National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Anchal Ora
Student ID: x18135846

School of Computing
National College of Ireland

Supervisor: Dr. Catherine Mulwa

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Anchal Ora
Student ID:	x18135846
Programme:	Data Analytics
Year:	2019
Module:	MSc Research Project
Supervisor:	Dr. Catherine Mulwa
Submission Due Date:	12/12/2019
Project Title:	Configuration Manual
Word Count:	
Page Count:	23

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	28th January 2020

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Anchal Ora
x18135846

1 Introduction

The configuration manual demonstrates the implementation stages of the research **"Spam Detection in Short Message Service Using Natural Language Processing and Machine Learning Techniques"**. The motive of the research is to develop a model that detects spam SMS efficiently and in minimal time. To build the model, the combination of techniques like Natural Language Processing for feature extraction like Bag of Words, TF-IDF along with machine learning algorithms such as XGBoost, LightGBM, Bernoulli Naïve Bayes, Support Vector Machine, and Random Forest were implemented. This manual explains the hardware and the software project specifications to implement the project in Chapter 2. Chapter 3 explains the Data Preparation followed by Chapter 4 which describes the implementation steps in detail and the output generated.

2 System Specification

The project hardware and software requirements are the key points to be considered before implementing any project. For the project, the requirements are mentioned as follows:

2.0.1 Hardware Configuration

Implementation of the project is carried out on the laptop having the hardware configuration as depicted in Figure 1 :

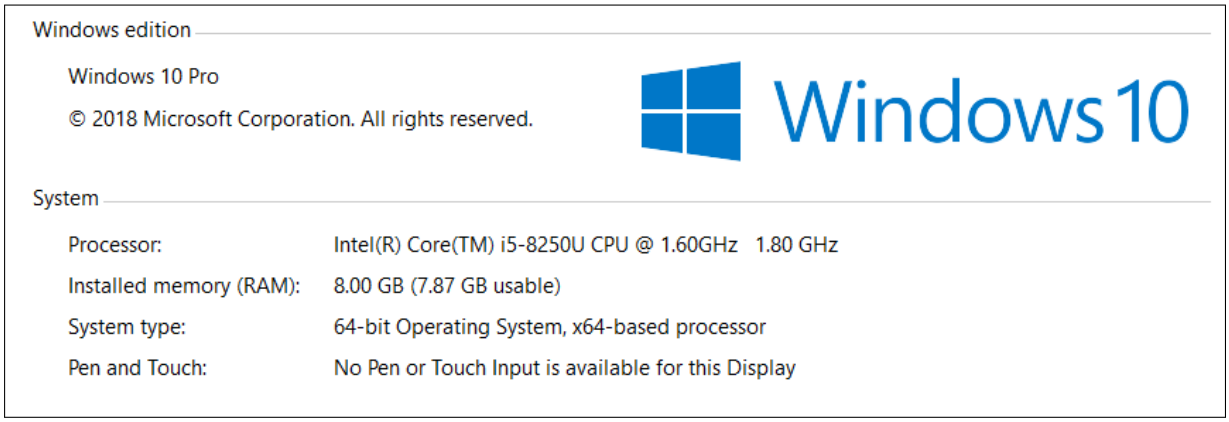

Figure 1: Hardware Configuration

2.0.2 Software Configuration

1. **Google Colab**: The implementation of the project was done in Python using Google Colab Notebook (similar to Jupyter Notebook) which is hosted on the cloud and can be accessed from the browser [1]. For the project, no external installation or downloads were required as all the libraries are pre-installed in the notebook. To code in Google Colab, the following steps were undertaken:
a. Open the Link:
`https://colab.research.google.com/notebooks/welcome.ipynb#recent=true`
b. The below window appears (refer Figure 2), click on "New Python 3 Notebook":

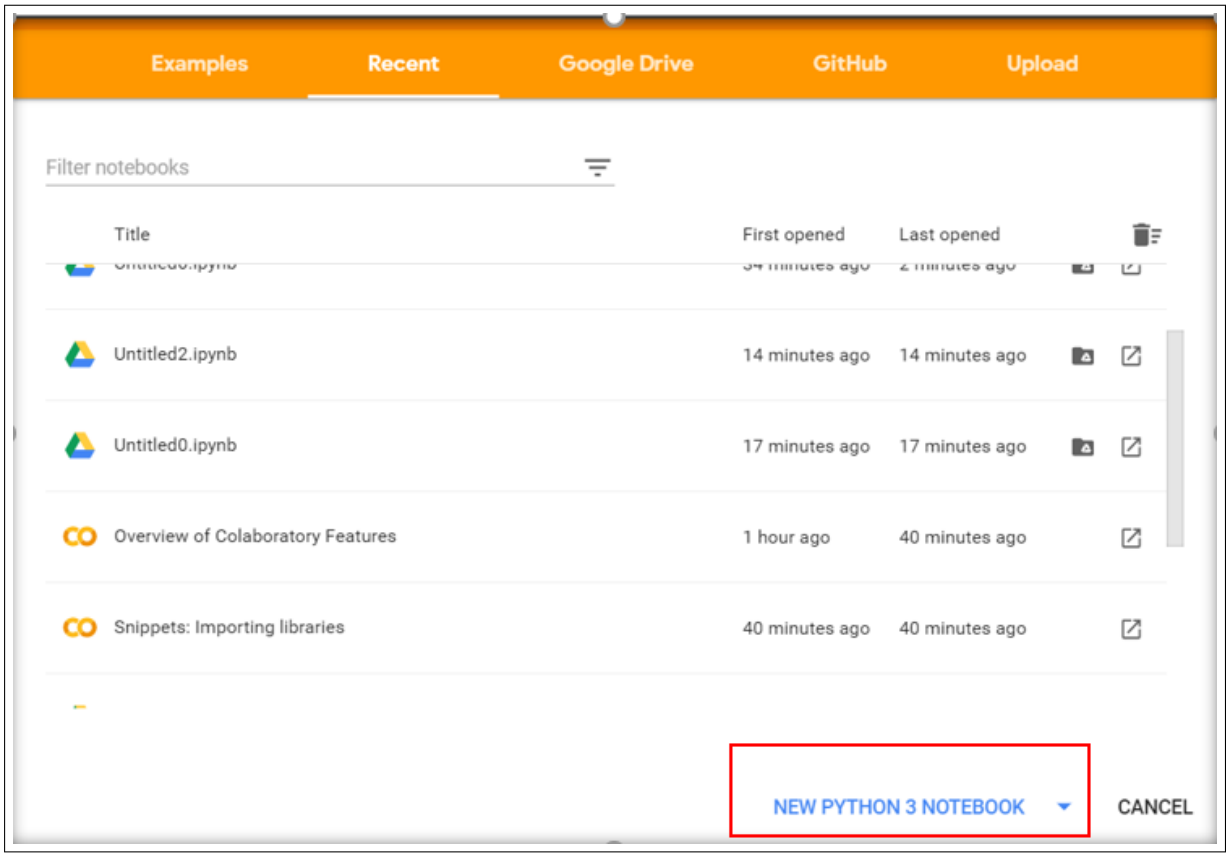

Figure 2: Colab Notebook

c. This opens the code editor where the code can be written and can be saved directly Google Drive.

2. **Microsoft Power BI**: Power BI tool is used for visualization of results.

3 Data Preparation

3.1 Data Loading

The data was fetched from the Kaggle website which was already present in .csv format and was downloaded from the below link:

[1] https://colab.research.google.com/notebooks/welcome.ipynb#scrollTo=ipS1ETT7sF9w

https://www.kaggle.com/uciml/sms-spam-collection-dataset
The dataset was first uploaded in Colab from the local machine using the code shown in the Figure 3 and then the initial level of data was prepared so that exploratory analysis of the data can be carried out.

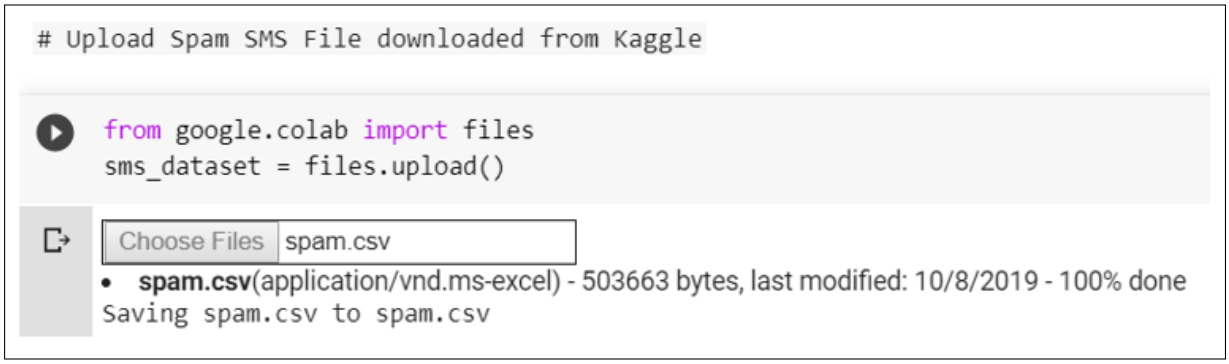

Figure 3: Uploading SMS Data in Colab

After the data was uploaded, the required libraries for exploratory analysis and pre-processing were then imported (refer Figure 4).

Figure 4: Importing Libraries

The data in the form of csv was then loaded in Pandas Data Frame using the read_csv function. The unnecessary columns were then removed and the column labels were made appropriate. The following code (Figure 5) is implemented :

```
# Importing the CSV file

sms_dataset = pd.read_csv("spam.csv",delimiter=',',encoding='latin-1')

# Printing the top 5 rows

sms_dataset.head()
```

	v1	v2	Unnamed: 2	Unnamed: 3	Unnamed: 4
0	ham	Go until jurong point, crazy.. Available only ...	NaN	NaN	NaN
1	ham	Ok lar... Joking wif u oni...	NaN	NaN	NaN
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	NaN	NaN	NaN
3	ham	U dun say so early hor... U c already then say...	NaN	NaN	NaN
4	ham	Nah I don't think he goes to usf, he lives aro...	NaN	NaN	NaN

```
# Removing the unncessary columns

sms_dataset=sms_dataset.drop(['Unnamed: 2', 'Unnamed: 3', 'Unnamed: 4'],axis='columns',inplace=False)

#Changing the column name
sms_dataset.columns=['SMStype','Text']

#Print the data
sms_dataset.head()
```

	SMStype	Text
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...

Figure 5: Data Preparation

3.2 Exploratory Data Analysis

In the exploratory Data Analysis, missing values were checked. The text data distribution was observed to follow Zipf's distribution. The class imbalance was found and resolved using the down-sampling technique. The word cloud for ham and spam was generated, which shows the top words of each category. Then, the correlation between the length and SMS type was calculated which can indicate if the length was a good feature to consider or not.

a. **Missing Values Analysis** : As can be seen in the dataset there were no missing values present (in Figure 6).

```
# Checking the missing values

print(sms_dataset.isnull().sum())

SMStype    0
Text       0
dtype: int64
```

Figure 6: Data Preparation

b. **Analyzing Text Data Distribution** : The text data distribution was analyzed using a Numpy histogram and it can be seen as the data follows Zipf's distribution which is very commonly seen for the text data [2] (Figure 8). The graph was plotted between the Frequency and Rank of the words.

```
# Checking the Distribution of Text Data

from itertools import islice
from scipy import special
import re

#Retrieve SMS spam dataset
freq = {}
smsfile = open("spam.csv", 'r',encoding = "ISO-8859-1")
smsfile = smsfile.read()
terms = re.findall(r'(\b[A-Za-z][a-z]{2,9}\b)', smsfile)
#build dict of words based on frequency
for term in terms:
    count = freq.get(term,0)
    freq[term] = count + 1


#limit words to 1000
number = 1000
freq = {key:value for key,value in islice(freq.items(), 0, number)}

f = np.fromiter(freq.values(), dtype=float)


#Calculate zipf and plot the data
a = 2. #  distribution parameter
count, bins, ignored = plt.hist(f[f<50], 50, density=True)
x = np.arange(1., 50.)
y = x**(-a) / special.zetac(a)
plt.plot(x, y/max(y), linewidth=2, color='r')
plt.xlabel("Rank")
plt.ylabel("Frequency")
plt.show()
```

Figure 7: Code for analyzing the text Data Distribution

[2]https://towardsdatascience.com/another-twitter-sentiment-analysis-with-python-part-3-zipfs-law-data-visualisation-fc9eadda71e7

Output:

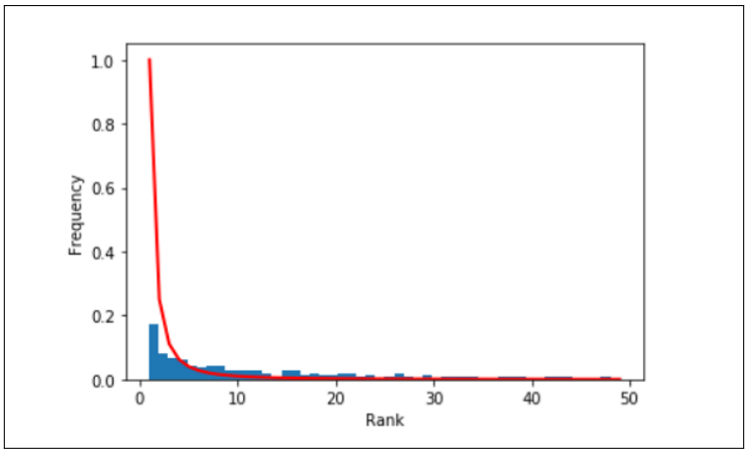

Figure 8: Zipf's Distribution of Text Data

c. **Examining the count of Spam and Ham Messages** : The count of Ham and Spam messages were examined and it was found that there was a huge class imbalance shown in Figure 9.

Figure 9: Before Data Sampling

To handle this problem, the down-sampling technique was applied. It was implemented using sample function from the Random module of python and the ham class was brought to the same level of the spam class(Figure 10).

```
# Downsampling Majority class (ham) to Minority class (spam)

# Class count
ham, spam = sms_dataset.SMStype.value_counts()

# Divide by class
class_ham = sms_dataset[sms_dataset['SMStype'] == 'ham']
class_spam = sms_dataset[sms_dataset['SMStype'] == 'spam']

class_ham_under= class_ham.sample(spam, random_state=5)
sms_dataset = pd.concat([class_ham_under, class_spam], axis=0)
sms_dataset.to_csv('spam_afterSampling.csv')
files.download('spam_afterSampling.csv')

print('Count after Random down-sampling:')
print(sms_dataset.SMStype.value_counts())

bar=sms_dataset.SMStype.value_counts().plot(kind='bar', title='Number of SMS based on Type',color= '#66CDAA');
bar.set_xlabel("Message Type")
bar.set_ylabel('Count of SMS')
```

```
Count after Random down-sampling:
spam    747
ham     747
Name: SMStype, dtype: int64
Text(0, 0.5, 'Count of SMS')
```

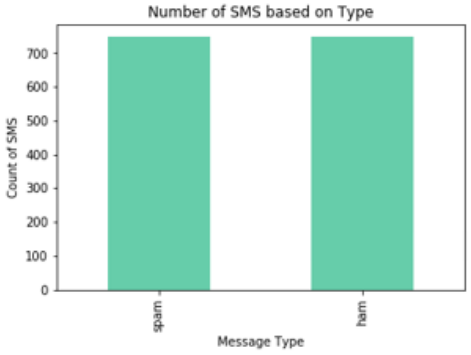

Figure 10: After Data Sampling

d. **Analyzing the most frequent word in Ham and Spam by WordCloud** : Generating the Word Cloud for Spam and Ham (shown in Figure 11) using WordCloud module which shows the most occurring words of the document. The argument passed for generating WordCloud for spam was spam Text and similarly for ham.

Figure 11: Word Cloud for Spam and Ham

e. **Encoding SMS Type, Adding Length feature and Finding the Correlation between Length and SMS Type**: The categorical variable that is the SMS type was encoded using LabelEncoder() function which assigns the value between 0 to one less than the number of categories. As suggested by the researcher (Agarwal et al. (2016)) that the length parameter can enhance the performance, hence the length has been calculated which takes the number of characters into account excluding space.

Then the correlation was calculated using corr() function and correlation was visualized using heatmap from seaborn package between the length feature and the SMS Type as shown in Figure 12.

```
# Label Encoding of Dependent Variable and adding length variable
```

```
labelencoder = LabelEncoder()
sms_dataset['SMStype']=labelencoder.fit_transform(sms_dataset['SMStype'])
sms_dataset['Text_Length'] = sms_dataset['Text'].apply(lambda x : len(x)-x.count(' '))
sms_dataset['Text_Length'].describe()
sms_dataset.head()
```

	SMStype	Text	Text_Length
5226	0	Prabha..i'm soryda..realy..frm heart i'm sory	41
3920	0	Do 1 thing! Change that sentence into: \Becaus...	96
2945	0	make that 3! 4 fucks sake?! x	23
1048	0	I walked an hour 2 c u! doesnåÕt that show I c...	61
2699	0	Oh baby of the house. How come you dont have a...	59

```
# Calculating the correlation between Length feature and SMS type
```

```
correlation= sms_dataset.corr()
print(correlation)

sns.heatmap(correlation,
            xticklabels=correlation.columns.values,
            yticklabels=correlation.columns.values)
```

```
               SMStype  Text_Length
SMStype       1.000000     0.601167
Text_Length   0.601167     1.000000
<matplotlib.axes._subplots.AxesSubplot at 0x7f6b6ea75710>
```

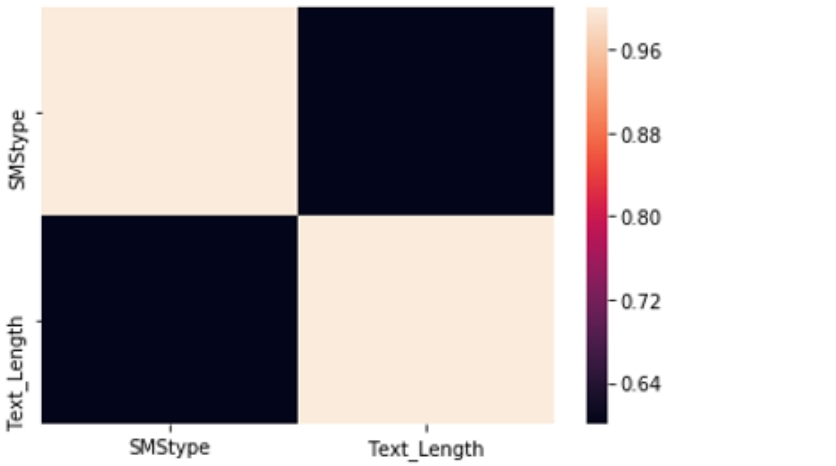

Figure 12: Correlation Matrix between Length and SMS Type

3.3 Data Pre-processing

The pre-processing steps include the removal of non-English characters using a regular expression. The words were then converted to token using word_tokenize from NLTK and

then stopwords were removed using the stopwords package. Lastly, lemmatization was applied using the WordNetLemmatizer package from NLTK and the data was normalized using the lower() function. The processed data was exported and verified. The output of the cleaned data is shown in Figure 13.

```
# Data Pre-processing
```

```
# Removing special characters and numbers
sms_dataset['Text'] = sms_dataset['Text'].astype('str')
sms_dataset['Text'] = sms_dataset['Text'].str.replace(r'[^\w\s]+', " ")
sms_dataset['Text'] = sms_dataset['Text'].str.replace('\d+', " ")

# Tokenize the words
sms_dataset['Text'] = sms_dataset.apply(lambda row: nltk.word_tokenize(row['Text']), axis=1)

# Remove stop words

stopWords = stopwords.words('english')
sms_dataset['Text']= list(map(lambda line: list(filter(lambda word: word not in stopWords, line)), sms_dataset['Text']))


# Lemmatization and lowering thecase

filtered_data = []
for item in sms_dataset['Text']:
    words = [WordNetLemmatizer().lemmatize(word).lower() for word in item]
    filtered_data.append(words)
sms_dataset['Text'] = filtered_data

sms_dataset.to_csv('Data_afterPreprocessing.csv')
files.download('Data_afterPreprocessing.csv')

sms_dataset.head()
```

	SMStype	Text	Text_Length
5226	0	[prabha, soryda, realy, frm, heart, sory]	41
3920	0	[thing, change, sentence, want, concentrate, e...	96
2945	0	[make, fuck, sake, x]	23
1048	0	[walked, hour, c, u, doesnâõt, show, care, won...	61
2699	0	[oh, baby, house, come, dont, new, picture, fa...	59

Figure 13: Pre-processing of Dataset

4 Implementation, Evaluation and Results

The implementation explains the Feature Extraction, Feature Selection, and application of Machine Learning models. There was a total of 6 combinations of feature matrix where 5 models have been applied. The features were extracted as Unigram, Bigram, and TF-IDF. The evaluation is done with and without length feature. Features were selected using Chi-Square. The kbest feature was selected, where k = 300 for all the matrix as it gave optimal results. After feature selection, machine learning models like XGBoost, LightGBM, Bernoulli Naive Bayes, SVM and Random Forest have been implemented and the models were evaluated on the accuracy, precision, recall, F1-Score and Execution Time. The suitable libraries were imported for implementation as shown in the Figure 14.

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from xgboost import XGBClassifier
from sklearn.naive_bayes import BernoulliNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from lightgbm import LGBMClassifier
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import f1_score, recall_score, precision_score, accuracy_score, confusion_matrix
from time import time
```

Figure 14: Imported Required Libraries

4.1 Unigram without Length Feature

For the Unigram score model, CountVectorizer from sklearn.feature_extraction was imported which was used to create a Unigram matrix where the ngram_range was passed as (1,1). After the matrix was created, the feature selection technique that is Chi-Square was implemented where the top 300 features were picked as it gave the optimal results. After this model like XGBoost, LightGBM, Bernoulli Naïve Bayes, SVM, and Random Forest were implemented with default parameters, except using the linear kernel for SVM as linear Kernel works well for sparse data.[3]
Code:

```
# Bag of Words--- unigram
def funcUnigram():
    unigram = CountVectorizer(tokenizer=lambda doc: doc, lowercase=False,ngram_range=(1,1))
    bag_of_words= unigram.fit_transform(sms_dataset['Text'])
    bow= bag_of_words.toarray()
#Chi Square :Feature Selection
    Y=sms_dataset['SMStype']
    chi2_selector = SelectKBest(chi2, k=300)
    X = chi2_selector.fit_transform(bow,Y)
# Trying to fit all models
    models = [
            ('bernoulli_nb', BernoulliNB(alpha=1.0, binarize=0.0, class_prior=None, fit_prior=True)),
            ('svm',SVC(gamma='auto',kernel='linear')),
            ('xgboost', XGBClassifier()),
            ('random_forest', RandomForestClassifier(n_estimators = 1000, random_state = 42)),
            ('lgb',LGBMClassifier()),
            ]
    scores = pd.DataFrame([], columns=['model', 'accuracy', 'precision', 'Recall','F1-Score'])

    for model in models:

        start_time = time()

        i = 1
        element00 = 0
        element01 = 0
        element10 = 0
        element11 = 0
        cv_accuracy_score =[]
        cv_precision_score =[]
        cv_recall_score =[]
        cv_f1_score =[]
```

[3]https://www.svm-tutorial.com/2014/10/svm-linear-kernel-good-text-classification/

```
        cv = StratifiedKFold(n_splits = 10, random_state = 5, shuffle=True)

        Ykf = Y.values

        for train_index, test_index in cv.split(X,Ykf):
            X_train, X_test = X[train_index], X[test_index]
            y_train, y_test = Ykf[train_index],Ykf[test_index]

            model[1].fit(X_train,y_train)
            y_pred = model[1].predict(X_test)

            end_time = time()
            execution_time = (end_time) - (start_time)

            cm = (confusion_matrix(y_test, y_pred)).tolist()
            #print('Model : ',model[0],'\nConfusion Matrix: \n',confusion_matrix(y_test, y_pred))

            element00 += cm[0][0]
            element01 += cm[0][1]
            element10 += cm[1][0]
            element11 += cm[1][1]

            cv_accuracy_score.append(accuracy_score(y_test, y_pred))
            cv_precision_score.append(precision_score(y_test, y_pred))
            cv_recall_score.append(recall_score(y_test, y_pred))
            cv_f1_score.append(f1_score(y_test, y_pred))

            i += 1

        cnf_mtx = np.matrix([[element00, element01], [element10, element11]])
        print('Model : ',model[0],'\nConfusion Matrix Sum: \n',cnf_mtx)
        print('cv: ',cv_accuracy_score,'\nMean cv score: ',np.mean(cv_accuracy_score))
        scores = scores.append({
                'model':model[0],
                'accuracy' : np.mean(cv_accuracy_score),
                'precision': np.mean(cv_precision_score),
                'Recall': np.mean(cv_recall_score),
                'F1-Score': np.mean(cv_f1_score),
                'run_time': execution_time
            }, ignore_index=True)
    print(scores)
funcUnigram()
```

Figure 15: Code for Unigram Model without Length

The train and test dataset were divided using Stratified 10-fold cross-validation and to evaluate the models, mean Accuracy, mean Precision, mean Recall, mean F1-score and Execution time was calculated. The confusion matrix was also plotted by taking the sum of all the 10 runs. The output generated was:

```
Model :  bernoulli_nb
Confusion Matrix Sum:
 [[746   1]
 [ 52 695]]
cv:  [0.9733333333333334, 0.9866666666666667, 0.96, 0.9533333333333334, 0.9666666666666667, 0.94, 0.96, 0.9662162162162162, 0.9594594594594594, 0.9797297297297297]
Mean cv score:  0.9645405405405405
Model :  svm
Confusion Matrix Sum:
 [[732  15]
 [ 41 706]]
cv:  [0.9666666666666667, 0.98, 0.96, 0.9466666666666667, 0.9666666666666667, 0.9533333333333334, 0.96, 0.9662162162162162, 0.9662162162162162, 0.9594594594594594]
Mean cv score:  0.9625225225225226
Model :  xgboost
Confusion Matrix Sum:
 [[719  28]
 [ 65 682]]
cv:  [0.9333333333333333, 0.9533333333333334, 0.9533333333333334, 0.94, 0.9133333333333333, 0.9533333333333334, 0.9333333333333333, 0.9459459459459459, 0.9324324324324325, 0.918918918918919]
Mean cv score:  0.9377297297297297
Model :  random_forest
Confusion Matrix Sum:
 [[722  25]
 [ 56 691]]
cv:  [0.9533333333333334, 0.9733333333333334, 0.9466666666666667, 0.96, 0.92, 0.94, 0.9333333333333333, 0.9459459459459459, 0.9459459459459459, 0.9391891891891891]
Mean cv score:  0.9457747747747748
Model :  lgb
Confusion Matrix Sum:
 [[717  30]
 [ 39 708]]
cv:  [0.9666666666666667, 0.9733333333333334, 0.9533333333333334, 0.9733333333333334, 0.9533333333333334, 0.9533333333333334, 0.9333333333333333, 0.9391891891891891, 0.9594594594594594, 0.932
Mean cv score:  0.9537747747747748
          model  accuracy  precision    Recall  F1-Score   run_time
0   bernoulli_nb  0.964541   0.998592  0.930414  0.963115   0.116155
1            svm  0.962523   0.979691  0.945171  0.961764   1.666668
2        xgboost  0.937730   0.961285  0.912937  0.936219   8.195967
3  random_forest  0.945775   0.965916  0.925045  0.944678  31.284493
4            lgb  0.953775   0.959925  0.947784  0.953439   1.155866
```

Figure 16: Output for Unigram Model without Length

4.2 Unigram with Length Feature

Performed the same step as above but including the length feature. Code:

```
# Bag of Words--- unigramwithlength
def funcUnigramWithLength():
    unigram = CountVectorizer(tokenizer=lambda doc: doc, lowercase=False,ngram_range=(1,1))
    bag_of_words= unigram.fit_transform(sms_dataset['Text'])
    bow= bag_of_words.toarray()
#Chi Square :Feature Selection
    Y=sms_dataset['SMStype']
    chi2_selector = SelectKBest(chi2, k=300)
    X = chi2_selector.fit_transform(bow,Y)
    length = np.array(sms_dataset['Text_Length'])
    X = np.column_stack((length,X))
# Trying to fit all models
    models = [
            ('bernoulli_nb', BernoulliNB(alpha=1.0, binarize=0.0, class_prior=None, fit_prior=True)),
            ('svm',SVC(gamma='auto',kernel='linear')),
            ('xgboost', XGBClassifier()),
            ('random_forest', RandomForestClassifier(n_estimators = 1000, random_state = 42)),
            ('lgb',LGBMClassifier()),
            ]
    scores = pd.DataFrame([], columns=['model', 'accuracy', 'precision', 'Recall','F1-Score'])

    for model in models:

        start_time = time()

        i = 1
        element00 = 0
        element01 = 0
        element10 = 0
        element11 = 0
        cv_accuracy_score =[]
        cv_precision_score =[]
        cv_recall_score =[]
        cv_f1_score =[]

        cv = StratifiedKFold(n_splits = 10, random_state = 5, shuffle=True)

        Ykf = Y.values

        for train_index, test_index in cv.split(X,Ykf):
            X_train, X_test = X[train_index], X[test_index]
            y_train, y_test = Ykf[train_index],Ykf[test_index]

            model[1].fit(X_train,y_train)
            y_pred = model[1].predict(X_test)
```

```
            end_time = time()
            execution_time = (end_time) - (start_time)

            cm = (confusion_matrix(y_test, y_pred)).tolist()
            #print('Model : ',model[0],'\nConfusion Matrix: \n',confusion_matrix(y_test, y_pred))

            element00 += cm[0][0]
            element01 += cm[0][1]
            element10 += cm[1][0]
            element11 += cm[1][1]

            cv_accuracy_score.append(accuracy_score(y_test, y_pred))
            cv_precision_score.append(precision_score(y_test, y_pred))
            cv_recall_score.append(recall_score(y_test, y_pred))
            cv_f1_score.append(f1_score(y_test, y_pred))

            i += 1

        cnf_mtx = np.matrix([[element00, element01], [element10, element11]])
        print('Model : ',model[0],'\nConfusion Matrix Sum: \n',cnf_mtx)
        print('cv: ',cv_accuracy_score,'\nMean cv score: ',np.mean(cv_accuracy_score))
        scores = scores.append({
                'model':model[0],
                'accuracy' : np.mean(cv_accuracy_score),
                'precision': np.mean(cv_precision_score),
                'Recall': np.mean(cv_recall_score),
                'F1-Score': np.mean(cv_f1_score),
                'run_time': execution_time
            }, ignore_index=True)
    print(scores)
funcUnigramWithLength()
```

Figure 17: Code for Unigram Model With Length

Output generated was:

```
Model :  bernoulli_nb
Confusion Matrix Sum:
 [[746   1]
 [ 52 695]]
cv:  [0.9733333333333334, 0.9866666666666667, 0.96, 0.9533333333333334, 0.9666666666666667, 0.94, 0.96, 0.9662162162162162, 0.9594594594594594, 0.9797297297297297]
Mean cv score:  0.9645405405405405
Model :  svm
Confusion Matrix Sum:
 [[733  14]
 [ 45 702]]
cv:  [0.9666666666666667, 0.9733333333333334, 0.9466666666666667, 0.96, 0.9666666666666667, 0.94, 0.96, 0.9594594594594594, 0.9797297297297297, 0.9527027027027027]
Mean cv score:  0.9605225225225226
Model :  xgboost
Confusion Matrix Sum:
 [[716  31]
 [ 53 694]]
cv:  [0.9533333333333334, 0.9533333333333334, 0.9533333333333334, 0.9333333333333333, 0.94, 0.9466666666666667, 0.9333333333333333, 0.9527027027027027, 0.95270270270270
Mean cv score:  0.943765765765766
Model :  random_forest
Confusion Matrix Sum:
 [[722  25]
 [ 40 707]]
cv:  [0.9533333333333334, 0.9733333333333334, 0.96, 0.9666666666666667, 0.94, 0.96, 0.9466666666666667, 0.9594594594594594, 0.9662162162162162, 0.9391891891891891]
Mean cv score:  0.9564864864864864
Model :  lgb
Confusion Matrix Sum:
 [[722  25]
 [ 43 704]]
cv:  [0.9666666666666667, 0.9666666666666667, 0.96, 0.9466666666666667, 0.96, 0.9333333333333333, 0.9466666666666667, 0.9527027027027027, 0.9662162162162162, 0.94594594
Mean cv score:  0.9544864864864865
          model  accuracy  precision    Recall  F1-Score   run_time
0   bernoulli_nb  0.964541   0.998592  0.930414  0.963115   0.162245
1           svm  0.960523   0.980793  0.939802  0.959594  13.515427
2       xgboost  0.943766   0.957937  0.929009  0.942729   7.686625
3  random_forest  0.956486   0.966047  0.946450  0.955995  32.770090
4           lgb  0.954486   0.965925  0.942432  0.953803   1.200719
```

Figure 18: Output for Unigram Model With Length

4.3 Bigram Without Length Feature

For the Bigram score model, CountVectorizer from sklearn.feature_extraction was imported which was used to create a Unigram matrix where the ngram_range was passed as (2,2) and the same steps were followed as for Unigram model.The code and output are shown in below Figure 19 and 20. Code:

```
def funcBigram():
    bigram = CountVectorizer(tokenizer=lambda doc: doc, lowercase=False,ngram_range=(2,2))
    bag_of_words= bigram.fit_transform(sms_dataset['Text'])
    bow= bag_of_words.toarray()
#Chi Square :Feature Selection
    Y=sms_dataset['SMStype']
    chi2_selector = SelectKBest(chi2, k=300)
    X = chi2_selector.fit_transform(bow,Y)
# Trying to fit all models
    models = [
            ('bernoulli_nb', BernoulliNB(alpha=1.0, binarize=0.0, class_prior=None, fit_prior=True)),
            ('svm',SVC(gamma='auto',kernel='linear')),
            ('xgboost', XGBClassifier()),
            ('random_forest', RandomForestClassifier(n_estimators = 1000, random_state = 42)),
            ('lgb',LGBMClassifier()),
            ]
    scores = pd.DataFrame([], columns=['model', 'accuracy', 'precision', 'Recall','F1-Score'])

    for model in models:

        start_time = time()

        i = 1
        element00 = 0
        element01 = 0
        element10 = 0
        element11 = 0
        cv_accuracy_score =[]
        cv_precision_score =[]
        cv_recall_score =[]
        cv_f1_score =[]
```

```
        cv = StratifiedKFold(n_splits = 10, random_state = 5, shuffle=True)

        Ykf = Y.values

        for train_index, test_index in cv.split(X,Ykf):
            X_train, X_test = X[train_index], X[test_index]
            y_train, y_test = Ykf[train_index],Ykf[test_index]

            model[1].fit(X_train,y_train)
            y_pred = model[1].predict(X_test)

            end_time = time()
            execution_time = (end_time) - (start_time)

            cm = (confusion_matrix(y_test, y_pred)).tolist()


            element00 += cm[0][0]
            element01 += cm[0][1]
            element10 += cm[1][0]
            element11 += cm[1][1]

            cv_accuracy_score.append(accuracy_score(y_test, y_pred))
            cv_precision_score.append(precision_score(y_test, y_pred))
            cv_recall_score.append(recall_score(y_test, y_pred))
            cv_f1_score.append(f1_score(y_test, y_pred))

            i += 1

        cnf_mtx = np.matrix([[element00, element01], [element10, element11]])
        print('Model : ',model[0],'\nConfusion Matrix Sum: \n',cnf_mtx)
        print('cv: ',cv_accuracy_score,'\nMean cv score: ',np.mean(cv_accuracy_score))
        scores = scores.append({
                'model':model[0],
                'accuracy' : np.mean(cv_accuracy_score),
                'precision': np.mean(cv_precision_score),
                'Recall': np.mean(cv_recall_score),
                'F1-Score': np.mean(cv_f1_score),
                'run_time': execution_time
            }, ignore_index=True)
    print(scores)
funcBigram()
```

Figure 19: Code for Bigram Model Without Length

Output generated was:

```
Model :  bernoulli_nb
Confusion Matrix Sum:
 [[747   0]
 [260 487]]
cv:  [0.8066666666666666, 0.8333333333333334, 0.84, 0.8333333333333334, 0.8533333333333334, 0.82, 0.7866666666666666, 0.8378378378378378, 0.8378378378378378, 0.8108108108108109]
Mean cv score:  0.8259819819819819
Model :  svm
Confusion Matrix Sum:
 [[743   4]
 [179 568]]
cv:  [0.8666666666666667, 0.8866666666666667, 0.9066666666666666, 0.88, 0.88, 0.8666666666666667, 0.8533333333333334, 0.8648648648648649, 0.8918918918918919, 0.8783783783783784]
Mean cv score:  0.8775135135135136
Model :  xgboost
Confusion Matrix Sum:
 [[746   1]
 [305 442]]
cv:  [0.78, 0.8, 0.82, 0.7733333333333333, 0.8333333333333334, 0.7866666666666666, 0.7733333333333333, 0.8108108108108109, 0.8243243243243243, 0.75]
Mean cv score:  0.7951801801801801
Model :  random_forest
Confusion Matrix Sum:
 [[743   4]
 [176 571]]
cv:  [0.8666666666666667, 0.8866666666666667, 0.8933333333333333, 0.88, 0.9, 0.8666666666666667, 0.8666666666666667, 0.8513513513513513, 0.8918918918918919, 0.8918918918918919]
Mean cv score:  0.8795135135135135
Model :  lgb
Confusion Matrix Sum:
 [[746   1]
 [423 324]]
cv:  [0.74, 0.7466666666666667, 0.7, 0.74, 0.7466666666666667, 0.7133333333333334, 0.66, 0.7094594594594594, 0.7432432432432432, 0.6621621621621622]
Mean cv score:  0.7161531531531532
          model  accuracy  precision    Recall  F1-Score   run_time
0   bernoulli_nb  0.825982   1.000000  0.651964  0.788685   0.144221
1           svm  0.877514   0.993219  0.760396  0.860900   2.507051
2       xgboost  0.795180   0.997436  0.591712  0.741716   7.654397
3  random_forest  0.879514   0.993120  0.764414  0.863512  40.739040
4           lgb  0.716153   0.996154  0.433658  0.601816   0.372195
```

Figure 20: Output for Bigram Model Without Length

4.4 Bigram With Length Feature

Performed the same step as above along with the length feature Code:

```
# Bag of Words--- bigram
def funcBigramWithLength():
    bigram = CountVectorizer(tokenizer=lambda doc: doc, lowercase=False,ngram_range=(2,2))
    bag_of_words= bigram.fit_transform(sms_dataset['Text'])
    bow= bag_of_words.toarray()
#Chi Square :Feature Selection
    Y=sms_dataset['SMStype']
    chi2_selector = SelectKBest(chi2, k=300)
    X = chi2_selector.fit_transform(bow,Y)
    length = np.array(sms_dataset['Text_Length'])
    X = np.column_stack((length,X))
# Trying to fit all models
    models = [
            ('bernoulli_nb', BernoulliNB(alpha=1.0, binarize=0.0, class_prior=None, fit_prior=True)),
            ('svm',SVC(gamma='auto',kernel='linear')),
            ('xgboost', XGBClassifier()),
            ('random_forest', RandomForestClassifier(n_estimators = 1000, random_state = 42)),
            ('lgb',LGBMClassifier()),
            ]
    scores = pd.DataFrame([], columns=['model', 'accuracy', 'precision', 'Recall','F1-Score'])

    for model in models:

        start_time = time()

        i = 1
        element00 = 0
        element01 = 0
        element10 = 0
        element11 = 0
        cv_accuracy_score =[]
        cv_precision_score =[]
        cv_recall_score =[]
        cv_f1_score =[]
```

```
        cv = StratifiedKFold(n_splits = 10, random_state = 5, shuffle=True)

        Ykf = Y.values

        for train_index, test_index in cv.split(X,Ykf):
            X_train, X_test = X[train_index], X[test_index]
            y_train, y_test = Ykf[train_index],Ykf[test_index]

            model[1].fit(X_train,y_train)
            y_pred = model[1].predict(X_test)

            end_time = time()
            execution_time = (end_time) - (start_time)

            cm = (confusion_matrix(y_test, y_pred)).tolist()
           # print('Model : ',model[0],'\nConfusion Matrix: \n',confusion_matrix(y_test, y_pred))

            element00 += cm[0][0]
            element01 += cm[0][1]
            element10 += cm[1][0]
            element11 += cm[1][1]

            cv_accuracy_score.append(accuracy_score(y_test, y_pred))
            cv_precision_score.append(precision_score(y_test, y_pred))
            cv_recall_score.append(recall_score(y_test, y_pred))
            cv_f1_score.append(f1_score(y_test, y_pred))

            i += 1

        cnf_mtx = np.matrix([[element00, element01], [element10, element11]])
        print('Model : ',model[0],'\nConfusion Matrix Sum: \n',cnf_mtx)
        print('cv: ',cv_accuracy_score,'\nMean cv score: ',np.mean(cv_accuracy_score))
        scores = scores.append({
                'model':model[0],
                'accuracy' : np.mean(cv_accuracy_score),
                'precision': np.mean(cv_precision_score),
                'Recall': np.mean(cv_recall_score),
                'F1-Score': np.mean(cv_f1_score),
                'run_time': execution_time
            }, ignore_index=True)
    print(scores)
funcBigramWithLength()
```

Figure 21: Code for Bigram Model With Length

Output generated was:

```
Model :  bernoulli_nb
Confusion Matrix Sum:
 [[747   0]
 [260 487]]
cv:  [0.8066666666666666, 0.8333333333333334, 0.84, 0.8333333333333334, 0.8533333333333334, 0.82, 0.7866666666666666, 0.8378378378378378, 0.8378378378378378, 0.8108108108108109]
Mean cv score:  0.8259819819819819
Model :  svm
Confusion Matrix Sum:
 [[735  12]
 [174 573]]
cv:  [0.8666666666666667, 0.9, 0.9, 0.8733333333333333, 0.8933333333333333, 0.8666666666666667, 0.8533333333333334, 0.8513513513513513, 0.8918918918918919, 0.8581081081081081]
Mean cv score:  0.8754684684684685
Model :  xgboost
Confusion Matrix Sum:
 [[650  97]
 [ 84 663]]
cv:  [0.9066666666666666, 0.86, 0.8733333333333333, 0.8733333333333333, 0.8733333333333333, 0.9, 0.8933333333333333, 0.8783783783783784, 0.8716216216216216, 0.8581081081081081]
Mean cv score:  0.8788108108108108
Model :  random_forest
Confusion Matrix Sum:
 [[687  60]
 [108 639]]
cv:  [0.9066666666666666, 0.8733333333333333, 0.8733333333333333, 0.8666666666666667, 0.9, 0.9066666666666666, 0.86, 0.8851351351351351, 0.9121621621621622, 0.8918918918918919]
Mean cv score:  0.8875855855855855
Model :  lgb
Confusion Matrix Sum:
 [[656  91]
 [ 88 659]]
cv:  [0.92, 0.8733333333333333, 0.88, 0.8666666666666667, 0.8866666666666667, 0.9133333333333333, 0.8666666666666667, 0.8716216216216216, 0.8783783783783784, 0.8445945945945946]
Mean cv score:  0.880126126126126
          model  accuracy  precision    Recall  F1-Score   run_time
0   bernoulli_nb  0.825982   1.000000  0.651964  0.788685   0.142750
1           svm  0.875468   0.979750  0.767063  0.860071  36.287825
2       xgboost  0.878811   0.872973  0.887532  0.879769   7.789369
3  random_forest  0.887586   0.915643  0.855477  0.883778  39.336061
4           lgb  0.880126   0.878927  0.882072  0.880002   0.630908
```

Figure 22: Output for Bigram Model With Length

4.5 TF-IDF Without Length Feature

For the TF-IDF score model, TFIDFVectorizer from sklearn.feature_extraction was imported where the ngram_range was passed as (1,1) and the same steps were followed as for above models.The code and output are shown in below Figure 21 and 22. Code:

```
[ ]  # Bag of Words--- tfidf
     def funcTfIDF():
         tfidf = TfidfVectorizer(tokenizer = lambda doc: doc, lowercase = False,ngram_range = (1, 1))
         doc_matrix = tfidf.fit_transform(sms_dataset['Text'])
         bow= doc_matrix.toarray()
     #Chi Square :Feature Selection
         Y=sms_dataset['SMStype']
         chi2_selector = SelectKBest(chi2, k=300)
         X = chi2_selector.fit_transform(bow,Y)
     # Trying to fit all models
         models = [
                 ('bernoulli_nb', BernoulliNB(alpha=1.0, binarize=0.0, class_prior=None, fit_prior=True)),
                 ('svm',SVC(gamma='auto',kernel='linear')),
                 ('xgboost', XGBClassifier()),
                 ('random_forest', RandomForestClassifier(n_estimators = 1000, random_state = 42)),
                 ('lgb',LGBMClassifier()),
                 ]
         scores = pd.DataFrame([], columns=['model', 'accuracy', 'precision', 'Recall','F1-Score'])

         for model in models:

             start_time = time()

             i = 1
             element00 = 0
             element01 = 0
             element10 = 0
             element11 = 0
             cv_accuracy_score =[]
             cv_precision_score =[]
             cv_recall_score =[]
             cv_f1_score =[]
```

```
            cv = StratifiedKFold(n_splits = 10, random_state = 5, shuffle=True)

            Ykf = Y.values

            for train_index, test_index in cv.split(X,Ykf):
                X_train, X_test = X[train_index], X[test_index]
                y_train, y_test = Ykf[train_index],Ykf[test_index]

                model[1].fit(X_train,y_train)
                y_pred = model[1].predict(X_test)

                end_time = time()
                execution_time = (end_time) - (start_time)

                cm = (confusion_matrix(y_test, y_pred)).tolist()
                #print('Model : ',model[0],'\nConfusion Matrix: \n',confusion_matrix(y_test, y_pred))

                element00 += cm[0][0]
                element01 += cm[0][1]
                element10 += cm[1][0]
                element11 += cm[1][1]

                cv_accuracy_score.append(accuracy_score(y_test, y_pred))
                cv_precision_score.append(precision_score(y_test, y_pred))
                cv_recall_score.append(recall_score(y_test, y_pred))
                cv_f1_score.append(f1_score(y_test, y_pred))

                i += 1

            cnf_mtx = np.matrix([[element00, element01], [element10, element11]])
            print('Model : ',model[0],'\nConfusion Matrix Sum: \n',cnf_mtx)
            print('cv: ',cv_accuracy_score,'\nMean cv score: ',np.mean(cv_accuracy_score))
            scores = scores.append({
                    'model':model[0],
                    'accuracy' : np.mean(cv_accuracy_score),
                    'precision': np.mean(cv_precision_score),
                    'Recall': np.mean(cv_recall_score),
                    'F1-Score': np.mean(cv_f1_score),
                    'run_time': execution_time
                }, ignore_index=True)
        print(scores)
    funcTfIDF()
```

Figure 23: Code for TF-IDF Model Without Length

Output generated was:

```
Model :  bernoulli_nb
Confusion Matrix Sum:
 [[746   1]
 [ 51 696]]
cv:  [0.9733333333333334, 0.98, 0.9666666666666667, 0.96, 0.98, 0.94, 0.9466666666666667, 0.9594594594594594, 0.972972972972973, 0.972972972972973]
Mean cv score:  0.9652072072072073
Model :  svm
Confusion Matrix Sum:
 [[740   7]
 [ 70 677]]
cv:  [0.94, 0.98, 0.9466666666666667, 0.9533333333333334, 0.9666666666666667, 0.9266666666666666, 0.94, 0.9391891891891891, 0.9594594594594594, 0.9324324324324325]
Mean cv score:  0.9484414414414413
Model :  xgboost
Confusion Matrix Sum:
 [[723  24]
 [ 63 684]]
cv:  [0.94, 0.9533333333333334, 0.9533333333333334, 0.9533333333333334, 0.92, 0.94, 0.9266666666666666, 0.9594594594594594, 0.9391891891891891, 0.9324324324324325]
Mean cv score:  0.9417747747747747
Model :  random_forest
Confusion Matrix Sum:
 [[721  26]
 [ 39 708]]
cv:  [0.96, 0.9733333333333334, 0.9666666666666667, 0.9533333333333334, 0.9533333333333334, 0.9466666666666667, 0.94, 0.9459459459459459, 0.9662162162162162, 0.9594594594594594]
Mean cv score:  0.9564954954954954
Model :  lgb
Confusion Matrix Sum:
 [[716  31]
 [ 33 714]]
cv:  [0.9666666666666667, 0.9733333333333334, 0.9666666666666667, 0.96, 0.9666666666666667, 0.9333333333333333, 0.94, 0.9391891891891891, 0.9797297297297297, 0.9459459459459459]
Mean cv score:  0.957153153153153
          model  accuracy  precision    Recall  F1-Score   run_time
0   bernoulli_nb  0.965207   0.998611  0.931748  0.963813   0.150926
1           svm  0.948441   0.990037  0.906288  0.945939   2.602606
2       xgboost  0.941775   0.966626  0.915694  0.940192   7.926118
3  random_forest  0.956495   0.964662  0.947802  0.956033  38.320872
4           lgb  0.957153   0.958324  0.955838  0.956951   1.705751
```

Figure 24: Output for TF-IDF Model Without Length

4.6 TF-IDF With Length Feature

Performed the same step as above along with the length feature Code:

```
[ ]  # Bag of Words--- tfidf
     def funcTfIDFWithLength():
         tfidf = TfidfVectorizer(tokenizer = lambda doc: doc, lowercase = False,ngram_range = (1, 1))
         doc_matrix = tfidf.fit_transform(sms_dataset['Text'])
         bow= doc_matrix.toarray()
     #Chi Square :Feature Selection
         Y=sms_dataset['SMStype']
         chi2_selector = SelectKBest(chi2, k=300)
         X = chi2_selector.fit_transform(bow,Y)
         length = np.array(sms_dataset['Text_Length'])
         X = np.column_stack((length,X))
         #df=pd.DataFrame(X_kbest)
         #df.rename(columns={0:'Text_Length'}, inplace=True)
         #df.head()
     # Trying to fit all models
         models = [
                 ('bernoulli_nb', BernoulliNB(alpha=1.0, binarize=0.0, class_prior=None, fit_prior=True)),
                 ('svm',SVC(gamma='auto',kernel='linear')),
                 ('xgboost', XGBClassifier()),
                 ('random_forest', RandomForestClassifier(n_estimators = 1000, random_state = 42)),
                 ('lgb',LGBMClassifier()),
                 ]
         scores = pd.DataFrame([], columns=['model', 'accuracy', 'precision', 'Recall','F1-Score'])

         for model in models:

             start_time = time()

             i = 1
             element00 = 0
             element01 = 0
             element10 = 0
             element11 = 0
             cv_accuracy_score =[]
             cv_precision_score =[]
             cv_recall_score =[]
             cv_f1_score =[]
```

```
            cv = StratifiedKFold(n_splits = 10, random_state = 5, shuffle=True)

            Ykf = Y.values

            for train_index, test_index in cv.split(X,Ykf):
                X_train, X_test = X[train_index], X[test_index]
                y_train, y_test = Ykf[train_index],Ykf[test_index]

                model[1].fit(X_train,y_train)
                y_pred = model[1].predict(X_test)

                end_time = time()
                execution_time = (end_time) - (start_time)

                cm = (confusion_matrix(y_test, y_pred)).tolist()
                #print('Model : ',model[0],'\nConfusion Matrix: \n',confusion_matrix(y_test, y_pred))

                element00 += cm[0][0]
                element01 += cm[0][1]
                element10 += cm[1][0]
                element11 += cm[1][1]

                cv_accuracy_score.append(accuracy_score(y_test, y_pred))
                cv_precision_score.append(precision_score(y_test, y_pred))
                cv_recall_score.append(recall_score(y_test, y_pred))
                cv_f1_score.append(f1_score(y_test, y_pred))

                i += 1

            cnf_mtx = np.matrix([[element00, element01], [element10, element11]])
            print('Model : ',model[0],'\nConfusion Matrix Sum: \n',cnf_mtx)
            print('cv: ',cv_accuracy_score,'\nMean cv score: ',np.mean(cv_accuracy_score))
            scores = scores.append({
                    'model':model[0],
                    'accuracy' : np.mean(cv_accuracy_score),
                    'precision': np.mean(cv_precision_score),
                    'Recall': np.mean(cv_recall_score),
                    'F1-Score': np.mean(cv_f1_score),
                    'run_time': execution_time
                }, ignore_index=True)
        print(scores)
    funcTfIDFWithLength()
```

Figure 25: Code for TF-IDF Model With Length

Output generated was:

```
Model :  bernoulli_nb
Confusion Matrix Sum:
 [[746   1]
 [ 51 696]]
cv:  [0.9733333333333334, 0.98, 0.9666666666666667, 0.96, 0.98, 0.94, 0.9466666666666667, 0.9594594594594594, 0.972972972972973, 0.972972972972973]
Mean cv score:  0.9652072072072073
Model :  svm
Confusion Matrix Sum:
 [[727  20]
 [ 50 697]]
cv:  [0.96, 0.9866666666666667, 0.9466666666666667, 0.9533333333333334, 0.96, 0.9333333333333333, 0.9333333333333333, 0.9527027027027027, 0.9594594594594594, 0.9459459459459459]
Mean cv score:  0.9531441441441441
Model :  xgboost
Confusion Matrix Sum:
 [[718  29]
 [ 54 693]]
cv:  [0.9533333333333334, 0.96, 0.9533333333333334, 0.9466666666666667, 0.9466666666666667, 0.9466666666666667, 0.9333333333333333, 0.9459459459459459, 0.9459459459459459, 0.9121621621621622]
Mean cv score:  0.9444054054054053
Model :  random_forest
Confusion Matrix Sum:
 [[727  20]
 [ 34 713]]
cv:  [0.96, 0.98, 0.9666666666666667, 0.9666666666666667, 0.9666666666666667, 0.96, 0.94, 0.972972972972973, 0.9594594594594594, 0.9662162162162162]
Mean cv score:  0.9638648648648648
Model :  lgb
Confusion Matrix Sum:
 [[719  28]
 [ 41 706]]
cv:  [0.96, 0.9666666666666667, 0.9666666666666667, 0.94, 0.9666666666666667, 0.9466666666666667, 0.9466666666666667, 0.9459459459459459, 0.9527027027027027, 0.9459459459459459]
Mean cv score:  0.9537927927927928
          model  accuracy  precision    Recall  F1-Score   run_time
0   bernoulli_nb  0.965207   0.998611  0.931748  0.963813   0.156936
1            svm  0.953144   0.972263  0.933099  0.952114  30.285465
2        xgboost  0.944405   0.960310  0.927658  0.943216   7.919159
3  random_forest  0.963865   0.973089  0.954505  0.963466  34.957316
4            lgb  0.953793   0.962077  0.945081  0.953326   1.707481
```

Figure 26: Output for TF-IDF Model With Length

4.7 Conclusion

Looking at the results and analyzing it in the technical report, it was seen that Bernoulli Naive Bayes with TF-IDF with length outperformed amongst other model with an accuracy of 0.965 and execution time of 0.15 seconds followed by LightGBM with accuracy of 0.954 and execution time of 1.708 seconds.Hence these 2 models are the best fit for the problem.

References

Agarwal, S., Kaur, S. and Garhwal, S. (2016). SMS spam detection for Indian messages, *Proceedings on 2015 1st International Conference on Next Generation Computing Technologies, NGCT 2015* (September): 634–638.