Configuration Manual

MSc Research Project
Programme Name

Atul Vasant Lambhate
Student ID: x20203624

School of Computing
National College of Ireland

Supervisor: Dr. Abubakr Siddig

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name	Atul Vasant Lambhate
Student ID	x20203624
Programme	MSc. Data Analytics
Year:	2021/2022
Module:	MSc. Research Project
Supervisor:	Dr. Abubakr Siddig
Submission Due Date:	19th September 2022
Project Title:	Sentiment Analysis of Spam Reviews Using Bert-Large with SoftMax Classifier
Word Count:	962
Page Count:	15

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	Atul Vasant Lambhate
Date	19th September 2022

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

Atul Vasant Lambhate
Student ID: x20203624
19th September 2022

1 Introduction

This Configuration Manual lists together all prerequisites needed to duplicate the studies and its effects on a specific setting. A glimpse of the source for Data Importing & EDA (Exploratory Data Analysis) and after that Data Pre-processing while taking into consideration about Bert tokenizer, all the created algorithms, and Evaluations is also supplied, together with the necessary hardware components as well as Software applications. The report is organized as follows, with details relating environment configuration provided in Section 2.
Information about data gathering is detailed in Section 3. Data pre-processing including EDA are included in Section 4's information extraction section. In section 5, the Bert tokenizer is described. Insights on Feature Selection are provided in Section 6. Data scalability and train-test splits are covered in Section 7 for both the purpose of both training and testing models. Details well about models that were created and tested are provided in Section 8. How the results are calculated and shown is described in Section 9.

2 System Requirements

The specific needs for hardware as well as software to put the research into use are detailed in this section.

2.1 Hardware Requirements

The necessary hardware specs are shown in Figure 1 below. MacOs M1 Chip, macOS 10.15.x (Catalilna) operating system, 8GB RAM, 256GB Storage, 24'' Display.

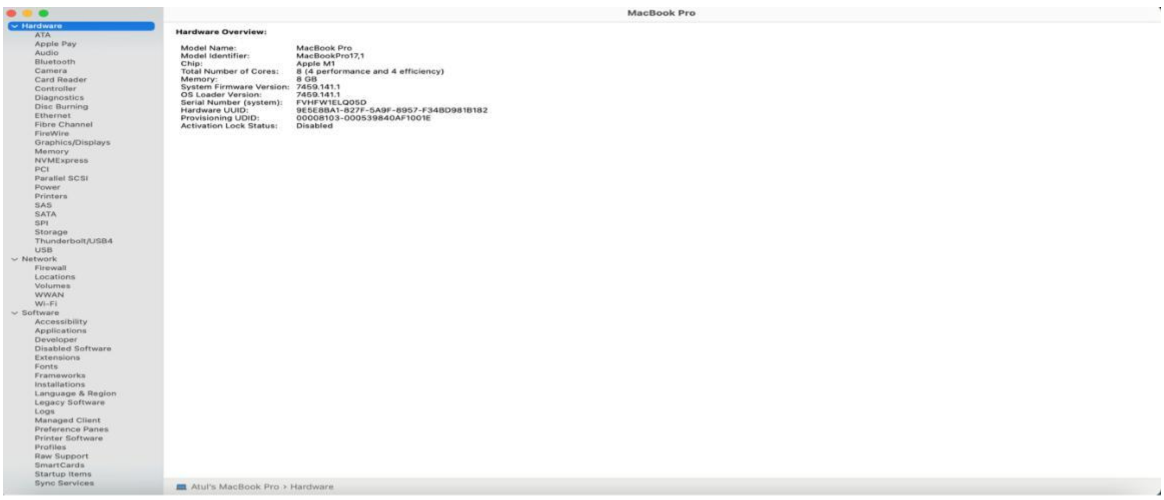


Figure 1: Hardware Requirements

2.2 Software Requirements

- Anaconda 3 for MacOs (Version 4.8.0)
- Jupyter Notebook (Version 6.0.3)
- Python (Version 3.7.6)

2.3 Code Execution

The code can be run in jupyter notebook and also in Google Collab. The jupyter notebook comes with Anaconda 3, run the jupyter notebook from startup. This will open jupyter notebook in web browser. The web browser will show the folder structure of the system, move to the folder where the code file is located. Open the code file from the folder and to run the code, go to Kernel menu and run all cells.
Similarly, Uploading the dataset on GoogleDrive and connecting it with the Google Collab can make the code run after installing some packages described in figure 2.

3 Data Collection

The information came from a publicly accessible Kaggle source, https://www.kaggle.com/datasets/eswarchandt/amusic-reviews is the link of the dataset. The data contains 10261 data points for user reviews including 9 unique features

4 Data Exploration

Figure 2 includes a list of every Python library necessary to complete the project.

```
# importing the necessary packages
import pandas as pd
import numpy as np
import nltk
nltk.download('punkt')
nltk.download('stopwords')
import matplotlib.pyplot as plt
from nltk.corpus import stopwords
from sklearn.preprocessing import StandardScaler
import string
from datetime import datetime
from tqdm import tqdm
import warnings
warnings.filterwarnings('ignore')
from sklearn.decomposition import PCA
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.naive_bayes import GaussianNB, CategoricalNB
import tensorflow as tf
from tensorflow.keras.models import Sequential
import tensorflow_hub as hub
from tensorflow.keras import layers
import bert, transformers
from bert import tokenization
from transformers import BertTokenizer
from sklearn.metrics import accuracy_score
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
```

Figure 2: Necessary Python libraries

The Figure 3 represents the block of code to check data information and the total number of missing values for each feature column.

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10261 entries, 0 to 10260
Data columns (total 9 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   reviewerID      10261 non-null  object
 1   asin            10261 non-null  object
 2   reviewerName    10234 non-null  object
 3   helpful         10261 non-null  object
 4   reviewText      10254 non-null  object
 5   overall         10261 non-null  float64
 6   summary         10261 non-null  object
 7   unixReviewTime  10261 non-null  int64
 8   reviewTime      10261 non-null  object
dtypes: float64(1), int64(1), object(7)
memory usage: 721.6+ KB
```

Missing Value

```
data.isnull().sum()
```

```
reviewerID         0
asin               0
reviewerName      27
helpful            0
reviewText         7
overall            0
summary            0
unixReviewTime     0
reviewTime         0
dtype: int64
```

Figure 3: EDA for Checking Data Information and Missing Values

As seen in Figure 4, The review text analysis is done in code block for word, uppercase and special character count.

```
data['WordCount'] = [len(title.split()) for title in data['reviewText']]
data['UppercaseCount'] = [sum(char.isupper() for char in title) for title in data['reviewText']]
data['SpecialCount'] = [sum(char in string.punctuation for char in title) for title in data['reviewText']]
```

Figure 4: EDA for Review text

In figure 5, the sentiment is set based on the user ratings.

```
data['overall'].value_counts()
```

```
5.0     6932
4.0     2083
3.0      772
2.0      250
1.0      217
Name: overall, dtype: int64
```

```
range = {"low": 4, "high": 5}
data["Sentiment"]=0
data["Sentiment"].loc[data["overall"] <= range["low"]] = 0
data["Sentiment"].loc[data["overall"] >= range["high"]] = 1
data.head()
# 0- negative, 1- positive
```

Figure 5: Sentiment Scores

The Figure 6, illustrate the code to de-contract the words and clear the punctuations.

```
import re

def decontracted(phrase):
    # This function decontract words like it's to it is.

    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase

def cleanpunc(sentence): #function to clean the word of any punctuation or special characters
    cleaned = re.sub(r'[?|!|\'|"|#]',r'',sentence)
    cleaned = re.sub(r'[.|,|)|(|\|/]',r' ',cleaned)
    return  cleaned
```

Figure 6: Cleaning words

The Figure 7, illustrate the code to clean the review text, each word, de-contracted and cleaned.

```
# Here we are cleaning the data using functions define above, removing stopword and reducing words to there root words.

i=0
str1=' '
final_string=[]
all_positive_words=[] # store words from +ve reviews here
all_negative_words=[] # store words from -ve reviews here.
s=''
stop = set(stopwords.words('english')) #set of stopwords
sno = nltk.stem.SnowballStemmer('english') #initialising the snowball stemmer
for sent in tqdm(data['reviewText']):
    filtered_sentence=[]

    sent = decontracted(sent)
    for w in sent.split():
        for cleaned_words in cleanpunc(w).split():
            if((cleaned_words.isalpha()) & (len(cleaned_words)>2)):
                if(cleaned_words.lower() not in stop):
                    s=(sno.stem(cleaned_words.lower())).encode('utf8')
                    filtered_sentence.append(s)
                    if (data['Sentiment'].values)[i] == 1:
                            all_positive_words.append(s) #list of all words used to describe positive reviews
                    if (data['Sentiment'].values)[i] == 0:
                            all_negative_words.append(s) #list of all words used to describe negative reviews reviews
                else:
                    continue
            else:
                continue

    str1 = b" ".join(filtered_sentence) #final string of cleaned words
    final_string.append(str1)
    i+=1

data["clean_review"] = final_string
data['clean_review']=data['clean_review'].str.decode("utf-8")
```

Figure 7: Sentiment Scores

Figures 8 and 9 show the code used to create word clouds for both positive and also negative assessments.

```
from wordcloud import WordCloud

print("Wordcloud of words present in positive class : \n")
wordcloud = WordCloud(width = 800, height = 800,background_color ='white', min_font_size = 10).generate(b' '.join(all_
ords).decode("utf-8"))
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)
plt.show()
```

```
Wordcloud of words present in positive class :
```

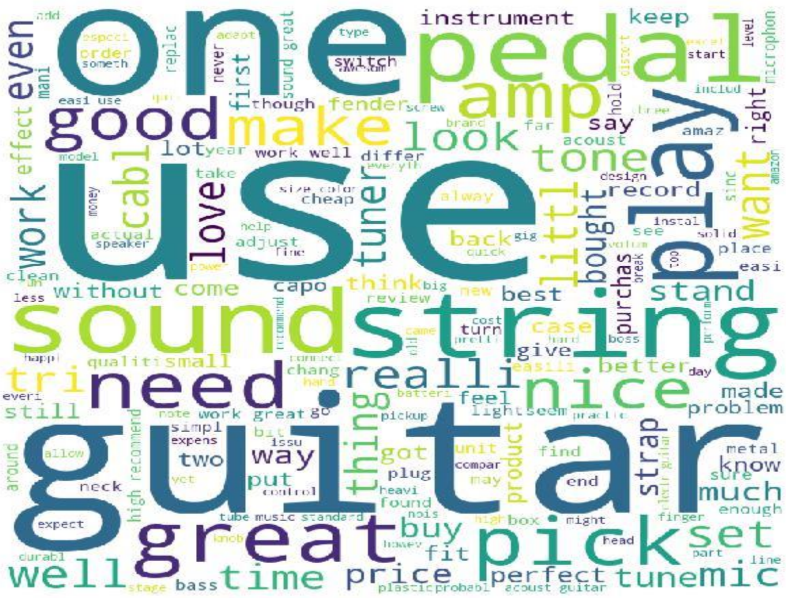


Figure 8: Word cloud for positive feedback

```
print("Wordcloud of words present in negative class : \n")
wordcloud = WordCloud(width = 800, height = 800,background_color ='white', min_font_size = 10).generate(b' '.
ords).decode("utf-8"))
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)
plt.show()
```

```
Wordcloud of words present in negative class :
```

Figure 9: Word cloud of negative emotions

5 Bert Tokenizer

The Figure 10, illustrate the code to tokenize based in large bert tokenizer.

```
tokenizer = BertTokenizer.from_pretrained('bert-large-uncased')
```

```
tokenized_texts = [tokenizer.tokenize(com) for com in X.clean_review]
tokenized_texts = [sent[:len(tokenized_texts)] for sent in tokenized_texts]
bigBertData = [tokenizer.convert_tokens_to_ids(com) for com in tokenized_texts]
bigBertData = tf.keras.preprocessing.sequence.pad_sequences(bigBertData, maxlen=len(tokenized_texts)+2, truncating='post', paddi
ng='post')
bigBertData.shape
```

```
(10254, 10256)
```

Figure 10: Large Bert tokenizer

The Figure 11, illustrate the code to tokenize based in small bert tokenizer.

```
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
```

```
tokenized_texts = [tokenizer.tokenize(com) for com in X.clean_review]
tokenized_texts = [sent[:len(tokenized_texts)] for sent in tokenized_texts]
smallBertData = [tokenizer.convert_tokens_to_ids(com) for com in tokenized_texts]
smallBertData = tf.keras.preprocessing.sequence.pad_sequences(smallBertData, maxlen=len(tokenized_texts)+2, truncating='post', p
adding='post')
smallBertData.shape
```

```
(10254, 10256)
```

Figure 11: SmallBert tokenizer

6 Feature Selection

PCA is utilized for feature extraction. It is a technique for decreasing the size of a metric by taking it from having many columns to few. After that the data is scaled.
Figure 12 and 13 shows the process for both Large Bert data and Small Bert data.

```
bigBertData = pca.fit_transform(bigBertData)
bigBertData.shape
```

```
(10254, 13)
```

```
scaler = StandardScaler()
print(scaler.fit(bigBertData))
print(scaler.mean_)
```

```
StandardScaler()
[ 3.50750614e-12 -3.17604703e-12 -5.30405531e-14  5.11247071e-13
  3.37335257e-13 -7.86250808e-13  4.83263300e-13  3.03519687e-13
 -1.75641397e-13  1.15048330e-12  5.68190273e-13  9.54242125e-13
  3.48311458e-13]
```

```
bigBertData = scaler.transform(bigBertData)
bigBertData
```

```
array([[-0.5836619 ,  0.2735953 ,  0.23317427, ...,  0.35248407,
         0.04163063,  0.3212214 ],
       [ 0.24529684, -1.16830239,  0.68279888, ...,  0.54813942,
         0.94430909,  1.30654113],
```

Figure 12: Feature selection and Scaling for Large Bert Data

```
scaler = StandardScaler()
print(scaler.fit(smallBertData))
print(scaler.mean_)
```

```
StandardScaler()
[5753.54759118 7104.6217086  6685.02837917 ...    0.            0.
    0.        ]
```

```
smallBertData = scaler.transform(smallBertData)
smallBertData
```

```
array([[-0.60865082, -0.41715287, -0.00773906, ...,  0.        ,
         0.        ,  0.        ],
       [-0.292731  , -0.07083544,  0.21053346, ...,  0.        ,
         0.        ,  0.        ],
       [ 2.60985672, -0.60569676, -0.55380711, ...,  0.        ,
         0.        ,  0.        ],
       ...,
       [ 1.23350672, -0.25183758, -0.2352932 , ...,  0.        ,
         0.        ,  0.        ],
       [-0.62224608, -0.75773856,  0.82296501, ...,  0.        ,
         0.        ,  0.        ],
       [-0.10018815, -0.67749428, -0.46655997, ...,  0.        ,
         0.        ,  0.        ]])
```

```
smallBertData = pca.fit_transform(smallBertData)
smallBertData.shape
```

Figure 13: Feature selection and Scaling for Small Bert Data

Figure 14 and 15 below, shows the implementation of data splitting. The test dataset contains 1000 records, and the remaining are in training set.

```
trainX= bigBertData[1000:]
Y_train=  y[1000:]
print(trainX.shape, Y_train.shape)
```

```
(9254, 13) (9254,)
```

```
testX= bigBertData[:1000]
Y_test = y[:1000]
print(testX.shape, Y_test.shape)
```

```
(1000, 13) (1000,)
```

Figure 14: Data splitting Large Bert Data

```
trainX= smallBertData[:1000]
Y_train=  y[:1000]
print(trainX.shape, Y_train.shape)
```

```
(1000, 13) (1000,)
```

```
testX= smallBertData[1000:]
Y_test = y[1000:]
print(testX.shape, Y_test.shape)
```

```
(9254, 13) (9254,)
```

```
trainX
```

```
array([[-0.97981749,  1.47508436,  1.82594611, ..., -0.15531164,
        -0.87954037, -0.53453805],
       [-0.86845673,  1.17414884,  1.16539114, ...,  0.16627586,
         1.3741936 ,  1.07691461],
       [-0.8345458 ,  0.99381646,  0.85123719, ...,  0.26026831,
         2.29384147,  1.71315822],
       ...,
       [-0.97829328,  1.48004441,  1.82062968, ..., -0.13074199,
        -0.88546981, -0.58441685],
       [ 0.16205453, -2.18241242, -5.02144671, ...,  0.62305087,
         0.56290718, -1.37155017],
       [-0.99237572,  1.5053561 ,  1.89550149, ..., -0.18632018,
        -1.173987  , -0.78128917]])
```

Figure 15: Data splitting Small Bert Data

7 Machine Learning Models

7.1 Large Bert Models

7.1.1 SVM

SVM

```
[ ] model = SVC(kernel= 'rbf', C= 10, gamma= 'scale')
```

```
[ ] model.fit(trainX, Y_train)
```

```
SVC(C=10)
```

```
[ ] test_pred = model.predict(testX)
```

```
[ ] bigbertsvmacc = accuracy_score(test_pred,Y_test)*100
    print("Accuracy: %.2f%%" % bigbertsvmacc)
```

```
Accuracy: 66.10%
```

```
[ ] bigBertScore.append(["SVM",bigbertsvmacc])
```

Figure 16: Implementation of Large Bert SVM

7.1.2 Naïve Bayes

```
param_grid_nb = {
    'var_smoothing': np.logspace(0,-9, num=100)
}
model = GridSearchCV(estimator=GaussianNB(), param_grid=param_grid_nb, verbose=1, cv=10, n_jobs=-1)
```

```
model.fit(trainX, Y_train)
```

```
Fitting 10 folds for each of 100 candidates, totalling 1000 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    4.2s
[Parallel(n_jobs=-1)]: Done 912 tasks      | elapsed:    6.5s
[Parallel(n_jobs=-1)]: Done 1000 out of 1000 | elapsed:    7.2s finished
```

```
GridSearchCV(cv=10, estimator=GaussianNB(), n_jobs=-1,
             param_grid={'var_smoothing': array([1.00000000e+00, 8.11130831e-01, 6.57933225e-01, 5.33669923e-01,
       4.32876128e-01, 3.51119173e-01, 2.84803587e-01, 2.31012970e-01,
       1.87381742e-01, 1.51991108e-01, 1.23284674e-01, 1.00000000e-01,
       8.11130831e-02, 6.57933225e-02, 5.33669923e-02, 4.32876128e-02,
       3.51119173e-02, 2.848035...
       1.23284674e-07, 1.00000000e-07, 8.11130831e-08, 6.57933225e-08,
       5.33669923e-08, 4.32876128e-08, 3.51119173e-08, 2.84803587e-08,
       2.31012970e-08, 1.87381742e-08, 1.51991108e-08, 1.23284674e-08,
       1.00000000e-08, 8.11130831e-09, 6.57933225e-09, 5.33669923e-09,
       4.32876128e-09, 3.51119173e-09, 2.84803587e-09, 2.31012970e-09,
       1.87381742e-09, 1.51991108e-09, 1.23284674e-09, 1.00000000e-09])},
             verbose=1)
```

```
test_pred = model.predict(testX)
```

```
bigbertnbacc = accuracy_score(test_pred,Y_test)*100
print("Accuracy: %.2f%%" % bigbertnbacc)
```

```
Accuracy: 66.60%
```

```
bigBertScore.append(["Naive Bayes",bigbertnbacc])
```

Figure 17: Implementation of Large Bert Naïve Bayes

7.1.3 LSTM

```
trainX = np.reshape(trainX, (trainX.shape[0], trainX.shape[1], 1))
trainX.shape
```

```
(9254, 13, 1)
```

```
testX = np.reshape(testX, (testX.shape[0], testX.shape[1], 1))
testX.shape
```

```
(1000, 13, 1)
```

```
model = Sequential()
#(trainX.shape[1],1)model.add(tf.keras.layers.BatchNormalization(input_shape=(13,)))
model.add(layers.LSTM(units=256, input_shape=(trainX.shape[1],1), activation ='sigmoid'))
model.add(layers.Dense(64, activation ='sigmoid'))
model.add(layers.Dropout(0.1))
model.add(layers.Dense(1, activation ='sigmoid'))
model.compile(optimizer=tf.keras.optimizers.SGD(learning_rate=0.075), loss=tf.keras.losses.BinaryCrossentropy(), metrics=['accur
acy'])
```

```
checkpoint = ModelCheckpoint('../bigbert_model_best.h5', monitor='val_loss', verbose=1, save_best_only=True, mode='min')
```

```
bigberthistory= model.fit(trainX, Y_train, batch_size = 64,
                    validation_data = (testX, Y_test),
                    callbacks = [checkpoint])
```

```
145/145 [==============================] - ETA: 0s - loss: 0.6349 - accuracy: 0.6748
Epoch 00001: val_loss improved from inf to 0.63805, saving model to ../bigbert_model_best.h5
145/145 [==============================] - 5s 36ms/step - loss: 0.6349 - accuracy: 0.6748 - val_loss: 0.6380 - val_accuracy: 0.
6680
```

```
test_pred = np.round(model.predict(testX))
```

```
bigbertlstmacc = accuracy_score(test_pred,Y_test)*100
print("Accuracy: %.2f%%" % bigbertlstmacc)
```

```
Accuracy: 66.80%
```

```
bigBertScore.append(["LSTM",bigbertlstmacc])
```

Figure 18: Implementation of Large Bert LSTM

7.2 Small Bert Models

7.2.1 SVM

```
model = SVC(kernel= 'rbf', C= 10, gamma= 'auto')
```

```
model.fit(trainX, Y_train)
```

```
SVC(C=10, gamma='auto')
```

```
test_pred = model.predict(testX)
```

```
smallbertsvmacc = accuracy_score(test_pred,Y_test)*100
print("Accuracy: %.2f%%" % smallbertsvmacc)
```

```
Accuracy: 65.27%
```

```
smallBertScore.append(["SVM",smallbertsvmacc])
```

Figure 19: Implementation of Small Bert SVM

7.2.2 Naïve Bayes

```
param_grid_nb = {
    'var_smoothing': np.logspace(0,-9, num=100)
}
model = GridSearchCV(estimator=GaussianNB(), param_grid=param_grid_nb, verbose=1, cv=10, n_jobs=-1)
```

```
model.fit(trainX, Y_train)
```

```
Fitting 10 folds for each of 100 candidates, totalling 1000 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  56 tasks      | elapsed:    0.1s
[Parallel(n_jobs=-1)]: Done 1000 out of 1000 | elapsed:    1.0s finished
```

```
GridSearchCV(cv=10, estimator=GaussianNB(), n_jobs=-1,
             param_grid={'var_smoothing': array([1.00000000e+00, 8.11130831e-01, 6.57933225e-01, 5.33669923e-01,
       4.32876128e-01, 3.51119173e-01, 2.84803587e-01, 2.31012970e-01,
       1.87381742e-01, 1.51991108e-01, 1.23284674e-01, 1.00000000e-01,
       8.11130831e-02, 6.57933225e-02, 5.33669923e-02, 4.32876128e-02,
       3.51119173e-02, 2.848035...
       1.23284674e-07, 1.00000000e-07, 8.11130831e-08, 6.57933225e-08,
       5.33669923e-08, 4.32876128e-08, 3.51119173e-08, 2.84803587e-08,
       2.31012970e-08, 1.87381742e-08, 1.51991108e-08, 1.23284674e-08,
       1.00000000e-08, 8.11130831e-09, 6.57933225e-09, 5.33669923e-09,
       4.32876128e-09, 3.51119173e-09, 2.84803587e-09, 2.31012970e-09,
       1.87381742e-09, 1.51991108e-09, 1.23284674e-09, 1.00000000e-09])},
             verbose=1)
```

```
test_pred = model.predict(testX)
```

```
smallbertnbacc = accuracy_score(test_pred,Y_test)*100
print("Accuracy: %.2f%%" % smallbertnbacc)
```

```
Accuracy: 67.46%
```

```
smallBertScore.append(["Naive Bayes",smallbertnbacc])
```

Figure 20: Implementation of Small Bert Naïve Bayes

7.2.3 LSTM

```
trainX = np.reshape(trainX, (trainX.shape[0], trainX.shape[1], 1))
trainX.shape
```

```
(1000, 13, 1)
```

```
testX = np.reshape(testX, (testX.shape[0], testX.shape[1], 1))
testX.shape
```

```
(9254, 13, 1)
```

```
model = Sequential()
model.add(layers.LSTM(units=256, input_shape=(trainX.shape[1],1), activation ='softmax'))
model.add(layers.Dense(64, activation ='sigmoid'))
model.add(layers.Dropout(0.1))
model.add(layers.Dense(1, activation ='sigmoid'))
model.compile(optimizer=tf.keras.optimizers.SGD(learning_rate=0.075), loss=tf.keras.losses.BinaryCrossentropy(), metrics=['accur
acy'])
```

```
checkpoint = ModelCheckpoint('../smallbert_model_best.h5', monitor='val_loss', verbose=1, save_best_only=True, mode='man')
smallberthistory= model.fit(trainX, Y_train, validation_data=(testX, Y_test), epochs=10, verbose=True, callbacks=checkpoint)
```

```
WARNING:tensorflow:ModelCheckpoint mode man is unknown, fallback to auto mode.
Epoch 1/10
31/32 [============================>.] - ETA: 0s - loss: 0.6479 - accuracy: 0.6633
Epoch 00001: val_loss improved from inf to 0.63123, saving model to ../smallbert_model_best.h5
32/32 [==============================] - 5s 169ms/step - loss: 0.6478 - accuracy: 0.6630 - val_loss: 0.6312 - val_accuracy: 0.6
769
Epoch 2/10
31/32 [============================>.] - ETA: 0s - loss: 0.6376 - accuracy: 0.6593
Epoch 00002: val_loss did not improve from 0.63123
32/32 [==============================] - 5s 144ms/step - loss: 0.6371 - accuracy: 0.6600 - val_loss: 0.6323 - val_accuracy: 0.6
769
Epoch 3/10
31/32 [============================>.] - ETA: 0s - loss: 0.6479 - accuracy: 0.6643
```

```
test_pred = np.round(model.predict(testX))
```

```
smallbertlstmacc = accuracy_score(test_pred,Y_test)*100
print("Accuracy: %.2f%%" % smallbertlstmacc)
```

```
Accuracy: 67.69%
```

```
smallBertScore.append(["LSTM",smallbertlstmacc])
```

Figure 21: Implementation of Small Bert LSTM

8 Model result

This section explains the performance of the models.

8.1 Model Scores

```
bigBertScore = pd.DataFrame(bigBertScore)
bigBertScore.columns=['Models', 'Score']
bigBertScore
```

	Models	Score
0	SVM	66.0
1	Naive Bayes	66.6
2	LSTM	66.8

```
smallBertScore = pd.DataFrame(smallBertScore)
smallBertScore.columns=['Models', 'Score']
smallBertScore
```

	Models	Score
0	SVM	65.269073
1	Naive Bayes	67.462719
2	LSTM	67.689648

Figure 22: Model Performance

8.2 Large Bert Accuracy

```
# summarize history for accuracy
plt.bar(bigBertScore['Models'], bigBertScore['Score'])
plt.title('Big Bert Model Accuracy')
plt.ylabel('Score')
plt.xlabel('Models')
plt.show()
```

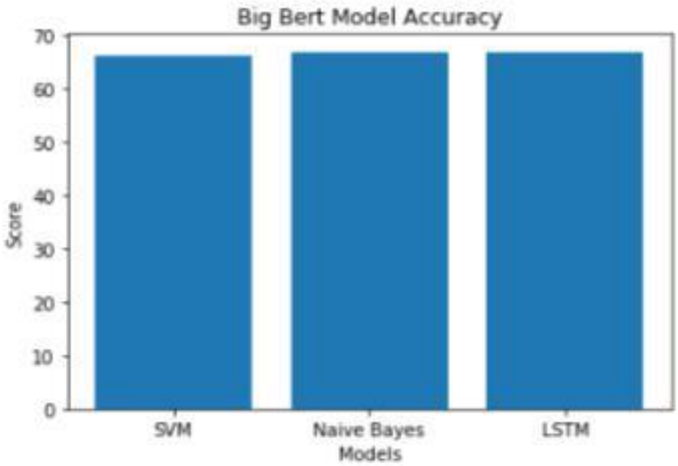


Figure 21: Large Bert Accuracy

8.3 Small Bert Accuracy

```
# summarize history for accuracy
plt.bar(smallBertScore['Models'], smallBertScore['Score'])
plt.title('Small Bert Model Accuracy')
plt.ylabel('Score')
plt.xlabel('Models')
plt.show()
```

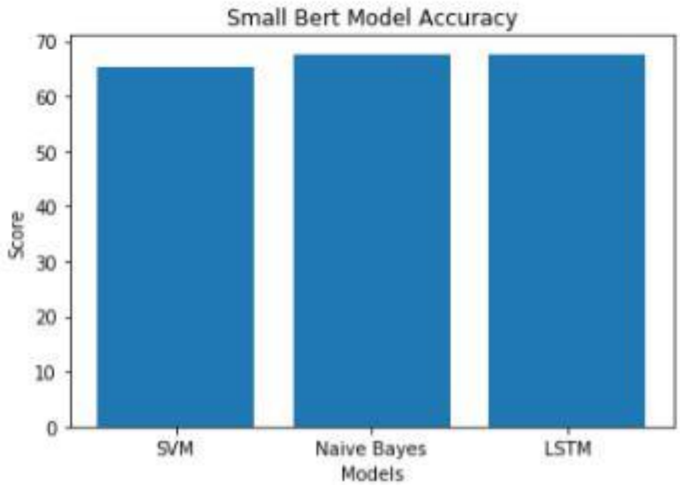
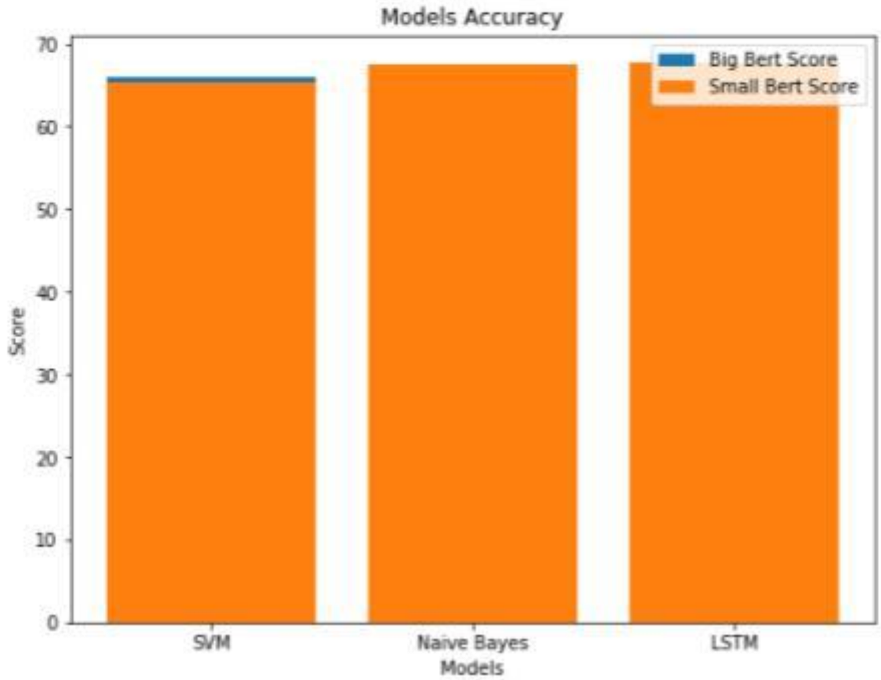


Figure 22: Small Bert accuracy

8.4 Model Scores

```
# summarize history for accuracy
plt.figure(figsize=(8,6))
plt.bar(bigBertScore['Models'], bigBertScore['Score'], label="Big Bert Score")
plt.bar(smallBertScore['Models'], smallBertScore['Score'], label="Small Bert Score")
plt.title('Models Accuracy')
plt.ylabel('Score')
plt.xlabel('Models')
plt.legend()
plt.show()
```



References

Data Source:
https://www.kaggle.com/datasets/eswarchandt/amusic-reviews

Code Reference for De-Contracting Words and Cleaning Punctuations.
https://stackoverflow.com/a/47091490/4084039

https://github.com/google-research/bert

https://colah.github.io/posts/2015-08-Understanding-LSTMs/