Configuration Manual

MSc Research Project
Data Analytics

Ramyaa Rajasekar
Student ID: x21122881

School of Computing
National College of Ireland

Supervisor: Qurrat Ul Ain

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Ramyaa Rajasekar
Student ID:	x21122881
Programme:	Data Analytics
Year:	2022-2023
Module:	MSc Research Project
Supervisor:	Qurrat Ul Ain
Submission Due Date:	01/02/2023
Project Title:	Configuration Manual
Word Count:	802
Page Count:	9

Signature:	Ramyaa Rajasekar
Date:	1st February 2023

Configuration Manual

Ramyaa Rajasekar
x21122881

1 Introduction

The motive of this paper is to outline the software requirements and provide an overview of the code artifacts implemented to achieve the objective of the research, in developing traditional machine learning, Ensemble Learning, and deep learning models with feature transformation and hyperparameter tuning to classify the gene variants using the clinical literature.

2 System Specifications

The project was implemented on a cloud platform Google Colab, which is well known for coding and executing machine learning and deep learning models by integrating tensor flow and Keras. The execution speed is increased compared to the CPU as it utilizes the GPU and TPU when required.

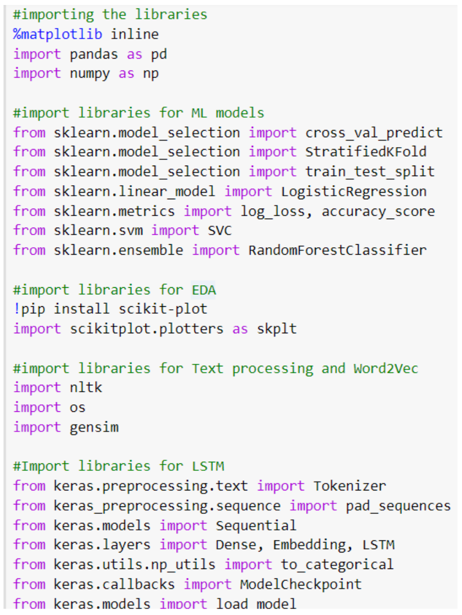
- Google Colab: Intel Xeon CPU @2.20 GHz
- RAM: 13 GB
- Disk Space: 78 GB
- Processor: Intel R Core (i5)
- System RAM: 16 GB
- Operating System: Windows 11 64 bit

2.1 Software Requirements

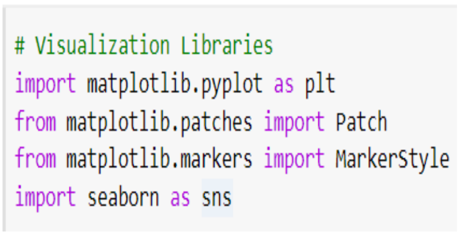
The programming language used to implement the classification approach is Python, as it is free and open source and has various integrated libraries which can be utilized for implementing NLP techniques, Word Embedding techniques, and Tenser flow and Keras versions for deep learning models.

3 Importing Python Libraries

One of the main advantages of the collab is, it is not required to install all the libraries during every session, it has multiple pre-installed libraries which can be directly imported such as numpy, panda, matplotlib,scikit-plot,gensim,nltk, and model building libraries.



```
#importing the libraries
%matplotlib inline
import pandas as pd
import numpy as np

#import libraries for ML models
from sklearn.model_selection import cross_val_predict
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import log_loss, accuracy_score
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier

#import libraries for EDA
!pip install scikit-plot
import scikitplot.plotters as skplt

#import libraries for Text processing and Word2Vec
import nltk
import os
import gensim

#Import libraries for LSTM
from keras.preprocessing.text import Tokenizer
from keras_preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Dense, Embedding, LSTM
from keras.utils.np_utils import to_categorical
from keras.callbacks import ModelCheckpoint
from keras.models import load_model
```

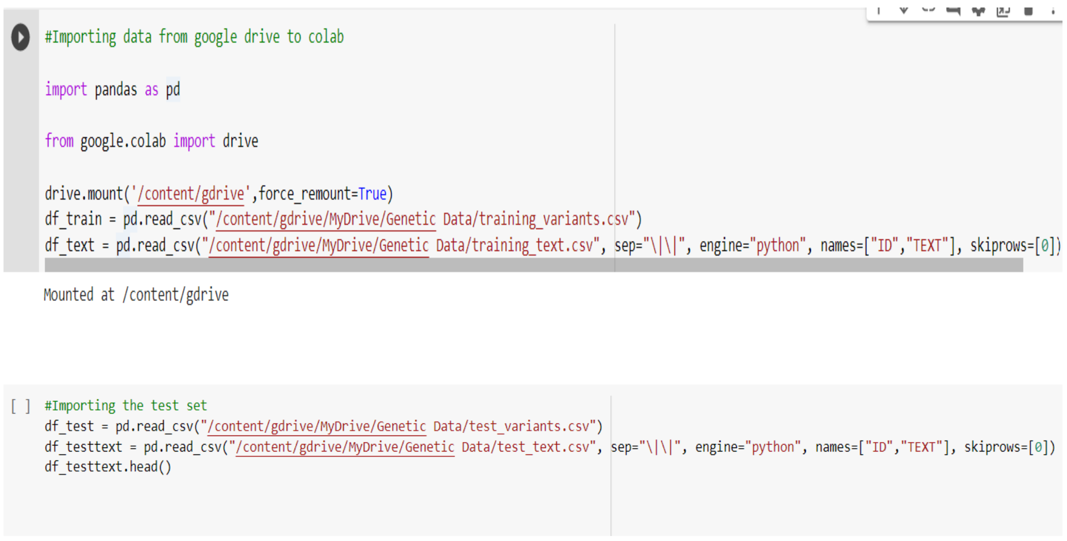


```
# Visualization Libraries
import matplotlib.pyplot as plt
from matplotlib.patches import Patch
from matplotlib.markers import MarkerStyle
import seaborn as sns
```

Figure 1: Import Libraries

4 Loading the data

The data is categorized into different CSV files as Training and Testing Variants and Text files. As the text file is unstructured data, to reduce the time taken to load in the collab, have imported it from Google drive.



```
#Importing data from google drive to colab

import pandas as pd

from google.colab import drive

drive.mount('/content/gdrive',force_remount=True)
df_train = pd.read_csv("/content/gdrive/MyDrive/Genetic Data/training_variants.csv")
df_text = pd.read_csv("/content/gdrive/MyDrive/Genetic Data/training_text.csv", sep="\|\|", engine="python", names=["ID","TEXT"], skiprows=[0])
```

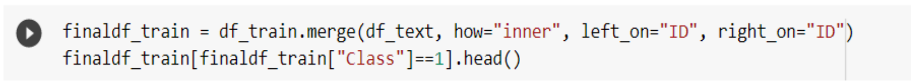
```
Mounted at /content/gdrive
```

```
#Importing the test set
df_test = pd.read_csv("/content/gdrive/MyDrive/Genetic Data/test_variants.csv")
df_testtext = pd.read_csv("/content/gdrive/MyDrive/Genetic Data/test_text.csv", sep="\|\|", engine="python", names=["ID","TEXT"], skiprows=[0])
df_testtext.head()
```

Figure 2: Loading the train and test data

4.1 Merging the Data

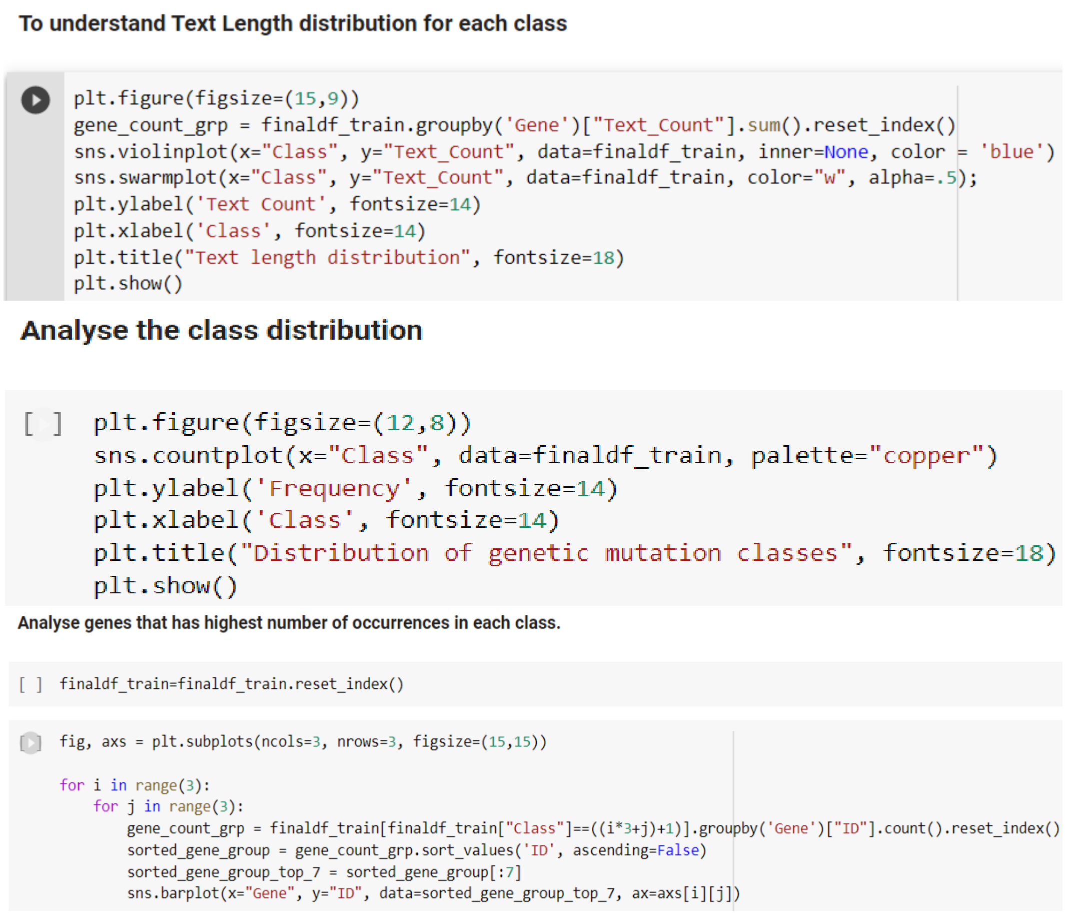
As the training set comprises gene variants and text files, it is required to merge it as a single data frame and proceed with the further process.



```
finaldf_train = df_train.merge(df_text, how="inner", left_on="ID", right_on="ID")
finaldf_train[finaldf_train["Class"]==1].head()
```

Figure 3: Merging the data

5 Exploratory Data Analysis



To understand Text Length distribution for each class

```
plt.figure(figsize=(15,9))
gene_count_grp = finaldf_train.groupby('Gene')["Text_Count"].sum().reset_index()
sns.violinplot(x="Class", y="Text_Count", data=finaldf_train, inner=None, color = 'blue')
sns.swarmplot(x="Class", y="Text_Count", data=finaldf_train, color="w", alpha=.5);
plt.ylabel('Text Count', fontsize=14)
plt.xlabel('Class', fontsize=14)
plt.title("Text length distribution", fontsize=18)
plt.show()
```

Analyse the class distribution

```
[ ] plt.figure(figsize=(12,8))
    sns.countplot(x="Class", data=finaldf_train, palette="copper")
    plt.ylabel('Frequency', fontsize=14)
    plt.xlabel('Class', fontsize=14)
    plt.title("Distribution of genetic mutation classes", fontsize=18)
    plt.show()
```

Analyse genes that has highest number of occurrences in each class.

```
[ ] finaldf_train=finaldf_train.reset_index()
```

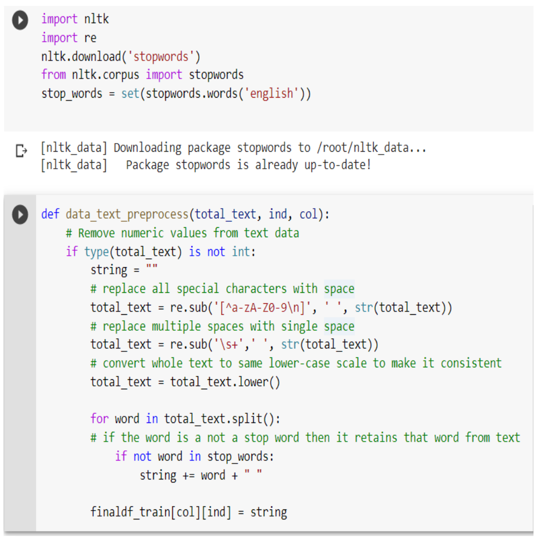
```
fig, axs = plt.subplots(ncols=3, nrows=3, figsize=(15,15))

for i in range(3):
    for j in range(3):
        gene_count_grp = finaldf_train[finaldf_train["Class"]==((i*3+j)+1)].groupby('Gene')["ID"].count().reset_index()
        sorted_gene_group = gene_count_grp.sort_values('ID', ascending=False)
        sorted_gene_group_top_7 = sorted_gene_group[:7]
        sns.barplot(x="Gene", y="ID", data=sorted_gene_group_top_7, ax=axs[i][j])
```

Figure 4: Visualisation of data

Exploratory Data Analysis (EDA) was implemented to analyze the data and get pertinent knowledge about the gene variations and clinical text correlations.EDA is performed for the Text column to get insights about the distribution of text for every class, for the Gene column and assume which gene affects the individual and categorizes to more number cancer classes, for the class column to understand the number of samples available for each class and get insights about the data linearity.

6 Text Data Pre-processing

As the data chosen is in textual format, it is required to apply natural language processing techniques along with the handling of missing values which would upheld the model-building process and performance.

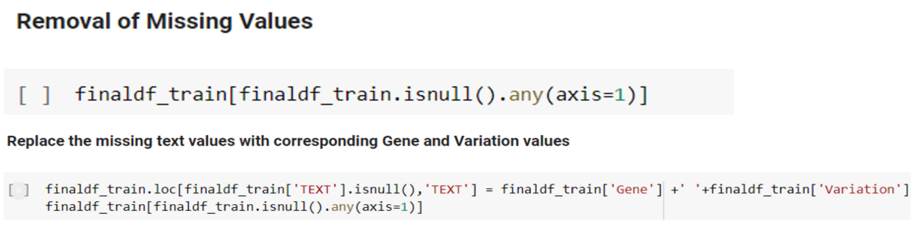


```
import nltk
import re
nltk.download('stopwords')
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
def data_text_preprocess(total_text, ind, col):
    # Remove numeric values from text data
    if type(total_text) is not int:
        string = ""
        # replace all special characters with space
        total_text = re.sub('[^a-zA-Z0-9\n]', ' ', str(total_text))
        # replace multiple spaces with single space
        total_text = re.sub('\s+',' ', str(total_text))
        # convert whole text to same lower-case scale to make it consistent
        total_text = total_text.lower()

        for word in total_text.split():
        # if the word is a not a stop word then it retains that word from text
            if not word in stop_words:
                string += word + " "

        finaldf_train[col][ind] = string
```

Figure 5: Pre-processing using NLP



Removal of Missing Values

```
[ ]  finaldf_train[finaldf_train.isnull().any(axis=1)]
```

Replace the missing text values with corresponding Gene and Variation values

```
[ ]  finaldf_train.loc[finaldf_train['TEXT'].isnull(),'TEXT'] = finaldf_train['Gene'] +' '+finaldf_train['Variation']
     finaldf_train[finaldf_train.isnull().any(axis=1)]
```

7 Tranformation of Data using Embedding Technique

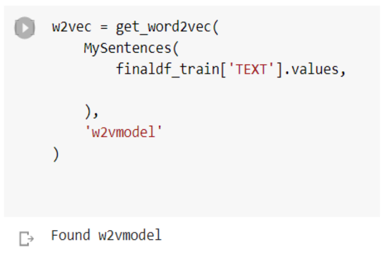
The word embedding technique utilized to transform the text into a numerical vector is Word2Vec by importing the genism library. This technique is applied for Gene and Variation column and visualization was performed with t-SNE and K means clustering to understand how the words are related and embedding has been performed.

To convert a document of multiple words into a single vector using our trained word2vec, we take the average or mean of the vectors of that sentence. A transformer (with a sklearn interface) is defined to convert a document into its corresponding vector.

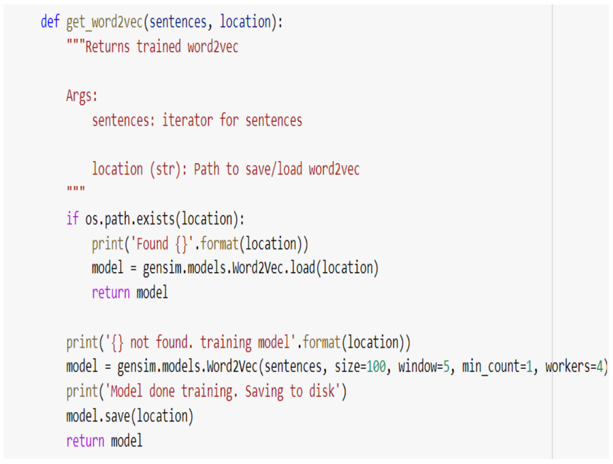
8 Machine Learning and Deep Learning Models

Initially, The dataset was split into Train and Test split with an 80 to 20 ratio. For the purpose of model evaluation, the train set split is performed and with the best-performed model, the prediction was done on the test set provided. Following to that nine different models were built and compared before and after applying hyperparameter tuning where ever it was required. The models built are Logistic Regression, Random Forest, Support Vector Machine, K nearest neighbor, Gradient Boosting, Majority Voting Classifier, and Long Short Term memory (LSTM).

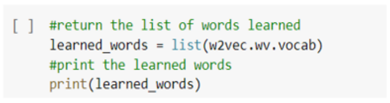
Training the word2vec model for Text column

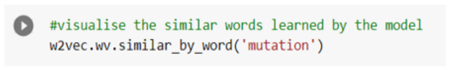

```
w2vec = get_word2vec(
    MySentences(
        finaldf_train['TEXT'].values,

    ),
    'w2vmodel'
)
```

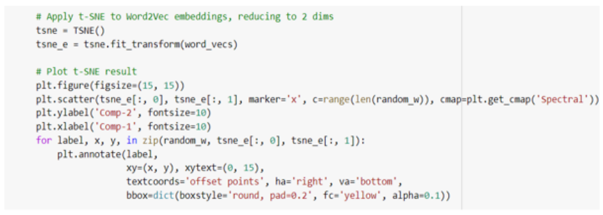
```
Found w2vmodel
```

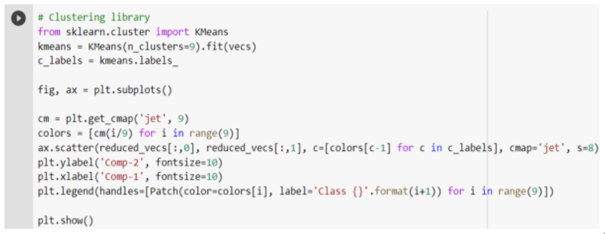

```
def get_word2vec(sentences, location):
    """Returns trained word2vec

    Args:
        sentences: iterator for sentences

        location (str): Path to save/load word2vec
    """
    if os.path.exists(location):
        print('Found {}'.format(location))
        model = gensim.models.Word2Vec.load(location)
        return model

    print('{} not found. training model'.format(location))
    model = gensim.models.Word2Vec(sentences, size=100, window=5, min_count=1, workers=4)
    print('Model done training. Saving to disk')
    model.save(location)
    return model
```

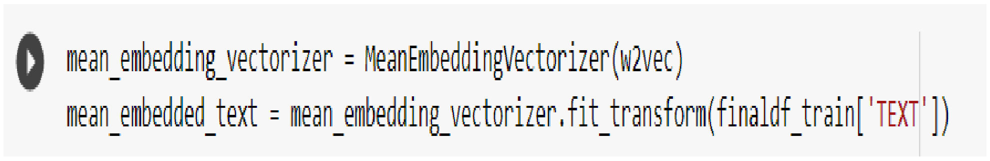
To visualise the list of vocabs learned by the model

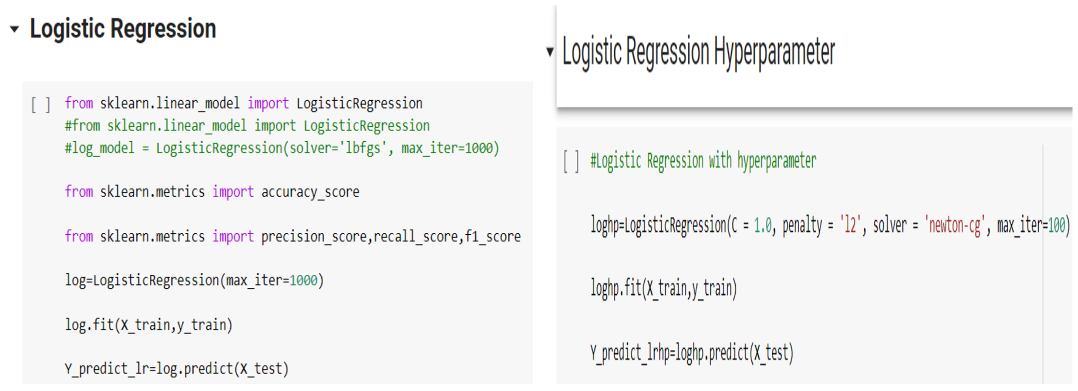

```
[ ] #return the list of words learned
    learned_words = list(w2vec.wv.vocab)
    #print the learned words
    print(learned_words)
```


```
#visualise the similar words learned by the model
w2vec.wv.similar_by_word('mutation')
```


```
# Apply t-SNE to Word2Vec embeddings, reducing to 2 dims
tsne = TSNE()
tsne_e = tsne.fit_transform(word_vecs)

# Plot t-SNE result
plt.figure(figsize=(15, 15))
plt.scatter(tsne_e[:, 0], tsne_e[:, 1], marker='x', c=range(len(random_w)), cmap=plt.get_cmap('Spectral'))
plt.ylabel('Comp-2', fontsize=10)
plt.xlabel('Comp-1', fontsize=10)
for label, x, y, in zip(random_w, tsne_e[:, 0], tsne_e[:, 1]):
    plt.annotate(label,
                 xy=(x, y), xytext=(0, 15),
                 textcoords='offset points', ha='right', va='bottom',
                 bbox=dict(boxstyle='round, pad=0.2', fc='yellow', alpha=0.1))
```


```
# Clustering library
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=9).fit(vecs)
c_labels = kmeans.labels_

fig, ax = plt.subplots()

cm = plt.get_cmap('jet', 9)
colors = [cm(i/9) for i in range(9)]
ax.scatter(reduced_vecs[:,0], reduced_vecs[:,1], c=[colors[c-1] for c in c_labels], cmap='jet', s=8)
plt.ylabel('Comp-2', fontsize=10)
plt.xlabel('Comp-1', fontsize=10)
plt.legend(handles=[Patch(color=colors[i], label='Class {}'.format(i+1)) for i in range(9)])

plt.show()
```

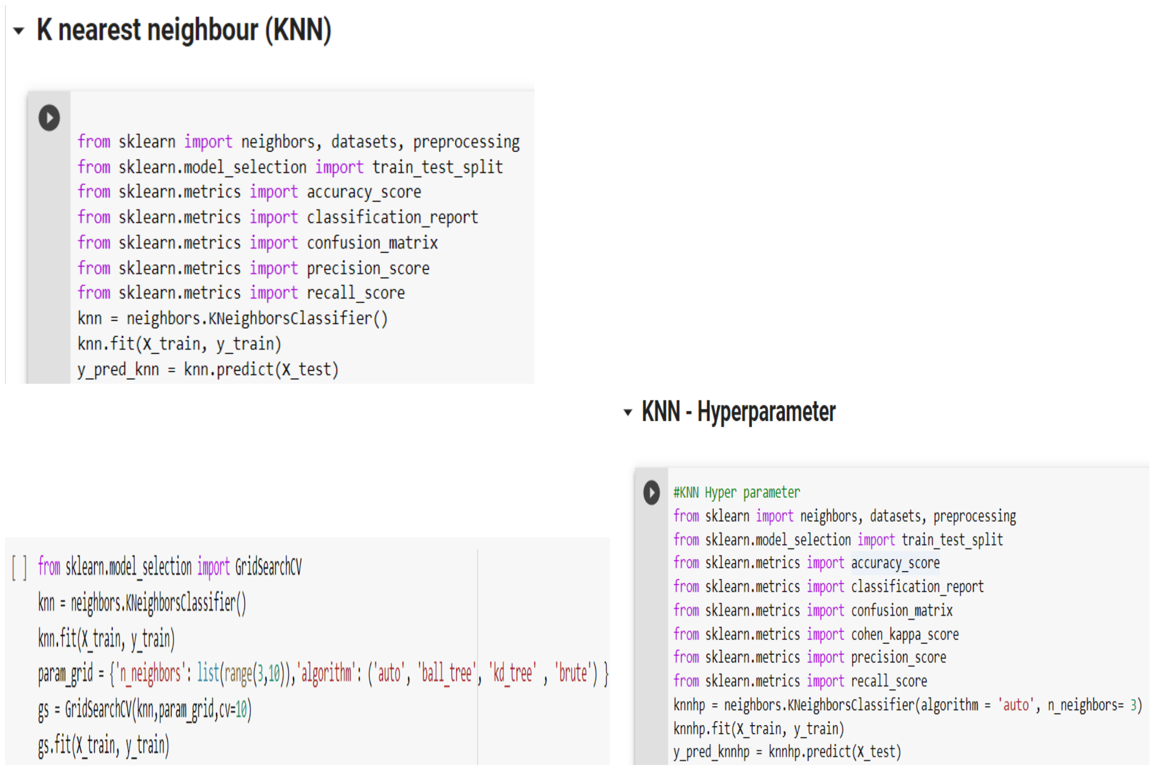

```
mean_embedding_vectorizer = MeanEmbeddingVectorizer(w2vec)
mean_embedded_text = mean_embedding_vectorizer.fit_transform(finaldf_train['TEXT'])
```



```
from sklearn.linear_model import LogisticRegression
#from sklearn.linear_model import LogisticRegression
#log_model = LogisticRegression(solver='lbfgs', max_iter=1000)

from sklearn.metrics import accuracy_score

from sklearn.metrics import precision_score,recall_score,f1_score

log=LogisticRegression(max_iter=1000)

log.fit(X_train,y_train)

Y_predict_lr=log.predict(X_test)
```

```
#Logistic Regression with hyperparameter

loghp=LogisticRegression(C = 1.0, penalty = 'l2', solver = 'newton-cg', max_iter=100)

loghp.fit(X_train,y_train)

Y_predict_lrhp=loghp.predict(X_test)
```

Figure 6: Logistic Regression

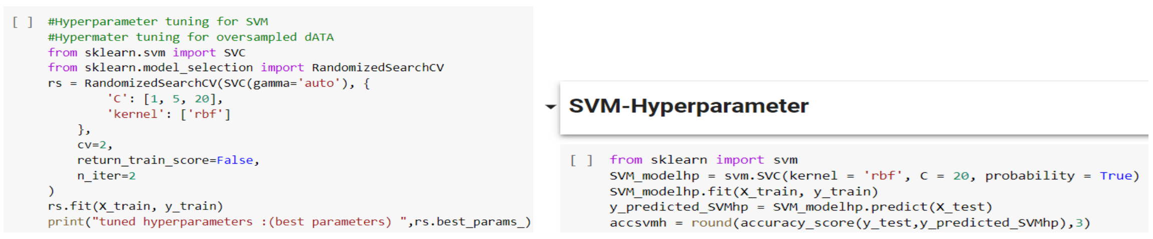


```
from sklearn import neighbors, datasets, preprocessing
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
knn = neighbors.KNeighborsClassifier()
knn.fit(X_train, y_train)
y_pred_knn = knn.predict(X_test)
```

```
#KNN Hyper parameter
from sklearn import neighbors, datasets, preprocessing
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import cohen_kappa_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
knnhp = neighbors.KNeighborsClassifier(algorithm = 'auto', n_neighbors= 3)
knnhp.fit(X_train, y_train)
y_pred_knnhp = knnhp.predict(X_test)
```

```
from sklearn.model_selection import GridSearchCV
knn = neighbors.KNeighborsClassifier()
knn.fit(X_train, y_train)
param_grid = {'n_neighbors': list(range(3,10)),'algorithm': ('auto', 'ball_tree', 'kd_tree' , 'brute') }
gs = GridSearchCV(knn,param_grid,cv=10)
gs.fit(X_train, y_train)
```

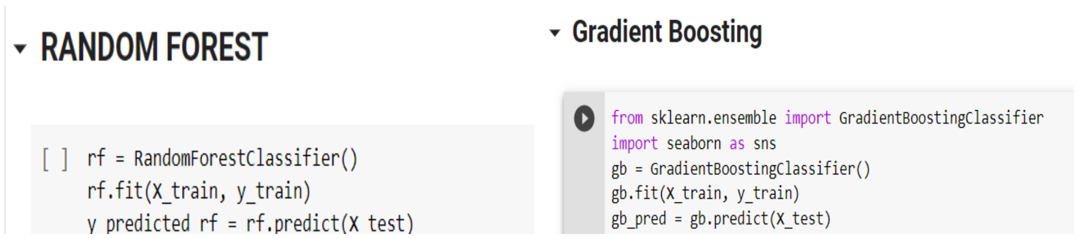
Figure 7: K Nearest Neighbour



```
#Hyperparameter tuning for SVM
#Hypermater tuning for oversampled dATA
from sklearn.svm import SVC
from sklearn.model_selection import RandomizedSearchCV
rs = RandomizedSearchCV(SVC(gamma='auto'), {
        'C': [1, 5, 20],
        'kernel': ['rbf']
    },
    cv=2,
    return_train_score=False,
    n_iter=2
)
rs.fit(X_train, y_train)
print("tuned hyperparameters :(best parameters) ",rs.best_params_)
```

```
from sklearn import svm
SVM_modelhp = svm.SVC(kernel = 'rbf', C = 20, probability = True)
SVM_modelhp.fit(X_train, y_train)
y_predicted_SVMhp = SVM_modelhp.predict(X_test)
accsvmh = round(accuracy_score(y_test,y_predicted_SVMhp),3)
```

Figure 8: Support Vector Machine



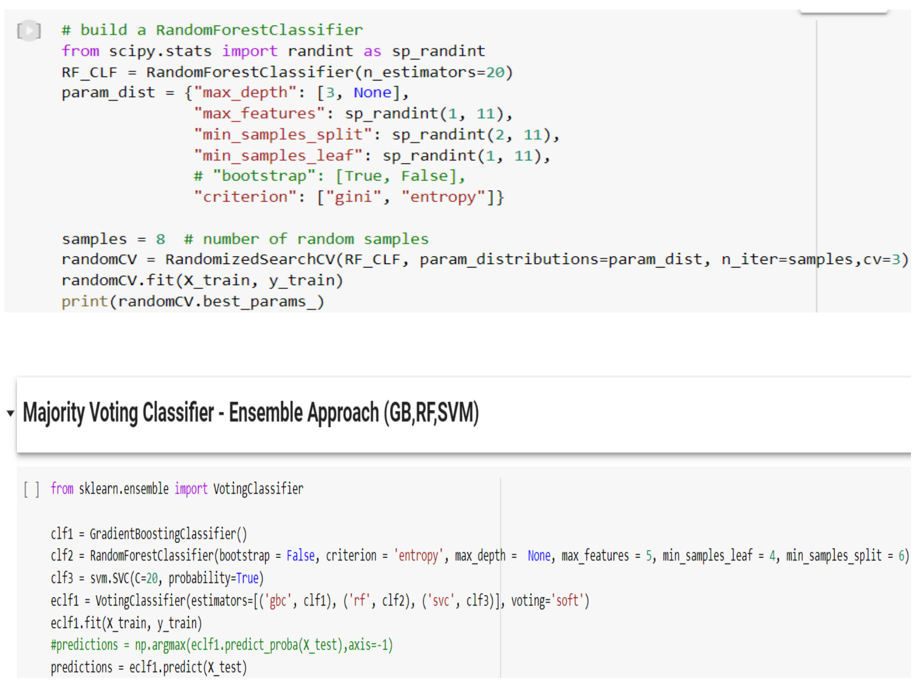
RANDOM FOREST

```
rf = RandomForestClassifier()
rf.fit(X_train, y_train)
y_predicted_rf = rf.predict(X_test)
```

Gradient Boosting

```
from sklearn.ensemble import GradientBoostingClassifier
import seaborn as sns
gb = GradientBoostingClassifier()
gb.fit(X_train, y_train)
gb_pred = gb.predict(X_test)
```

Figure 9: Random Forest and Gradient Boosting

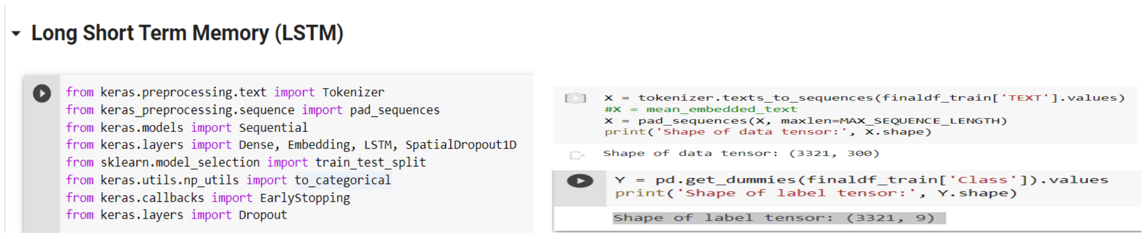


```
# build a RandomForestClassifier
from scipy.stats import randint as sp_randint
RF_CLF = RandomForestClassifier(n_estimators=20)
param_dist = {"max_depth": [3, None],
              "max_features": sp_randint(1, 11),
              "min_samples_split": sp_randint(2, 11),
              "min_samples_leaf": sp_randint(1, 11),
              # "bootstrap": [True, False],
              "criterion": ["gini", "entropy"]}

samples = 8  # number of random samples
randomCV = RandomizedSearchCV(RF_CLF, param_distributions=param_dist, n_iter=samples,cv=3)
randomCV.fit(X_train, y_train)
print(randomCV.best_params_)
```

Majority Voting Classifier - Ensemble Approach (GB,RF,SVM)

```
from sklearn.ensemble import VotingClassifier

clf1 = GradientBoostingClassifier()
clf2 = RandomForestClassifier(bootstrap = False, criterion = 'entropy', max_depth =  None, max_features = 5, min_samples_leaf = 4, min_samples_split = 6)
clf3 = svm.SVC(C=20, probability=True)
eclf1 = VotingClassifier(estimators=[('gbc', clf1), ('rf', clf2), ('svc', clf3)], voting='soft')
eclf1.fit(X_train, y_train)
#predictions = np.argmax(eclf1.predict_proba(X_test),axis=-1)
predictions = eclf1.predict(X_test)
```

Figure 10: Ensemble Classifier



Long Short Term Memory (LSTM)

```
from keras.preprocessing.text import Tokenizer
from keras_preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Dense, Embedding, LSTM, SpatialDropout1D
from sklearn.model_selection import train_test_split
from keras.utils.np_utils import to_categorical
from keras.callbacks import EarlyStopping
from keras.layers import Dropout
```

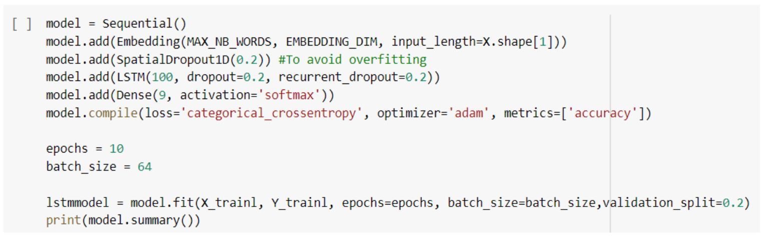
```
X = tokenizer.texts_to_sequences(finaldf_train['TEXT'].values)
#X = mean_embedded_text
X = pad_sequences(X, maxlen=MAX_SEQUENCE_LENGTH)
print('Shape of data tensor:', X.shape)
```

```
Shape of data tensor: (3321, 300)
```

```
Y = pd.get_dummies(finaldf_train['Class']).values
print('Shape of label tensor:', Y.shape)
```

```
Shape of label tensor: (3321, 9)
```

Figure 11: Import libraries and tensor data format for LSTM

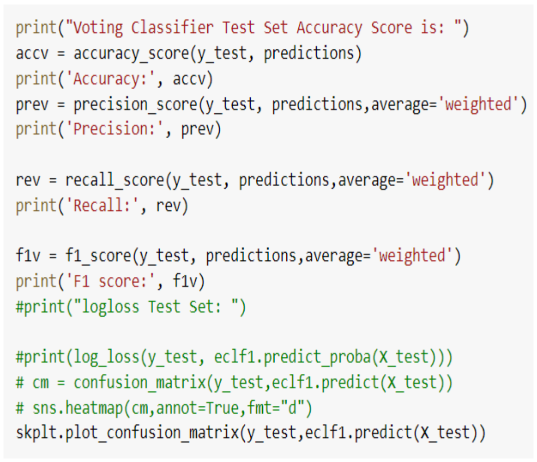


```
model = Sequential()
model.add(Embedding(MAX_NB_WORDS, EMBEDDING_DIM, input_length=X.shape[1]))
model.add(SpatialDropout1D(0.2)) #To avoid overfitting
model.add(LSTM(100, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(9, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

epochs = 10
batch_size = 64

lstmmodel = model.fit(X_trainl, Y_trainl, epochs=epochs, batch_size=batch_size,validation_split=0.2)
print(model.summary())
```

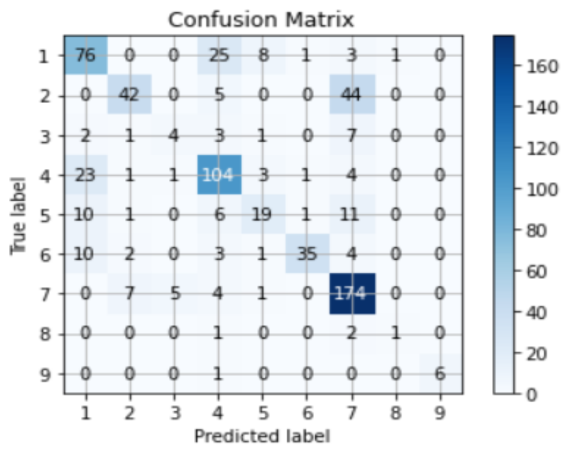
Figure 12: LSTM Model

9 Model Evaluation

The models which are implemented are evaluated using classification metrics such as Confusion Matrix, Accuracy, Precision, Recall, F1 Score and Log loss. As shown in figure 13 and 14, it was observed that the Voting Ensemble classifier and Random Forest gave the highest accuracy but the Ensemble has the miminal log loss and proves as best performed model.Whereas KNN and LSTM attained the least accuracy and KNN with the highest logloss, hence it is considered as a least performed model.

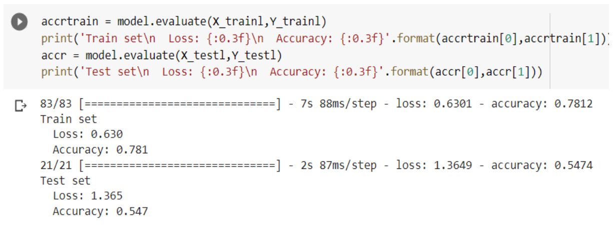


```
print("Voting Classifier Test Set Accuracy Score is: ")
accv = accuracy_score(y_test, predictions)
print('Accuracy:', accv)
prev = precision_score(y_test, predictions,average='weighted')
print('Precision:', prev)

rev = recall_score(y_test, predictions,average='weighted')
print('Recall:', rev)

f1v = f1_score(y_test, predictions,average='weighted')
print('F1 score:', f1v)
#print("logloss Test Set: ")

#print(log_loss(y_test, eclf1.predict_proba(X_test)))
# cm = confusion_matrix(y_test,eclf1.predict(X_test))
# sns.heatmap(cm,annot=True,fmt="d")
skplt.plot_confusion_matrix(y_test,eclf1.predict(X_test))
```



	1	2	3	4	5	6	7	8	9
1	76	0	0	25	8	1	3	1	0
2	0	42	0	5	0	0	44	0	0
3	2	1	4	3	1	0	7	0	0
4	23	1	1	104	3	1	4	0	0
5	10	1	0	6	19	1	11	0	0
6	10	2	0	3	1	35	4	0	0
7	0	7	5	4	1	0	174	0	0
8	0	0	0	1	0	0	2	1	0
9	0	0	0	1	0	0	0	0	6

Figure 13: Evaluation of Ensemble Classifier



```
accrtrain = model.evaluate(X_trainl,Y_trainl)
print('Train set\n  Loss: {:0.3f}\n  Accuracy: {:0.3f}'.format(accrtrain[0],accrtrain[1]))
accr = model.evaluate(X_testl,Y_testl)
print('Test set\n  Loss: {:0.3f}\n  Accuracy: {:0.3f}'.format(accr[0],accr[1]))
```

```
83/83 [==============================] - 7s 88ms/step - loss: 0.6301 - accuracy: 0.7812
Train set
  Loss: 0.630
  Accuracy: 0.781
21/21 [==============================] - 2s 87ms/step - loss: 1.3649 - accuracy: 0.5474
Test set
  Loss: 1.365
  Accuracy: 0.547
```

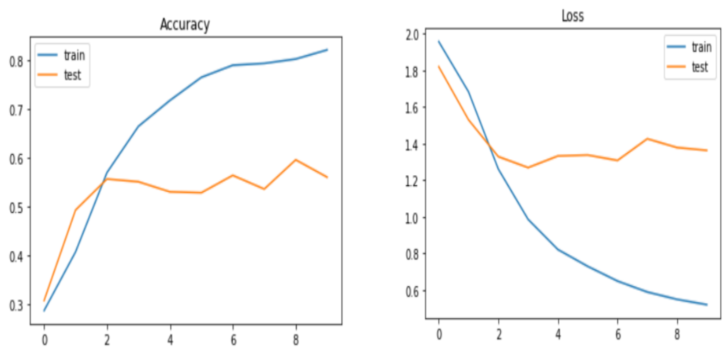
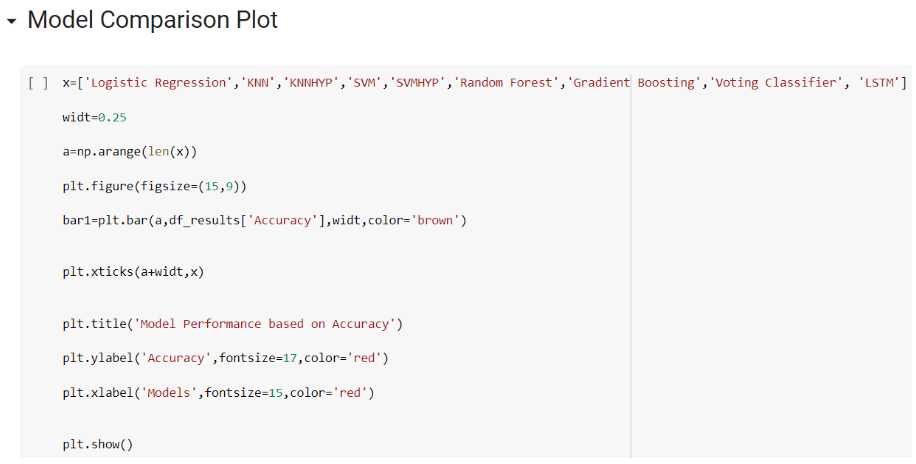


Figure 14: Evaluation of LSTM

10 Resuts

By comparing the model performance as shown in the figure 15, it is evident Ensemble classifier attained better accuracy with least mispredicted classes.



Model Comparison Plot

```
x=['Logistic Regression','KNN','KNNHYP','SVM','SVMHYP','Random Forest','Gradient Boosting','Voting Classifier', 'LSTM']
widt=0.25
a=np.arange(len(x))
plt.figure(figsize=(15,9))
bar1=plt.bar(a,df_results['Accuracy'],widt,color='brown')

plt.xticks(a+widt,x)

plt.title('Model Performance based on Accuracy')
plt.ylabel('Accuracy',fontsize=17,color='red')
plt.xlabel('Models',fontsize=15,color='red')

plt.show()
```

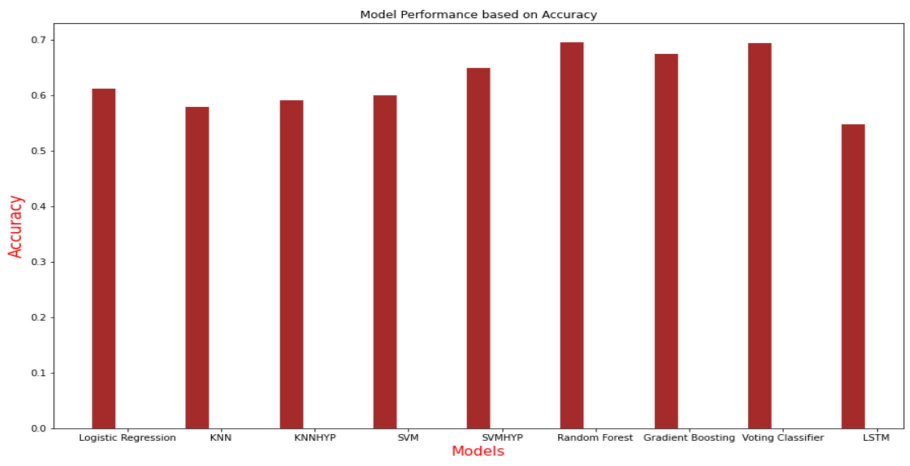


Figure 15: Model Comparison