

# Configuration Manual

MSc Research Project  
Data Analytics

Gayathri Chengalakkattu Ajayan  
Student ID: X21157103

School of Computing  
National College of Ireland

Supervisor: Prof. Jorge Basilio

National College of Ireland  
Project Submission Sheet  
School of Computing



<b>Student Name:</b>	Gayathri Chengalakkattu Ajayan
<b>Student ID:</b>	X21157103
<b>Programme:</b>	Data Analytics
<b>Year:</b>	2022
<b>Module:</b>	MSc Research Project
<b>Supervisor:</b>	Prof. Jorge Basilio
<b>Submission Due Date:</b>	15/12/2022
<b>Project Title:</b>	Configuration Manual
<b>Word Count:</b>	1025
<b>Page Count:</b>	18

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.

<b>Signature:</b>	
<b>Date:</b>	15th December 2022

**PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:**

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

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

<b>Office Use Only</b>	
Signature:	
Date:	
Penalty Applied (if applicable):	

# Configuration Manual

Gayathri Chengalakkattu Ajayan  
X21157103

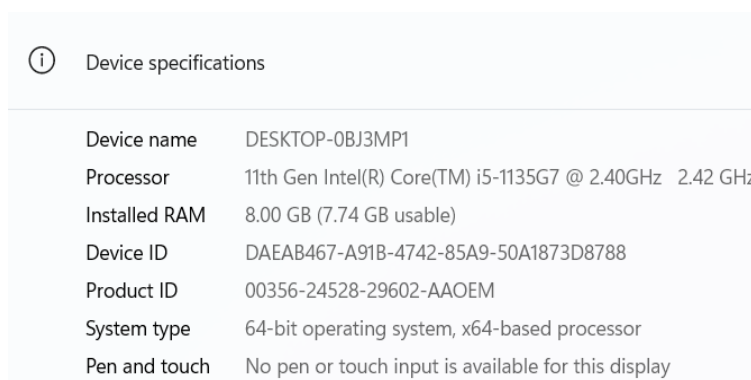
## 1 Introduction

The configuration manual provides detailed instructions for putting the study topic "Genuine Online Reviewer Identification using Machine Learning Techniques" into practice. The next sections outline the necessary hardware and software for implementation. The necessary programming code, related goal, and output outcomes are shown in order. The main goal of the study is to create a CNN-LSTM Hybrid DL Sentiment Analysis model that, using the collected data, can identify the Genuine Reviewer with a comparatively high accuracy rate. It is essential to recognize the fake comments and remove them from the dataset. In this project, many Natural Language Processing (NLP) techniques can be used for this filtering. A review dataset is utilized to train the hybrid Convolution Neural Network-Long Short Term Memory (CNN-LSTM) Machine Learning (ML) system to determine if a comment or review is favorable or negative using the sentimental analysis method. The Genuine Reviewer is selected using the combined results of the crowdsourcing method utilized in this study. The many technologies employed to produce the results are described in the sections that follow.

## 2 System Requirements

The project's system requirements are discussed in this section, and prior to conducting computing experiments, knowing of the system requirements is always advantageous.

### 2.1 Hardware Requirements



The image shows a screenshot of the Windows System Information window. At the top, there is a title bar with an information icon and the text "Device specifications". Below this, a table lists various system details:

Device name	DESKTOP-0BJ3MP1
Processor	11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz 2.42 GHz
Installed RAM	8.00 GB (7.74 GB usable)
Device ID	DAEAB467-A91B-4742-85A9-50A1873D8788
Product ID	00356-24528-29602-AAOEM
System type	64-bit operating system, x64-based processor
Pen and touch	No pen or touch input is available for this display

Figure 1: Hardware Requirements

## 2.2 Software Requirements

1. **Python 3** - The whole implementation step, including dataset cleaning and final model deployment, was done in Python.
2. **Microsoft Office 365 Excel** –Gipson (2022) Datasets were imported and exported using Excel. This study made use of datasets in .CSV file type.
3. **Google Colab** – fuat (2018) Python code was created and put to use in the Colab component of the Google Cloud Platform. In the Google Colab, the classification model's evaluation was also done. In essence, it is a Google Cloud collaboration of the Jupyter environment. To reserve a colab session, all you need is a Google account.

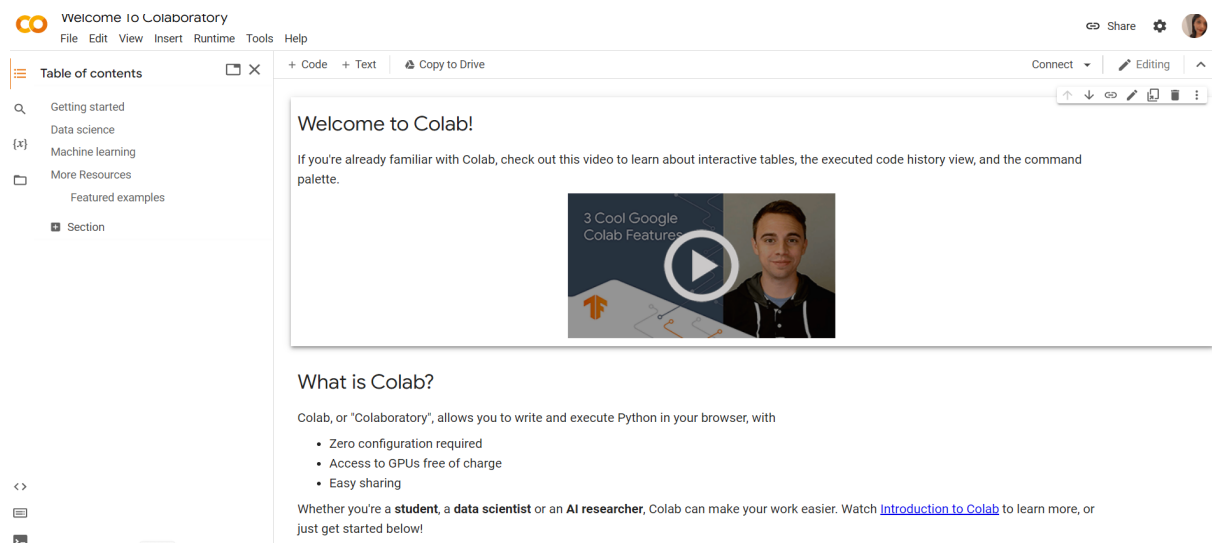


Figure 2: Software Requirements

## 3 Data Pre-Processing

### 3.1 Google Colab - Step wise Instructions

1. In order to set up an environment in Google Colab, an account must first be created.
2. After creating the account, we will be on the main page as depicted in figure 2.
3. A new notebook needs to be created in order to start the coding. As illustrated in figure 3, we must first open the file before selecting New Notebook.
4. As seen in figure 4, the Google Colab provides the ability to switch the runtime from a local machine to a GPU in order to run the code more rapidly and without any latency.

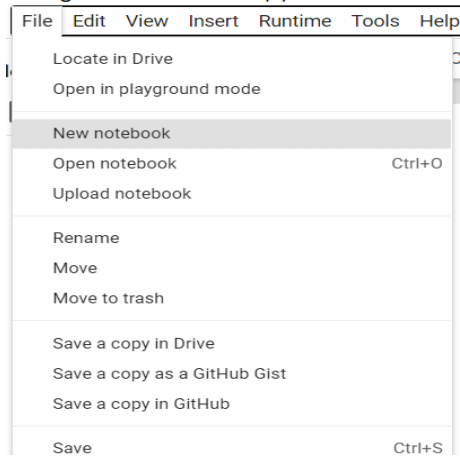


Figure 3: Colab Notebook

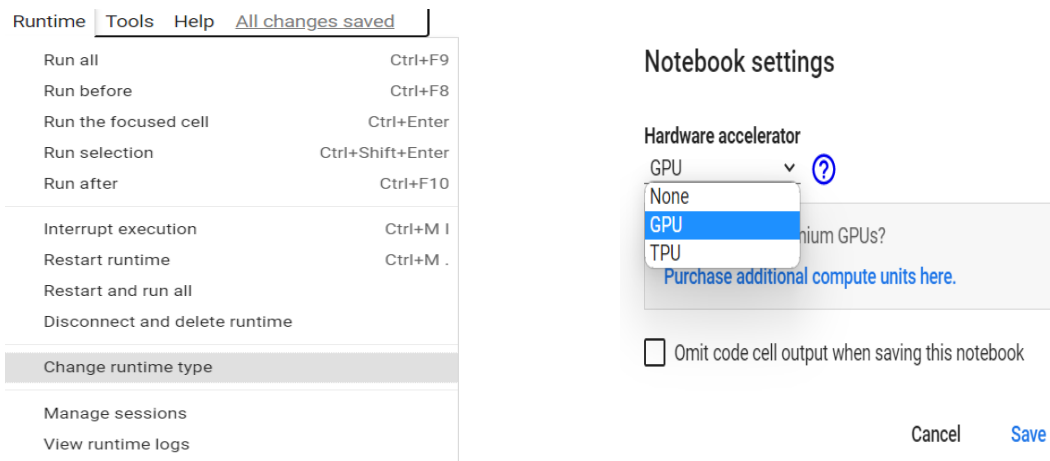


Figure 4: Changing the runtime and GPU allocation

## 3.2 Installion of Packages and Importing Raw Data

The base dataset which is the sentiment dataset was acquired from <https://www.kaggle.com/datasets/yasserh/amazon-product-reviews-dataset>.

Then second dataset for the Genuine Reviewer identification is generated by the researcher herself using Mockaroo <https://www.mockaroo.com/>, which is an online data generator tool. Each of the two datasets is obtained from its respective source and saved in .csv format on a local drive. Installing packages is necessary before importing and pre-processing data.

```
import warnings
warnings.filterwarnings("ignore")

import pandas as pd
pd.set_option('display.max_columns',None)
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
plt.rcParams['font.size']=15
%matplotlib inline
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,classification_report,confusion_matrix
from tqdm.notebook import tqdm
import pickle
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.utils import to_categorical
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Embedding,Conv1D,MaxPooling1D,Bidirectional,Dense,Dropout,Flatten,LSTM
from tensorflow.keras.optimizers import RMSprop,Adam
```

Figure 5: Importing Libraries

## 3.3 Importing of the Datasets

The necessary datasets are imported from Google Drive after being uploaded there. Data is imported using the Panda library, which is imported as pd.

The first dataset is named as 7817-1.csv and the second dataset is named as mockaroo-dataset.csv.

```
[ ] from google.colab import drive
drive.mount('/content/drive')
```

Mounted at /content/drive

Figure 6: Importing data into Google Colaboratory

### Variable Description:

As shown in figure 7 and figure 8 *df* is the variable into which the base dataset is loaded and *data* is the variable into which the second dataset, which is created using Mockaroo: an online data generator tool, has been loaded.

## Dataset Loading

```
df=pd.read_csv("/content/drive/MyDrive/7817_1.csv")
df.head()
```

Figure 7: First Dataset Loading

## Data Loading

```
[ ] data=pd.read_csv('/content/drive/MyDrive/GenuineReviewClassification/input/mockaroo_dataset.csv')
data=data.dropna()
data.head()
```

Figure 8: Second Dataset Loading

## 3.4 Data Pre-processing

### 3.4.1 EDD - Exploratory Data Analysis

1. The dataframes are first examined for the shape and printed as seen in figures 9 and 10.

```
[ ] #Print the total number of rows and columns in the dataset/dataframe
df.shape

(1597, 27)
```

Figure 9: Shape of DataFrame "df" associated with first dataset

```
[ ] data.shape

(1894, 3)
```

Figure 10: Shape of DataFrame "data" associated with second dataset

2. Storing only those columns to the dataframe "df" that are necessary for the following steps, as shown in figure 11.
3. Figure 12 given above illustrates how null values are identified and eliminated from dataframes.
4. The names of the selected columns are changed to "text" and "label," and figure 13 given above displays the sentiment of the "text" column, which contains the review content.

```
#Storing only the required columns to the dataframe "df"
df=df[["reviews.text","reviews.rating"]]
#Printing the final datafaramame
df.head()
```

	reviews.text	reviews.rating
0	I initially had trouble deciding between the p...	5.0
1	Allow me to preface this with a little history...	5.0
2	I am enjoying it so far. Great for reading. Ha...	4.0
3	I bought one of the first Paperwhites and have...	5.0
4	I have to say upfront - I don't like coroporat...	5.0

Figure 11: Storing only the required columns to the dataframe "df"

```
#Checking for null values inside the dataframe
df.isnull().sum()
```

reviews.text	0
reviews.rating	420

```
#Removig the null values
df=df.dropna().reset_index(drop=True)
```

Figure 12: Checking and Removing Null Values

```
#Renaming the columns' names to "text" & "label"
df.columns=["text","label"]
df.head()

#Text Sentiment
df['label']=df['label'].apply(lambda x: "positive" if x>=3.0 else "negative")
```

Figure 13: Text Sentiment



5. The exploratory data analysis stage reveals that the variable "label," which represents the sentiment of the review text, has a class imbalance. The solution to this class imbalance is upsampling as shown in figure 14.

```
from sklearn.utils import resample
oversampled=[]
for label in df['label'].unique():
    oversampled.append(resample(df.loc[df['label']==label],n_samples=1500))
df=pd.concat(objs=oversampled,ignore_index=True)
df=df.sample(frac=1).reset_index(drop=True)
```

Figure 14: Upsampling

### 3.4.2 Text Cleaning

The column "text" that holds the customer reviews text in the base dataset and the column "Review" that holds the reviews in the second dataset are cleaned using same set of procedures as shown in figure 15. Both datasets are cleaned using the same set of procedures to avoid misclassification.

```
#from cleantext import clean
#tqdm is a library in Python which is used for creating Progress Meters or Progress Bars.
from tqdm.notebook import tqdm
cleaned_sentence=[]
for sentence in tqdm(df['text'].values):
    sent=expandContractions(sentence)
    sent=nlp(sent.lower())
    cleaned_sentence.append(text_cleaning(sent))
df['text']=cleaned_sentence
df['text']=df['text'].apply(str)
```

Figure 15: Text Cleaning

Texts were normalized then punctuations, stopwords were removed and the contractions were expanded with the functions `textcleaning()` and `expandContractions()` with the following codes as shown in figure 16 in the next page. In the `expandContractions()`, we have used a pickle file named as `cword-dict.pkl`. It is basically a dictionary with contractions as key and their expansions as the value. It has been created by the researcher herself after referring few resources.

```

def text_cleaning(docx):
    sentence=[word.lemma_.strip() if word.lemma_ != "-PRON-" else word.lower_ for word in docx]
    sentence=[word for word in sentence if word not in stopwords and word not in punctuations]
    sentence=[word for word in sentence if len(word)>1 and word.isalpha()]
    sentence=" ".join(sentence)
    return sentence

#Expand the contractions
import re
import pickle

with open("/content/drive/MyDrive/GenuineReviewClassification/input/cword_dict.pkl",mode='rb') as file:
    cList=pickle.load(file)
    c_re = re.compile('%s' % '|'.join(cList.keys()))

def expandContractions(text, c_re=c_re):
    def replace(match):
        return cList[match.group()]
    return c_re.sub(replace, text)

```

Figure 16: textcleaning() and expandContractions()

### 3.4.3 Word Cloud

With the use of a word cloud and the code in figure 17, significant textual data points are highlighted. The frequency or relevance of each word is indicated by its size in a word cloud, a data visualization tool for displaying text data.

```

from wordcloud import WordCloud, STOPWORDS
stopwords=set(STOPWORDS)
#The FreqDist function gives the user the frequency distribution of all the words in the text
from nltk import FreqDist
doc_list = list(df['text'])
words = [sublist for sublist in doc_list]
word= FreqDist(words)
plt.figure(figsize=(18,8))
wordcloud = WordCloud(width = 1000,background_color = 'lightcyan', height = 500,stopwords=stopwords).generate(" ".join(str(v) for v in word))
plt.imshow(wordcloud)
plt.axis("off")
plt.title(label='Wordcloud for frequency words',fontsize=20)
plt.show()

```

Figure 17: Word Cloud

### 3.4.4 Label Encoding

The following code encoded the label column.

**Positive: 0**

**Negative:1**

```

df['label']=df['label'].map({'positive':0,'negative':1})
#Shuffle DataFrame rows
df=df.sample(frac=1).reset_index(drop=True)
df.head(5)

```

Figure 18: Label Encoding

### 3.4.5 Vectorization Technique to create Bag-of-Words

Text can be represented in vectors most simply as a collection of words. Each column in a vector represents a word. How many times a word appears in a sentence is shown by the values in a row's cells. The texts are turned into sequences using the following code, which is displayed in figure 19. The total number of unique tokens is then reported.

```
tokenizer=Tokenizer()
tokenizer.fit_on_texts(df['text'].values)
sequence_data=tokenizer.texts_to_sequences(df['text'].values)

print(sequence_data[:5],"\n")
vocab_size=len(tokenizer.word_index)+1
print("Unique Tokens size : {}".format(len(tokenizer.word_index)+1))
```

Figure 19: Vectorization

Pad sequences is then used to ensure that all sequences in a list have the same length with the following code as shown in figure 20.

```
#Pad_sequences is used to ensure that all sequences in a list have the same length.
pad_text=pad_sequences(sequences=sequence_data,maxlen=200,padding='post',truncating='post')
print(pad_text[:5])
```

Figure 20: Pad Sequences

**The whole text cleaning and vectorization techniques are applied for both datasets to avoid misclassification.**

Figure 21 illustrates how the column "label," which was originally a numpy array of integers to represent the various categories in the data, was converted using the to-categorical() function into a numpy array or matrix with binary values and as many columns as there were categories in the data.

```
y=to_categorical(y=y,num_classes=2)
print(y)
```

Figure 21: to-categorical()

### 3.4.6 Data set Splitting into train, validation and testing

The base dataset is then splitted into train, validation and test sets as depicted in figure 22.

```
[ ] X_train,X_test,y_train,y_test=train_test_split(pad_text,y,test_size=0.3,random_state=37)
#random_state=42
print(X_train.shape,X_test.shape,y_train.shape,y_test.shape)

(2100, 200) (900, 200) (2100, 2) (900, 2)
```

Figure 22: Dataset splitting into train and testing sets

### 3.4.7 Glove word Embedding

GloVe is an unsupervised learning method for building word vector representations (Global Vectors for Word Representation). In other words, GloVe gives us the ability to naturally map each word in a corpus of text to a location in a high-dimensional space. As a result, related terms will be put in one group.

The GloVe file **glove.6B.50d.txt** has been obtained from: <https://www.kaggle.com/datasets/watts2/glove6b50d.txt> and stored in the folder GenuineReviewerIdentification.

```
embeddings_index = {}
with open('/content/drive/MyDrive/GenuineReviewClassification/input/glove.6B.50d.txt', encoding='utf-8') as f:
    for line in f:
        values = line.split()
        word = values[0]
        coefs = np.asarray(values[1:], dtype='float32')
        embeddings_index[word] = coefs
    f.close()

embedding_dimention = 50
def embedding_matrix_creator(embedding_dimention, word_index):
    embedding_matrix = np.zeros((len(word_index)+1, embedding_dimention))
    for word, i in word_index.items():
        embedding_vector = embeddings_index.get(word)
        if embedding_vector is not None:
            embedding_matrix[i] = embedding_vector
    return embedding_matrix
embedding_matrix = embedding_matrix_creator(50, word_index=tokenizer.word_index)
print("Glove Loded!")
```

Figure 23: Glove word Embedding

## 4 Model Implementation

The steps involved in putting the model into practice are discussed in this section. There are various libraries that must be installed before the model construction process can begin. In figure 24, those libraries are depicted.

```
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Embedding,Conv1D,MaxPooling1D,Bidirectional,Dense,Dropout,Flatten,LSTM
from tensorflow.keras.optimizers import RMSprop,Adam
```

Figure 24: Model implementation libraries

## 4.1 CNN-LSTM Model Implementation

The CNN-LSTM model's implementation procedures are displayed in the figure 25. The processes needed to create the CNN-LSTM model initialization are shown in the figure, along with the steps involved in model compilation. The model initialization was then started using the following parameters using the `model.fit()` function. Finally, the optimizer is initialized for model training. Thus the Sentiment Analysis System is built using the Training Data with the CNN-LSTM model.

```
model=Sequential()
model.add(Embedding(input_dim=vocab_size, output_dim=50, input_length=pad_text.shape[1],weights=[embedding_matrix]))
model.add(Conv1D(filters=64,kernel_size=3,padding='valid',activation='relu'))
model.add(Conv1D(filters=64,kernel_size=3,padding='valid',activation='relu'))
model.add(MaxPooling1D())
model.add(Bidirectional(LSTM(200,return_sequences=True,dropout=0.2)))
model.add(Bidirectional(LSTM(200)))
model.add(Flatten())
model.add(Dropout(0.3))
model.add(Dense(512,activation='relu'))
model.add(Dense(2,activation='sigmoid'))
model.compile(loss='binary_crossentropy',optimizer=Adam(learning_rate=1e-4),metrics=['accuracy'])

model.summary()

history=model.fit(x=X_train,y=y_train,batch_size=32,epochs=8,validation_data=(X_test,y_test))
```

Figure 25: Model Building

## 5 Evaluation of Implemented Model

The necessary steps for the evaluation of the model are discussed in this section.

### 5.1 Evaluation of CNN-LSTM model

#### 5.1.1 Accuracy and Loss Plots

The figure 26 in the next page shows the code with the all required steps for implementing the accuracy and loss plots for the model.

Both accuracy plot and loss plots have plotted using the following snippet of code.

Validation accuracy and validation loss plots are also plotted with this code as shown in figure 26.

```

with plt.style.context(style='fivethirtyeight'):
    plt.figure(figsize=(18,8))
    plt.rcParams['font.size']=20
    plt.subplot(121)
    plt.plot(history.history['accuracy'],label='Accuracy')
    plt.plot(history.history['val_accuracy'],label='Val_Accuracy')
    plt.title("Accuracy Plots")
    plt.xlabel("Epochs")
    plt.ylabel("Accuracy")
    plt.legend()
    plt.subplot(122)
    plt.plot(history.history['loss'],label='Loss')
    plt.plot(history.history['val_loss'],label='Val_Loss')
    plt.title("Loss Plots")
    plt.xlabel("Epochs")
    plt.ylabel("Loss")
    plt.legend()
    plt.show()

```

Figure 26: Accuracy and Loss Graphs

### 5.1.2 Model Prediction

Figure 27 given below shows the code for the model prediction.

```

[ ] model_pred=model.predict(x=X_test,batch_size=32,verbose=1)
    print(model_pred)

```

Figure 27: Model Prediction

As shown in the figure 28 `argmax()` is used to convert the probabilities or the categorical values back to the labels.

```

[ ] pred_label=[]
    for i in range(len(model_pred)):
        pred_label.append(np.argmax(model_pred[i]))
    true_label=[]
    for i in range(len(y_test)):
        true_label.append(np.argmax(y_test[i]))
    print(true_label[:50])

```

Figure 28: `argmax()`

### 5.1.3 Result Analysis

As illustrated below in the figure 29, figure 30 and figure 31, metrics including the classification report, confusion matrix and accuracy score imported from the Sklearn package are used to evaluate the classification model.

#### Classification Report:

```
print(classification_report(y_true=true_label,y_pred=pred_label,target_names=['positive','negative']))
```

	precision	recall	f1-score	support
positive	0.96	0.98	0.97	465
negative	0.98	0.96	0.97	435
accuracy			0.97	900
macro avg	0.97	0.97	0.97	900
weighted avg	0.97	0.97	0.97	900

Figure 29: Classification Report

#### Confusion Matrix:

```
▶ from sklearn.metrics import confusion_matrix
matrix = confusion_matrix(y_true=true_label,y_pred=pred_label)
print("\n")
print(matrix)
print("\n");
import seaborn as sns
class_names = ["positive","negative"]
fig,ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks,class_names)
plt.yticks(tick_marks,class_names)
cm=confusion_matrix(y_true=true_label,y_pred=pred_label)
df_cm = pd.DataFrame(cm, index=class_names, columns=class_names, )
sns.heatmap(df_cm,annot=True,cmap="Blues",fmt="d")
sns.set(style='whitegrid', palette='muted', font_scale=1.5)
ax.xaxis.set_label_position('bottom')
plt.tight_layout()
plt.ylabel('Actual label')
plt.xlabel('Predicted label');
#plt.show()
```

Figure 30: Confusion Matrix

### Accuracy score:

```
[ ] model_accuracy=accuracy_score(y_true=true_label,y_pred=pred_label)
    print("Overall validated accuracy of the model is {:.2f}%".format(model_accuracy*100.0))
```

Figure 31: Accuracy score

#### 5.1.4 Model Saving

The Sentiment Model is saved as shown below in the figure 32.

##### **Model Saving**

```
[ ] model.save(filepath="model/ConvolutionalLongShortTermMemory.h5")
```

Figure 32: Model Saving

#### 5.1.5 Model Loading

The Sentiment Model is then loaded for the Genuine Reviewer Identification as shown below:

```
[ ] model=load_model(filepath='content/drive/MyDrive/GenuineReviewClassification/model/ConvolutionalLongShortTermMemory.h5')
```

Figure 33: Model Loading

## 6 Genuine Reviewer Identification

The sentiment associated with each reviews in the second dataset is found and is merged to the dataframe under the column name "sentiment" using the following snippet of code.

### Labels:

Positive Sentiment: 0

Negative Sentiment: 1

```
[ ] #Sentiment
    data['sentiment']=pred_label
    data=data.sample(frac=1).reset_index(drop=True)
    data.head()
```

Figure 34: Sentiment Identification

In the next step, a list of all the unique products are developed from the second dataset



```
unique_products=list(data['product_name'].unique())
print(unique_products)
```

Figure 35: Unique Products List

with the column "product name".

Then a product dictionary is created with product name as the key and value is a list that contains the sentiment as shown in figure 36 and figure 37.

```
product_dictionary={}
for product in unique_products:
    product_dictionary[product]=[]
print(product_dictionary)
```

Figure 36: Empty Product Dictionary Creation

```
for index,features in data.iterrows():
    for key,value in product_dictionary.items():
        if features['product_name']==key:
            if features['sentiment']==0:
                product_dictionary[key].append(0)
            else:
                product_dictionary[key].append(1)
        else:
            continue
```

Figure 37: Product Dictionary

Then the overall sentiment is calculated for the each product is calculated which is the product crowd sentiment analysis. The whole procedure is depicted in figure 38.

```
overall_sentiment=[]
for key,value in product_dictionary.items():
    positive=value.count(0)
    negative=value.count(1)
    if positive>negative:
        overall_sentiment.append('positive')
    else:
        overall_sentiment.append("negative")

unique_overall_df=pd.DataFrame(data={'product_name':unique_products,'overall_status':overall_sentiment})
unique_overall_df.head()
```

Figure 38: Product Crowd Sentiment Analysis

Nextly, User Crowd Sourcing Behavior is evaluated. A list of all the unique users are developed from the second dataset with the column "Name".

```
unique_users=data['Name'].unique()
print(list(unique_users))
```

Figure 39: Unique Users List

Then an user dictionary is created as shown in figure 40 and figure 41 with user name as the key and value is a list that contains the information that whether the user sentiment is with the product crowd sentiment or the user sentiment is against the product crowd sentiment.

```
users_dictionary={}
for user in unique_users:
    users_dictionary[user]=[]
print(users_dictionary)
```

Figure 40: Empty User Dictionary Creation

```

for indices1,features1 in data.iterrows(): #user
    for indices2,features2 in unique_overall_df.iterrows(): #product_crowd
        if features1['product_name']==features2['product_name']:
            sent=''
            if features1['sentiment']==0:
                sent='positive'
            else:
                sent='negative'
            if sent==features2['overall_status']:
                for key,value in users_dictionary.items():
                    if features1['Name']==key:
                        users_dictionary[key].append("WithCrowd")
            else:
                for key,value in users_dictionary.items():
                    if features1['Name']==key:
                        users_dictionary[key].append("AgainstCrowd")

```

Figure 41: User Dictionary

Then two empty lists are created to store the total numbers of users' whose sentiment is with the product crowd sentiment and whose sentiment is against the product crowd sentiment as shown in the figure 42.

```

with_crowd=[]
against_crowd=[]
for key,value in users_dictionary.items():
    with_crowd.append(value.count('WithCrowd'))
    against_crowd.append(value.count('AgainstCrowd'))

```

Figure 42: User lists

Finally, users are rewarded with marks depending on user sentiment. Each user will be provided with +2 marks for each genuine review and it will be stored in the variable "GenuineReviewCounts". Then each user will be provided with -2 marks for each non genuine review and it will be stored in the variable "NotGenuineReviewCounts". The difference of both will give the genuine review score. Genuine Reviewer Rating is done using the snippet code given in the figure 43. The top K Genuine Reviewers are listed in the descending order of their scores.

```

#Users are given +2 marks for genuine reviews and -2 marks for fake reviews
#Difference of both scores will give the Genuine Score for each user
result=pd.DataFrame(data={'Users':users_dictionary.keys(),'GenuineReviewCounts': with_crowd,'NotGenuineReviewCounts': against_crowd})
result=result.fillna(0)
result['GenuineScore']=(result['GenuineReviewCounts']*2) - (result['NotGenuineReviewCounts']*2)
result=result.sort_values(by='GenuineScore',ascending=False).reset_index(drop=True)
result.head()

```

Figure 43: Genuine Reviewer Rating

## 7 Conclusion

Therefore, if any interested third parties recreate the step-by-step implementation described in this paper, it will always function. As a result, the research was successful in achieving its stated goals.

## References

- fuat (2018). Google Colab Free GPU Tutorial, <https://medium.com/deep-learning-turkey/google-colab-free-gpu-tutorial-e113627b9f5d>. [Online; accessed 10-December-2022].
- Gipson, S. (2022). How to Import/Convert CSV to Excel, <https://www.guru99.com/import-csv-data-excel.html>. [Online; accessed 10-December-2022].