Configuration Manual

MSc Research Project
Data Analytics

Yash Rajendra Pilankar
Student ID: x19216858

School of Computing
National College of Ireland

Supervisor: Dr. Rejwanul Haque

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Yash Rajendra Pilankar
Student ID:	x19216858
Programme:	Data Analytics
Year:	2021
Module:	MSc Research Project
Supervisor:	Dr. Rejwanul Haque
Submission Due Date:	31/01/2022
Project Title:	Configuration Manual
Word Count:	1242
Page Count:	12

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:	Yash Rajendra Pilankar
Date:	30th January 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

Yash Rajendra Pilankar
x19216858

1 Introduction

For the given research extensive analysis and various experiments were performed. This configuration manual shows detailed specification about the hardware and software requirements along with the steps taken from data gathering to model implementation that can help to replicate the given project.

2 System Specification and Requirements

The section shows details about the Hardware and Software configurations which was required and utilized for implementation of the research.

2.1 Hardware Specification

Table 2 shows about the Hardware configuration required for the experimentation. For the given research local machine was utilized as different investigation were carried out.

Hardware	Dell inspiron 5000
Processor	Intel(R) Core(TM) i5-5200U CPU @ 2.20GHz
RAM	12.0 GB DDR-4
System Type	64-bit operating system, x64-based processor
Graphics / Graphics Card	2GB - Nvidia 920M Chipset

Table 1: Hardware Specifications

2.2 Software specification and Requirements

The required software and libraries were installed on Windows 10 Pro Operating System. The program was implemented on Jupyter Notebook 6.1.4 version. Following Table shows the libraries along with the version taken into consideration. These libraries were helpful from data gathering using Tweepy, cleaning data using SpaCy, Model building with tensorflow and SKlearn, and evaluation based on Matplotlib and Seaboard.Other supporting libraries like pandas, numpy, regular expression, string were used throughout for storing, access and manipulation of data which was fetched from Twitter.

Libraries	Version
Python	3.8.5
Tweepy	3.10.0
Tensorflow	2.7.0
Tensorflow Hub	0.12.0
Sci-kit Learn	0.24.2
SpaCy	3.2.0
Seaborn	0.11.0
Imblearn	0.8.0
nltk	3.5
Pandas	1.1.3
Numpy	1.19.2
Malplotlib	3.3.2
Seaborn	0.11.0

Table 2: Libraries Required

3 Data Preparation

As the dataset was not readily available data was gathered for the given research. The below given are the steps followed to gain data from Twitter.

Step 1 - Setup Twitter API: Since, data was fetched from Twitter therefore to get authorization token Twitter Developer Account [1] needs to be setup. Further, based on the approval from Twitter a Twitter application can be created where it will provide authentication keys [2] which was required to access Tweets. The figure 1 shows dashboard

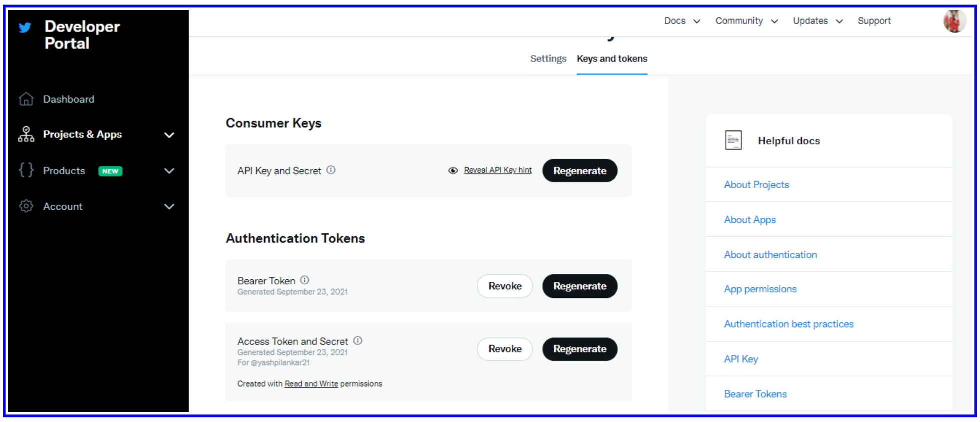


Figure 1: Twitter Developer Account Dashboard

where the keys can be accessed.

Step 2: Data Collection using Tweepy: The data was collected using Tweepy [3] library that can be used to access Twitter API to fetch the data from Twitter. Before fetching the Tweet, the API needs to be setup with help of authentication tokens as seen in figure 2. The data was gathered in 2 ways, first the tweets were fetched from user

[1] https://developer.twitter.com/
[2] https://developer.twitter.com/en/docs/authentication/overview
[3] https://docs.tweepy.org/en/stable/api.html

```python
import tweepy #https://github.com/tweepy/tweepy
import csv
import pandas as pd

#Twitter API credentials
consumer_key = "YOUR CONSUMER KEY"
consumer_secret = "YOUR SECRET KEY"
access_key = "YOUR ACCESS KEY"
access_secret = "YOUR ACCESS SECRET KEY"
auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_key, access_secret)
api = tweepy.API(auth)
```

Figure 2: Setup API with Authentication token

accounts which are associated with Human Rights. As seen in the figure 3 , a function was created to fetch Tweet of certain accounts like Amnesty, Human Rights Watch and activist like Nadia Murad and Malala which are current active Human Rights activist and NGO's that stands for equal Human Rights for all people around the world.

```python
def get_all_tweets(screen_name):
    #Twitter only allows access to a users most recent 3240 tweets with this method

    #authorize twitter, initialize tweepy
    auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
    auth.set_access_token(access_key, access_secret)
    api = tweepy.API(auth)

    #initialize a list to hold all the tweepy Tweets
    alltweets = []

    #make initial request for most recent tweets (200 is the maximum allowed count)
    new_tweets = api.user_timeline(screen_name = screen_name,count=200, include_rts = False, exclude_replies = True, tweet_mo

    #save most recent tweets
    alltweets.extend(new_tweets)
    #print(type(new_tweets))
    #save the id of the oldest tweet less one
    oldest = alltweets[-1].id - 1
    df = pd.DataFrame(alltweets)
    #print(alltweets)
    #keep grabbing tweets until there are no tweets left to grab
    while len(new_tweets) > 0:
        print(f"getting tweets before {oldest}")

        #all subsiquent requests use the max_id param to prevent duplicates
        new_tweets = api.user_timeline(screen_name = screen_name,count=200,max_id=oldest, tweet_mode="extended",include_rts =


        #save most recent tweets
        alltweets.extend(new_tweets)

        #update the id of the oldest tweet less one
        oldest = alltweets[-1].id - 1

        print(f"...{len(alltweets)} tweets downloaded so far")


#     #transform the tweepy tweets into a 2D array that will populate the csv
    outtweets = [[tweet.id_str, tweet.created_at, tweet.full_text, tweet.entities, tweet.user.screen_name] for tweet in alltw

    #print(outtweets)
    #write the csv
    tweetdf = pd.DataFrame(outtweets, columns =['Tweet_ID', 'Created_at', 'tweet_text','tweet_entities','tweet_username'])
    datatoexcel = pd.ExcelWriter(screen_name+'_tweets.xlsx')

    # write DataFrame to excel
    tweetdf.to_excel(datatoexcel)

    # save the excel
    datatoexcel.save()
#     with open(f'new_{screen_name}_tweets.csv', 'w', encoding="utf-8") as f:
#         writer = csv.writer(f)
#         writer.writerow(["id","created_at","text"])
#         writer.writerows(outtweets)

#     pass


if __name__ == '__main__':
    #get_all_tweets('amnesty')
    usernamelist = ['amnesty','Refugees', 'UNESCO', 'UN', 'UNHumanRights', 'hrw', 'un_hrc', 'humanrightsirl', 'irishcentrehr',
    for user in usernamelist:
        get_all_tweets(user)
```

Figure 3: Method 1 for Fetching Tweets of Users

Further, more data was gathered by shooting different situational queries like "attack on civilians", "ban on education", "child abuse" etc. The figure 4, shows the same functionality as discussed above but various filters were applied like exclude retweets, get recent tweets, and language set to English.Only constrain was Twitter allows only 1000 Tweet per search call therefore, multiple calls were required to get relevant data.

Step 3: Data Annotation: The given data was annotated manually based on

```
#query to be sent
query = 'attack on civilians'

#Fetch Tweets Based on Query using search method
fetched_tweets = tweepy.Cursor(api.search,q=query + ' -filter:retweets', result_type='recent',
                               timeout=999999, lang='en',tweet_mode="extended" ).items(1000)

#Get relevant columns from the fetched Tweets data
outtweets = [[tweet.id_str, tweet.created_at, tweet.full_text, tweet.entities,
              tweet.user.screen_name] for tweet in fetched_tweets]
tweetdf = pd.DataFrame(outtweets, columns =['Tweet_ID', 'Created_at', 'tweet_text','tweet_entities','tweet_username'])

#Store in Excel

datatoexcel = pd.ExcelWriter(query+'_tweets.xlsx')
tweetdf.to_excel(datatoexcel)
datatoexcel.save()
```

Figure 4: Method 2 for Fetching Tweets of Users

the understanding. The Target Variable was set to yes if it was a factual tweet about Human Right Violation else target was set to no for all the excels fetched and were finally combined into single excel data which can be seen in figure 5.

Tweet_ID	Created_at	tweet_text	tweet_entities	tweet_username	target
1453697848084680704	2021-10-28 12:19:10	As the climate crisis gets worse, so do the threats to our rights. As this year's UN Climate Conference #COP26 takes place, it is imperative that all governments adopt new and improved emission reduction targets. They must take action now. https://t.co/WPMw2jS254	{'hashtags': [{'text': 'COP26', 'indices':	amnesty	yes
1453687004634533899	2021-10-28 11:36:04	When #G20 leaders met in 2020, 1.5 million people had died of #Covid19. Since then, another 3.5m lives have been lost, while many G20 members are sitting on millions of surplus doses. As @g20org leaders gather in Rome, they must ensure that their promises are matched by action ✊ https://t.co/OKDS7LrEN0	{'hashtags': [{'text': 'G20', 'indices': [5,	amnesty	no

Figure 5: Annotation of Tweets

Step 4: Data Cleaning and Pre-processing using SpaCy: The combine dataset was uploaded and accessed for cleaning and pre-processing which was performed using Natural Language Processing based SpaCy[4] Library. The follow figure 6 shows function create for cleaning tweets

```
nlp = spacy.load('en_core_web_sm')

def spacy_cleaner(text):
    #Encode text to UTF-8
    try:
        decoded = unidecode.unidecode(codecs.decode(text, 'unicode_escape'))
    except:
        decoded = unidecode.unidecode(text)
    #Handle Apostrophe
    apostrophe_handled = re.sub("’", "'", decoded)
    expanded = ' '.join([contraction_mapping[t] if t in contraction_mapping else t for t in apostrophe_handled.split(" ")])
    #parse the text
    parsed = nlp(expanded)
    #Create Tokens
    final_tokens = []
    #Remove whitespace, URLS, Numbers, Usertags
    for t in parsed:
        if t.is_punct or t.is_space or t.like_num or t.like_url or str(t).startswith('@'):
            pass
        else:
            #Lemmatize and Stemming
            if t.lemma_ == '-PRON-':
                final_tokens.append(str(t))
            else:
                sc_removed = re.sub("[^a-zA-Z]", '', str(t.lemma_))
                if len(sc_removed) > 1:
                    final_tokens.append(sc_removed)
    joined = ' '.join(final_tokens)
    spell_corrected = re.sub(r'(.)\1+', r'\1\1', joined)
    return spell_corrected
```

Figure 6: Cleaning and Pre-processing of Tweets

[4]https://spacy.io/api

4 Experimental Setup

This section will give detailed steps for setting up the parameters for different models.

4.1 Experiment 1 - Classification using Machine Learning

As the Dataset was imbalanced as seen in the figure 7 the ratio was of around 90% of tweets not about Human Rights Violation and only 10% tweets were about Human Rights Violation.

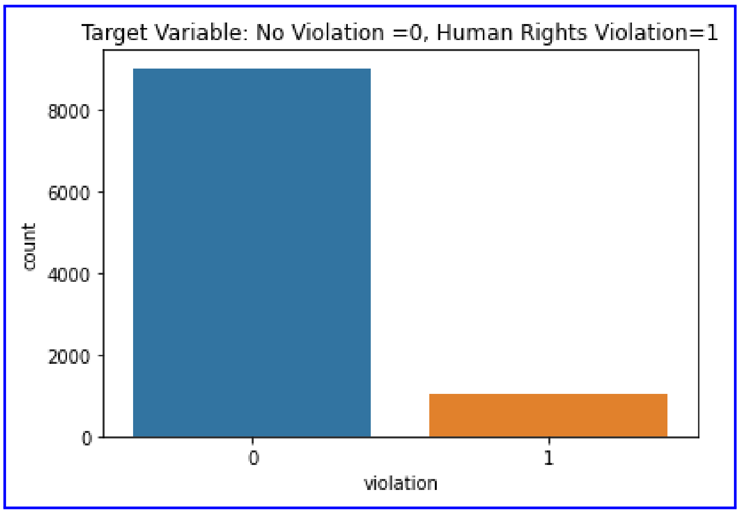


Figure 7: Imbalanced Dataset

The Dataset was balanced by using SMOTE technique that up samples minority class. But before data was given to SMOTE it was tokenized using Term Frequency - Inverse Document Frequency(TF-IDF) that generated vectorized data as seen in the figure 8.

Figure 8: balanced Dataset

The usage of Random Forest Classifier was ispired by Fitri et al. (2019). For Hyper-parameter i.e. n-estimator for Random Forest Classifier was found using GridSearchCV as it gives the best score as observed in the figure 9. Also, the criterion set was entropy as it helps for information gain.

```
print("Apply GridSearchCV in Random Forest Classifier to get best n_estimators")
rfc = RandomForestClassifier(criterion='entropy', max_features='auto', random_state=1)
grid_param = {'n_estimators': [200, 250, 300, 350, 400, 450]}
gd_sr = GridSearchCV(estimator=rfc, param_grid=grid_param, cv=5)
gd_sr.fit(X_train, y_train)
best_parameters = gd_sr.best_params_
print(best_parameters)
best_result = gd_sr.best_score_ # Mean cross-validated score of the best_estimator
print(best_result)
```

```
Apply GridSearchCV in Random Forest Classifier to get best n_estimators
{'n_estimators': 250}
0.9786400591278639
```

Figure 9: Hyper-Parameter tuning for Random Forest

Based on the parameter the Model was fit over the dataset which was 75-25 train-test split as depicted in figure 10.

```
print("Applying Random Forest based on GridSearchCV n_estimators")
rfc = RandomForestClassifier(n_estimators=best_parameters['n_estimators'], criterion='entropy', max_features='auto',
                             random_state=1)
rfc.fit(X_train,y_train)
rfc_pred = rfc.predict(X_test)
cf_matrix_rfc = confusion_matrix(y_test, rfc_pred)
print(classification_report(y_test, rfc_pred))
sns.heatmap(cf_matrix_rfc, annot = True, cmap='Blues', fmt='g')
plt.title(label ="Confusion Matrix for Random Forest Classifier ")
```

Figure 10: Random Forest Classifier Model

Further, observed in the figure 11 the Results were obtained based on the model prediction.

```
Applying Random Forest based on GridSearchCV n_estimators
              precision    recall  f1-score   support

           0       0.98      0.99      0.98      2255
           1       0.99      0.98      0.98      2255

    accuracy                           0.98      4510
   macro avg       0.98      0.98      0.98      4510
weighted avg       0.98      0.98      0.98      4510

Text(0.5, 1.0, 'Confusion Matrix for Random Forest Classifier ')
```

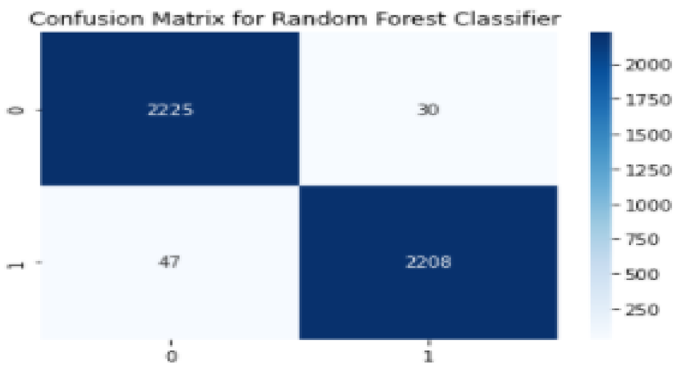


Figure 11: Results for Random Forest Classifier

Similarly, the model of Support Vector Machine was developed with same balanced Dataset and results were obtained which can be observed in figure. As the research was majorly motivate from Alhelbawy et al. (2020) as they used SVM which gave good results.

```
svmc = svm.SVC()
svmc.fit(X_train,y_train)
svmc_pred = svmc.predict(X_test)
svmc_cf_matrix = confusion_matrix(y_test, svmc_pred)
print(classification_report(y_test, svmc_pred))
sns.heatmap(svmc_cf_matrix, annot = True, cmap='Blues', fmt='g')
plt.title(label ="Confusion Matrix for SVM Classifier ")
```

```
              precision    recall  f1-score   support

           0       0.99      0.98      0.99      2255
           1       0.98      0.99      0.99      2255

    accuracy                           0.99      4510
   macro avg       0.99      0.99      0.99      4510
weighted avg       0.99      0.99      0.99      4510
```

```
Text(0.5, 1.0, 'Confusion Matrix for SVM Classifier ')
```

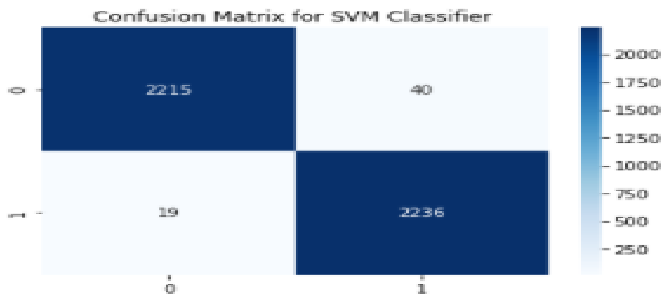


Figure 12: Model built and Results for Support Vector Machine

4.2 Experiment 2 - Classification using One Class Classification Technique

For the given experiment imbalanced dataset was given as input and the model was fit on majority class is what the concept of One Class Classification is all about as noted by Seliya et al. (2021). Here one Class Support Vector Machine was utilized as it has extension over the given technique. The following figure 13 shows the implementation along with the classification report.

```
trainX, testX, trainy, testy = train_test_split(X_tf,dataset.violation, test_size=0.5, random_state=2,
                                                stratify=dataset.violation)
# define outlier detection model
model = OneClassSVM(gamma='scale', nu=0.01)
# fit on majority class
trainX_ = trainX.toarray()[trainy==0]
model.fit(trainX_)
yhat = model.predict(testX.toarray())
# mark inliers 1, outliers -1
testy[testy == 1] = -1
testy[testy == 0] = 1
cf_matrix_OCSVM = confusion_matrix(testy, yhat)
print(classification_report(testy, yhat))
```

```
              precision    recall  f1-score   support

          -1       0.09      0.25      0.13       529
           1       0.89      0.69      0.78      4510

    accuracy                           0.64      5039
   macro avg       0.49      0.47      0.45      5039
weighted avg       0.80      0.64      0.71      5039
```

Figure 13: Model built and Results for One Class Support Vector Machine

4.3 Experiment 3 - Classification using Functional Neural Network

For the given experiment BERT base model was utilized which has 12 encoders with 768 hidden layers. Following figure 15 shows the initialization of BERT pre-processor and BERT encoder.

```
import tensorflow as tf
import tensorflow_hub as hub
import tensorflow_text as text
bert_preprocess = hub.KerasLayer("https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3")
bert_encoder = hub.KerasLayer("https://tfhub.dev/tensorflow/bert_en_uncased_L-12_H-768_A-12/4")
```

Figure 14: Pre-process and Encoder initialization of BERT

Further, the BERT Model was built over Functional Neural Network as depicted in figure 15.

```
def bertModel():
    # Bert Layers
    text_input = tf.keras.layers.Input(shape=(), dtype=tf.string, name='text')
    preprocessed_text = bert_preprocess(text_input)
    outputs = bert_encoder(preprocessed_text)

    # Neural network layers
    l = tf.keras.layers.Dropout(0.1, name="dropout")(outputs['pooled_output'])
    l = tf.keras.layers.Dense(1, activation='sigmoid', name="output")(l)

    # Use inputs and outputs to construct a final model
    return tf.keras.Model(inputs=[text_input], outputs = [l])
```

```
bertmodel = bertModel()
bertmodel.summary()
```

Figure 15: Functional Model built over BERT

based on the parameters set and number of hidden layer the model was build and summary can be seen in figure 16 where there are 3 hidden layer and rest was input and output layer.

```
____________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
====================================================================================================
 text (InputLayer)              [(None,)]            0           []

 keras_layer_2 (KerasLayer)     {'input_mask': (Non  0           ['text[0][0]']
                                e, 128),
                                 'input_type_ids':
                                (None, 128),
                                 'input_word_ids':
                                (None, 128)}

 keras_layer_3 (KerasLayer)     {'default': (None,   109482241   ['keras_layer_2[1][0]',
                                768),                             'keras_layer_2[1][1]',
                                 'encoder_outputs':               'keras_layer_2[1][2]']
                                 [(None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768)],
                                 'pooled_output': (
                                None, 768),
                                 'sequence_output':
                                 (None, 128, 768)}

 dropout (Dropout)              (None, 768)          0           ['keras_layer_3[1][13]']

 output (Dense)                 (None, 1)            769         ['dropout[0][0]']

====================================================================================================
Total params: 109,483,010
Trainable params: 769
Non-trainable params: 109,482,241
```

Figure 16: Summary of Functional Neural Network Model

The model was compiled where the parameters like optimizer, loss and metrics were setup which can be seen in figure 17.

```
METRICS = [
      tf.keras.metrics.BinaryAccuracy(name='accuracy'),
      tf.keras.metrics.Precision(name='precision'),
      tf.keras.metrics.Recall(name='recall')
]

bertmodel.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=METRICS)
```

Figure 17: Fine-Tune parameters

As dataset was imbalanced, Class weights were calculated depicted in figure as it was also one of the parameter set which training.

```
class_wts = compute_class_weight('balanced', np.unique(y_train),y_train)
```

```
class_wts
```

```
array([0.55861916, 4.76481715])
```

Figure 18: calculating Class Weight

The model was further fit over the dataset along with class weights and was run for 10 epochs displayed in figure 19 and evaluated as seen in figure 20.

```
bertmodel.fit(X_train, y_train, epochs=10, class_weight= clswt )
```

```
Epoch 1/10
237/237 [==============================] - 2790s 12s/step - loss: 0.6450 - accuracy: 0.6193 - precision: 0.1621 - recall: 0.630
5
Epoch 2/10
237/237 [==============================] - 3107s 13s/step - loss: 0.5725 - accuracy: 0.7011 - precision: 0.2174 - recall: 0.711
2
Epoch 3/10
237/237 [==============================] - 2739s 12s/step - loss: 0.5289 - accuracy: 0.7394 - precision: 0.2542 - recall: 0.766
7
Epoch 4/10
237/237 [==============================] - 2620s 11s/step - loss: 0.5073 - accuracy: 0.7499 - precision: 0.2647 - recall: 0.778
1
Epoch 5/10
237/237 [==============================] - 2607s 11s/step - loss: 0.4985 - accuracy: 0.7581 - precision: 0.2693 - recall: 0.761
7
Epoch 6/10
237/237 [==============================] - 2699s 11s/step - loss: 0.4813 - accuracy: 0.7667 - precision: 0.2854 - recall: 0.813
4
Epoch 7/10
237/237 [==============================] - 2561s 11s/step - loss: 0.4733 - accuracy: 0.7699 - precision: 0.2848 - recall: 0.789
4
Epoch 8/10
237/237 [==============================] - 2763s 12s/step - loss: 0.4714 - accuracy: 0.7737 - precision: 0.2892 - recall: 0.793
2
Epoch 9/10
237/237 [==============================] - 2781s 12s/step - loss: 0.4599 - accuracy: 0.7766 - precision: 0.2943 - recall: 0.807
1
Epoch 10/10
237/237 [==============================] - 2714s 11s/step - loss: 0.4520 - accuracy: 0.7888 - precision: 0.3082 - recall: 0.813
4
```

Figure 19: Training of Functional Neural Network

```
bertmodel.evaluate(X_test, y_test)
```

```
79/79 [==============================] - 898s 11s/step - loss: 0.5359 - accuracy: 0.7536 - precision: 0.2842 - recall: 0.8902

[0.5359407663345337,
 0.7535714507102966,
 0.2841596007347107,
 0.8901515007019043]
```

Figure 20: Evaluation of Functional Neural Network

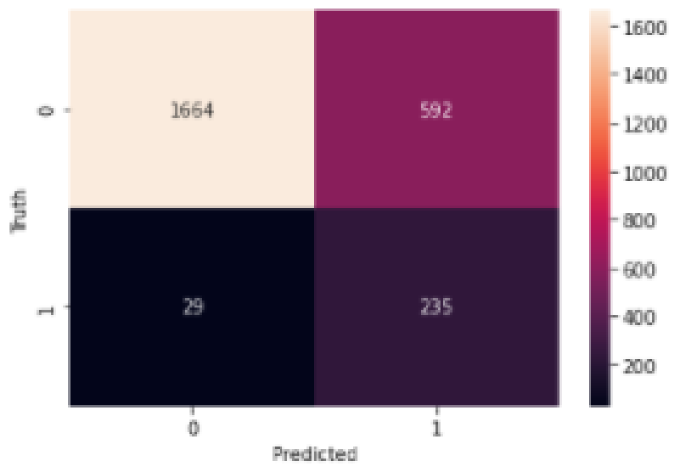
Post evaluation the Model Results were obtained as seen in figure.

```
bert_predicted = bertmodel.predict(X_test)
bert_predicted = bert_predicted.flatten()
bert_predicted = np.where(bert_predicted > 0.5, 1, 0)
from sklearn.metrics import confusion_matrix, classification_report

cm = confusion_matrix(y_test, bert_predicted)
```

```
from matplotlib import pyplot as plt
import seaborn as sn
sn.heatmap(cm, annot=True, fmt='d')
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

```
Text(33.0, 0.5, 'Truth')
```




```
print(classification_report(y_test, bert_predicted))
```

```
              precision    recall  f1-score   support

           0       0.98      0.74      0.84      2256
           1       0.28      0.89      0.43       264

    accuracy                           0.75      2520
   macro avg       0.63      0.81      0.64      2520
weighted avg       0.91      0.75      0.80      2520
```

Figure 21: Results of Functional Neural Network

References

Alhelbawy, A., Lattimer, M., Kruschwitz, U., Fox, C. and Poesio, M. (2020). An nlp-powered human rights monitoring platform, *Expert Systems with Applications* **153**: 113365.
URL: *https://www.sciencedirect.com/science/article/pii/S0957417420301901*

Fitri, V. A., Andreswari, R. and Hasibuan, M. A. (2019). Sentiment analysis of social media twitter with case of anti-lgbt campaign in indonesia using naïve bayes, decision

tree, and random forest algorithm, *Procedia Computer Science* **161**: 765–772. The Fifth Information Systems International Conference, 23-24 July 2019, Surabaya, Indonesia.
URL: *https://www.sciencedirect.com/science/article/pii/S1877050919318927*

Seliya, N., Abdollah Zadeh, A. and Khoshgoftaar, T. M. (2021). A literature review on one-class classification and its potential applications in big data, *Journal of big data* **8**(1).
URL: *http://dx.doi.org/10.1186/s40537-021-00514-x*