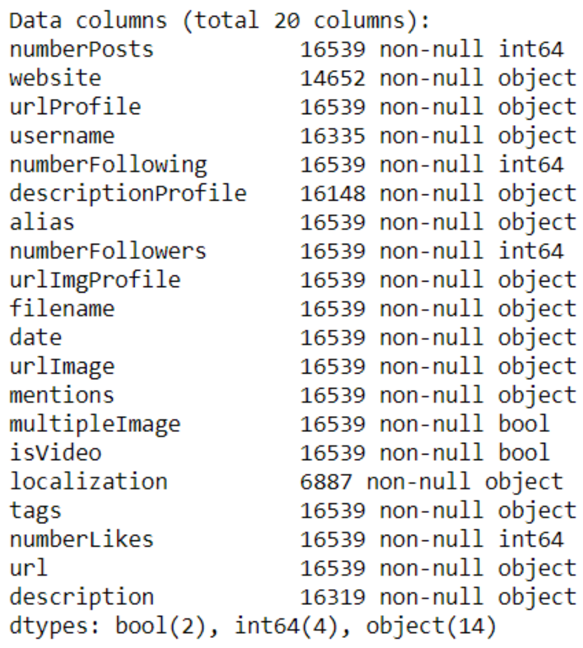
Some observations from the dashboard:

- Distribution of a number of posts is heavily left skewed i.e. the frequency of the number of posts till 10000 is very high as compared to the rest of the datasets.
- The second plot shows that majority of the post has less than 30k likes. It is important to notice that, to prepare this plot some data was filtered out as it was acting as noise in the dataset.
- The third plot shows the relationship between a number of followers and number of likes. The correlation between these two attributes came out to be 0, that means there is no relation that by having a higher number of followers, a user will have a high number of likes.
- The last plot shows the number of accounts users follows and the number of likes they get. The different colors represent the different places that the posts have been geotagged. On further exploration, Costa Maya in Mexico came out to be highest liked place followed by Miami, Florida.

To predict the number of likes makes this a regression problem as there are no labels to classify the data into. Therefore, the models used in this will be different as compared to the previous chapter.

3.3 Data Preprocessing

The first description of the data did give some insights into the datasets, but there are many more attributes present in the dataset. These are given below:



```
Data columns (total 20 columns):
numberPosts           16539 non-null int64
website               14652 non-null object
urlProfile            16539 non-null object
username              16335 non-null object
numberFollowing       16539 non-null int64
descriptionProfile    16148 non-null object
alias                 16539 non-null object
numberFollowers       16539 non-null int64
urlImgProfile         16539 non-null object
filename              16539 non-null object
date                  16539 non-null object
urlImage              16539 non-null object
mentions              16539 non-null object
multipleImage         16539 non-null bool
isVideo               16539 non-null bool
localization          6887 non-null object
tags                  16539 non-null object
numberLikes           16539 non-null int64
url                   16539 non-null object
description           16319 non-null object
dtypes: bool(2), int64(4), object(14)
```

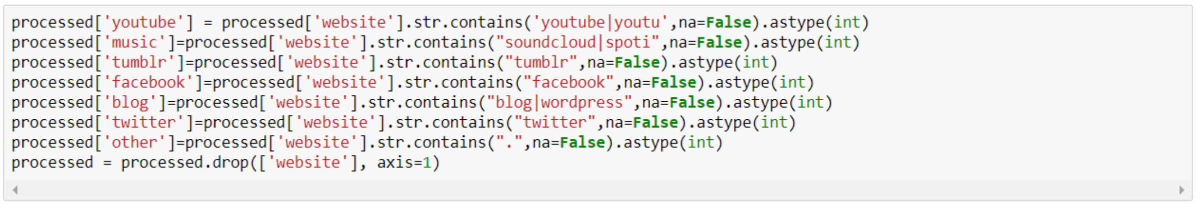
Figure 41. Dataset structure

There are total 20 attributes out of which 14 are an object type and 4 are a numeric type. As the approach to this problem is regression analysis, attributes such as – 'username', 'urlProfile', 'urlImgProfile', 'filename', 'url', urlImage' and 'isVideo'. Other attributes such as 'descriptionProfile', 'description', 'tags', 'mentions' and 'localization' are kept aside because using these natural language features can be built which can be used to train the model.

3.4 Feature Extraction

There are two more attributes, from which features can be extracted using one-hot encoding. These are:

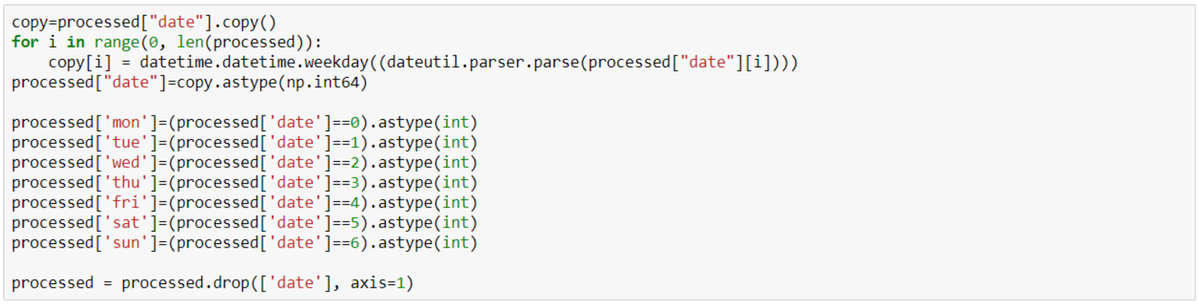
- Website: each user has a website option that they can fill to promote their website on Instagram. This can be anything, ranging from YouTube, Snapchat or Facebook to their personal websites as well. So, 7 features were created and the website attribute was dropped from the dataset. These features can have either 1 or 0 as their values, where 1 means that the user has that respected site mentioned in their profile and the rest 6 features will be 0 for that user. The code snippet is given below:



```
processed['youtube'] = processed['website'].str.contains('youtube|youtu',na=False).astype(int)
processed['music']=processed['website'].str.contains("soundcloud|spoti",na=False).astype(int)
processed['tumblr']=processed['website'].str.contains("tumblr",na=False).astype(int)
processed['facebook']=processed['website'].str.contains("facebook",na=False).astype(int)
processed['blog']=processed['website'].str.contains("blog|wordpress",na=False).astype(int)
processed['twitter']=processed['website'].str.contains("twitter",na=False).astype(int)
processed['other']=processed['website'].str.contains(".",na=False).astype(int)
processed = processed.drop(['website'], axis=1)
```

Figure 42. one-hot encoding: website

- Date: the date attribute is a timestamp on which the post was uploaded. This was converted to the days of the week and then using one-hot encoding each day was assigned a number from 0 to 6 starting from Monday and ending on Sunday.



```
copy=processed["date"].copy()
for i in range(0, len(processed)):
    copy[i] = datetime.datetime.weekday((dateutil.parser.parse(processed["date"][i])))
processed["date"]=copy.astype(np.int64)

processed['mon']=(processed['date']==0).astype(int)
processed['tue']=(processed['date']==1).astype(int)
processed['wed']=(processed['date']==2).astype(int)
processed['thu']=(processed['date']==3).astype(int)
processed['fri']=(processed['date']==4).astype(int)
processed['sat']=(processed['date']==5).astype(int)
processed['sun']=(processed['date']==6).astype(int)

processed = processed.drop(['date'], axis=1)
```

Figure 43. one-hot encoding: date

Just as a verification process that the features being built make some sense or not dashboards were created for the above two features, which are given below:

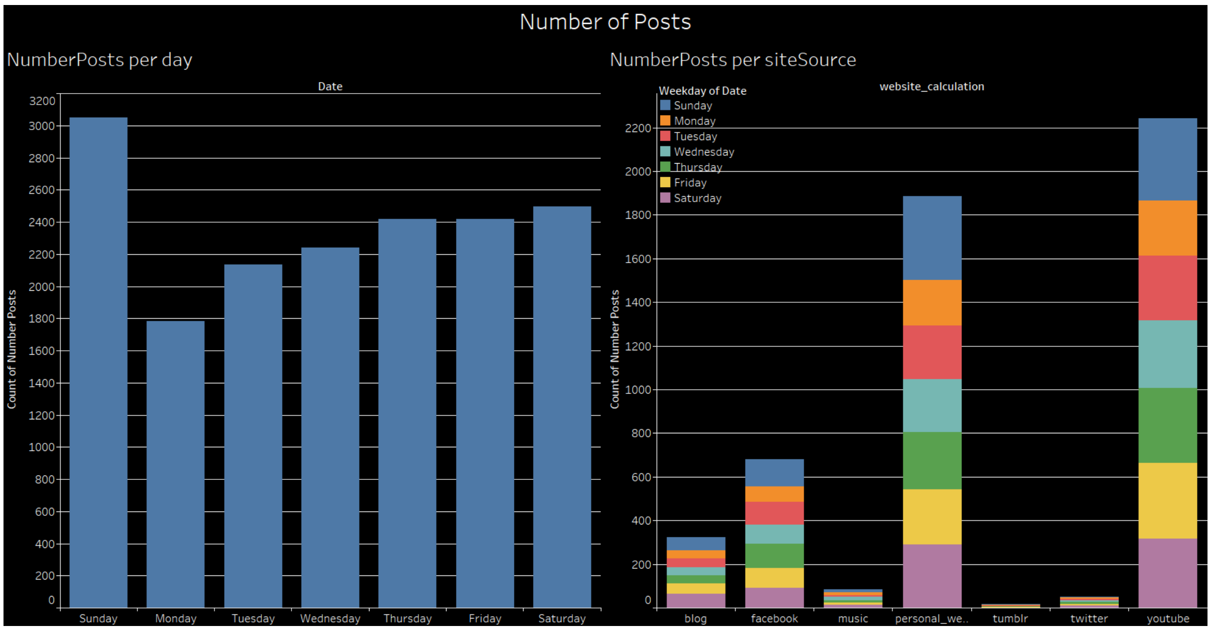


Figure 44. Dashboard for number of posts per day

The dashboard given above has two plots, the first one tells the number of posts uploaded according to days of the week and the second one shows the distribution of pictures posted by users according to the websites that they have in their profile. Clearly, a lot if these influencers have YouTube channels in their website sections followed by personal websites and then their Facebook accounts. As seen, most number of posts are uploaded on Sundays and the least on Mondays. This is understandable given the nature of people who use social media. Just to verify the first plot, percentage plot was created, shown below:

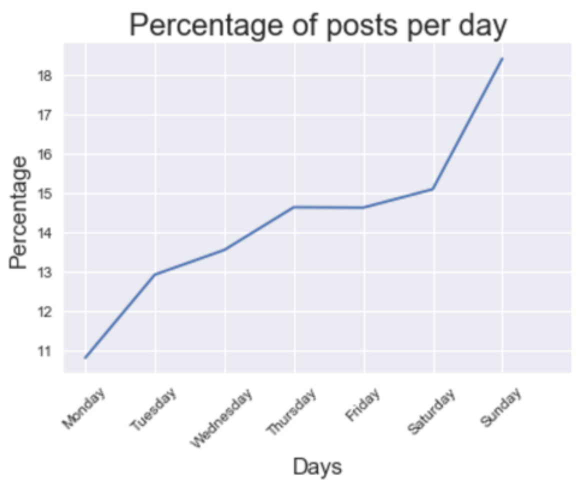


Figure 45. percentage of posts per day

The second dashboard emphasizes on Likes per post according to the days the posts have been uploaded and according to the websites that the users had mentioned in their profiles. This dashboard is given below:

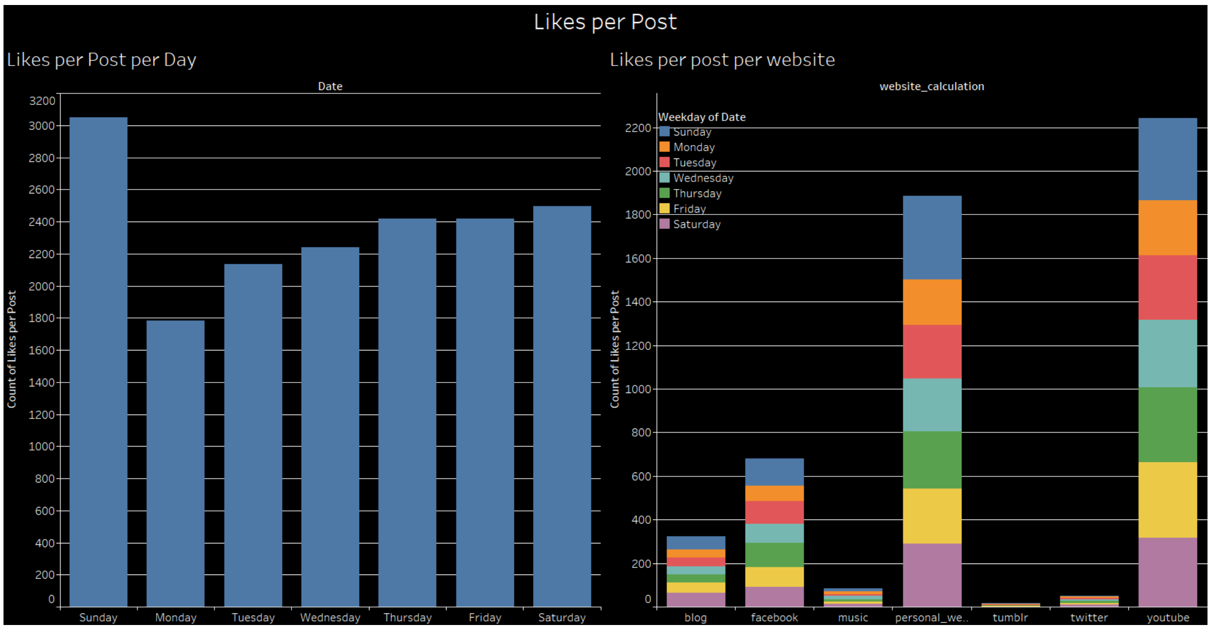
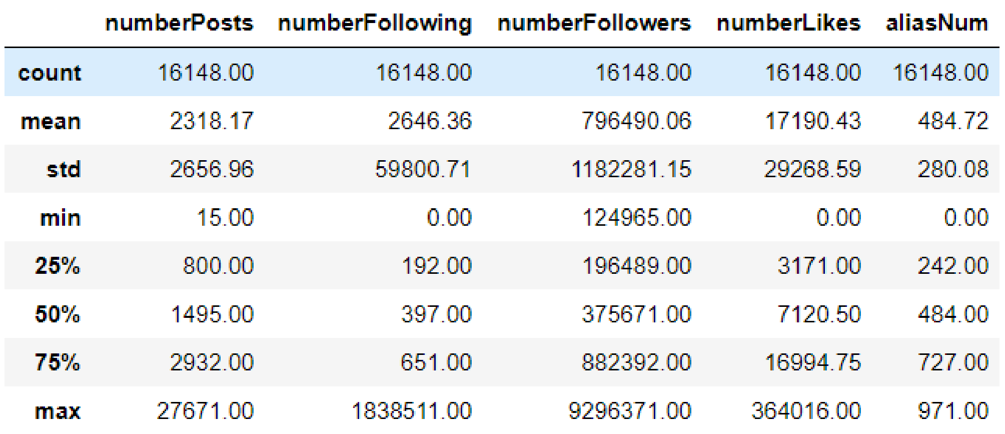


Figure 46. Dashboard for Likes per posr per day

As observed, most liken on posts are given on Sundays and the least on Mondays. This can also be because of the percentages of posts being uploaded on these dates. Sundays collectively have a share of around 18% of data whereas Mondays have a mere share of 10% of data.

3.4.1 Natural Language Processing

As previously discussed, attributes such as user's description, description of the profiles, description of the images all these can be used to create natural language features. This involved a lot of text cleaning as users generally tend to add a lot of emoticons and special characters in their descriptions of posts and profiles. The first step was to change the 'alias' which represents username using which the user had created their profile into numeric labels so that textual data can be eliminated from the dataset. The next step was to filter out posts that had more than 200k likes because these posts create noise in the dataset which ultimately affects the final results of the prediction model. This step changes the number of posts from 16539 to 16148. The description of the new dataset is given below:

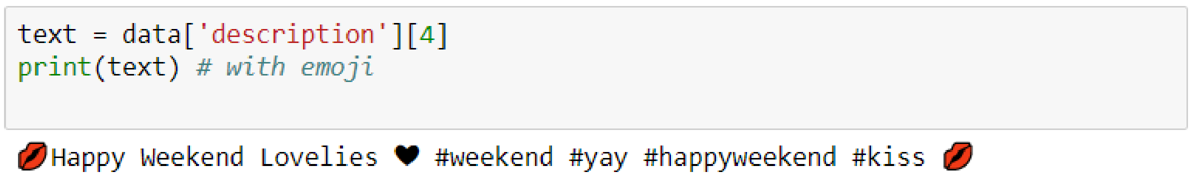


	numberPosts	numberFollowing	numberFollowers	numberLikes	aliasNum
count	16148.00	16148.00	16148.00	16148.00	16148.00
mean	2318.17	2646.36	796490.06	17190.43	484.72
std	2656.96	59800.71	1182281.15	29268.59	280.08
min	15.00	0.00	124965.00	0.00	0.00
25%	800.00	192.00	196489.00	3171.00	242.00
50%	1495.00	397.00	375671.00	7120.50	484.00
75%	2932.00	651.00	882392.00	16994.75	727.00
max	27671.00	1838511.00	9296371.00	364016.00	971.00

Figure 47. Data filtering

The disparity in the dataset is minimized as the mean of number of Likes now becomes 17190.43 which previously was 24414.26 and the third quartile value changes to 16994.75 from 18357. Feature extraction was performed on various attributes that are described below:

- Post description



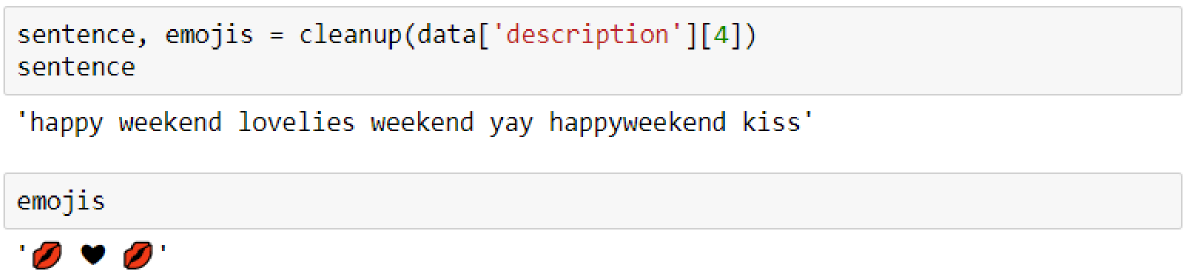
```
text = data['description'][4]
print(text) # with emoji
```

```
💋Happy Weekend Lovelies 🖤 #weekend #yay #happyweekend #kiss 💋
```

Figure 48. post description

The figure above contains a general post description with hashtags and emojis. The challenge is to clean up such descriptions related to all the posts, separate the emojis and from the remaining text create text corpus that can be then converted into a sparse matrix of words. This approach is somewhat like what was taken in the NLP section of the last chapter.

The above example of post description after cleaning up, becomes like the one shown in the figure below:



```
sentence, emojis = cleanup(data['description'][4])
sentence
```

```
'happy weekend lovelies weekend yay happyweekend kiss'
```

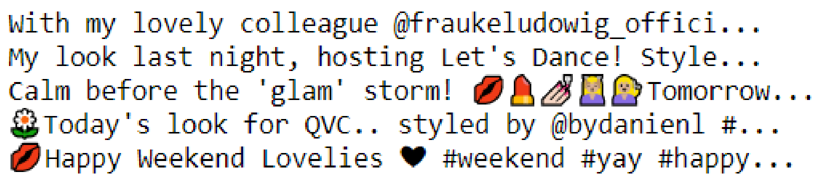
```
emojis
```

```
'💋 🖤 💋'
```

Figure 49. emoji removed

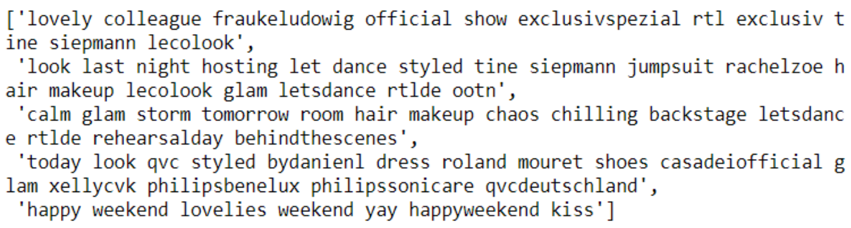
The code snippets for this approach are given in Appendix 2 – Part B.

When this approach is applied to the entire 'description' attribute, it cleans up the text which then can be used to create a sparse dataset of words. The figure below shows the difference between original data and clean data:



```
With my lovely colleague @fraukeludowig_offici...
My look last night, hosting Let's Dance! Style...
Calm before the 'glam' storm! 💋💄💅💆💇Tomorrow...
🌼Today's look for QVC.. styled by @bydanienl #...
💋Happy Weekend Lovelies 🖤 #weekend #yay #happy...
```

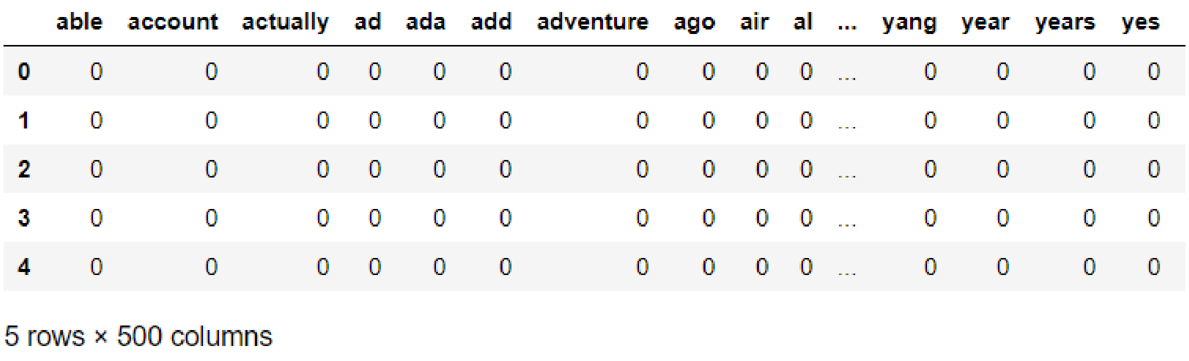
Figure 50. Original post description



```
['lovely colleague fraukeludowig official show exclusivspezial rtl exclusiv t
ine siepmann lecolook',
 'look last night hosting let dance styled tine siepmann jumpsuit rachelzoe h
air makeup lecolook glam letsdance rtlde ootn',
 'calm glam storm tomorrow room hair makeup chaos chilling backstage letsdanc
e rtlde rehearsalday behindthescenes',
 'today look qvc styled bydanienl dress roland mouret shoes casadeiofficial g
lam xellycvk philipsbenelux philipssonicare qvcdeutschland',
 'happy weekend lovelies weekend yay happyweekend kiss']
```

Figure 51. Processed posts description

Count Vectorizer is used to assign numeric labels to these words which first converts words to vectors and then assigns them labels. The maximum features limit is set to 500 because as it can be observed, after text preprocessing and cleaning of description, there are many words which are being misspelled. The results after performing count vectorizer are then converted into an array. A pandas dataframe is creates using this array and the column names are retrieved using the 'get_feature_names' function present in Count Vectorizer. The dataset obtained is shown in the figure below:



	able	account	actually	ad	ada	add	adventure	ago	air	al	...	yang	year	years	yes
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0

5 rows × 500 columns

Figure 52. sparse matrix from post description

Also, the emojis from the description of the posts were extracted into a different list and the top 37 emojis used by social media influencers are mentioned in the figure below. This can also be interpreted as if a user on Instagram wants high number of likes, they can use these emojis to boost their likes.

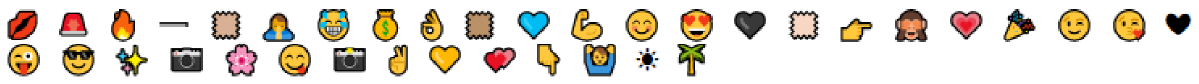
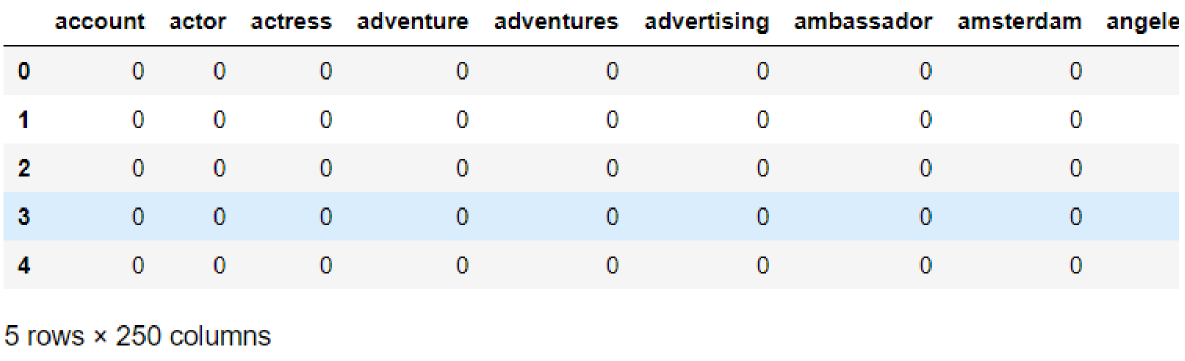


Figure 53. top 37 most used emojis on Instagram

- User Description

The description a user writes about themselves contains a treasure of data if processed correctly. The steps applied for post descriptions were used to extract features from user description. The only difference here is the limit of maximum features in the Count Vectorizer step was set to 250.

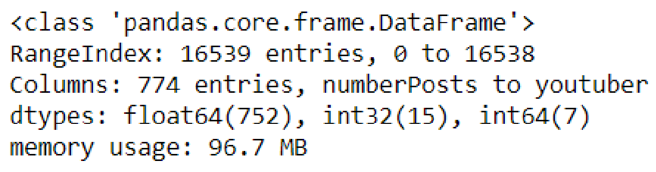


	account	actor	actress	adventure	adventures	advertising	ambassador	amsterdam	angele
0	0	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	0	
2	0	0	0	0	0	0	0	0	
3	0	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	

5 rows × 250 columns

Figure 54. Sparse matrix using profile description

Note: Other attributes such as localization, hashtags and mentions were also processed as natural language features, but they are nothing else other than keywords, were not included in the NLP dataset as introducing this many degree of features and that too as a sparse matrix will only result in 'the curse of dimensionality'. Therefore, only description of posts and profiles as features were included in the dataset and then this was concatenated with the previous dataset built in feature building topic to give out a dataset of shape 16539 rows and 774 columns.



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16539 entries, 0 to 16538
Columns: 774 entries, numberPosts to youtuber
dtypes: float64(752), int32(15), int64(7)
memory usage: 96.7 MB
```

Figure 55. final dataset to be used for prediction

3.5 Prediction Models

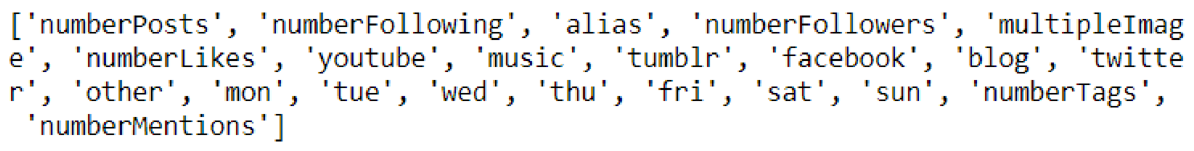
3.5.1 ANNs – Keras API

Just to check what kind of results neural networks would give, Keras API running on top of TensorFlow was used.

Keras is a high level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation. *Being able to go from idea to result with the least possible delay is key to doing good research*.

For this project, Keras is used on top of TensorFlow [20] and uses the GPU of the system, which is NVIDIA GEFORCE 940M having 2GB capacity so that the CPU can be used for other purposes while the neural network model is being trained. This capability was used to its utmost, as neural networks models were trained using GPU of the system because it can take 2 hours to 2 days to construct a fully operational model and during this time the CPU was used to train other traditional machine learning models for different datasets included in this dissertation.

The dataset here used did not had any natural language features but only had one-hot encoded versions of website and dates. The attributes involved in this stage of prediction are given below:



```
['numberPosts', 'numberFollowing', 'alias', 'numberFollowers', 'multipleImag
e', 'numberLikes', 'youtube', 'music', 'tumblr', 'facebook', 'blog', 'twitte
r', 'other', 'mon', 'tue', 'wed', 'thu', 'fri', 'sat', 'sun', 'numberTags',
 'numberMentions']
```

Figure 56. attributes used for prediction

The dataset was divided into training and testing dataset, X and Y respectively where X had all the attributes other than number of likes whose value had to be predicted which was therefore placed in Y. Three different models of neural networks were created, just to see how their results vary. The evaluation metric used on all of them was mean square error as this is a regression problem. These are mentioned below:

- Neural network – standard model

Input layers – 1 (21 neurons)

Output layers – 1 (1 neuron)

Input dimensions – 21

Number of epochs = 100

Results: 2256319031.999982 (1802579849.58) MSE

- Neural network – larger model

Input layers – 1 (21 neurons)

Output layers – 1 (1 neuron)

Hidden layer – 1 (8 neurons)

Input dimensions – 21

Number of epochs = 50

Result: 2795003169.545209 (2536374719.56) MSE

- Neural network – larger model

Input layers – 1 (30 neurons)

Output layers – 1 (1 neuron)

Input dimensions – 21

Number of epochs = 50

Result: 2758380933.48 (2221797636.55) MSE

These results obtained are not at all good. Mean square error in billions just proves that neural networks didn't performed very well even when there was ample of data supplied that too which was only numerical data.

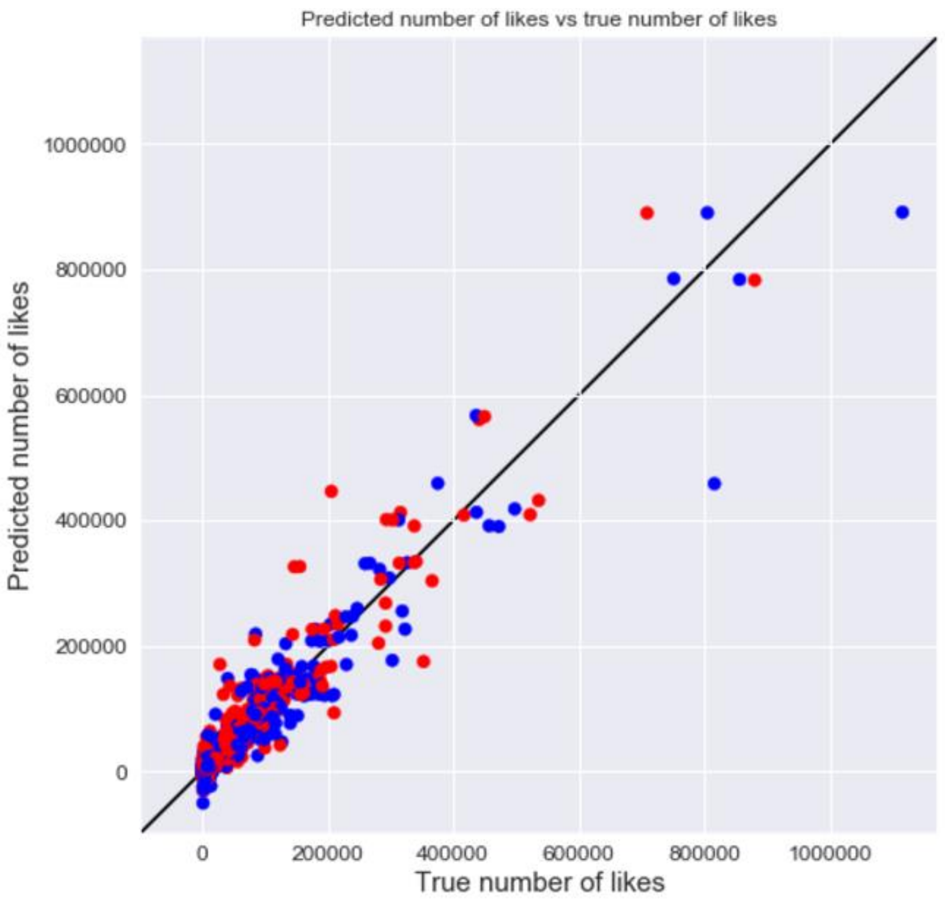
Note: This step was the first step performed when dataset was created after extraction. Observing the results, it was decided feature extraction using natural language processing is required.

3.5.2 Regression – Linear Regression [27]

The most basic step to check the behaviour of dataset and to decide what should be done next to improve the efficiency of the model is to start the prediction with Linear Regression. There are many ways the dataset for this step can be supplied. Two different iterations of the dataset were supplied, one where the features extracted from NLP i.e. description of posts and emojis from posts were concatenated with the one-hot encoding dataset and the other where the latter dataset was concatenated with description of posts and description of profiles. Both the results were somewhat similar therefore the discussion here is respect to the second dataset being used.

Creating the training and testing sets for this problem was somewhat peculiar because a simple train_test_split could not suffice. As it is already known, each user has 17 posts, the first three posts were taken as testing set and the rest as training set for each user. These sets were then divided into the standard X_train, X_test, y_train, y_test sets to be used in the prediction model. The approach for this is mentioned in Appendix 2 – Part C.

The score for linear regression model came out to be 91.4% which is not that bad for starters. The plot between true values of likes and predicted values of likes is given below:



Also, the correlation between the predicted values and the actual values comes out to be 95.674% which means that these two attributes have a high positive correlation. The approach is mentioned in Appendix 2 – Part D

3.5.3 Gradient Boosting Regressor [28]

It is an ensemble approach for boosting algorithms which is implemented using forward stagewise additive modelling. This class of algorithm starts with an empty ensemble and incorporates new members sequentially. At each stage, the model that maximizes the predictive performance of the ensemble as a whole is added, without altering those already in the ensemble. Optimizing is done by allowing the next model to focus on those training instances on which the ensemble performs poorly. This is exactly what boosting does by giving those instances larger weights [29].

The same set of training and testing sets were used. The accuracy of the model came out to be 91% with root mean square error of 19162. Comparing this with the results of ANNs there is an increase of 90% in performance of the model. The implementation of this algorithm can be seen in Appendix 2 – Part E.

3.5.4 AdaBoost Regression [30]

AdaBoost is a meta algorithm, therefore the underlying principle of classification and regression using AdaBoost are same. This algorithm was introduced in 1995 by Freund and Schapire [31]. The core principle of AdaBoost is to fit a sequence of weak learners (i.e. models that are only slightly better than random guessing, such as small decision tree) on repeatedly modified versions of the data. The prediction from all of them are then combined through a weighted majority vote (or sum) to produce the final prediction. For regression, the algorithm implements AdaBoost.R2 [32].

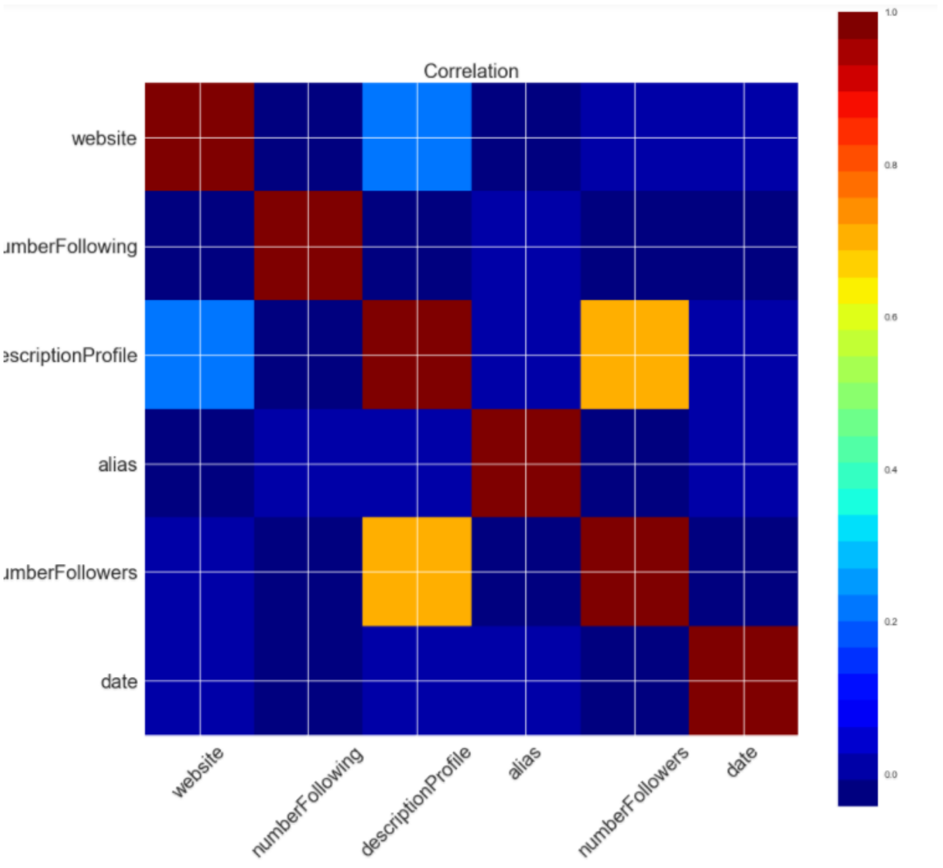
AdaBoost is a special case of Gradient Boosting technique. Both methods use a set of weak learners. They try to boost these weak learners into a strong learner. Gradient Boosting generates learners during the learning process. It build first learner to predict the value and calculates the loss i.e. the difference between the outcome of the first learner and the real value. Then it builds a second learner to predict the loss after the first step. This continues until a certain threshold is achieved.

AdaBoost requires a user to specify a set of weak learners, in this case the base estimator is Random Forest Regressor. It learns the weights of how to add these learners to be a strong learner. The weight of each learner is learned by whether it predicts a sample correctly or not. If a learner does not predict a sample correctly, the weight of the learner is reduced a bit. It repeats this process until a certain converge.

Therefore, to compare the performance of these two algorithms the loss function is selected as Root Mean Square Error, which for AdaBoost comes out to be 18671.5. The percentage increase in the performance from Gradient Boosting to AdaBoost is 2.56% which can have a significant impact at a grand scale. The implementation of this algorithm can be seen in Appendix 2 – Part F.

3.5.5 Lasso Regression

One final step before concluding this chapter is to check whether multi-collinearity exists in the dataset or not. For this a plot for correlation of the entire dataset was constructed, which is given below:



On observing the plot, there is weak or no correlation in the entire dataset. The only strong correlation is among the same type of features which is supposed to be there. If there would have been strong correlation, it would mean that similar type of data is being repeated and regression wouldn't have given the results that it gave. Also, in such scenarios Ridge regression has to be used. But as there is no multi-collinearity Lasso Regression can be used to further minimize the root mean square error.

Lasso approach is known for regression shrinkage [33]. The lasso minimizes the residual sum of squares subject to the sum of absolute value of the coefficients being less than constant. Because of the nature of this constraint it tends to produce some coefficients that are exactly

0 and hence gives interpretable models. It is same as linear regression but it is trained with L1 prior as regularizer. The optimization objective for Lasso is given in the equation below:

```
(1 / (2 * n_samples)) * ||y - Xw||^2_2 + alpha * ||w||_1
```

……………………………………………….(2)

Using the same training and testing sets and applying lasso on it, the accuracy came out to be 92.26% which is slightly less than linear regression but the root mean square error was 17707, the minimum of all the approaches. Performance evaluation of all the models I given in the figure below:

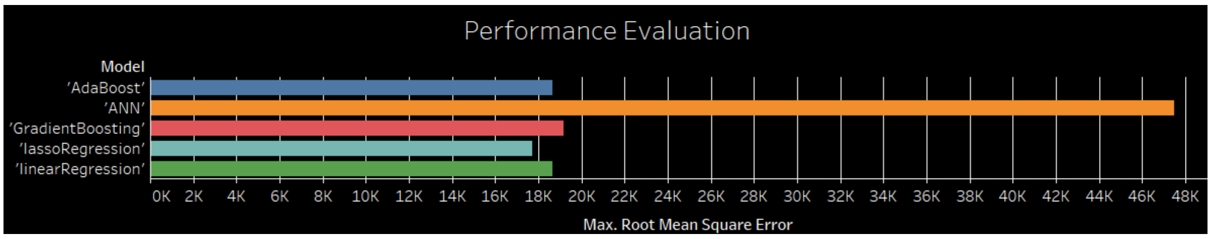


Figure 57. performance comparison of prediction models

3.6 Conclusion and Future Work

The most important finding of this chapter is ensemble models using feature building can outperform Artificial Neural networks even though this time the dataset was not that small as compared to previous chapter. Also, while working with prediction of values, two things should always be checked:

- Interpretability of the dataset, whether it is important or not
- Multi-collinearity in the dataset, whether it exists or not

These two factors in regression shape up the approaches i.e. the algorithms that have to be used. To further improve the performance, truncated singular value decomposition can be implemented as this dataset had a textual sparse matrix which could have introduced dimensionality in the dataset. Truncated SVD performs linear dimensionality reduction thus making the dataset more interpretable. Also, the model developed can be used in online marketing to find influencers that yield the most impressions for a given post.

Chapter 4 – Stock market prediction

4.1 Introduction

4.1.1 History of stocks

The concept of stocks and share markets is old to mankind. It all started in 12th century France where the *courretiers de change* were concerned with managing and regulating the debts of agricultural communities on behalf of the banks. Because these men also traded with debts, they could be called the first brokers [34]. In 17th and 18th century, the Dutch pioneering several financial innovations helped lay the foundations of modern financial system [35][36][37][38].

4.1.2 Function and Purpose of Stock Markets

There are stock markets now in virtually every developed and most developing economics, with the world's largest markets being in the United States, United Kingdom, Japan, India, China, Canada, Germany, France, South Korea and the Netherlands [39]. The stock market is one of the most important ways for companies to raise money, along with debt markets which are generally more imposing but do not trade publicly [40].History has shown that the price of stocks and other assets is an important part of the dynamics of economic activity. An economy where the stock market is on the rise is considered to be an up-and-coming economy. The stock market is often considered the primary indicator of a country's economic strength and development [41].

4.1.3 Stock market crash

This phenomenon is often defined as a sharp dip in share prices of stocks listed on the stock exchanges. In parallel with various economic factors, a reason for stock market crashes is also due to panic and investing public's loss of confidence. Often, stock market crashes end speculative economic bubbles. One of the most famous stock market crashes started October

19, 1987 – Black Monday. The crash began in Hong Kong and quickly spread around the world. By the end of October, stock markets in Hong Kong had fallen 45.5%, Australia 41.8%, Spain 31%, The United Kingdom 26.4%, the United States 22.68%, and Canada 22.5%. Black Monday itself was the largest one-day percentage decline in stock market history – the Dow Jones fell by 22.6% in a day [42].

4.1.4 Irrational behaviour of the stock market

Sometimes, the market seems to react irrationally to economic or financial news, even if the news is likely to have no real effect on the fundamental value of securities itself [43]. But this may be more apparent than real, since often such news has been anticipated, and a counterreaction may occur if the news is better (or worse) than expected. Therefore, the stock market may be swayed in either direction by press release, rumours, euphoria and mass panic. A part of the approach used in this chapter for prediction of changes in stock market covers the impact of tweets over a particular company.

4.2 Background

- Tobias Preis and his colleagues Helen Susannah Moat and H. Eugene Stanley introduced a method to identify online precursors for stock market moves, using trading strategies based on search volume data provided by Google Trends [44]. Their analysis of Google search volume for 98 terms of varying financial relevance suggests that increase in search volume for financially relevant search terms tend to precede large losses in financial markets [45][46].

- In a study published in Scientific Reports in 2013, [47] Helen Susannah Moat, Tobias Preis and colleagues demonstrated a link between changes in the number of views of English Wikipedia articles relating to financial topics and subsequent large stock market moves [48].

- The use of Text Mining together with Machine Learning algorithms received more attention in the last years, with the use of textual content from Internet as input to predict price changes in Stocks and other financial markets. A paper in 2014, "Text mining for market prediction: A systematic review" [49], is believed to be the first effort to provide a comprehensive review from a holistic and interdisciplinary point of view. This work intended to accomplish facilitation of integration of research activities from different fields on the topic of market prediction based on online text mining and provision of a study-framework to isolate the problem or different aspects of it in order to clarify the path for further improvement.
- The collective mood of Twitter messages has been linked to stock market performance in the paper "Twitter mood predicts the stock market" [50]. The study, however, has been criticized for its methodology [51], with the biggest problem being the advertising of results being biased 'by selection'.

4.3 Methodology

The approach to prove that sentiments from twitter do have some effect on the stock price three methods are implemented:

- Stock prediction using data from S&P-500 and other big stock markets as there is some correlation between markets around the world [52].
- Stock prediction of Reliance Industries [53] using tweets.
- Using Facebook's latest released package 'Prophet' to predict the stocks for the next 30 days.

4.3.1 S&P-500 Stock Prediction

4.3.1.1 Goal

The goal here is to first classify the stocks into two different classes, 'Up' when the stock price goes up and 'Down' when it goes down over a period of time. Then prediction is made using this model. The approach is discussed in the sections below.

4.3.1.2 Data Source

As mentioned previously as the world markets are linked to each other i.e. changes caused in one market creates ripples across all the other markets, therefore the data gathered here is of the following markets:

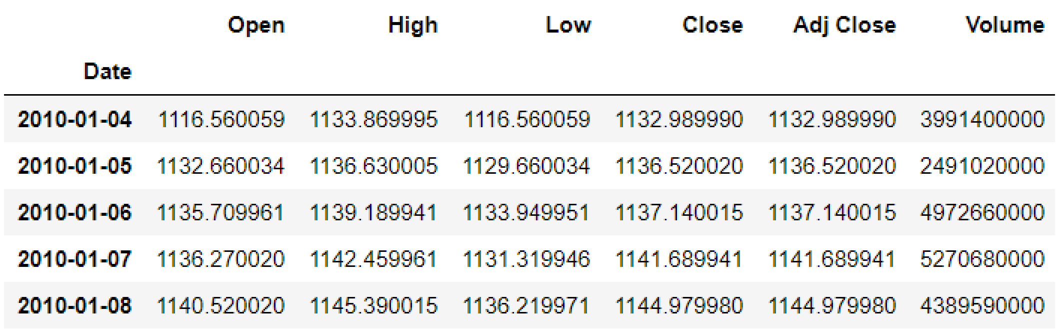
- S&P – 500 from Yahoo Finance (^GSPC) [54]
- Nasdaq Composite from Yahoo Finance (^IXIC) [55]
- Dow Jones Industrial Average from Quandl (.DJI) [56]
- 5 Years US Treasury from Yahoo Finance (^FVX) [57]
- Hong Kong Hang Seng from Yahoo Finance (^HIS) [58]
- Frankfurt DAX from Yahoo Finance (^GDAXI) [59]
- Paris CAC 40 from Yahoo Finance (^FCHI) [60]
- Tokyo Nikkei-225 from Yahoo Finance (^N225) [61]
- London FTSE-100 from Yahoo Finance (^FTSE) [62]
- Oil Price US Department for Energy from Quandl [63]
- Gold Price (Bundesbank) from Quandl [64]
- Dollar in Euros Rate Exchange from Quandl [65]
- Dollar in Japanese Yen Rate from Quandl [66]
- Dollar in Australian Dollars Rate from Quandl [67]

The above-mentioned datasets are collected using either yahoo finance [68] or Quandl [69]. Using Python's package pandas_datareader data can be extracted, by providing three parameters. The symbol of the dataset, start date and end date. For all of these datasets that have been extracted, the start date is set to 1st January 2010 and the end date is 31st December 2016. To extract data from Quandl, Python has a package called 'quandl'. Total of 14 datasets are created out of which the first 9 are stock datasets and the rest 5 are prices of commodities that affect the world market directly or indirectly.

4.3.1.3 Extraction using Yahoo Financials

4.3.1.3.1 Data Preprocessing

The S&P-500 dataset was created by fetching the values of stocks from yahoo finance over a duration of 6 years and then it was stored in a pandas dataframe. Pandas automatically detects if the dataset has to be treated as a timeseries dataset or not by looking at the date attribute. So, it creates the date as the index of the dataset and converts the dataset into a timeseries. The first 5 instances in the timeseries are given below:



	Open	High	Low	Close	Adj Close	Volume
Date						
2010-01-04	1116.560059	1133.869995	1116.560059	1132.989990	1132.989990	3991400000
2010-01-05	1132.660034	1136.630005	1129.660034	1136.520020	1136.520020	2491020000
2010-01-06	1135.709961	1139.189941	1133.949951	1137.140015	1137.140015	4972660000
2010-01-07	1136.270020	1142.459961	1131.319946	1141.689941	1141.689941	5270680000
2010-01-08	1140.520020	1145.390015	1136.219971	1144.979980	1144.979980	4389590000

Figure 58. S&P-500 stock prices

The most important attribute here in the above dataset is Adj (Adjusted) Close. The closing price is just the price of that stock at the close of the trading day. The adjusted closing price uses the closing price as a starting point, but it considers factors such as dividends, stock splits and new stock offerings. The adjusted close price represents a more accurate reflection of a stock's value, since distribution and new offerings can alter the closing price.

- Effects of dividends

  When a stock appreciates in value, the corporation can choose to reward stockholders with a dividend. The dividend can come either in the form of cash paid per share or as an additional percentage of shares. In either event, a dividend reduces the stock's value. The adjusted closing price shows the stock's value after posting a dividend. For example, if a share with a closing price of $100 paid a $5 dividend per share, the adjusted closing price would be $95.
- Stock Splits

  If the price of individual shares in a corporation is too high for investors to purchase in round numbers, the corporation may split its stock shares. When the number of shares increases, the value of each individual share drops. In the example above, if the company splits each $100 share into two $50 shares, the adjusted closing price from the day before the split is now $50. The adjustment reflects the stock split, rather than a single-day 50 percent drop in share price.

- New Offerings

  A corporation may choose to offer additional shares of stock to raise capital. In a 'rights offering', the current shareholders are given the option to purchase the new shares at reduced prices. When the new shares enter the market, the price of the existing shares drop. The adjusted closing price accounts for the new offerings and how they change the closing price. The calculations of the adjusted share price on new offerings are not as straightforward as those for other corporate actions.

The primary use for the adjusted closing price is as a means to develop an accurate track records of a stock's performance. The comparison of a stock's historical adjusted closing price to its current price shows the true rate of return. For example, if a stock's current closing share price is $100, and its closing price 10 years ago was $40, the stock shows a 250% return over the last decade. However, if the adjusted closing price from a decade ago was $25 , the stock shows a true 10-year return rate of 400 percent. The plot of Adjusted Closing price over the years is given below:

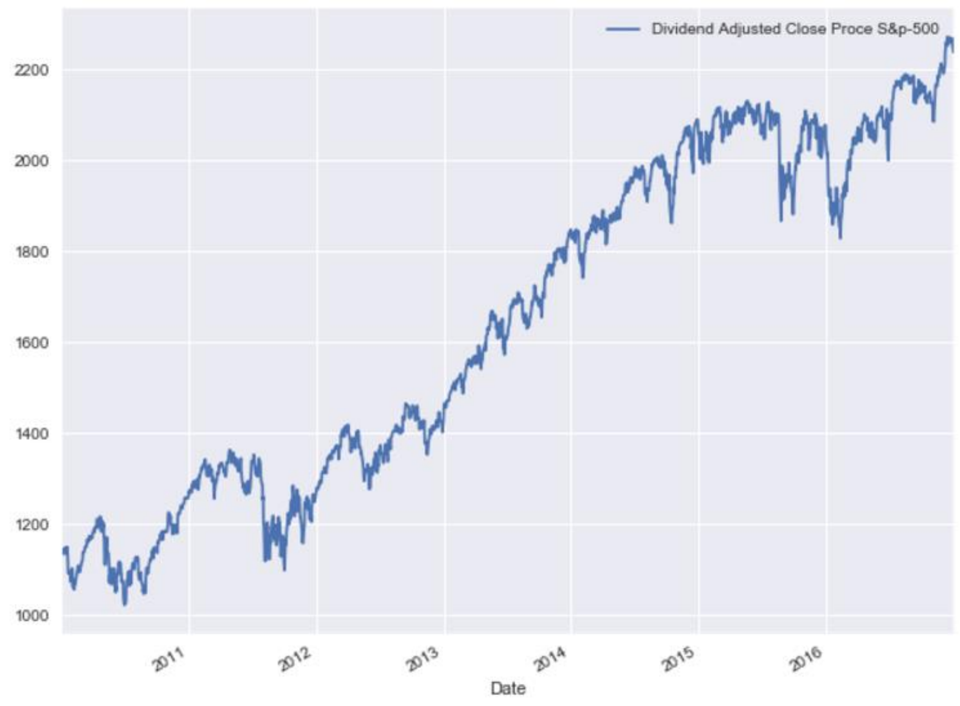


Figure 59. Adjusted close price of S&P-500

4.3.1.3.2 Feature Building

A new feature 'Return' is added in the timeseries which is percentage change of adjusted closing price for two consecutive days, that is going by the data in the dataset, adjusted closing price for the first two dates is 1132.989990 and 1136.520020 which makes the Return feature value 0.003116 for the second date in the dataset. The plot for 'Return' for S&P-500 is given below:

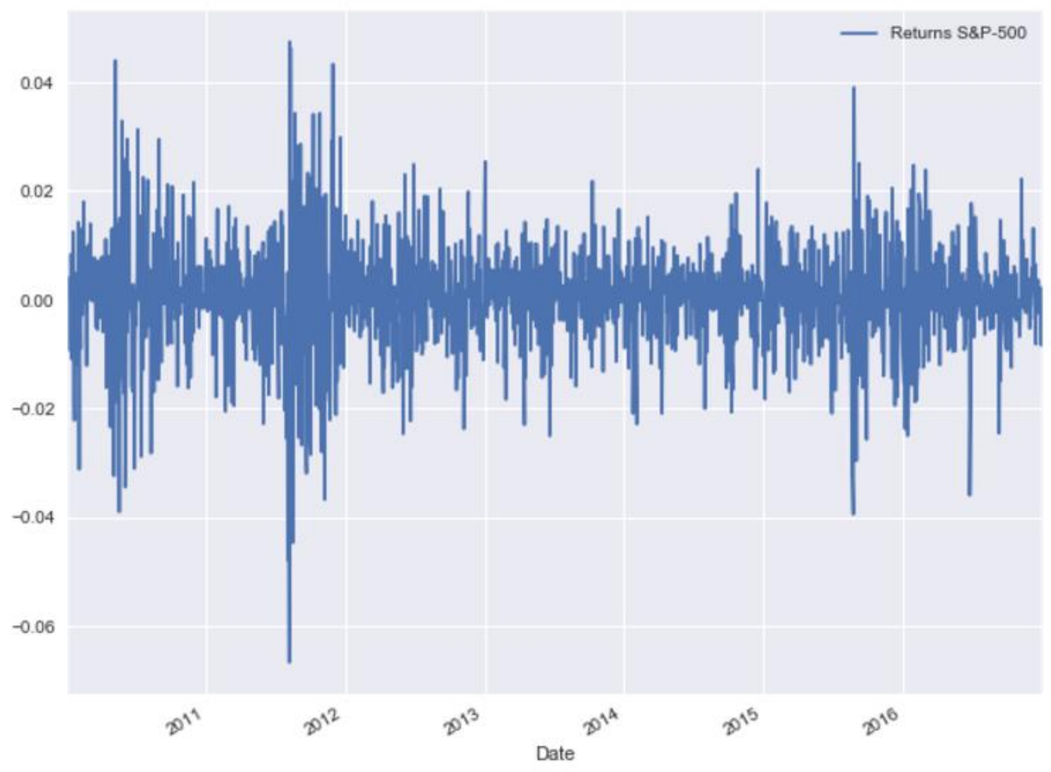


Figure 60. Percentage change of S&P-500

There is only 1 null value in the dataset which is the first value corresponding to 'Return' feature. Finally, the timeseries constructed has 1762 instances and 7 features.

The same process was applied to extract data for stock prices of other markets as well. The table given below illustrates the dataset formed after preprocessing and feature building with their respective number of rows and columns:

Market for stock prices	Instances	Features
Nasdaq Composite from Yahoo Finance (^IXIC)	1762	7
Dow Jones Industrial Average from Quandl (.DJI)	1762	7
5 Years US Treasury from Yahoo Finance (^FVX)	1762	7
Hong Kong Hang Seng from Yahoo Finance (^HIS)	1729	7
Frankfurt DAX from Yahoo Finance (^GDAXI)	1778	7
Paris CAC 40 from Yahoo Finance (^FCHI)	1790	7
Tokyo Nikkei-225 from Yahoo Finance (^N225)	1714	7
London FTSE-100 from Yahoo Finance (^FTSE)	1768	7

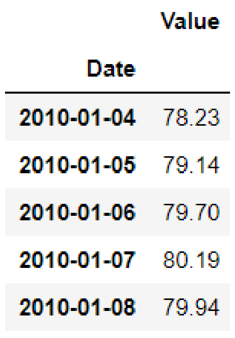
Figure 61. dimesnitons of different datasets

4.3.1.4 Extraction using Quandl

4.3.1.4.1 Data Preprocessing

The data for the other five datasets i.e. oil price, gold price, EUR vs USD, YEN vs USD and AUD vs USD was extracted from Quandl. Unlike Yahoo Financial, Quandl has an API for Python, R and MATLAB which allows extraction of data using any of the mentioned language. The format to fetch data is somewhat different i.e. instead of market symbols used previously, Quandl has keys assigned to each of the financial dataset that it has. An authorization token is required to connect the application which will fetch the data to Quandl and the start date and end date are mentioned while the call is made to extract the data.

The data for Oil Price US Department for Energy was extracted using the method mentioned above. It gave out two attributes date and value, which when saved in a pandas dataframe converted the data into a timeseries. The first five values are given below:



Date	Value
2010-01-04	78.23
2010-01-05	79.14
2010-01-06	79.70
2010-01-07	80.19
2010-01-08	79.94

Figure 62. Oil prices

The plot for oil prices over the duration from 2010 to 2016 is given below:

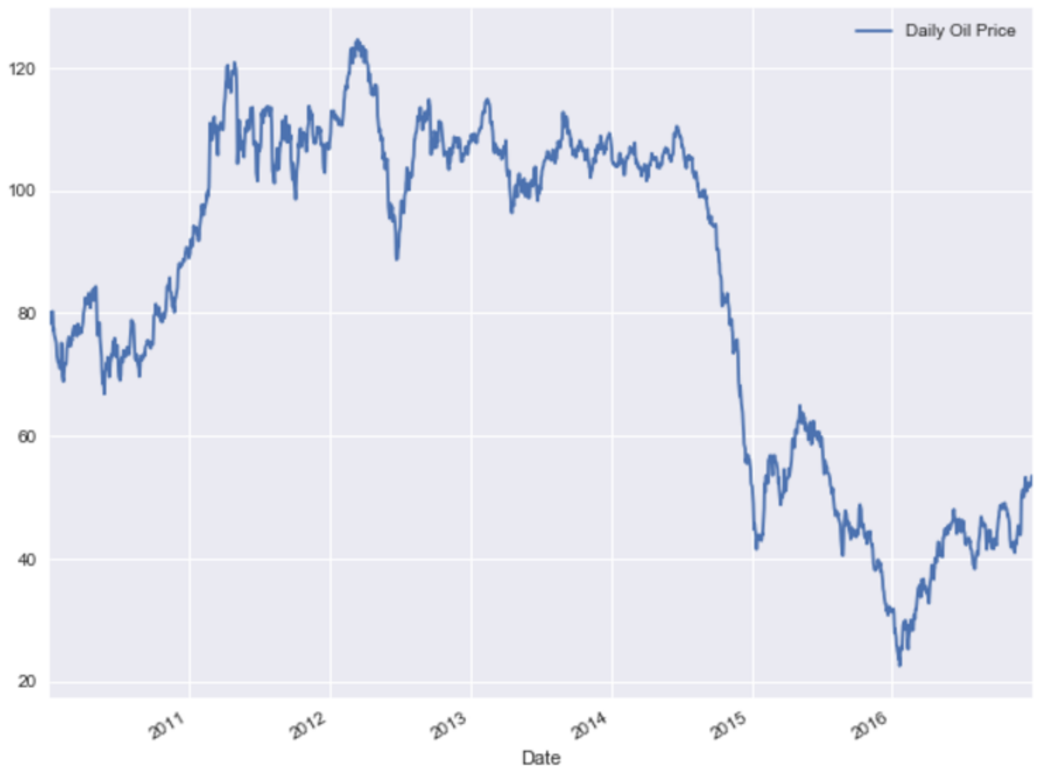


Figure 63. Oil prices for the past 7 years

4.3.1.4.2 Feature Building

A new feature 'Delta' was added which is again percentage change in the attribute 'Value' for two consecutive dates in the timeseries, just like 'Return' feature which was created for stock price datasets. The plot for percentage change for Oil prices is given below:

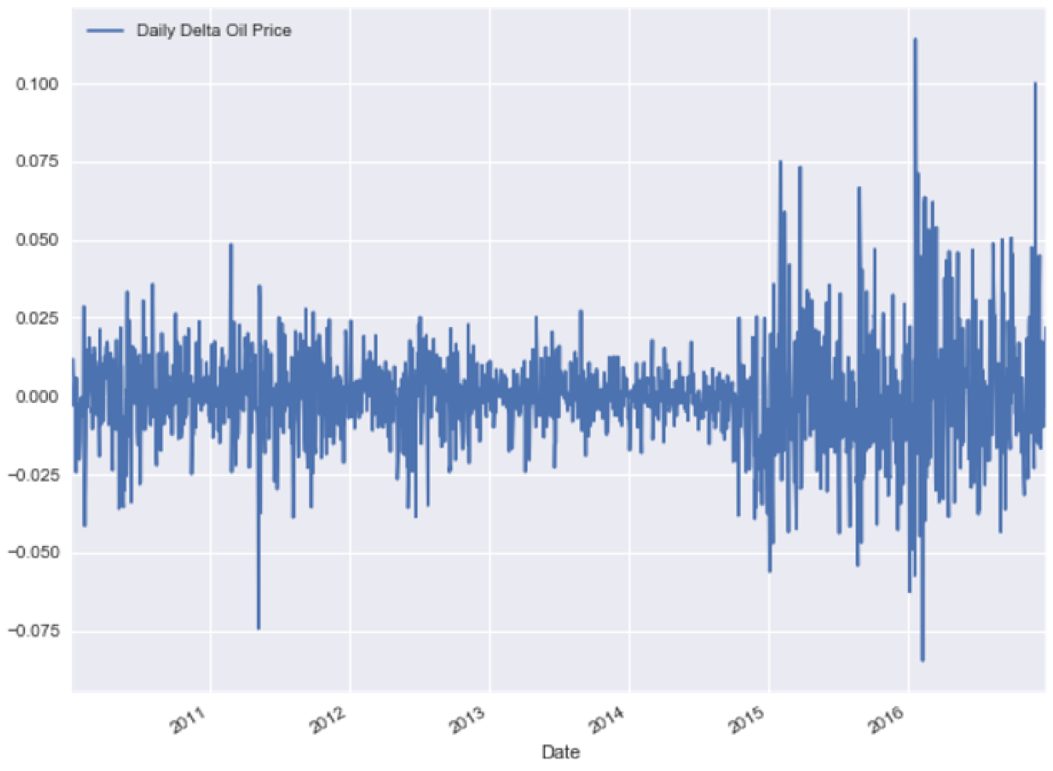


Figure 64. percentage change in oil prices

There is only 1 null value in the dataset which is the first value corresponding to 'Return' feature. Finally, the timeseries constructed has 1808 instances and 2 features.

The same process was applied to extract data for prices of other commodities as well. The table given below illustrates the dataset formed after preprocessing and feature building with their respective number of rows and columns:

Commodities	Instances	Features
Gold Price (Bundesbank) from Quandl	1589	2
Dollar in Euros Rate Exchange from Quandl	1794	2
Dollar in Japanese Yen Rate from Quandl	2555	2
Dollar in Australian Dollars Rate from Quandl	2457	2

Figure 65. dimensions of commoditeis datasets

The entire approach of extracting data for S&P-500 and Oil Prices can be found in Appendix 3 - Part A.

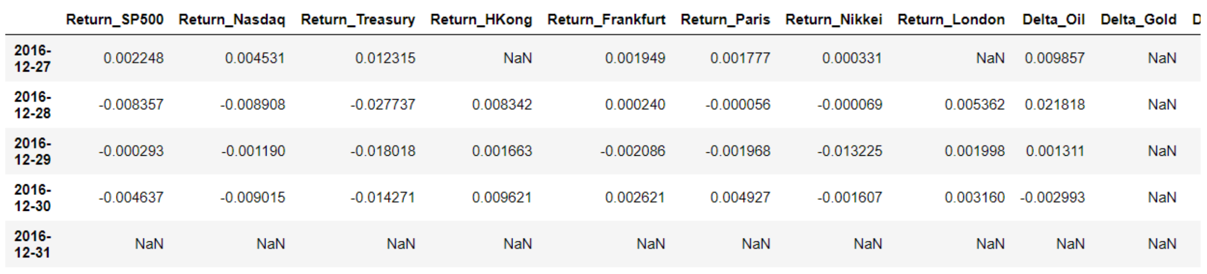
4.3.1.5 Data Preprocessing

After extraction of data from Yahoo Finance and Quandl, apart from the percentage change feature that was calculated for S&P-500, every other dataset is merged together as a single dataset. The code snippet for this is shown below:

```
to_be_merged = [nasdaq[['Return_Nasdaq']],
                treasuey[['Return_Treasury']],
                hkong[['Return_HKong']],
                frankfurt[['Return_Frankfurt']],
                paris[['Return_Paris']],
                nikkei[['Return_Nikkei']],
                london[['Return_London']],
                oil[['Delta_Oil']],
                gold[['Delta_Gold']],
                euro[['Delta_Euro']],
                yen[['Delta_Yen']],
                aus[['Delta_Aud']]]
```

Figure 66. Merging of datasets

The percentage change feature for S&P-500 is joined to the dataset obtained above, which gives the shape of the dataset as 2555 instances and 13 features. The last 5 instances of the dataset are shown below:

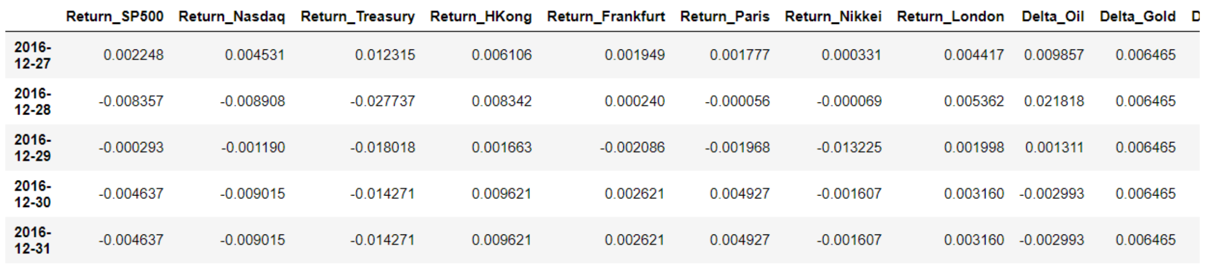


	Return_SP500	Return_Nasdaq	Return_Treasury	Return_HKong	Return_Frankfurt	Return_Paris	Return_Nikkei	Return_London	Delta_Oil	Delta_Gold	D
2016-12-27	0.002248	0.004531	0.012315	NaN	0.001949	0.001777	0.000331	NaN	0.009857	NaN	
2016-12-28	-0.008357	-0.008908	-0.027737	0.008342	0.000240	-0.000056	-0.000069	0.005362	0.021818	NaN	
2016-12-29	-0.000293	-0.001190	-0.018018	0.001663	-0.002086	-0.001968	-0.013225	0.001998	0.001311	NaN	
2016-12-30	-0.004637	-0.009015	-0.014271	0.009621	0.002621	0.004927	-0.001607	0.003160	-0.002993	NaN	
2016-12-31	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	

Figure 67. Merged dataset

As it can be observed there are many NaN's present in the dataset, these values are introduced in the dataset because the percentage changes are not available for all the dates i.e. the percentage change of adjusted close values that were calculated for Hong Kong market may not have the same dates as that of Germany market.

On further analysis, there were 8972 NaN's in the dataset. Therefore, to minimize these, pandas.Dataframe.interpolate [70] method was implemented which is used to perform interpolation at missing data points [71]. After this step, the number of NaN's in come down to 48. For these remaining null values, the values are filled by the mean values of the entire dataset. The dataset is modified which is shown in the figure below:



	Return_SP500	Return_Nasdaq	Return_Treasury	Return_HKong	Return_Frankfurt	Return_Paris	Return_Nikkei	Return_London	Delta_Oil	Delta_Gold	D
2016-12-27	0.002248	0.004531	0.012315	0.006106	0.001949	0.001777	0.000331	0.004417	0.009857	0.006465	
2016-12-28	-0.008357	-0.008908	-0.027737	0.008342	0.000240	-0.000056	-0.000069	0.005362	0.021818	0.006465	
2016-12-29	-0.000293	-0.001190	-0.018018	0.001663	-0.002086	-0.001968	-0.013225	0.001998	0.001311	0.006465	
2016-12-30	-0.004637	-0.009015	-0.014271	0.009621	0.002621	0.004927	-0.001607	0.003160	-0.002993	0.006465	
2016-12-31	-0.004637	-0.009015	-0.014271	0.009621	0.002621	0.004927	-0.001607	0.003160	-0.002993	0.006465	

Figure 68. Final dataset

4.3.1.5.1 Temporal Shifting

S&P-500 percentage change that was merged with the rest of the dataset as an outer join needs to be shifted backwards one day in order to have the return of today matched with the return on yesterday of the other predictors [72].

4.3.1.6 Classification Models

In order to perform prediction on the dataset, there needs to be labels in the dataset. Therefore, the feature 'Return_SP500' is converted into a binary feature in which the labels are 'Up' and 'Down'. When the percentage change is equal to or greater than 0, the label is

‘Up’ else it is ‘Down’. The data is then divided into training and testing sets with the testing set having all the labels i.e. the ‘UpDown’ feature and training containing all the other features.

The data was then split into the standard X_train, X_test, y_train, y_test with 80% of data being used for training and the rest for testing.

4.3.1.6.1 Linear Support Vector Machine [18]

The kernel implemented for SVM used was rbf (radial basis function), also known as Gaussian kernel. The classifier was trained and the model built was used to predict the values of the testing data. The accuracy came out to be 55.4%.

4.3.1.6.2 Random Forest Classifier [73]

A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and use averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is always the same as the original input sample size but the samples are drawn with the replacement if bootstrap=True.

The classifier was trained and the model built was used to predict the values of the testing data. The accuracy came out to be 86.3%.

4.3.1.6.3 K-Neighbours Classifier [74]

This is a non-parametric method used for classification and regression. The input consists of the k closest training examples in the feature space. The output is a class membership. An object is classified by a majority of its neighbours, with the object being assigned to the class most common among its k nearest neighbours. It is a lazy-learning algorithm where the function is only approximated locally and all computation is deferred until classification. The k-NN algorithm is among the simplest of all machine learning algorithms.

The classifier was trained and the model built was used to predict the values of the testing data. The accuracy came out to be 62.9%. The entire implementation is mentioned in Appendix 3 – Part B.

4.3.1.7 Conclusions

These testing accuracies may seem satisfactory for a domain such as stock prediction which is known for its fickleness because a small change in the trend can create huge impact. But there are two important points to consider here:

- The classification is being done on binary model. And testing accuracies of SVM and K-NN classifiers came out to be 55% and 62% respectively. This can be interpreted as, the model is anyways binary, 50% values will be always predicted correctly. That means the other label is being predicted correctly 5 percent of the times and 12 percent of the times for SVM and k-NN respectively, which is not good at all.
- Going along with the point mentioned above, Random Forest Classifier gave pretty good results i.e. an accuracy of 86.3%. But it being an ensemble model is expected to boost the results of other weak classifiers. Also, the duration for which the data is extracted for stock prices and commodities prices did not saw any major crashes in the financial world. Therefore, if economic bubbles are introduced in the timeseries the results are expected to go haywire.

4.3.2 Stock Predictions of 'Reliance' [75] using tweets.

4.3.2.1 Goal

This is the second method for predicting stocks. Unlike the last section, where classification was predicted on binary model, here the objective is not to perform classification but to predict how can tweets have any action on the stock prices of a company and is there any relation between these two entities.

4.3.2.2 Data Source

For this method, two datasets are needed. First the stock prices for the company in question and second tweets related to this company. The first part of the dataset can be easily downloaded from National Stock Exchange of India's [76] website. The second part is complicated.

4.3.2.2.1 Tweet extraction

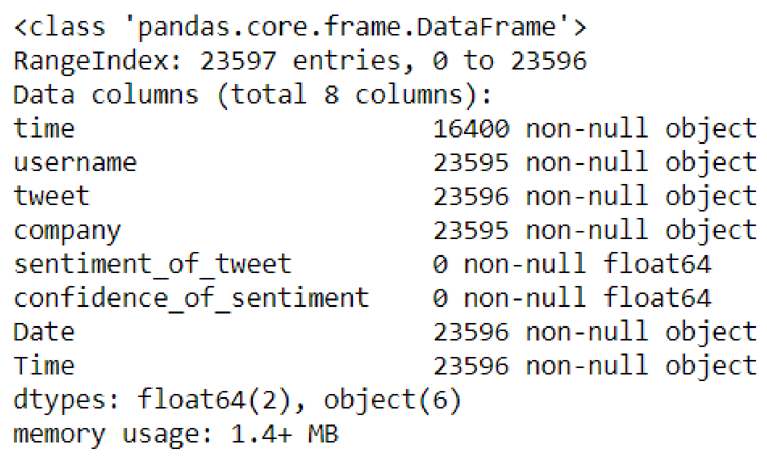
Python has a package, Twython [77] that can be used to extract tweets related to keywords. It can be used to query data for user information, twitter lists, timeline, direct messages, and anything else found in the Twitter API documentation [78].

The package was downloaded from Github using pip command. The requirements to use this package are that an application should be created using Twitter applications. This will generate certain keys and tokens which then can be used as authentication tokens in the code to extract data from twitter. The entire process is discussed in detail in a thread [79] on twitter community.

Once the credentials are created APP_KEY and APP_SECRET are supplied to Twython which then generates an ACCESS_TOKEN which can be used to perform search throughout twitter using a query. A query is generally a keyword or multiple keywords and using these Twython starts extracting metadata related to a tweet which has one or more of these keywords present in it. The incoming metadata can be saved as Json objects which can be used later for text processing. This entire approach can be seen in Appendix 3 – Part C.

4.3.2.3 Data Preprocessing

Data from JSON objects was extracted and saved into a CSV file. The CSV file was imported into the code as a pandas dataframe. The basic structure of this CSV file is given below:



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 23597 entries, 0 to 23596
Data columns (total 8 columns):
time                        16400 non-null object
username                    23595 non-null object
tweet                       23596 non-null object
company                     23595 non-null object
sentiment_of_tweet          0 non-null float64
confidence_of_sentiment     0 non-null float64
Date                        23596 non-null object
Time                        23596 non-null object
dtypes: float64(2), object(6)
memory usage: 1.4+ MB
```

Figure 69. Tweets of 'Reliance' dataset

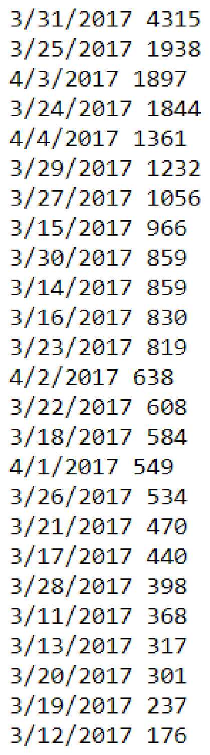
So, the dataframe has 23597 instances with 8 attributes. But two of these attributes were inserted manually i.e. sentiment_of_tweet and confidence_of_sentiment.

4.3.2.3.1 Calculating sentiment score

Python has a package Vader Sentiment [80] which can be used to calculate sentiment score of the tweets. This package is an implementation of VADER (Valance Aware Dictionary and Sentiment Reasoner) [81] which is a lexicon and rule based analysis tool that is specifically attuned to sentiment expressed in social media, and works well on texts from other domains. The calculations of sentiments score can be seen in Appendix 3 – Part D.

4.3.2.3.2 Adding the scores in dataset

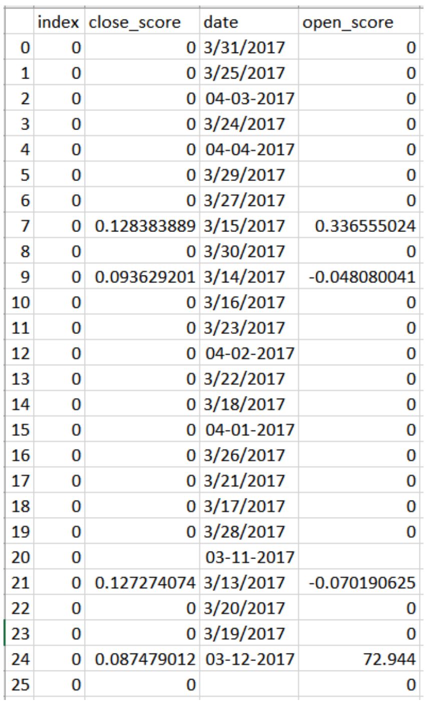
The duration chosen for analysis was 12th March 2017 to 4th April 2017 because at this point the company, Reliance had launched a new telecom product which was being promoted on twitter. So, this time had a good mixture of negative and positive sentiments for the company. The total number of tweets collected per day for the dates in the chosen time are given below:



```
3/31/2017 4315
3/25/2017 1938
4/3/2017 1897
3/24/2017 1844
4/4/2017 1361
3/29/2017 1232
3/27/2017 1056
3/15/2017 966
3/30/2017 859
3/14/2017 859
3/16/2017 830
3/23/2017 819
4/2/2017 638
3/22/2017 608
3/18/2017 584
4/1/2017 549
3/26/2017 534
3/21/2017 470
3/17/2017 440
3/28/2017 398
3/11/2017 368
3/13/2017 317
3/20/2017 301
3/19/2017 237
3/12/2017 176
```

Figure 70. total number of tweets collected per day

The next step involved creating a dataframe which had sentiment scores of tweets posted on different times throughout the day. As sentiments scores, according to times were already available in the dataframe just created, this time attribute was used to create sentiment scores at opening time of the stock market, for the company and at the closing time of the stock market, for the company. 9am to 4pm was determined the working time of the stock market for each day present in the dataframe. The results were saved in a JSON and a CSV file. The structure of CSV file is given below and the approach is mentioned in Appendix 3 – Part E:

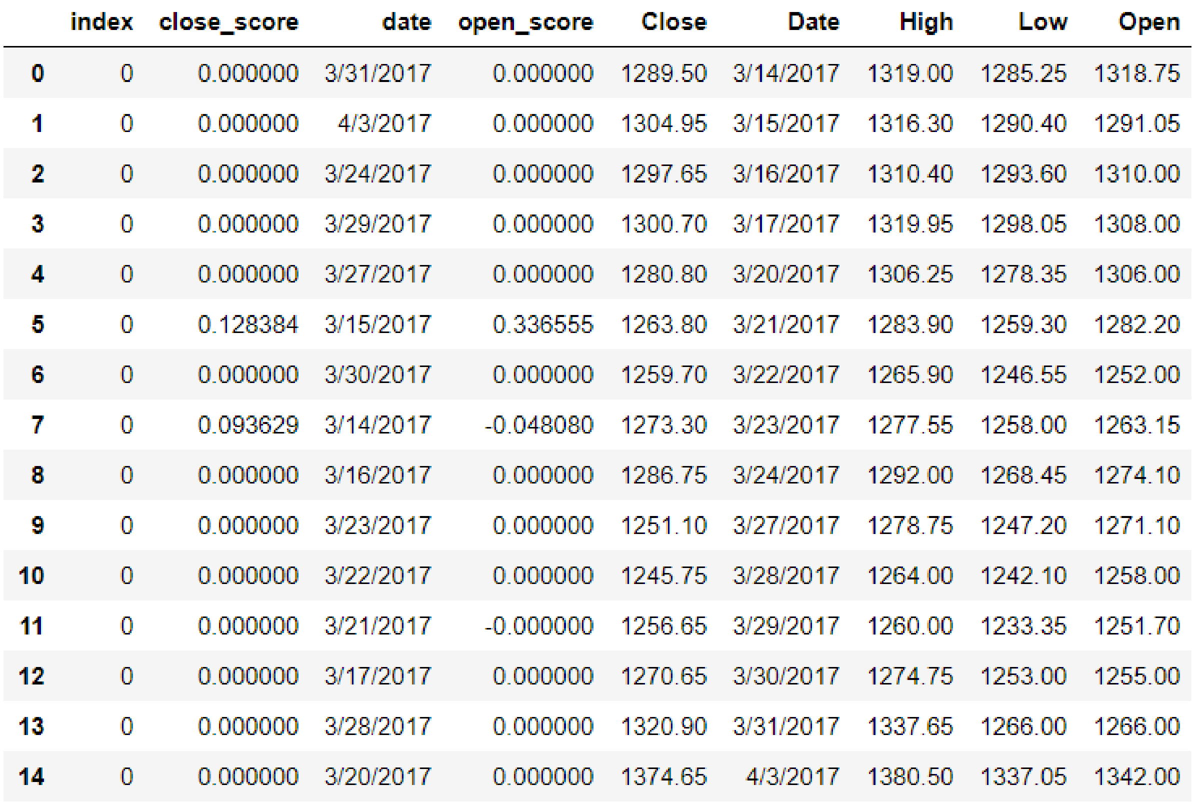


	index	close_score	date	open_score
0	0	0	3/31/2017	0
1	0	0	3/25/2017	0
2	0	0	04-03-2017	0
3	0	0	3/24/2017	0
4	0	0	04-04-2017	0
5	0	0	3/29/2017	0
6	0	0	3/27/2017	0
7	0	0.128383889	3/15/2017	0.336555024
8	0	0	3/30/2017	0
9	0	0.093629201	3/14/2017	-0.048080041
10	0	0	3/16/2017	0
11	0	0	3/23/2017	0
12	0	0	04-02-2017	0
13	0	0	3/22/2017	0
14	0	0	3/18/2017	0
15	0	0	04-01-2017	0
16	0	0	3/26/2017	0
17	0	0	3/21/2017	0
18	0	0	3/17/2017	0
19	0	0	3/28/2017	0
20	0		03-11-2017	
21	0	0.127274074	3/13/2017	-0.070190625
22	0	0	3/20/2017	0
23	0	0	3/19/2017	0
24	0	0.087479012	03-12-2017	72.944
25	0	0		0

Figure 71. close score and open score for a given date

4.3.2.3.3 Merging stock prices and sentiments scores

The file downloaded from national stock exchange website is merged with sentiments score. This is done by intersecting sets of dates in sentiment scores dataset and stock prices dataset. The final dataframe to be used is shown below:

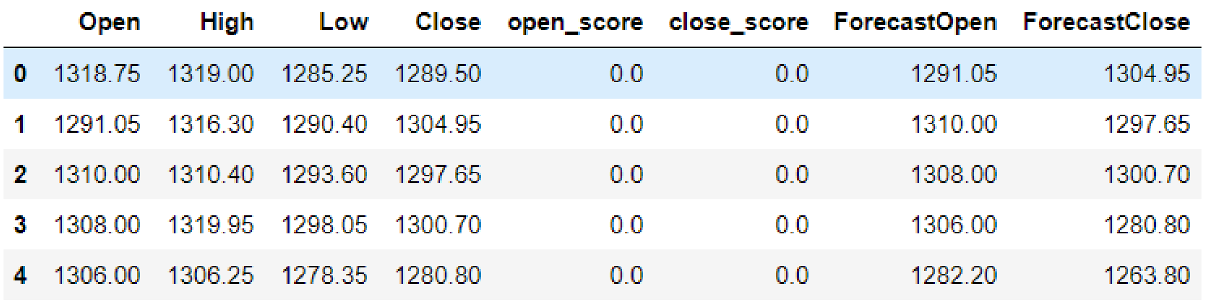


	index	close_score	date	open_score	Close	Date	High	Low	Open
0	0	0.000000	3/31/2017	0.000000	1289.50	3/14/2017	1319.00	1285.25	1318.75
1	0	0.000000	4/3/2017	0.000000	1304.95	3/15/2017	1316.30	1290.40	1291.05
2	0	0.000000	3/24/2017	0.000000	1297.65	3/16/2017	1310.40	1293.60	1310.00
3	0	0.000000	3/29/2017	0.000000	1300.70	3/17/2017	1319.95	1298.05	1308.00
4	0	0.000000	3/27/2017	0.000000	1280.80	3/20/2017	1306.25	1278.35	1306.00
5	0	0.128384	3/15/2017	0.336555	1263.80	3/21/2017	1283.90	1259.30	1282.20
6	0	0.000000	3/30/2017	0.000000	1259.70	3/22/2017	1265.90	1246.55	1252.00
7	0	0.093629	3/14/2017	-0.048080	1273.30	3/23/2017	1277.55	1258.00	1263.15
8	0	0.000000	3/16/2017	0.000000	1286.75	3/24/2017	1292.00	1268.45	1274.10
9	0	0.000000	3/23/2017	0.000000	1251.10	3/27/2017	1278.75	1247.20	1271.10
10	0	0.000000	3/22/2017	0.000000	1245.75	3/28/2017	1264.00	1242.10	1258.00
11	0	0.000000	3/21/2017	-0.000000	1256.65	3/29/2017	1260.00	1233.35	1251.70
12	0	0.000000	3/17/2017	0.000000	1270.65	3/30/2017	1274.75	1253.00	1255.00
13	0	0.000000	3/28/2017	0.000000	1320.90	3/31/2017	1337.65	1266.00	1266.00
14	0	0.000000	3/20/2017	0.000000	1374.65	4/3/2017	1380.50	1337.05	1342.00

Figure 72. pre-processed dataset

4.3.2.4 Forecasting

To evaluate the results of the model being used for forecasting, a shift of -1 was done on Open and Close attribute and was added in the dataset, thus changing the structure to:



	Open	High	Low	Close	open_score	close_score	ForecastOpen	ForecastClose
0	1318.75	1319.00	1285.25	1289.50	0.0	0.0	1291.05	1304.95
1	1291.05	1316.30	1290.40	1304.95	0.0	0.0	1310.00	1297.65
2	1310.00	1310.40	1293.60	1297.65	0.0	0.0	1308.00	1300.70
3	1308.00	1319.95	1298.05	1300.70	0.0	0.0	1306.00	1280.80
4	1306.00	1306.25	1278.35	1280.80	0.0	0.0	1282.20	1263.80

Figure 73. adding temporal shift for open price and close price

To implement forecasting for open prices and close prices three approaches are used:

- Traditional approach – Just using Open, Close, High and Close for training the model.
- Sentiment approach – Using open scores and close scores for training the model.

- Hybrid approach – Using all the attributes to train the model and predict the prices for next day.

These three methods have been implemented to forecast next day's Open price and Close price. The results are given below:

Evaluation metric – coefficient of determination R^2 of the prediction	Open Price	Close Price
Traditional approach	60.9	51.21
Sentiment Approach	-98.39	-848.922
Hybrid Approach	1.07	-27.33

Figure 74. Performance evaluation of different approaches

The evaluation metric used here is the coefficient of determination R^2 [82] of the prediction. The equation to determine R is:

$$r = \frac{n(\Sigma xy) - (\Sigma x)(\Sigma y)}{\sqrt{[n\Sigma x^2 - (\Sigma x)^2][n\Sigma y^2 - (\Sigma y)^2]}} \quad \text{...(3)}$$

Squaring this result and multiplying by 100 gives out the coefficient of determination. The best possible score is 1.0 and it can be negative because the model can be arbitrarily worse. Therefore, the results received for Hybrid Approaches for forecasting open and close prices outperforms the other two approaches.

4.3.2.5 Conclusion

Vader Sentiment performs well for online text data as other packages like Textblob and nltk were also tried but didn't gave a good result. The approach discussed in the Background work related to this section was sloppy and the results were being chosen and shown as the best. But in the approach taken for this section compared all the three methods of forecasting stock prices i.e. one without sentiments, second with just sentiments but no stock prices and the third included both.

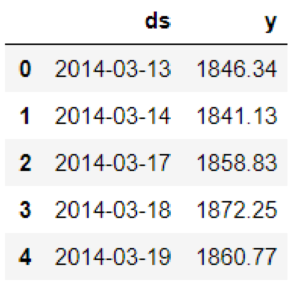
4.3.3 Stock Prediction using Facebook's Prophet

4.3.3.1 Goal:

Not long ago, Facebook released its forecasting library named as Prophet [83]. Python's package fbprophet can be used to use these libraries. The goal is to see how accurate Prophet can be and is it better in predicting stock prices than using sentiment approach.

4.3.3.2 Data Preprocessing

The data used here is S&P-500 stock prices covering a duration from 13th March 2014 to 13th March 2017. It was downloaded from Yahoo Finance website. The dataset had attributes like Open, Close, High, Low, Adjusted Close and Volume. But as it is previously discussed the most accurate representation of a stock price on any given day is its Adjusted Close Price, therefore all other attributes were dropped and just Adjusted close price attribute was included along with the dates. This is also done because Prophet library expects as input a Dataframe with one column containing the time information ad another column containing the metric that is needed to be forecasted. The dataset has the structure shown below:



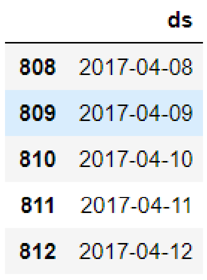
	ds	y
0	2014-03-13	1846.34
1	2014-03-14	1841.13
2	2014-03-17	1858.83
3	2014-03-18	1872.25
4	2014-03-19	1860.77

Figure 75. S&P-500 dataset with just adjusted close price

Before any analysis is done, log transformation is applied on 'y' attribute to try to convert non-stationary data to stationary. This also converts trends to more linear trends by decreasing the skewness in the data.

4.3.3.3 Forecasting

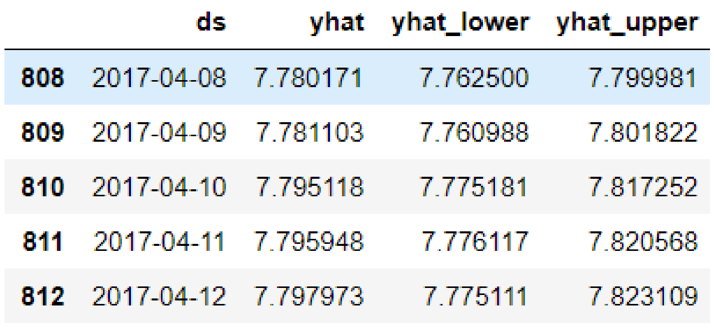
The next step involves training a model with the dataset. The authors of Prophet have abstracted away many of the inherent complexities if time series forecasting and have made it more intuitive for analysts and developers alike to work with time series data. The growth of the model is selected as linear in this case. After training values are predicted for the next 30 days. This is done by creating a future data frame using Prophet where the parameters include the period and frequency. Here the period is 30 and frequency is D i.e. days because a forecast of next 30 days is required. The dataframe created can be seen below:



	ds
808	2017-04-08
809	2017-04-09
810	2017-04-10
811	2017-04-11
812	2017-04-12

Figure 76. Future dataframe

Using this dataframe prediction is made for the next 30 days. The dataframe received as a result is given below:



	ds	yhat	yhat_lower	yhat_upper
808	2017-04-08	7.780171	7.762500	7.799981
809	2017-04-09	7.781103	7.760988	7.801822
810	2017-04-10	7.795118	7.775181	7.817252
811	2017-04-11	7.795948	7.776117	7.820568
812	2017-04-12	7.797973	7.775111	7.823109

Figure 77. Forecasting results

Prophet returns a large Dataframe with many interesting columns, but a subset of the output was created and the columns most relevant to forecasting are:

- ds: the datestamp of the forecasted value
- yhat: the forecasted value of the metric i.e. adjusted close price for the next 30 days.
- yhat_lower: the lower bounds of the forecast
- yhat_upper: the upper bound of the forecast.

A variation in values from the output presented above is to be expected as Prophet relies on Markov chain Monte Carlo (MCMC) methods to generate its forecast. MCMC is a stochastic process, so values will be slightly different each time.

The metric used to evaluate the results was root mean square error (RMSE) and evaluation was done by downloading data from yahoo finance for the next 30 days and comparing the predicted and actual values. The result came out to be 0.0214, which is much better than expected.

The forecasted plot is given below:

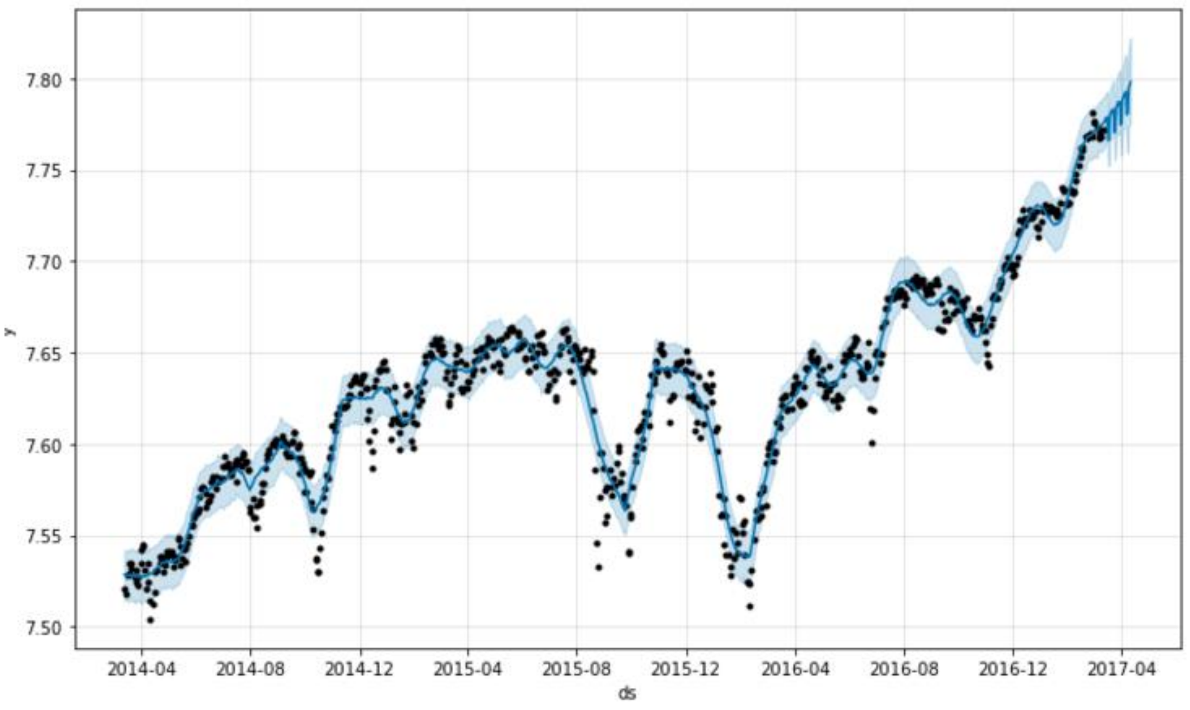


Figure 78. Forecast plot

Prophet plots the observed values of the time series as the black dots, the forecasted values as the blue line and the uncertainty intervals as the blues shaded region. The part at the end of the plot without any black dots is the prediction that Prophet made for the 30 days interval. The entire approach can be seen in Appendix 3 – Part F. The mathematical equation used for forecasting is given below:

$$g(t) = (k + \mathbf{a}(t)^\intercal \boldsymbol{\delta})t + (b + \mathbf{a}(t)^\intercal \boldsymbol{\gamma}), \qquad \text{...(4)}$$

4.4 Conclusion and Future Work

So, in this chapter three approaches were discussed. The first one that only included stock prices and was approaches as a classification problem belong to the algorithm based trading [84] approaches that were used in the late 1980s but with time this approach became obsolete and was replaced by High-frequency trading [85] in which instead on concentrating on prices at closing of the market at the end of the day, data is being collected for small intervals such as 5 minutes or 10 minutes and prediction are being made for the next 5 minutes. This is a very fine-grained approach for predicting stock values which requires very fast processing and high-end resources.

The second approach saw that making forecast using sentiments can be achieved with good results. For this the future work includes expansion from one company to multiple companies or even an entire market such as NASDAQ or NSE India. Also, more filtering can be done while performing text processing like removing tweets which do not concern the performance of the predictive model.

The third approach was the best as compared to the other two in the sense that it requires less amount of time to be processed and was done on very little amount of data. Even though Prophet still has some limits like lags cannot be created in the timeseries if this library has to be used. But, the biggest advantage of this library is that it is optimized for the forecast tasks, which typically have any of the following characteristics:

- hourly, daily, or weekly observations with at least a few months (preferably a year) of history
- strong multiple "human-scale" seasonalities: day of week and time of year
- important holidays that occur at irregular intervals that are known in advance (e.g. the Super Bowl)
- a reasonable number of missing observations or large outliers
- historical trend changes, for instance due to product launches or logging changes
- trends that are non-linear growth curves, where a trend hits a natural limit or saturates

Chapter 5 – Fake profile detection using Facebook data

5.1 Motivation

Imitation is said to be the highest form of flattery, but that is hardly the case on Facebook. Every day Facebook receives countless reports from people and page administrators that have had their profiles or pages copied by scammers or bullies. The most common scenario is someone copies the name, profile picture and other available photos of the intended victim. The next thing they do is block the person they are impersonating and send friend requests to everyone on the Victim's friends list. This is done to infiltrate their social networks. Once this is complete, the scammer has a variety of option at their disposal:

- Data mine the accounts they have friended under the bogus profile.
- Gather enough information and target close friends and family members for scamming purposes.
- Spam scam links to everyone on their friends list. This is common social engineering tactic used by online fraudsters. The goal is to use the trust of the victim's social circle to make the scam more believable.

Facebook reportedly has around 170 million fake users. Even though on March 12, 2015, Facebook purged fake accounts and millions of profiles disappeared overnight, the issue persists. While these accounts were deleted by Facebook, fake profiles are still being created.

In this chapter, more than classifying real profiles and fake profiles, the important topic to discover would be what are the factors that mark the profile as a fake profile i.e. which features of a profile can be used to label that profile as real or fake.

5.2 Background / Similar work

1. The paper "Detecting Fake Profiles in On-Line Social networks" [86] has discussed the mitigation of problems caused by detecting fake profiles of people who do not have a social media account. Their approach claims to be the first to analyse social network graphs from a dynamic point of view within the context of privacy threats.

2. "Automatic Detection of Fake Profiles in Online Social Networks" [87] discusses automation of detecting fake profiles using classification techniques like Support Vector Machine, Naïve Bayes and Decision Trees to classify the profiles into fake or genuine classes.
3. "An analysis of Social networks-based Sybil Defences" [88] shows that existing Sybil defences schemes, despite their considerable differences work by detecting local communities (i.e. clusters of nodes more tightly knit than the rest of the graph) around a trusted node.

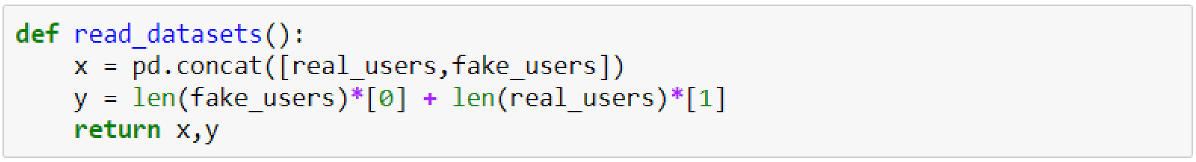
5.3 Data Source

Two CSV files are used for this task. One has all the profiles that are genuine and the other file has all the fake profiles. These files are fetched and saved into two different pandas dataframes that are namely, real_users and fake_users.

5.4 Data Preprocessing

The structure of both the CSV files was same. Therefore, apart from the instances the attributes will be the same. Exploring the structure of real_users gives out the information that it has 1481 instances and a total of 34 columns or attributes. Out of these 34 attributes 8 are of float type, 6 are of integer type and 20 are object.

The next logical step is to concatenate these two dataframes into 1 and label them. This is done by using a small function given below:



```
def read_datasets():
    x = pd.concat([real_users,fake_users])
    y = len(fake_users)*[0] + len(real_users)*[1]
    return x,y
```

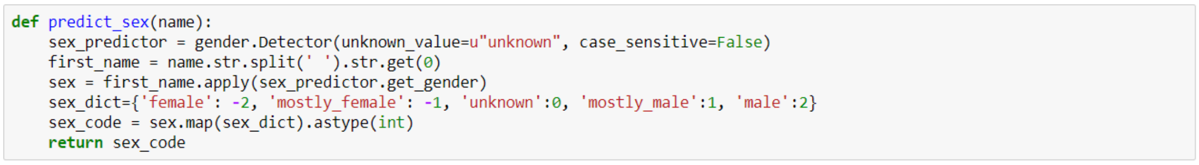
Figure 79. concatenating dataframes and calculation of labels

This function concatenates the real_users and fake_users dataset and returns the resulting dataframe into x and y calculates the length of these two dataframes and for fake users creates a list of 0 and for real users creates a list of 1. This list will act as the labels for classification models.

5.5 Feature Extraction

First, language codes are converted from string to a list of integers by enumerating this language attribute. Then these integer codes are assigned to their respective instances.

Second feature extraction done is to fetch the gender of the user using a Python package called Sex Machine [89], which uses the underlying data from the program “gender” by Jorg Michael [90]. The function takes the first name of a person and classifies the gender into Male , Female, mostly male, mostly female or ‘unknown which means androgynous. These 5 categories are then assigned an integer and then a variable sex_code is returned containing one of the 5 values according to classification by this package. The function is shown below:



```
def predict_sex(name):
    sex_predictor = gender.Detector(unknown_value=u"unknown", case_sensitive=False)
    first_name = name.str.split(' ').str.get(0)
    sex = first_name.apply(sex_predictor.get_gender)
    sex_dict={'female': -2, 'mostly_female': -1, 'unknown':0, 'mostly_male':1, 'male':2}
    sex_code = sex.map(sex_dict).astype(int)
    return sex_code
```

Figure 80. code snippet for sec machine

Finally, features are extracted from the dataframe columns. The columns selected to be features are:

- statuses count
- followers count
- friends count
- favourites count
- listed count
- sex code
- lang code

A thing to notice here is that all the features that are created are numeric in nature.

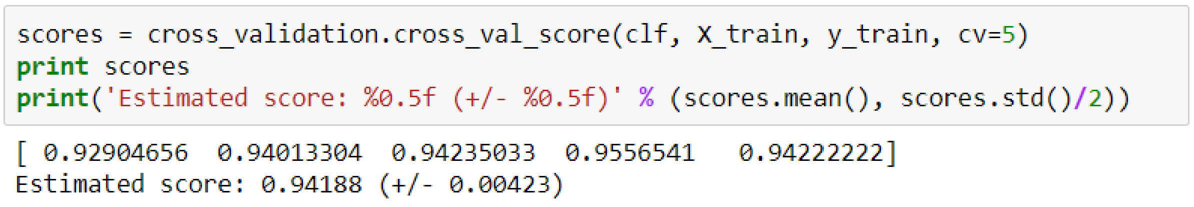
5.6 Model

5.6.1 Random Forest Classifier [73]

The dataset builds after feature selection is then split into the standard X_train, X_test, y_train, y_test. Here the testing size is 20% and training size is set to 80%. An instance of Random Forest with the number of decision trees to be used is set to 40 and OOB score set to true, is declared.

The Random Forest Classifier is trained using bootstrap aggregation, where each new tree is fit from a bootstrap sample of the training observations $z_i = (x_i, y_i)$. The out-of-bag OOB error is the average error for each z_i calculated using predictions from the trees that do not contain z_i in their respective bootstrap sample. This allows the random forest to be fit and validated whilst being trained [91].

The scores are calculated using 5-fold cross validation [92]. One round of cross-validation involves partitioning a sample of data into complementary subsets, performing the analysis on one subset and validating the analysis on the other subset. TO reduce variability, multiple rounds of cross-validation are performed using different partitions, and the validation results are combined over the rounds to estimate a final predictive model. In summary, cross-validation combines (averages) measures to fit (prediction error) to derive a more accurate estimate of model prediction performance. The estimated score of training the model comes out to be:



```
scores = cross_validation.cross_val_score(clf, X_train, y_train, cv=5)
print scores
print('Estimated score: %0.5f (+/- %0.5f)' % (scores.mean(), scores.std()/2))
```

```
[ 0.92904656  0.94013304  0.94235033  0.9556541   0.94222222]
Estimated score: 0.94188 (+/- 0.00423)
```

Figure 81. Random Forest scores

The learning curve depicting the training scores and cross-validation scores is depicted below:

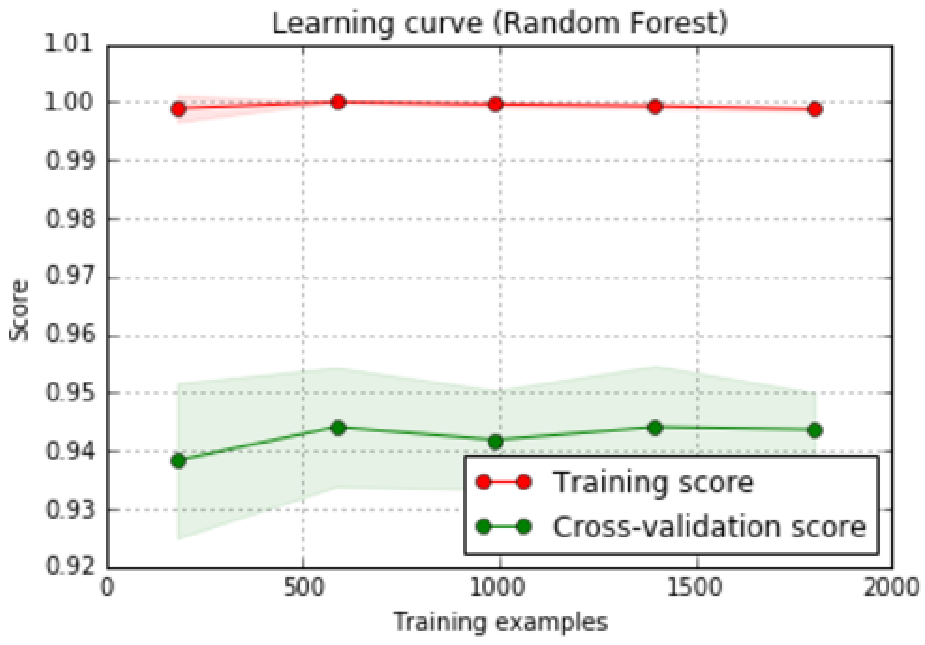


Figure 82. Random forest - learning curve

The accuracy on test data using this model came out to be 94.5%. Given below is the normalized confusion matrix and its plot:

```
Normalized confusion matrix
[[ 0.99626866  0.00373134]
 [ 0.10135135  0.89864865]]
```

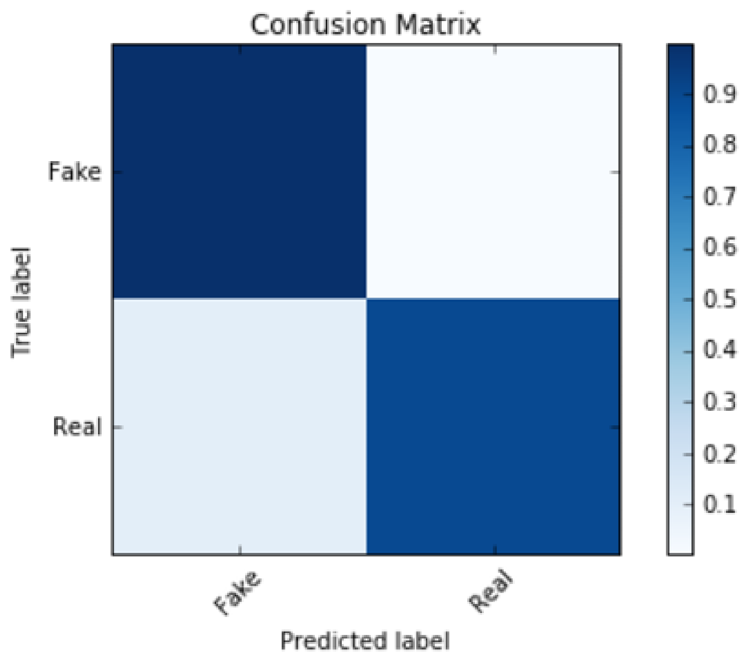


Figure 83. Random Forest - confusion matrix

5.6.1.1 Evaluation metric – Receiver Operating Characteristic (ROC) [93]

ROC Curves are used to see how well a classifier can separate positive and negative examples and to identify the best threshold for separating them. Initially this metric was used in Signal Detection Theory for distinguishing noise from not noise [94], so it's a way of showing the performance of Binary Classifiers. It's created by plotting the fraction of True Positives vs the fraction of False Positives.

$$\mathrm{tpr} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \qquad \mathrm{fpr} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}}$$

$$\mathrm{FP}$$

……………………… (5)

The plot for ROC with TPR and FPR is given below:

```
False Positive Rate:  [ 0.          0.00373134  1.        ]
True Positive Rate:  [ 0.          0.89864865  1.        ]
```

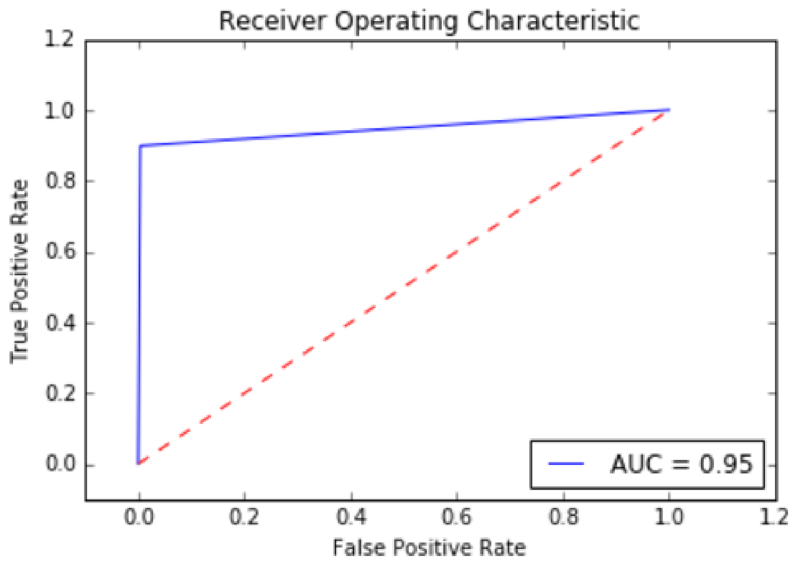
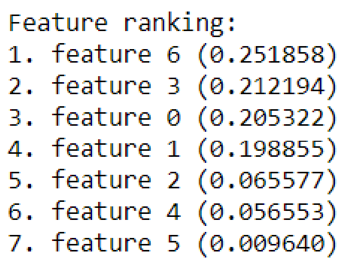


Figure 84. Random Forest - ROC curve

5.6.1.2 Feature Importance

Coming to the aim that was decided in the initial part of the chapter, a plot is created to answer what features make a profile real or fake. SO, the first step is to calculate the percentages of importances of the features used in classification. The ranks are shown below:



```
Feature ranking:
1. feature 6 (0.251858)
2. feature 3 (0.212194)
3. feature 0 (0.205322)
4. feature 1 (0.198855)
5. feature 2 (0.065577)
6. feature 4 (0.056553)
7. feature 5 (0.009640)
```

Figure 85. Random Forest - feature ranking

The plot for these percentages with feature name is given below:

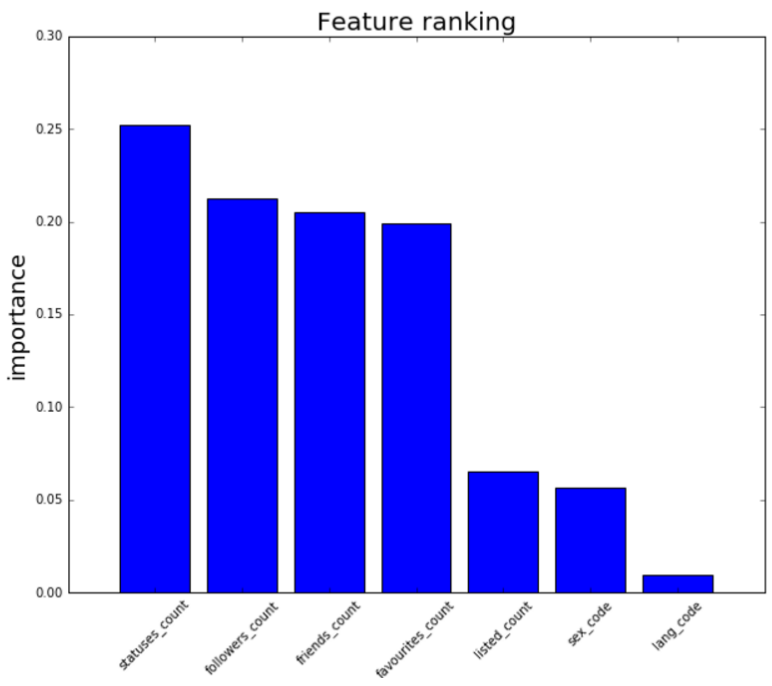


Figure 86. Random Forest - feature importance plot

5.7 Conclusion and Future Work

According to the goal that was set, the most important features from the dataset provided are ‘statuses count’ followed by ‘followers count’ and then ‘friends count’ which helped the classifier to detect whether the profile in question was fake or real. This result makes sense because a group of fake profiles will have status count in a particular range whereas status count for real profiles can be anything. Also, a fake profile will be following huge number of people but will have less users in its friend list. This won’t be true for a real profile because followers count and friends count both, can be arbitrary.

The future work for this chapter involves doing classification using neural networks and comparing the performance of and ensemble working on a dataset created using feature selection or a large neural network i.e. too many epochs and a really low learning rate but no features.

Chapter 6 – Intrusion Detection System

6.1 Introduction

An intrusion detection system monitors network traffic and monitors for suspicious activities and alerts the system or network administrator. In some cases, the IDS may also respond to anomalous or malicious traffic by acting such as blocking the user or source IP address from accessing the network.

An IDS needs only to detect threats and as such is placed out-of-band on the network infrastructure, meaning that it is not in the true real-time communication path between the sender and the receiver of the information. Rather, IDS solutions will often take advantage of a TAP or a SPAN port to analyse a copy of the inline traffic stream and thus ensuring that IDS does not impact inline network performance.

IDS was originally developed this way because at the time the depth of analysis required for intrusion detection could not be performed at a speed that could keep pace with components on the direct communications path of the network infrastructure.

As explained, the IDS is also a listen-only device. The IDS monitors traffic and reports its results to an administrator, but cannot automatically act to prevent a detected exploit from taking over the system. Attackers are capable of exploiting vulnerabilities very quickly once they enter the network, rendering the IDS an inadequate deployment for prevention device.

6.2 Background / Similar Work

1. "Random-Forest-Based Network Intrusion Detection Systems" [95] discusses how a hybrid model built on the principles of classification algorithms and anomaly detection outperforms the winning solution of the KDD99Cup challenge.
2. "An Intrusion-Detection Model Based on Fuzzy Class-Association-Rule Mining Using Genetic Network Programming" [96] describes a model based on genetic network programming for detecting network intrusions. GNP is an evolutionary optimization technique which uses directed graph structures instead of strings in genetic algorithm

or trees in genetic programming. The proposed method was implemented on KDD99Cup challenge and the results showed competitively high detection rates as compared to other machine-learning techniques and GNP with crisp data mining.

3. “Intrusion detection system with Wavelet and Neural Artificial Network Approach for Networks” [97] proposed an IDS based on ANN and applied it to KDD99Cup which showed high detection rates suggesting that the approach was very promising.
4. “A New Data-Mining Based Approach for Network Intrusion Detection” [98] proposed an ensemble of binary classifiers with feature selection and multiboosting simultaneously. This approach provided better performance in terms of accuracy and cost than the winner entry of the KDD99Cup challenge.

6.3 Motivation

IN 1999 KDD held The Third International Knowledge Discovery and Data Mining Tools Competition [99]. The competition task was to build a network intrusion detector, a predictive model capable of distinguishing between ‘bad’ connection, called intrusions or attacks, and ‘good’ connections.

But before this in 1998, DARPA Intrusion Detection Evaluation Program was prepared and managed by MIT Lincoln Labs. The objective was to survey and evaluate research in intrusion detection. A standard set of data to be audited, which included a wide variety of intrusions simulated military network environment, was provided. The 1999 KDD intrusion detection contest used a version of this dataset.

The goal here is to use the widely renowned KDD99Cup dataset and apply ensemble model without converting the dataset into a bivariate model which have been done in the papers discussed in the Background section and also by the winner of the challenge at that time and applying predictive analytics on the existing multivariate model and outperforming the winning solution.

6.4 Data Source

Lincoln Labs set up an environment to acquire nine weeks of raw TCP dump data for a local-area network simulating a typical U.S. Air Force Lan. They operated the LAN as if it were a true Air Force environment, but prepared it with multiple attacks.

The raw training data was about four gigabytes of compressed binary TCP dump data from seven weeks of network traffic. This was processed into about 5 million connection records. Similarly, the two weeks of test data yielded around two million connection records.

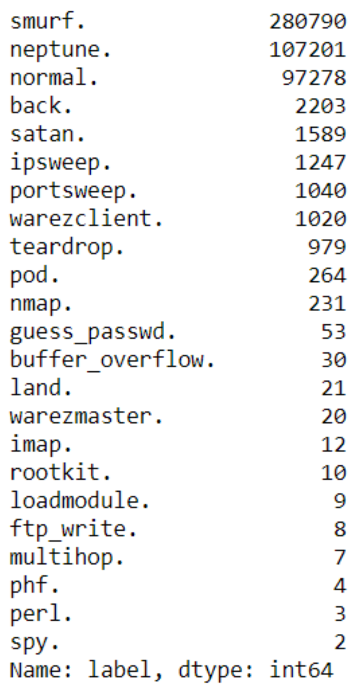
It is important to know that the test data is not from the same probability distribution as the training data, and it includes specific attack types not in the training data. This made the task more realistic. The datasets contain a total of 24 training attack types with an additional 14 types in the test data only.

The dataset is currently available on UCI's archives website [100] and has been characterized as a multivariate model having categorical and integer attributes with number of attributes equal to 42 and number of instances being 4 million.

6.5 Data Processing

The data while being fetched in pandas dataframe didn't had column names, therefore column names were entered manually and then the dataset was fetched. The description of the columns can be found here [101]. Initially, the dataset used for the implementation of classification algorithm was a 10% subset of the actual dataset. It had 42 attributes and 494021 instances.

The attribute 'label' is the prediction label that will be used in the model. It has 24 different types of attacks which along with their frequencies are shown below:



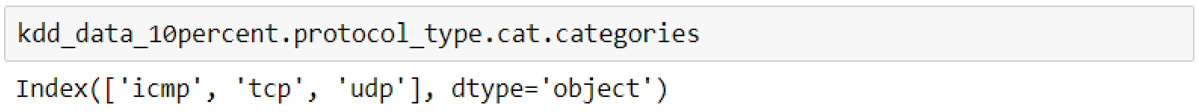
```
smurf.              280790
neptune.            107201
normal.              97278
back.                 2203
satan.                1589
ipsweep.              1247
portsweep.            1040
warezclient.          1020
teardrop.              979
pod.                   264
nmap.                  231
guess_passwd.           53
buffer_overflow.        30
land.                   21
warezmaster.            20
imap.                   12
rootkit.                10
loadmodule.              9
ftp_write.               8
multihop.                7
phf.                     4
perl.                    3
spy.                     2
Name: label, dtype: int64
```

Figure 87. 24 labels for types of attacks

Out of the 42 attributes, 4 of them are converted from objects to categories. These are:

- Protocol type – type of the protocol, e.g. tcp, udp, etc.
- Service – network service on the destination, e.g., http, telnet, etc.
- Flag – normal or error status of the connection
- Label – the attack labels.

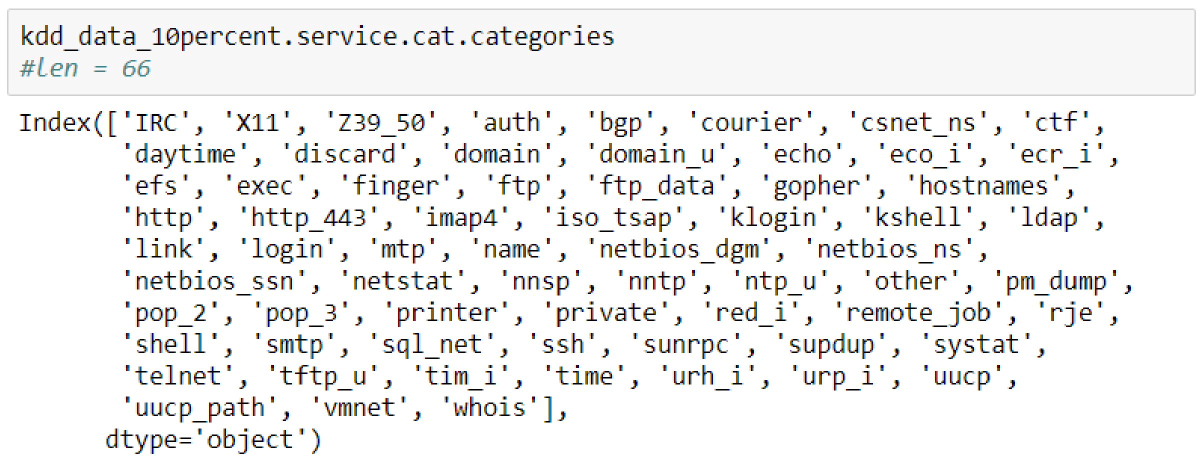
After converting them, the categories of these attributes were explored which are shown below:



```
kdd_data_10percent.protocol_type.cat.categories
```

```
Index(['icmp', 'tcp', 'udp'], dtype='object')
```

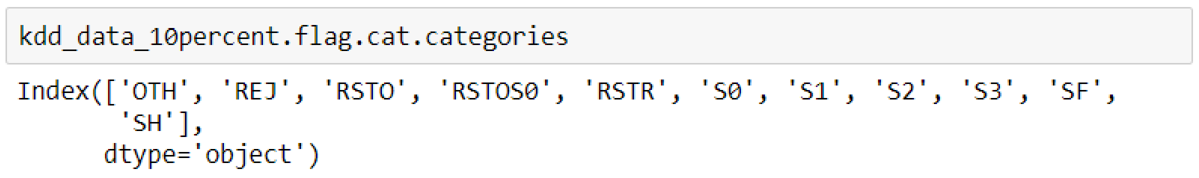
Figure 88. Protocol type - categories



```
kdd_data_10percent.service.cat.categories
#len = 66
```

```
Index(['IRC', 'X11', 'Z39_50', 'auth', 'bgp', 'courier', 'csnet_ns', 'ctf',
       'daytime', 'discard', 'domain', 'domain_u', 'echo', 'eco_i', 'ecr_i',
       'efs', 'exec', 'finger', 'ftp', 'ftp_data', 'gopher', 'hostnames',
       'http', 'http_443', 'imap4', 'iso_tsap', 'klogin', 'kshell', 'ldap',
       'link', 'login', 'mtp', 'name', 'netbios_dgm', 'netbios_ns',
       'netbios_ssn', 'netstat', 'nnsp', 'nntp', 'ntp_u', 'other', 'pm_dump',
       'pop_2', 'pop_3', 'printer', 'private', 'red_i', 'remote_job', 'rje',
       'shell', 'smtp', 'sql_net', 'ssh', 'sunrpc', 'supdup', 'systat',
       'telnet', 'tftp_u', 'tim_i', 'time', 'urh_i', 'urp_i', 'uucp',
       'uucp_path', 'vmnet', 'whois'],
      dtype='object')
```

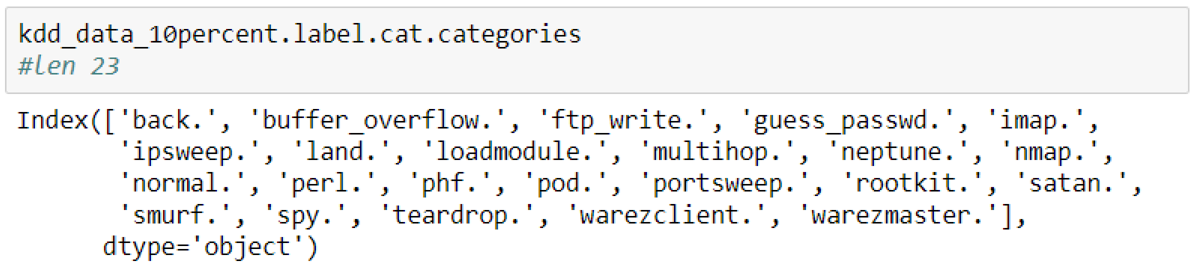
Figure 89. Service - categories



```
kdd_data_10percent.flag.cat.categories
```

```
Index(['OTH', 'REJ', 'RSTO', 'RSTOS0', 'RSTR', 'S0', 'S1', 'S2', 'S3', 'SF',
       'SH'],
      dtype='object')
```

Figure 90. Flag - categories



```
kdd_data_10percent.label.cat.categories
#len 23
```

```
Index(['back.', 'buffer_overflow.', 'ftp_write.', 'guess_passwd.', 'imap.',
       'ipsweep.', 'land.', 'loadmodule.', 'multihop.', 'neptune.', 'nmap.',
       'normal.', 'perl.', 'phf.', 'pod.', 'portsweep.', 'rootkit.', 'satan.',
       'smurf.', 'spy.', 'teardrop.', 'warezclient.', 'warezmaster.'],
      dtype='object')
```

Figure 91. label - categories

6.6 Feature Building

The only step performed for feature building is to convert the attributes of the dataset whose data type was integer into float.

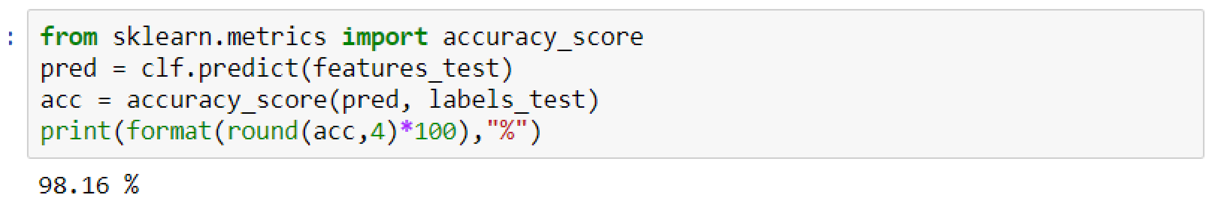
As the preprocessing and feature building was done on the 10% dataset, it now had to be implemented on the actual dataset. The actual dataset was fetched in the pandas dataframe and had 311029 instances and 42 attributes. All the steps of data preprocessing and feature building were repeated for the actual dataset. Finally, Min Max scaler was applied on the dataset by scaling each feature to a particular range.

6.7 Classification

The dataset was then split into X_train, X_test, y_train and y_test using train_test_split where 90% of data was used in training and 10% in testing. Now for the selection of the model, the dataset has more than two labels and the interpretability of the dataset does matter. Also, overfitting has to be avoided, therefore Random Forest Classifier goes best to achieve the result keeping in hindsight the requirements associated with the dataset.

6.7.1 Random Forest Classifier

The model is trained with a 100 decision trees and the evaluation metric is chosen to be accuracy of the model which comes out to be 98.16%. The code snippet for result can be seen below:



```
from sklearn.metrics import accuracy_score
pred = clf.predict(features_test)
acc = accuracy_score(pred, labels_test)
print(format(round(acc,4)*100),"%")
```

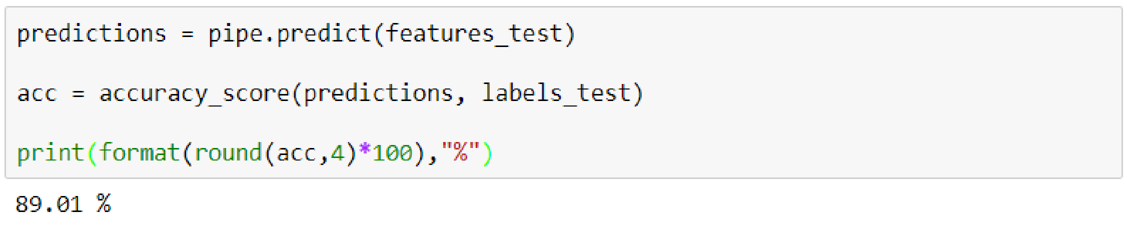
```
98.16 %
```

Figure 92. Random Forest - prediction accuracy

This result when compared to the approach taken by the winning entry of the KDD99Cup is better because in the latter k-NN classifier was used but that too on a bivariate model, but here the prediction is being done on a multivariate model.

6.7.2 Logistic Regression using Principle Component Analysis

A pipeline is created which first uses principle component analysis for dimensionality reduction and then logistic regression as the classifier. For PCA when the number of components is set to ‘mle’ and svd_solver is set to ‘full’, PCA will automatically try to detect the components that are needed to perform prediction. The code snippet and the result can be seen below:



```
predictions = pipe.predict(features_test)

acc = accuracy_score(predictions, labels_test)

print(format(round(acc,4)*100),"%")
```

```
89.01 %
```

Figure 93. Logistic regression - prediction accuracy

6.8 Conclusion

The primary aim which was to outperform the winning entry while doing classification on a multivariate model is achieved using the method implemented. Also, the paper discussed in the Background section of this chapter says that Random Forest Classifier has already been implemented for this dataset but the model there used for training was again a bivariate model. The second approach focuses on using logistic regression with PCA and the results achieved are not too bad because the data being used to train the model is not entire data because PCA has reduced the dimensionality of the dataset.

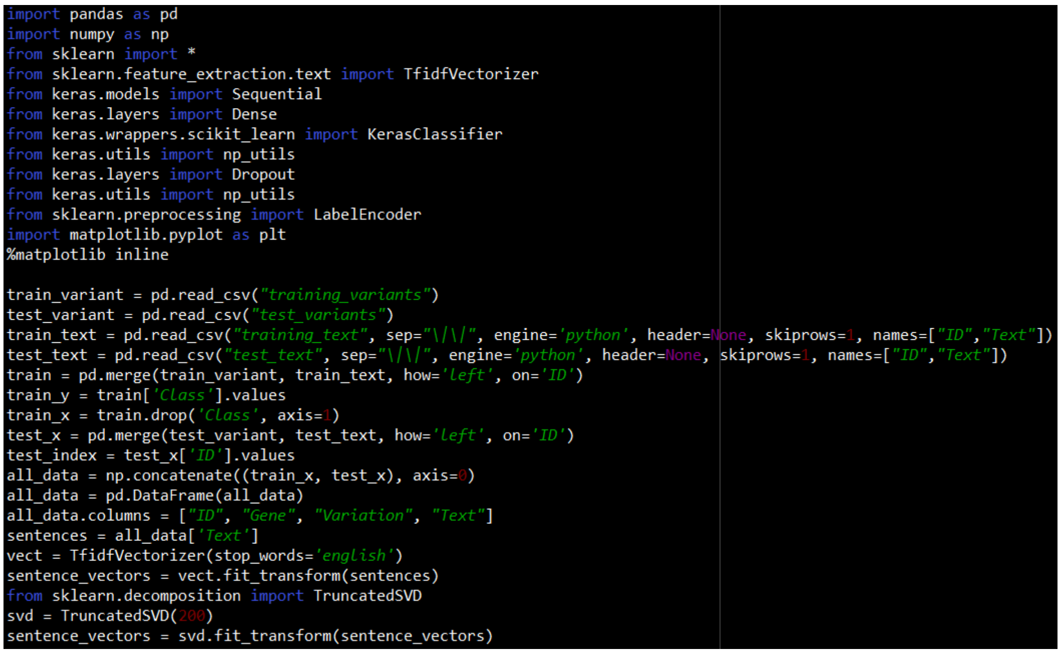
The only issue here lies in the dataset. In 2009 a paper [102] was published in IEEE symposium which had done a detailed analysis of the KDD CUP 99 dataset. Their study mentions that there are some problems with this dataset :

- Redundant Records – many records for some type of attacks are repeated
- Level of difficulty – evaluating results on the basis of accuracy, detection rate and false positive rate is not an appropriate option.

But still, the dataset is used because it is one of the most widely available datasets for creating an intrusion detection system.

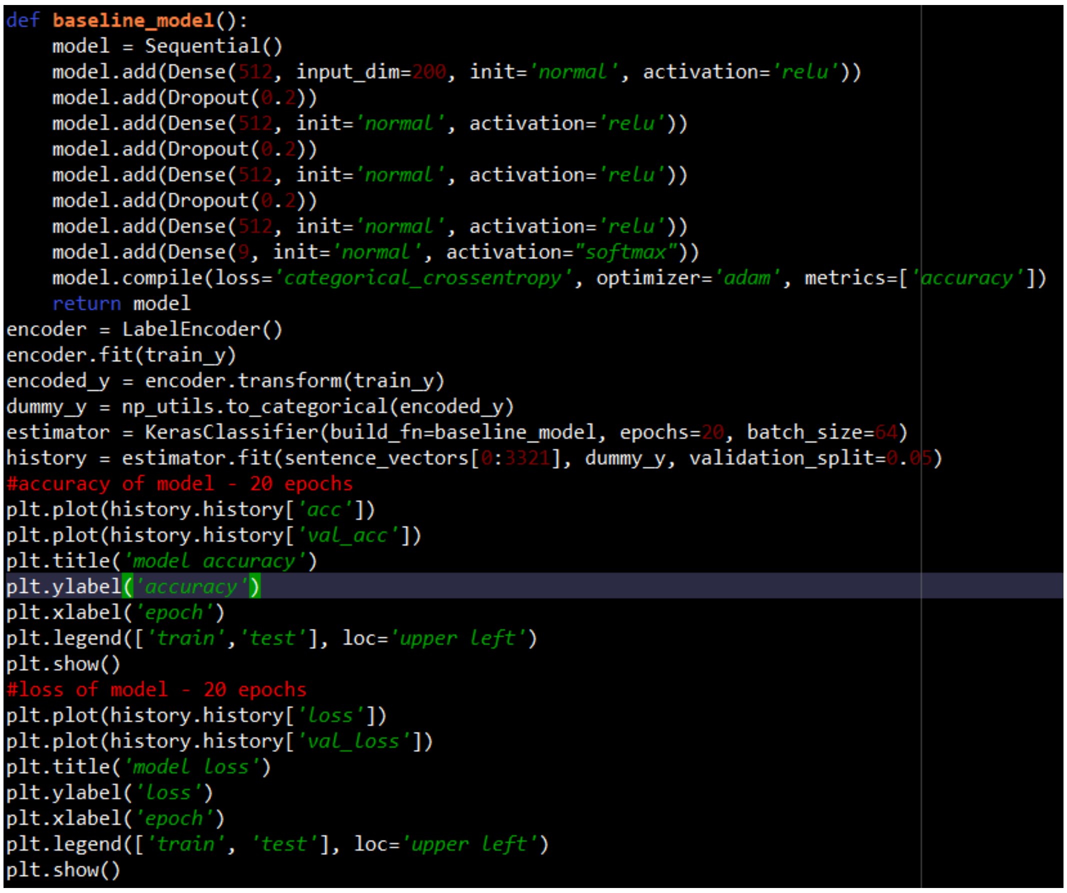
Appendix 1

Keras



```
import pandas as pd
import numpy as np
from sklearn import *
from sklearn.feature_extraction.text import TfidfVectorizer
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasClassifier
from keras.utils import np_utils
from keras.layers import Dropout
from keras.utils import np_utils
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt
%matplotlib inline

train_variant = pd.read_csv("training_variants")
test_variant = pd.read_csv("test_variants")
train_text = pd.read_csv("training_text", sep="\|\|", engine='python', header=None, skiprows=1, names=["ID","Text"])
test_text = pd.read_csv("test_text", sep="\|\|", engine='python', header=None, skiprows=1, names=["ID","Text"])
train = pd.merge(train_variant, train_text, how='left', on='ID')
train_y = train['Class'].values
train_x = train.drop('Class', axis=1)
test_x = pd.merge(test_variant, test_text, how='left', on='ID')
test_index = test_x['ID'].values
all_data = np.concatenate((train_x, test_x), axis=0)
all_data = pd.DataFrame(all_data)
all_data.columns = ["ID", "Gene", "Variation", "Text"]
sentences = all_data['Text']
vect = TfidfVectorizer(stop_words='english')
sentence_vectors = vect.fit_transform(sentences)
from sklearn.decomposition import TruncatedSVD
svd = TruncatedSVD(200)
sentence_vectors = svd.fit_transform(sentence_vectors)
```

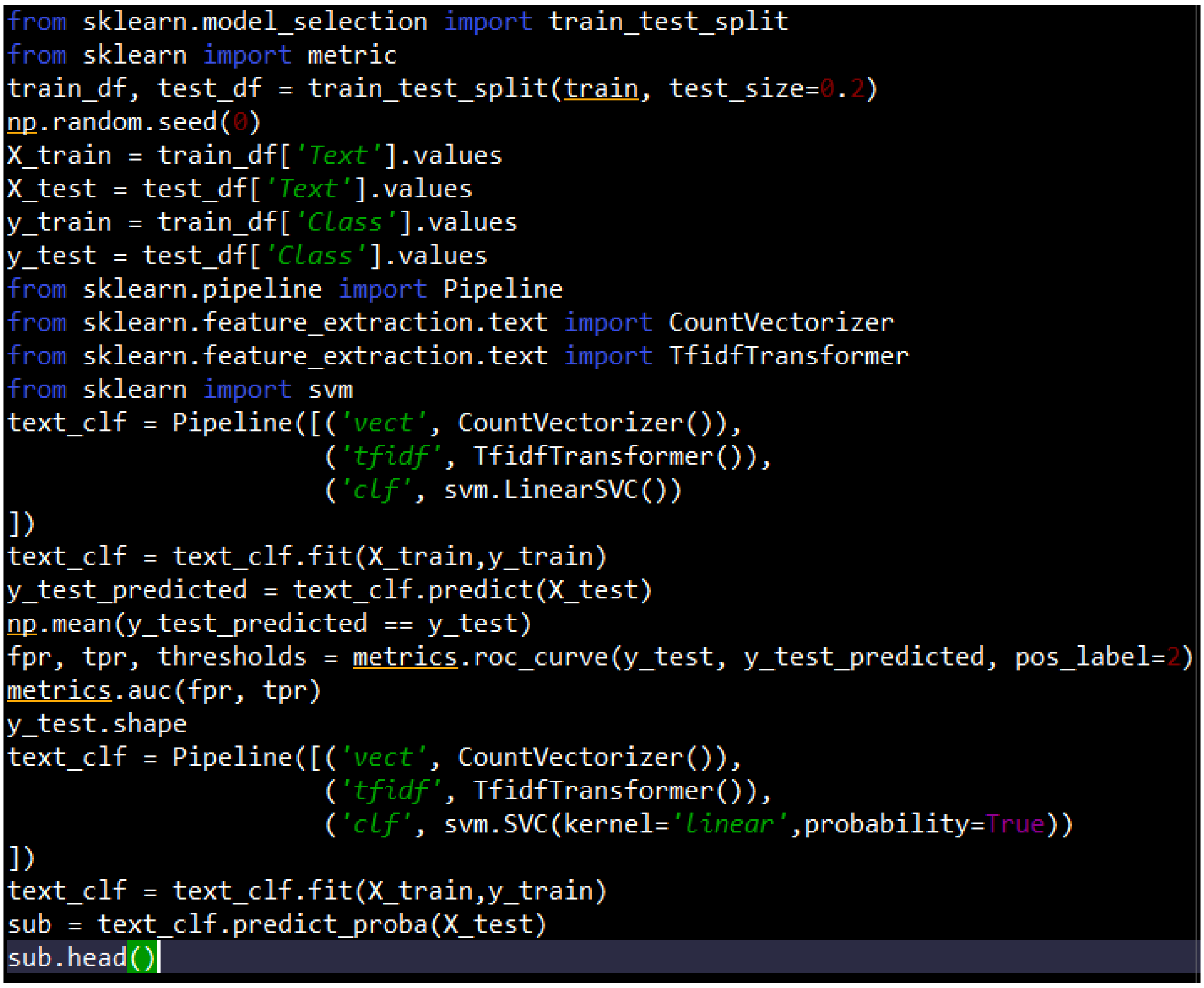
Figure 94. Keras - Part 1



```
def baseline_model():
    model = Sequential()
    model.add(Dense(512, input_dim=200, init='normal', activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(512, init='normal', activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(512, init='normal', activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(512, init='normal', activation='relu'))
    model.add(Dense(9, init='normal', activation="softmax"))
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
encoder = LabelEncoder()
encoder.fit(train_y)
encoded_y = encoder.transform(train_y)
dummy_y = np_utils.to_categorical(encoded_y)
estimator = KerasClassifier(build_fn=baseline_model, epochs=20, batch_size=64)
history = estimator.fit(sentence_vectors[0:3321], dummy_y, validation_split=0.05)
#accuracy of model - 20 epochs
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train','test'], loc='upper left')
plt.show()
#loss of model - 20 epochs
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

Figure 95. Keras - Part 2

SVM

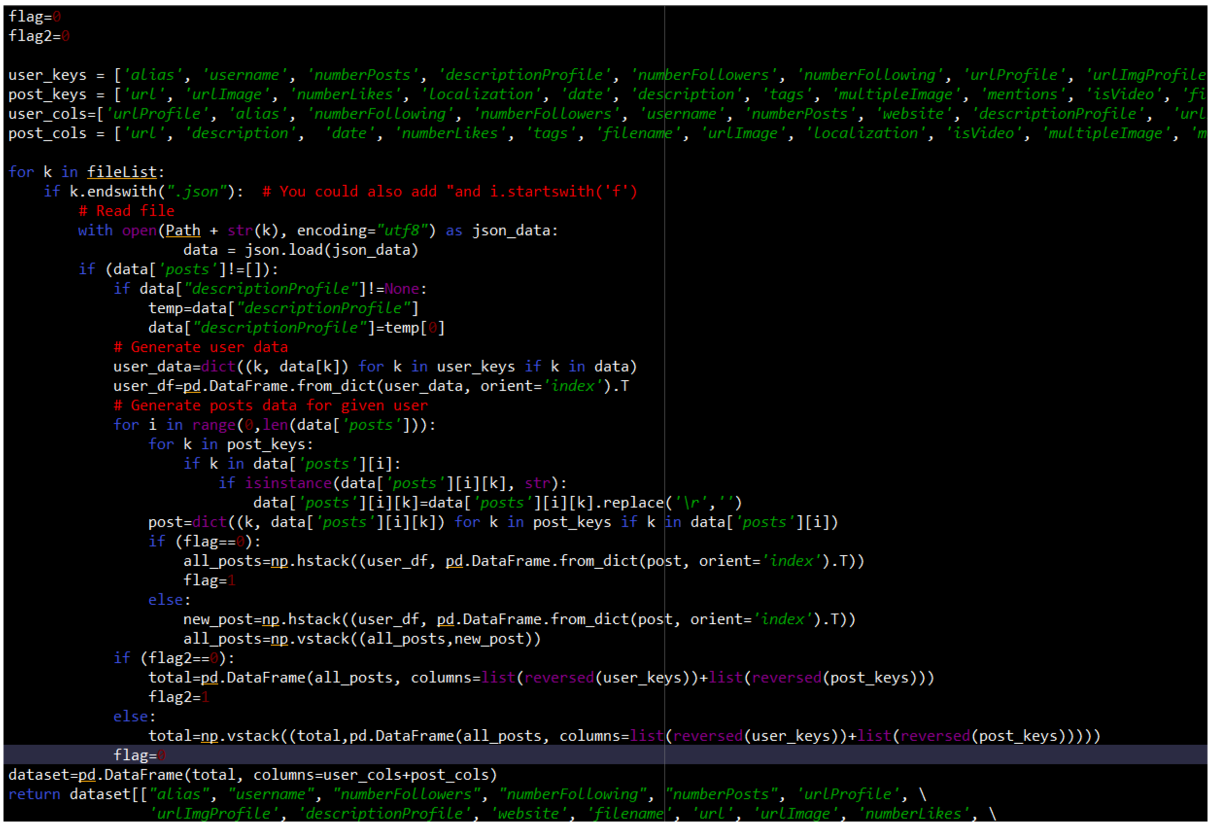


```
from sklearn.model_selection import train_test_split
from sklearn import metric
train_df, test_df = train_test_split(train, test_size=0.2)
np.random.seed(0)
X_train = train_df['Text'].values
X_test = test_df['Text'].values
y_train = train_df['Class'].values
y_test = test_df['Class'].values
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn import svm
text_clf = Pipeline([('vect', CountVectorizer()),
                     ('tfidf', TfidfTransformer()),
                     ('clf', svm.LinearSVC())
])
text_clf = text_clf.fit(X_train,y_train)
y_test_predicted = text_clf.predict(X_test)
np.mean(y_test_predicted == y_test)
fpr, tpr, thresholds = metrics.roc_curve(y_test, y_test_predicted, pos_label=2)
metrics.auc(fpr, tpr)
y_test.shape
text_clf = Pipeline([('vect', CountVectorizer()),
                     ('tfidf', TfidfTransformer()),
                     ('clf', svm.SVC(kernel='linear',probability=True))
])
text_clf = text_clf.fit(X_train,y_train)
sub = text_clf.predict_proba(X_test)
sub.head()
```

Figure 96. linear Support Vector Machine

Appendix 2

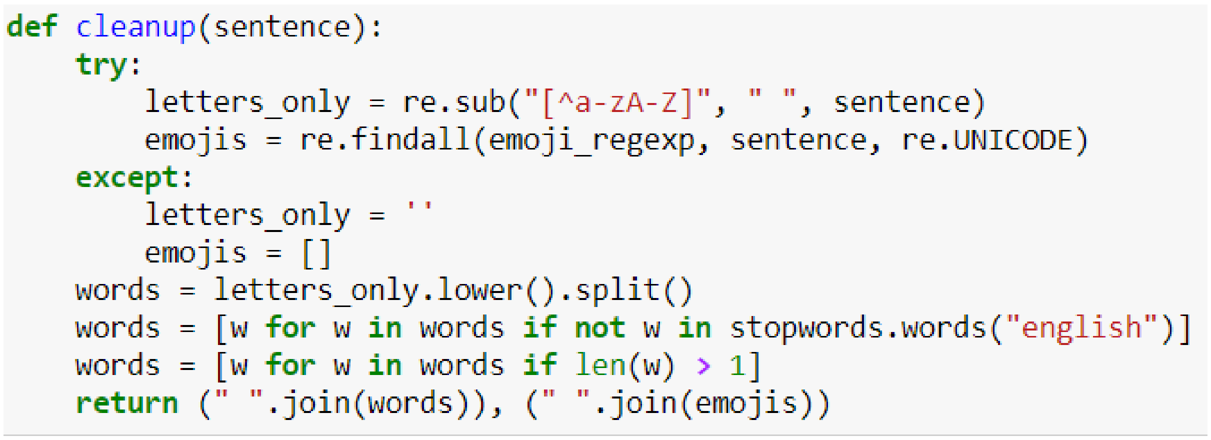
Part A



```
flag=0
flag2=0

user_keys = ['alias', 'username', 'numberPosts', 'descriptionProfile', 'numberFollowers', 'numberFollowing', 'urlProfile', 'urlImgProfile
post_keys = ['url', 'urlImage', 'numberLikes', 'localization', 'date', 'description', 'tags', 'multipleImage', 'mentions', 'isVideo', 'fi
user_cols=['urlProfile', 'alias', 'numberFollowing', 'numberFollowers', 'username', 'numberPosts', 'website', 'descriptionProfile',  'url
post_cols = ['url', 'description',  'date', 'numberLikes', 'tags', 'filename', 'urlImage', 'localization', 'isVideo', 'multipleImage', 'm

for k in fileList:
    if k.endswith(".json"):  # You could also add "and i.startswith('f')
        # Read file
        with open(Path + str(k), encoding="utf8") as json_data:
                    data = json.load(json_data)
        if (data['posts']!=[]):
            if data["descriptionProfile"]!=None:
                temp=data["descriptionProfile"]
                data["descriptionProfile"]=temp[0]
            # Generate user data
            user_data=dict((k, data[k]) for k in user_keys if k in data)
            user_df=pd.DataFrame.from_dict(user_data, orient='index').T
            # Generate posts data for given user
            for i in range(0,len(data['posts'])):
                for k in post_keys:
                    if k in data['posts'][i]:
                        if isinstance(data['posts'][i][k], str):
                            data['posts'][i][k]=data['posts'][i][k].replace('\r','')
                post=dict((k, data['posts'][i][k]) for k in post_keys if k in data['posts'][i])
                if (flag==0):
                    all_posts=np.hstack((user_df, pd.DataFrame.from_dict(post, orient='index').T))
                    flag=1
                else:
                    new_post=np.hstack((user_df, pd.DataFrame.from_dict(post, orient='index').T))
                    all_posts=np.vstack((all_posts,new_post))
            if (flag2==0):
                total=pd.DataFrame(all_posts, columns=list(reversed(user_keys))+list(reversed(post_keys)))
                flag2=1
            else:
                total=np.vstack((total,pd.DataFrame(all_posts, columns=list(reversed(user_keys))+list(reversed(post_keys)))))
            flag=0
dataset=pd.DataFrame(total, columns=user_cols+post_cols)
return dataset[["alias", "username", "numberFollowers", "numberFollowing", "numberPosts", 'urlProfile', \
                'urlImgProfile', 'descriptionProfile', 'website', 'filename', 'url', 'urlImage', 'numberLikes', \
```

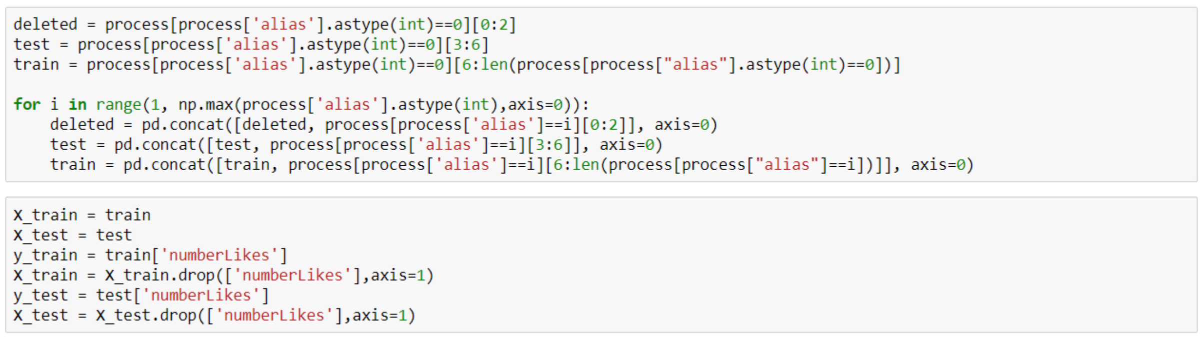
Figure 97. JSON to CSV

Part B



```
def cleanup(sentence):
    try:
        letters_only = re.sub("[^a-zA-Z]", " ", sentence)
        emojis = re.findall(emoji_regexp, sentence, re.UNICODE)
    except:
        letters_only = ''
        emojis = []
    words = letters_only.lower().split()
    words = [w for w in words if not w in stopwords.words("english")]
    words = [w for w in words if len(w) > 1]
    return (" ".join(words)), (" ".join(emojis))
```

Part C

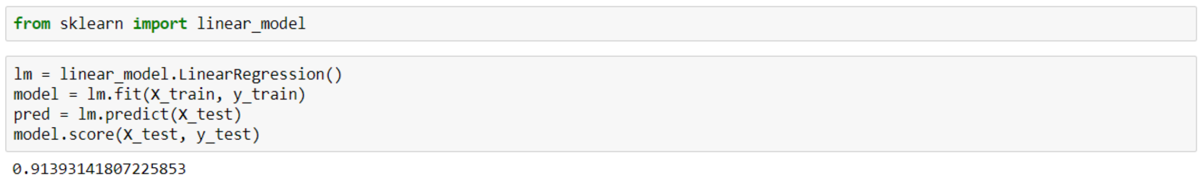


```
deleted = process[process['alias'].astype(int)==0][0:2]
test = process[process['alias'].astype(int)==0][3:6]
train = process[process['alias'].astype(int)==0][6:len(process[process["alias"].astype(int)==0])]

for i in range(1, np.max(process['alias'].astype(int),axis=0)):
    deleted = pd.concat([deleted, process[process['alias']==i][0:2]], axis=0)
    test = pd.concat([test, process[process['alias']==i][3:6]], axis=0)
    train = pd.concat([train, process[process['alias']==i][6:len(process[process["alias"]==i])]], axis=0)
```

```
X_train = train
X_test = test
y_train = train['numberLikes']
X_train = X_train.drop(['numberLikes'],axis=1)
y_test = test['numberLikes']
X_test = X_test.drop(['numberLikes'],axis=1)
```

Figure 98. Instagram data split into train and test

Part D



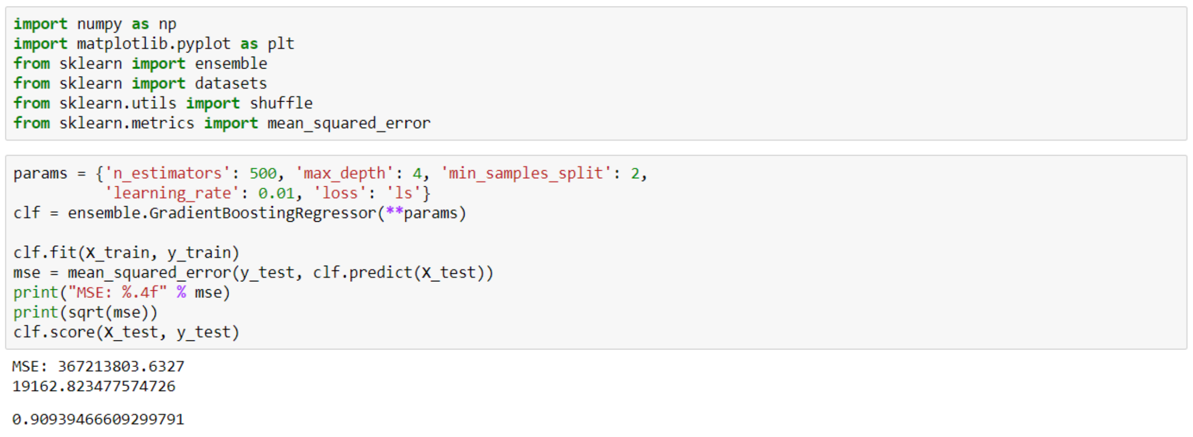
```
from sklearn import linear_model
```

```
lm = linear_model.LinearRegression()
model = lm.fit(X_train, y_train)
pred = lm.predict(X_test)
model.score(X_test, y_test)
```

```
0.91393141807225853
```

Figure 99. Linear regression

Part E



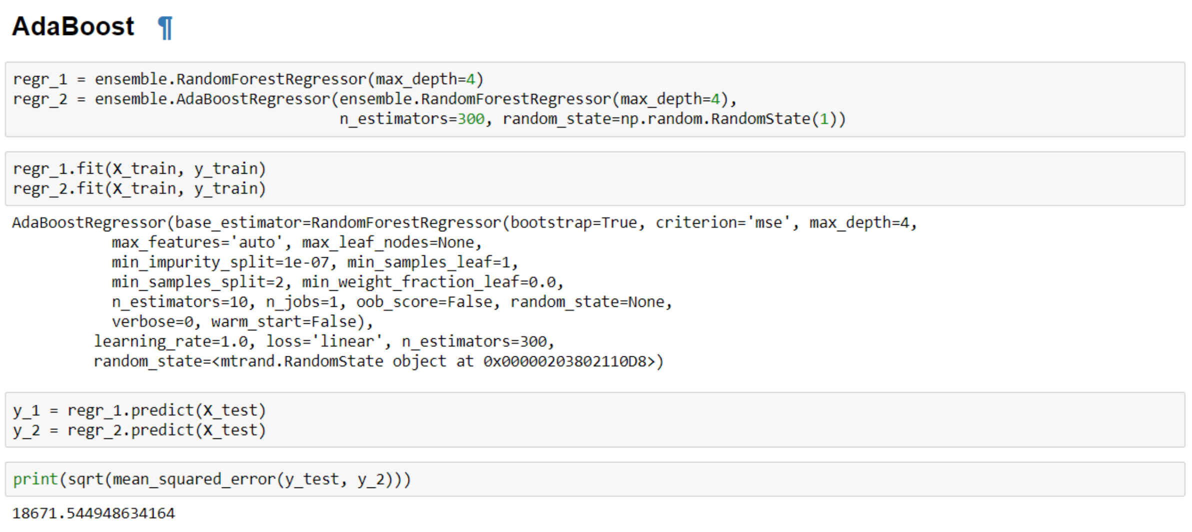
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import ensemble
from sklearn import datasets
from sklearn.utils import shuffle
from sklearn.metrics import mean_squared_error
```

```
params = {'n_estimators': 500, 'max_depth': 4, 'min_samples_split': 2,
          'learning_rate': 0.01, 'loss': 'ls'}
clf = ensemble.GradientBoostingRegressor(**params)

clf.fit(X_train, y_train)
mse = mean_squared_error(y_test, clf.predict(X_test))
print("MSE: %.4f" % mse)
print(sqrt(mse))
clf.score(X_test, y_test)
```

```
MSE: 367213803.6327
19162.823477574726

0.90939466609299791
```

Figure 100. Gradient Boosting

Part F



AdaBoost ¶

```
regr_1 = ensemble.RandomForestRegressor(max_depth=4)
regr_2 = ensemble.AdaBoostRegressor(ensemble.RandomForestRegressor(max_depth=4),
                                    n_estimators=300, random_state=np.random.RandomState(1))
```

```
regr_1.fit(X_train, y_train)
regr_2.fit(X_train, y_train)
```

```
AdaBoostRegressor(base_estimator=RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=4,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_split=1e-07, min_samples_leaf=1,
           min_samples_split=2, min_weight_fraction_leaf=0.0,
           n_estimators=10, n_jobs=1, oob_score=False, random_state=None,
           verbose=0, warm_start=False),
         learning_rate=1.0, loss='linear', n_estimators=300,
         random_state=<mtrand.RandomState object at 0x00000203802110D8>)
```

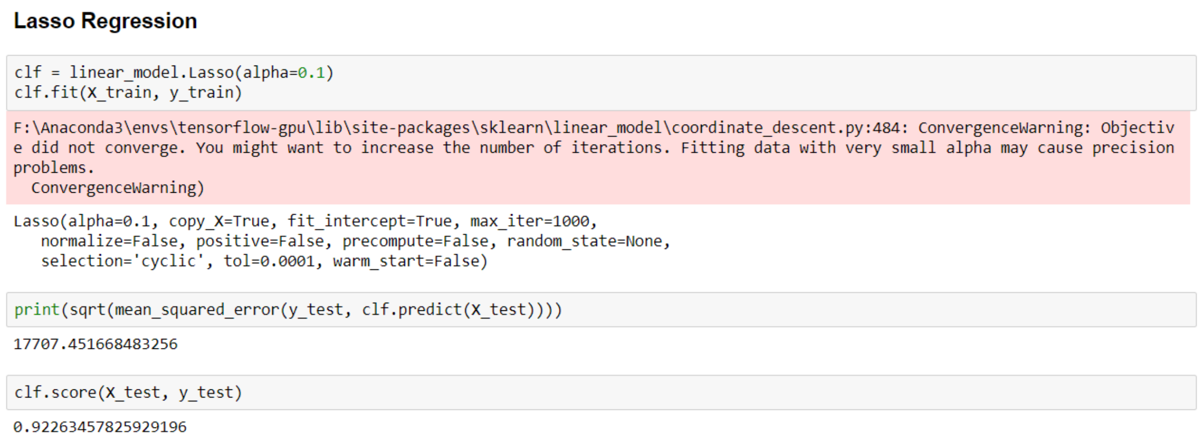
```
y_1 = regr_1.predict(X_test)
y_2 = regr_2.predict(X_test)
```

```
print(sqrt(mean_squared_error(y_test, y_2)))
```

```
18671.544948634164
```

Figure 101. AdaBoost regression

Lasso Regression



Lasso Regression

```
clf = linear_model.Lasso(alpha=0.1)
clf.fit(X_train, y_train)
```

```
F:\Anaconda3\envs\tensorflow-gpu\lib\site-packages\sklearn\linear_model\coordinate_descent.py:484: ConvergenceWarning: Objectiv
e did not converge. You might want to increase the number of iterations. Fitting data with very small alpha may cause precision
problems.
  ConvergenceWarning)
```

```
Lasso(alpha=0.1, copy_X=True, fit_intercept=True, max_iter=1000,
   normalize=False, positive=False, precompute=False, random_state=None,
   selection='cyclic', tol=0.0001, warm_start=False)
```

```
print(sqrt(mean_squared_error(y_test, clf.predict(X_test))))
```

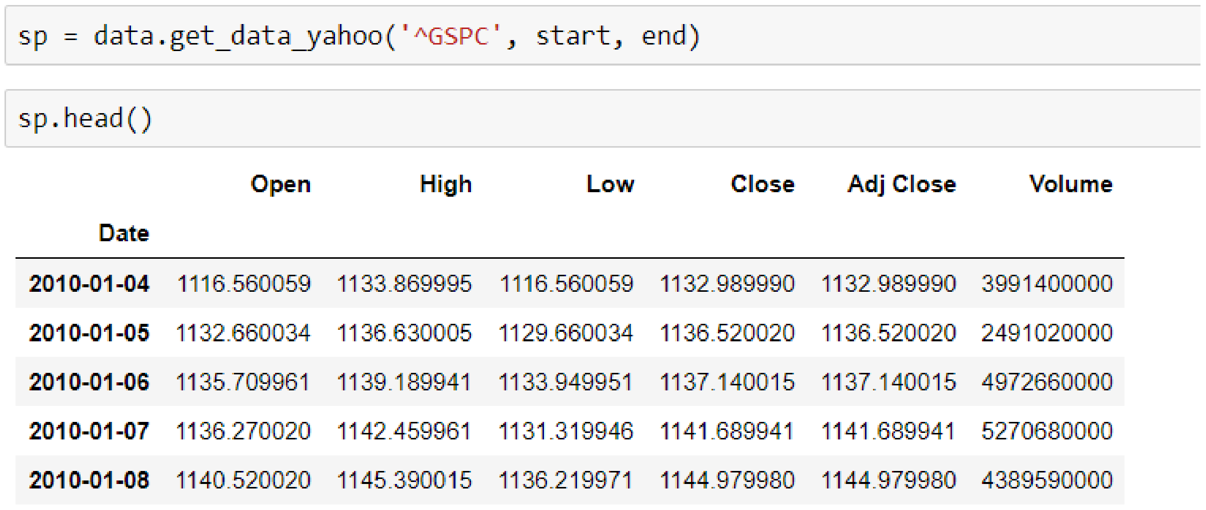
```
17707.451668483256
```

```
clf.score(X_test, y_test)
```

```
0.92263457825929196
```

Appendix 3

Part A

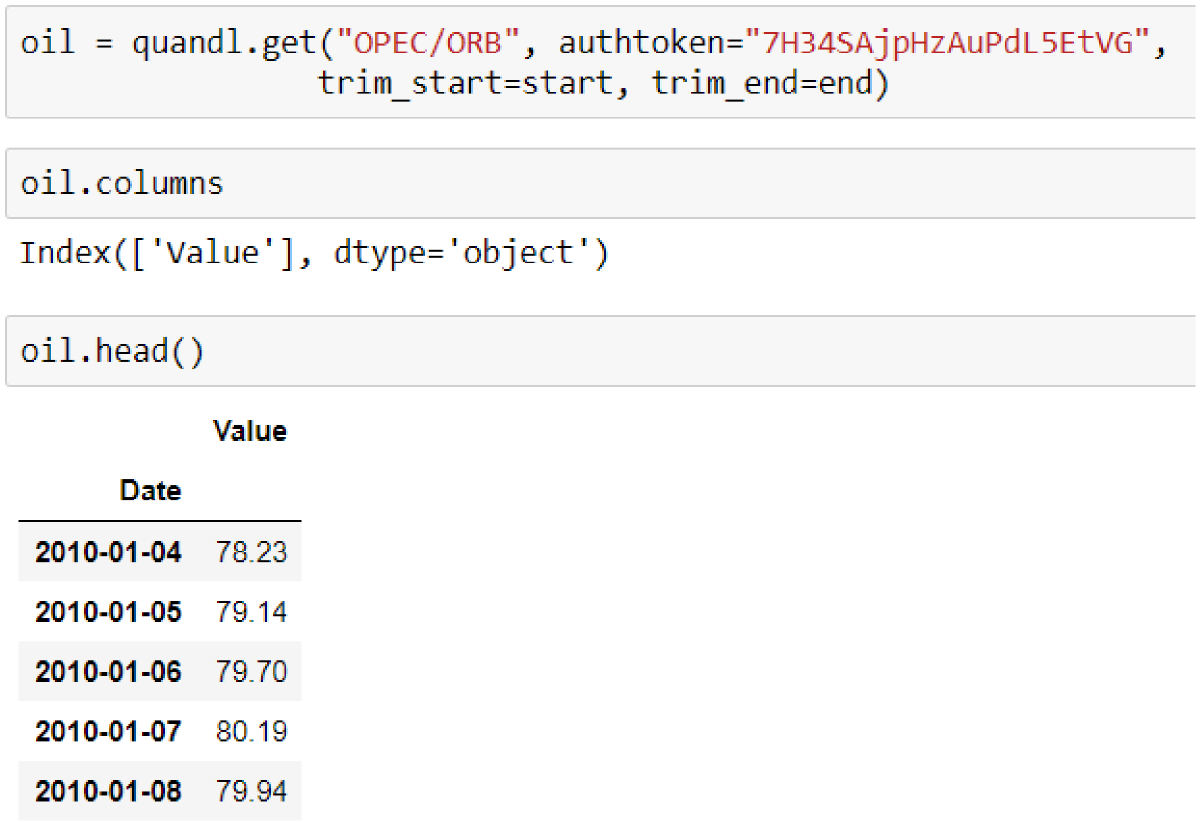


```
sp = data.get_data_yahoo('^GSPC', start, end)
```

```
sp.head()
```

Date	Open	High	Low	Close	Adj Close	Volume
2010-01-04	1116.560059	1133.869995	1116.560059	1132.989990	1132.989990	3991400000
2010-01-05	1132.660034	1136.630005	1129.660034	1136.520020	1136.520020	2491020000
2010-01-06	1135.709961	1139.189941	1133.949951	1137.140015	1137.140015	4972660000
2010-01-07	1136.270020	1142.459961	1131.319946	1141.689941	1141.689941	5270680000
2010-01-08	1140.520020	1145.390015	1136.219971	1144.979980	1144.979980	4389590000

Figure 102. S&P-500 prices



```
oil = quandl.get("OPEC/ORB", authtoken="7H34SAjpHzAuPdL5EtVG",
                    trim_start=start, trim_end=end)
```

```
oil.columns
```

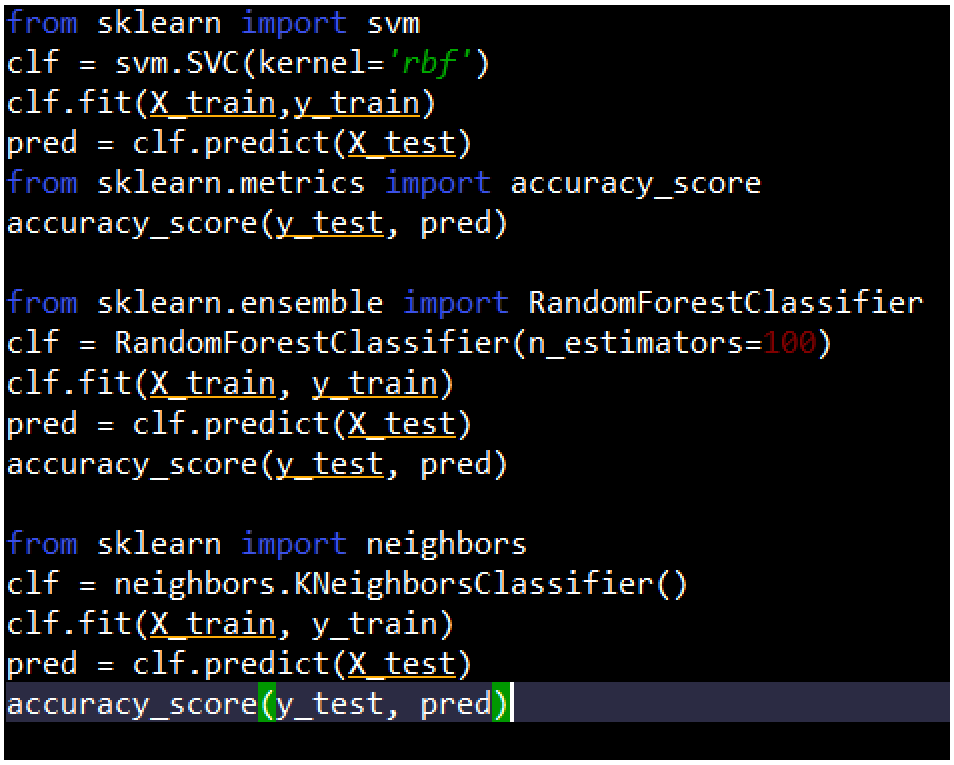
```
Index(['Value'], dtype='object')
```

```
oil.head()
```

Date	Value
2010-01-04	78.23
2010-01-05	79.14
2010-01-06	79.70
2010-01-07	80.19
2010-01-08	79.94

Figure 103. Oil prices

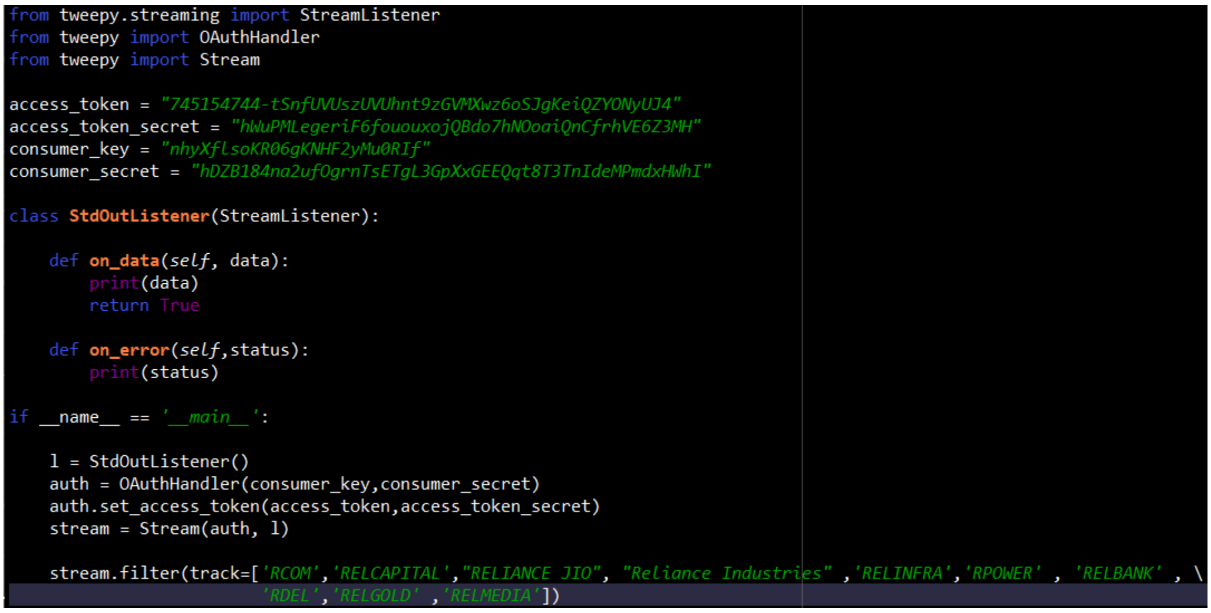
Part B



```
from sklearn import svm
clf = svm.SVC(kernel='rbf')
clf.fit(X_train,y_train)
pred = clf.predict(X_test)
from sklearn.metrics import accuracy_score
accuracy_score(y_test, pred)

from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(n_estimators=100)
clf.fit(X_train, y_train)
pred = clf.predict(X_test)
accuracy_score(y_test, pred)

from sklearn import neighbors
clf = neighbors.KNeighborsClassifier()
clf.fit(X_train, y_train)
pred = clf.predict(X_test)
accuracy_score(y_test, pred)
```

Figure 104. prediction models

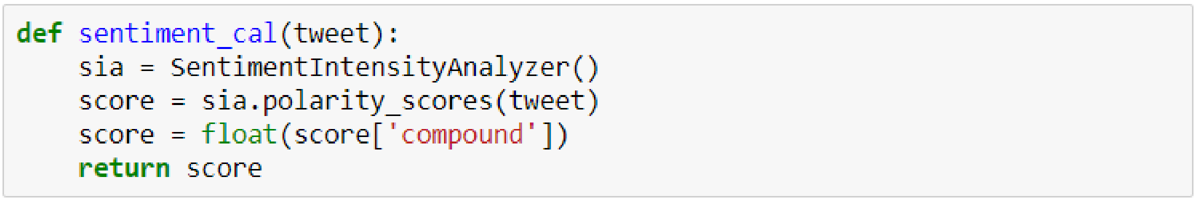
Part C



```
from tweepy.streaming import StreamListener
from tweepy import OAuthHandler
from tweepy import Stream

access_token = "745154744-tSnfUVUszUVUhnt9zGVMXwz6oSJgKeiQZYONyUJ4"
access_token_secret = "hWuPMLegeriF6fououxojQBdo7hNOoaiQnCfrhVE6Z3MH"
consumer_key = "nhyXflsoKR06gKNHF2yMu0RIf"
consumer_secret = "hDZB184na2ufOgrnTsETgL3GpXxGEEQqt8T3TnIdeMPmdxHWhI"

class StdOutListener(StreamListener):

    def on_data(self, data):
        print(data)
        return True

    def on_error(self,status):
        print(status)

if __name__ == '__main__':

    l = StdOutListener()
    auth = OAuthHandler(consumer_key,consumer_secret)
    auth.set_access_token(access_token,access_token_secret)
    stream = Stream(auth, l)

    stream.filter(track=['RCOM','RELCAPITAL',"RELIANCE JIO", "Reliance Industries" ,'RELINFRA','RPOWER' , 'RELBANK' , \
                         'RDEL','RELGOLD' ,'RELMEDIA'])
```

Figure 105. tweets scrapping

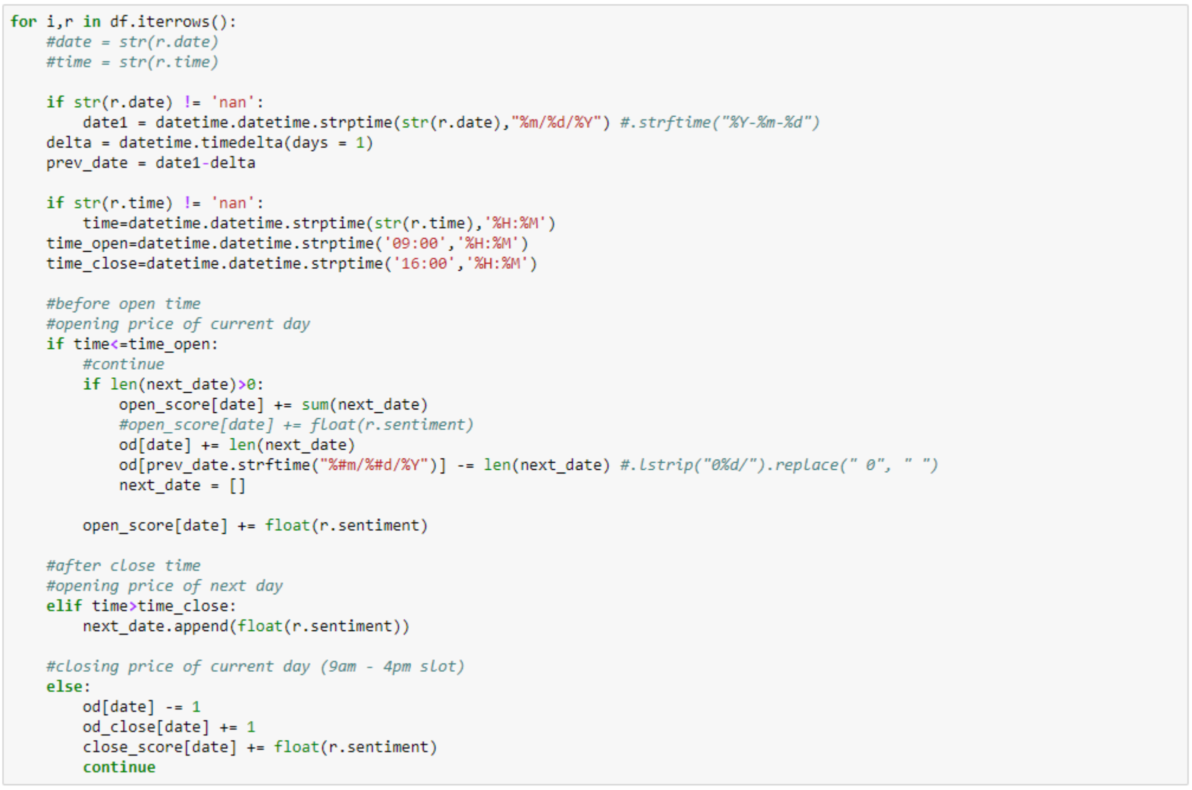
Part D



```python
def sentiment_cal(tweet):
    sia = SentimentIntensityAnalyzer()
    score = sia.polarity_scores(tweet)
    score = float(score['compound'])
    return score
```

Figure 106. Vader Sentiment

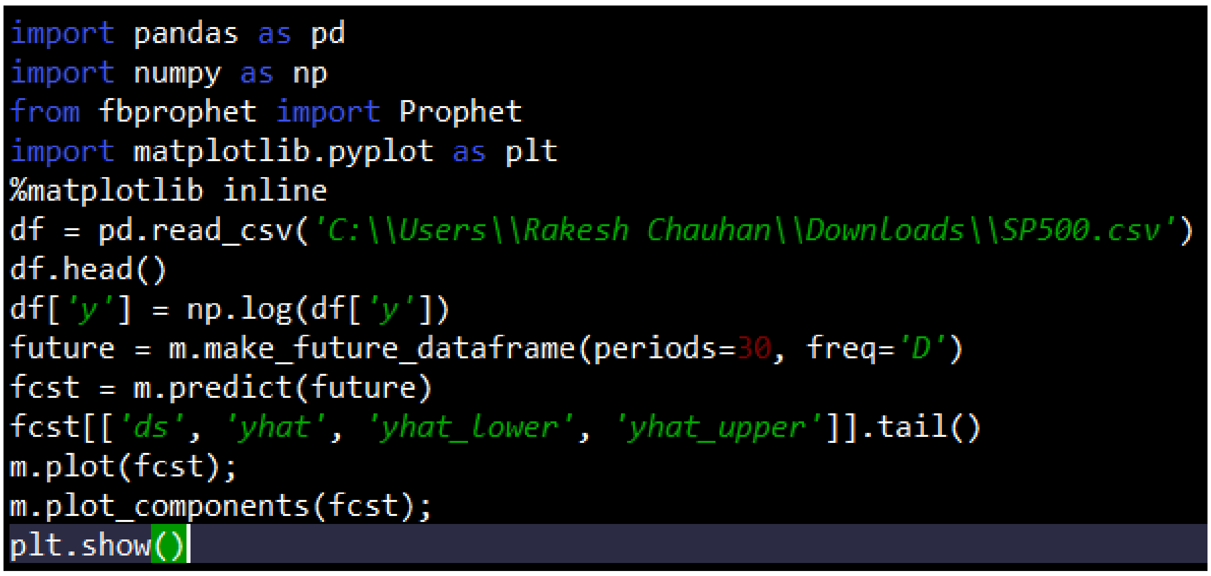
Part E



```python
for i,r in df.iterrows():
    #date = str(r.date)
    #time = str(r.time)

    if str(r.date) != 'nan':
        date1 = datetime.datetime.strptime(str(r.date),"%m/%d/%Y") #.strftime("%Y-%m-%d")
    delta = datetime.timedelta(days = 1)
    prev_date = date1-delta

    if str(r.time) != 'nan':
        time=datetime.datetime.strptime(str(r.time),'%H:%M')
    time_open=datetime.datetime.strptime('09:00','%H:%M')
    time_close=datetime.datetime.strptime('16:00','%H:%M')

    #before open time
    #opening price of current day
    if time<=time_open:
        #continue
        if len(next_date)>0:
            open_score[date] += sum(next_date)
            #open_score[date] += float(r.sentiment)
            od[date] += len(next_date)
            od[prev_date.strftime("%#m/%#d/%Y")] -= len(next_date) #.lstrip("0%d/").replace(" 0", " ")
            next_date = []

        open_score[date] += float(r.sentiment)

    #after close time
    #opening price of next day
    elif time>time_close:
        next_date.append(float(r.sentiment))

    #closing price of current day (9am - 4pm slot)
    else:
        od[date] -= 1
        od_close[date] += 1
        close_score[date] += float(r.sentiment)
        continue
```

Figure 107. open and close sentiment score calculation

Part F



```
import pandas as pd
import numpy as np
from fbprophet import Prophet
import matplotlib.pyplot as plt
%matplotlib inline
df = pd.read_csv('C:\\Users\\Rakesh Chauhan\\Downloads\\SP500.csv')
df.head()
df['y'] = np.log(df['y'])
future = m.make_future_dataframe(periods=30, freq='D')
fcst = m.predict(future)
fcst[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].tail()
m.plot(fcst);
m.plot_components(fcst);
plt.show()
```

Figure 108. Prophet implementation

Bibliography

[1] F. Meade, “Using Robust NLP and Machine Learning *,” 1997.

[2] http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

[3] http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html#sklearn.feature_extraction.text.CountVectorizer

[4] P. Domingos, “A few useful things to know about machine learning,” *Commun. ACM*, vol. 55, no. 10, p. 78, 2012.

[5] https://stats.stackexchange.com/questions/169156/explain-curse-of-dimensionality-to-a-child

[6] R. Blagus *et al.*, “SMOTE for high-dimensional class-imbalanced data,” *BMC Bioinformatics*, vol. 14, no. 1, p. 106, 2013.

[7] K. Pearson, “LIII. *On lines and planes of closest fit to systems of points in space*,” *Philos. Mag. Ser. 6*, vol. 2, no. 11, pp. 559–572, 1901.

[8] P. C. Hansen, “The truncated SVD as a method for regularization,” *Bit*, vol. 27, no. 4, pp. 534–553, 1987.

[9] P. Buhlmann and T. Hothorn, “Boosting algorithms: Regularization, predicitng and model fitting,” *Stat. Sci.*, vol. 22, no. 4, pp. 477–505, 2007.

[10] https://www.analyticsvidhya.com/blog/2015/11/quick-introduction-boosting-algorithms-machine-learning/

[11] A. Mayr, H. Binder, O. Gefeller, and M. Schmid, “The evolution of boosting algorithms: From machine learning to statistical modelling,” *Methods Inf. Med.*, vol. 53, no. 6, pp. 419–427, 2014.

[12] http://www.kdnuggets.com/2017/04/simple-understand-gradient-descent-algorithm.html

[13] Jerome H. Friedman, “Greedy Function Approximation: A Gradient Boosting Machine”, The Annals of Statistics, Vol. 29, No. 5(Oct 2001), pp 1189-1232

[14] https://www.pydoc.io/pypi/lightgbm-2.0.3/autoapi/plotting/index.html

[15] https://en.wikipedia.org/wiki/Loss_functions_for_classification

[16] https://github.com/Microsoft/LightGBM/issues/798

[17] http://scikit-learn.org/stable/modules/svm.html

[18] http://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html#sklearn.svm.LinearSVC

[19] https://keras.io/

[20] https://www.tensorflow.org/api_docs/python/

[21] Antonello Pasini, "Artificial neural networks for small dataset analysis", J Thorac Dis, 2015 May: 7(5): 953-960

[22] Q. Liu, Y. Wang, J. Li, Y. Jia, and Y. Ren, "Predicting User Likes in Online Media Based on Conceptualized Social Network Profiles," in *Web Technologies and Applications: APWeb 2014 Workshops, SNA, NIS, and IoTS, Changsha, China, September 5, 2014. Proceedings*, W. Han, Z. Huang, C. Hu, H. Zhang, and L. Guo, Eds. Cham: Springer International Publishing, 2014, pp. 82–92.

[23] Chen, K., Song, G.H., Sun, S.T.: Collaborative personalized tweet recommendation. In: SIGIR 2012, pp. 661–670. ACM (2012)

[24] https://applymagicsauce.com/documentation.html

[25] https://influence.iconosquare.com/

[26] https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.html

[27] K. H. Zou, K. Tuncali, and S. G. Silverman, "Correlation and simple linear regression," *Radiology*, vol. 227, no. 3, pp. 617–622, 2003.

[28] J. Elith, J. R. Leathwick, and T. Hastie, "A working guide to boosted regression trees," *J. Anim. Ecol.*, vol. 77, no. 4, pp. 802–813, 2008

[29] Witten, I. H., Frank, E., & Hall, M. A. (2013). Data mining: practical machine learning tools and techniques (3rd ed., 006.3'12-dc22). Burlington, MA: Morgan Kaufmann , Elsevier, pg. 362.

[30] R. E. Schapire, “A Short Introduction to Boosting,” *Society*, vol. 14, no. 5, pp. 771–780, 2009.

[31] Y. Freund, and R. Schapire, “A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting”, 1997.

[32] H. Drucker. “Improving Regressors using Boosting Techniques”, 1997.

[33] R. Tibshirani, “Regression Selection and Shrinkage via the Lasso,” *Journal of the Royal Statistical Society B*, vol. 58, no. 1. pp. 267–288, 1996.

[34] https://en.wikipedia.org/wiki/Stock_market#Early_history

[35] Tracy, James D. (1985). A Financial Revolution in the Habsburg Netherlands: Renten and Renteniers in the County of Holland, 1515–1565. (University of California Press, 300 pp)

[36] Goetzmann, William N.; Rouwenhorst, K. Geert (2005). The Origins of Value: The Financial Innovations that Created Modern Capital Markets. (Oxford University Press, ISBN 978-0195175714))

[37] Goetzmann, William N.; Rouwenhorst, K. Geert (2008). The History of Financial Innovation, in Carbon Finance, Environmental Market Solutions to Climate Change. (Yale School of Forestry and Environmental Studies, chapter 1, pp. 18–43

[38] Sylla, Richard (2015). "Financial Development, Corporations, and Inequality". (BHC-EBHA Meeting).

[39] "World Federation of Exchanges Monthly YTD Data". World-exchanges.org. Retrieved 2011-05-31

[40] "Equity market > Size relative to bond markets and bank assets". eurocapitalmarkets.org. Retrieved August 14, 2015.

[41] Mahipal Singh, 2011, ISBN 9788182055193, April 2011

[42] Shiller, Robert (2005). Irrational Exuberance (2d ed.). Princeton University Press. ISBN 0-691-12335-7.

[43] David Fabian (February 9, 2014). "Why The Market Doesn't Care Where You Think It Should Go". Seeking Alpha. Retrieved August 14, 2015.

[44] Philip Ball (April 26, 2013). "Counting Google searches predicts market movements". Nature.

Retrieved August 28, 2013

[45] Nick Bilton (April 26, 2013). "Google Search Terms Can Predict Stock Market, Study Finds". New York Times. Retrieved August 28, 2013.

[46] Jason Palmer (April 25, 2013). "Google searches predict market moves". BBC. Retrieved August 28, 2013.

[47] Helen Susannah Moat, Chester Curme, Adam Avakian, Dror Y. Kenett, H. Eugene Stanley and Tobias Preis (2013). "Quantifying Wikipedia Usage Patterns Before Stock Market Moves". Scientific Reports. 3: 1801. doi:10.1038/srep01801.

[48] "Wikipedia's crystal ball". Financial Times. May 10, 2013. Retrieved August 10, 2013.

[49] Khadjeh Nassirtoussi, Arman; Aghabozorgi, Saeed; Ying Wah, Teh; Ngo, David Chek Ling (2014-11-15). "Text mining for market prediction: A systematic review". Expert Systems with Applications. 41 (16): 7653–7670. doi:10.1016/j.eswa.2014.06.009.

[50] Bollen, Johan; Huina, Mao; Zeng, Xiao-Jun. "Twitter mood predicts the stock market". Cornell University. October 14, 2010. Retrieved November 7, 2013.

[51] sellthenews (14 April 2012). http://sellthenews.tumblr.com/post/21067996377/noitdoesnot. Retrieved August 28 2017.

[52] C. Kearney and V. Potì, "Correlation Dynamics in European Equity Markets Correlation Dynamics in European Equity Markets," *Res. Int. Bus. Financ.*, vol. 20, no. 3, pp. 305–321, 2006.

[53] https://en.wikipedia.org/wiki/Reliance_Industries. Retrieved August 28 2017.

[54] "Standard & Poor's 500 Index – S&P 500". Investopedia. Retrieved August 11, 2017.

[55] "NASDAQ Composite". Yahoo! Finance. Retrieved August 11, 2017.

[56] "Dow Jones Industrial Average". Quandl. Retrieved August 11, 2017.

[57] "Treasury Yield 5 Years". Yahoo Finance. Retrieved August 11, 2017.

[58] "HANG SENG INDEX". Yahoo Finance. Retrieved August 11, 2017.

[59] "DAX". Yahoo Finance. Retrieved August 11, 2017.

[60] "CAC 40". Yahoo Finance. Retrieved August 11, 2017.

[61] "Nikkei 225". Yahoo Finance. Retrieved August 11, 2017.

[62] "FTSE 100". Yahoo Finance. Retrieved August 11, 2017.

[63] "OPEC/ORB". Quandl. Retrieved August 11, 2017.

[64] "BUNDESBANK/BBK01_WT5511". Quandl. Retrieved August 11, 2017.

[65] "ECB/EURUSD". Quandl. Retrieved August 11, 2017.

[66] "CUR/JPY". Quandl. Retrieved August 11, 2017.

[67] "CUR/AUD". Quandl. Retrieved August 11, 2017.

[68] "https://en.wikipedia.org/wiki/Yahoo!_Finance". Wikipedia. Retrieved August 11, 2017.

[69] "https://www.quandl.com/about". Quandl. Retrieved August 11, 2017.

[70] "https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.interpolate.html#pandas.DataFrame.interpolate". Pydata. Retrieved August 12 2017.

[71] "https://pandas.pydata.org/pandas-docs/stable/missing_data.html". Pydata. Retrieved August 12 2017.

[72] C. A. O'Reilly, "Organizational Ambidexterity: Past, Present and Future," *Acad. Manag. Perspect.*, vol. 27, no. 4, pp. 324–338, 2013.

[73] "http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html". Scikit-learn. Retrieved August 12 2017.

[74] "http://ogrisel.github.io/scikit-learn.org/sklearn-tutorial/modules/generated/sklearn.neighbors.KNeighborsClassifier.html". Scikit-learn. Retrieved August 12 2017.

[75] "https://en.wikipedia.org/wiki/Reliance_Industries". Wikipedia. Retrieved August 13 2017.

[76] "https://en.wikipedia.org/wiki/National_Stock_Exchange_of_India". Wikipedia. Retrieved August 13 2017.

[77] “https://twython.readthedocs.io/en/latest/”. Twython documentation. Retrieved August 13 2017.

[78] “https://dev.twitter.com/docs”. Twitter. Retrieved August 13 2017.

[79] “https://twittercommunity.com/t/how-to-get-my-api-key/7033”. Twitter community. Retrieved August 13 2017.

[80] “https://github.com/cjhutto/vaderSentiment”. Github. Retrieved August 14 2017.

[81] C. J. Hutto and E. Gilbert, “Vader: A parsimonious rule-based model for sentiment analysis of social media text,” *Eighth Int. AAAI Conf. Weblogs …*, pp. 216–225, 2014.

[82] “http://www.statisticshowto.com/what-is-a-coefficient-of-determination/”. R squared. Retrieved on August 14 2017.

[83] S. J. Taylor and B. Letham, “Forecasting at Scale,” pp. 1–17, 2017.

[84] “http://www.investopedia.com/articles/active-trading/101014/basics-algorithmic-trading-concepts-and-examples.asp”. Investopedia. Retrieved August 14 2017.

[85] “http://www.investopedia.com/terms/h/high-frequency-trading.asp”. Investopedia. Retrieved August 14 2017.

[86] M. Conti, R. Poovendran and M. Secchiero, "FakeBook: Detecting Fake Profiles in On-Line Social Networks," 2012 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining, Istanbul, 2012, pp. 1071-1078.doi: 10.1109/ASONAM.2012.185

[87] R. N. Reddy and N. Kumar, “Automatic Detection of Fake Profiles in Online Social Networks,” 2012.

[88] B. Viswanath, A. Post, K. P. Gummadi, and A. Mislove, “An analysis of social network-based Sybil defenses,” *Proc. ACM SIGCOMM 2010 Conf. SIGCOMM - SIGCOMM ’10*, p. 363, 2010

[89] “https://pypi.python.org/pypi/SexMachine/”. PyPi. Retrieved June 30 2017.

[90] “https://autohotkey.com/board/topic/20260-gender-verification-by-forename-cmd-line-tool-db/”. Gender Verification. Retrieved June 30 2017.

[91] T. Hastie, R. Tibshirani and J. Friedman, “Elements of Statistical Learning Ed. 2”, p592-593, Springer, 2009.

[92] Devijver, Pierre A.; Kittler, Josef (1982). Pattern Recognition: A Statistical Approach. London, GB: Prentice-Hall.

[93] “https://stats.stackexchange.com/questions/105501/understanding-roc-curve”. Stack Exchange. Retrieved June 30 2017.

[94] “http://mlwiki.org/index.php/ROC_Analysis”. ML Wiki. Retrieved June 30 2017.

[95] Zhang, Jiong & Zulkernine, Mohammad & Haque, A. (2008). Random-Forests-Based Network Intrusion Detection Systems. Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on. 38. 649 - 659. 10.1109/TSMCC.2008.923876.

[96] S. Mabu, C. Chen, N. Lu, K. Shimada and K. Hirasawa, "An Intrusion-Detection Model Based on Fuzzy Class-Association-Rule Mining Using Genetic Network Programming," in IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), vol. 41, no. 1, pp. 130-139, Jan. 2011.doi: 10.1109/TSMCC.2010.2050685

[97] E. W. Tavares Ferreira, G. Arantes Carrijo, R. de Oliveira and N. Virgilio de Souza Araujo, "Intrusion Detection System with Wavelet and Neural Artifical Network Approach for Networks Computers," in IEEE Latin America Transactions, vol. 9, no. 5, pp. 832-837, Sept. 2011.doi: 10.1109/TLA.2011.6030997

[98] C. Dartigue, H. I. Jang, and W. Zeng, “A new data-mining based approach for network intrusion detection,” *Proc. 7th Annu. Commun. Networks Serv. Res. Conf. CNSR 2009*, pp. 372–377, 2009.

[99] “http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html”. KDD Cup. Retrieved June 20 2017.

[100] Stephen D. Bay and Dennis F. Kibler and Michael J. Pazzani and Padhraic Smyth. The UCI KDD Archive of Large Data Sets for Data Mining Research and Experimentation. SIGKDD Explorations, 2. 2000.

[101] Cost-based Modeling and Evaluation for Data Mining With Application to Fraud and Intrusion Detection: Results from the JAM Project by Salvatore J. Stolfo, Wei Fan, Wenke Lee, Andreas Prodromidis, and Philip K. Chan.

[102] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, “A detailed analysis of the KDD CUP 99 data set,” *IEEE Symp. Comput. Intell. Secur. Def. Appl. CISDA 2009*, no. Cisda, pp. 1–6, 2009.

[103] Guen VJ, Gamble C, Flajolet M, et al. CDK10/cyclin M is a protein kinase that controls ETS2 degradation and is deficient in STAR syndrome. Proceedings of the National Academy of Sciences of the United States of America. 2013;110(48):19525-19530. doi:10.1073/pnas.1306814110.

[104] S. Unger *et al.*, “Mutations in the cyclin family member FAM58A cause an X-linked dominant disorder characterized by syndactyly, telecanthus and anogenital and renal malformations.,” *Nat. Genet.*, vol. 40, no. 3, pp. 287–9, 2008.

[105] Olivier M, Hollstein M, Hainaut P. TP53 Mutations in Human Cancers: Origins, Consequences, and Clinical Use. Cold Spring Harbor Perspectives in Biology. 2010;2(1):a001008. doi:10.1101/cshperspect.a001008.

[106] M. Keniry and R. Parsons, “The role of PTEN signaling perturbations in cancer and in targeted therapy,” *Oncogene*, vol. 27, no. 41, pp. 5477–5485.

[107] L. F. Cranor, “A Framework for Reasoning About the Human in the Loop,” *Proc. 1st Conf. Usability, Psychol. Secur.*, p. 1:1--1:15, 2008.

[108] “http://cse512-15s.github.io/fp-marcotcr-bdol/”. Human in the loop ML. Retrieved September 5 2017.

[109] “https://docs.wso2.com/display/ML100/How+to+Select+an+Algorithm+in+WSO2+ML”. Algorithm selection. Retrieved September 5 2017.