Msc Data Analytics

MSc Research Project
Msc Data Analytics

Deborah Ebbu Kammu
Student ID: x20217561

School of Computing
National College of Ireland

Supervisor: Anderson Simiscuka

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Deborah Ebbu Kammu
Student ID:	x20217561
Programme:	MSc Data Analytics
Year:	2022
Module:	MSC Research Project
Supervisor:	Anderson Simiscuka
Submission Due Date:	152022
Project Title:	Research on Negative Post Identification in the Regional Language (Hindi)
Word Count:	500
Page Count:	7

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

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

Signature:	
Date:	15th December 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

Deborah Ebbu Kammu
x20217561

1 Introduction

How to successfully replicate the project is described in this documentation. All the details like hardware and Software requirements and libraries needed to implement this project is mentioned in this manual.

2 System Requirements

Hardware	Specification
Local System	HP PAVILION
RAM	16 GB
SSD	256 GB
CPU	AMD Ryzen 4000 series

Software	Specification
OS	Windows 11 Home (64-bit)
IDE	Jupyter Notebook, Google Colab
Programming Language	Python 3.8.2

Figure 1: System Requirements

3 Libraries Required

The following libraries were necessary for the code to operate. Some libraries are pre-installed with Python, while others need installation.
pandas, numpy, matplotlib, sklearn, tensorflow, keras, indic-nlp-library, googletrans
These libraires can be installed by using command - pip install (library name).

4 Dataset pre-processing and transforming

```
[11] import io
     df=pd.read_csv("/content/drive/MyDrive/emotions.csv")
     #df = pd.read_csv(io.BytesIO('emotions (1).csv'))
```

Figure 2: Importing Dataset

```
import googletrans
print(googletrans.LANGUAGES)

{'af': 'afrikaans', 'sq': 'albanian', 'am': 'amharic', 'ar': 'arabic', 'hy': 'armenian', 'az': 'azerbaijani', 'eu': 'basque', 'be':
```

```
from googletrans import Translator
import requests
import json
translator = Translator()
df['translated_text'] = df['Sentences'].apply(translator.translate, src='hi', dest='en').apply(getattr, args=('text',))
```

```
[16] df
```

	Sentences	Label	translated_text
0	मेरे ट्रैन का बुकिंग फेल हो रहा है बार बार	angry	my train booking is failing again and again
1	मेरे फ्लाइट का बुकिंग फेल हो रहा है बार बार	angry	my flight booking is failing again and again
2	क्या बेकार की बातें कर रहे हो	angry	what rubbish are you talking about
3	क्या बताओ यार मूड ही ख़राब है	angry	what tell me friend the mood is bad
4	आप ऐसे कैसे मेरा पैसा काट सकते हो	angry	how can you deduct my money like this
...	...	...	...
508	यार मेरी दिक्कत क्यों नहीं समझ रहे	sad	why don't you understand my problem
509	यार मेरे पैसे वापस कब मिलेंगे	sad	dude when will i get my money back
510	बेकार प्लान है इनका	sad	their plan is useless
511	बहुत दुखी हूँ मै तुम्हारे कारण	sad	i am so sad because of you

Figure 3: Translating Hindi words into English: The dataset consists of post in Hindi so it is translated in English so that it can be understandable for other users also if they don't know Hindi.

```
import collections
from collections import Counter
import re
def processText(text):
    text = text.lower()
    text = re.sub('((www.[^s]+)|(https?://[^s]+))','',text)
    text = re.sub('@[^s]+','',text)
    text = re.sub('[s]+', ' ', text)
    text = re.sub(r'#([^s]+)', r'1', text)

    return text
```

```
[21] for i in range(len(df)):
         df['Sentences'][i] = processText(df['Sentences'][i])
```

Figure 4: Processing Text: The sub() function searches for the pattern in the string and replaces the matched strings with the replacement (repl). If the sub() function couldn't find a match, it returns the original string. Otherwise, the sub() function returns the string after replacing the matches. Removing the unwanted links and mentions from twitter data and cleaning it.

```
from indicnlp.tokenize import indic_tokenize
def tokenization(indic_string):
    tokens = []
    for t in indic_tokenize.trivial_tokenize(indic_string):
        tokens.append(t)
    return tokens
df['Sentences'] = df['Sentences'].apply(lambda x: tokenization(x))
```

```
for i in range(len(df)):
    df['Sentences'][i] = [s.replace("\n", "") for s in df['Sentences'][i]]
```

Figure 5: Using Indicnlp for NLP and common text processing of Hindi (Indian languages). Trivial tokenizer tokenizes the punctuation boundaries (—, ;, :, etc). And it returns the lists of tokens.

```
import matplotlib.pyplot as plt
df_list = []
for i in range(len(df)):
    df_list +=df['Sentences'][i]
font = "/content/drive/MyDrive/gargi.ttf"
dictionary=Counter(df_list)
from wordcloud import WordCloud
wordcloud = WordCloud(width = 1000, height = 700,
                background_color ='white',
                min_font_size = 10, font_path= font).generate_from_frequencies(dictionary)
# plot the WordCloud image
plt.figure(figsize = (18, 8), facecolor = None)
plt.imshow(wordcloud,interpolation="bilinear")
plt.axis("off")
plt.tight_layout(pad = 0)
plt.show()
```

Figure 6: Upload "gargi.ttf" file which is uploaded in the code artifact. Word clouds, also known as tag clouds, are visual representations of word frequency that give terms that appear more frequently in a source text more emphasis. The word's frequency in the manuscript was indicated by how big it appeared in the image (s).

```
[21] y = df["Label"]
     X = df.drop(["Label"], axis = 1)
```

```
# 70/15/15 train/test/val split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.15, random_state=1)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.15, random_state=1)

X_train.head()
```

	Sentences	translated_text	word_count
171	[आये, होये, !, !, !, आज, तो, तूने, दिल, जीत, ल...	You have come! , , Today you have won my heart...	13
226	[शुक्रिया, तुम्हारा]	thank you	3
24	[यार, बार, बार, इंडिका, ही, क्यों, बुक, होता, है]	Why is Indica booked again and again?	10
33	[ऐसे, कैसे, चैनल्स, बंद, कर, दिए, आप]	How did you close the channels like this	8
167	[वाह, !, !, !, सस्ती, चीज, बता, दी, यार]	Wow ! , , Told the cheap thing man	9

Figure 7: Training Datasets and dividing the dataset into 3 subsets train, test, val to evaluate the performance of the model.

```
tk = Tokenizer(filters='!"#$%&()*+,-./:;<=>?@[\]^_`{"}~\t\n')

all_sentences = X_train + X_test + X_val

tk.fit_on_texts(X['Sentences'])

# + 1 for unknown token
vocab_size = len(tk.word_index) +1

X_train_seq = tk.texts_to_sequences(X_train['Sentences'])
X_test_seq = tk.texts_to_sequences(X_test['Sentences'])
X_val_seq = tk.texts_to_sequences(X_val['Sentences'])
# Initializing max length of sentence to 20 words
max_length = 20
```

Figure 8: Tokenizing each word in the sentence with maximum length=20, also eliminating the punctuations, line breaks, etc

```
tk.word_index

{'है': 1,
 'क्या': 2,
 'हो': 3,
 'नहीं': 4,
 '!': 5,
 'यार': 6,
 'कर': 7,
 'ये': 8,
 'बहुत': 9,
 'में': 10,
 'से': 11,
 'एक': 12,
 'की': 13,
 'होटल': 14,
```

Figure 9: Indexing the words.

```
[26] X_train_seq_pad = pad_sequences(X_train_seq, maxlen=max_length, padding='post')
     X_test_seq_pad = pad_sequences(X_test_seq, maxlen=max_length,padding='post')
     X_val_seq_pad = pad_sequences(X_val_seq, maxlen=max_length,padding='post')

#padding the sequences to make all the input sequences of the same length
le = LabelEncoder()
y_train_le = le.fit_transform(y_train)
y_test_le = le.transform(y_test)
y_val_le = le.transform(y_val)
y_train_oh = to_categorical(y_train_le)
y_test_oh = to_categorical(y_test_le)
y_val_oh = to_categorical(y_val_le)

[ ] X_train_seq_pad
```

Figure 10: Sequencing and padding the datasets to make all input sequence of the same length.

```
b_dims = 256

del = Sequential()
del.add(Embedding(vocab_size, emb_dims, input_length=max_length, embeddings_regularizer = tf.keras.regularizers.l2(0.000
del.add(LSTM(units = 16, dropout = 0.2,recurrent_dropout = 0.2))
del.add(Dense(4, activation='softmax'))

del.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

story = model.fit(X_train_seq_pad, y_train_oh, epochs = 128, batch_size = 256, validation_data=(X_val_seq_pad, y_val_oh)
```

Figure 11: Running epochs for training the data in the model.

```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```

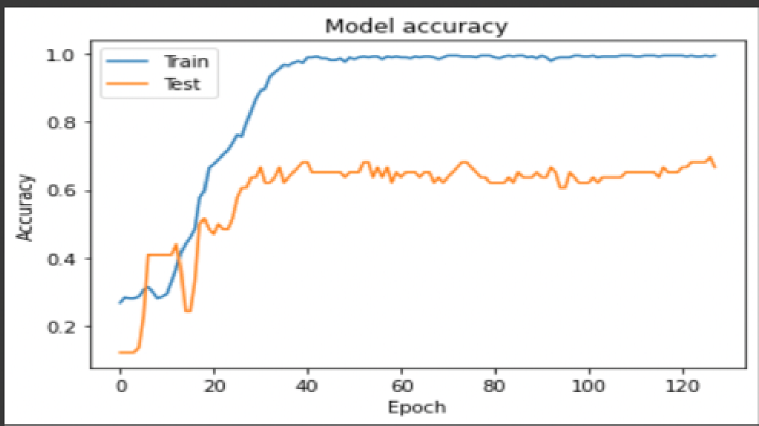

Figure 12: Plotting the accuracy of train and test data sets.

```
model.summary()

Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 20, 256)           176640
_________________________________________________________________
lstm (LSTM)                  (None, 16)                17472
_________________________________________________________________
dense (Dense)                (None, 4)                 68
=================================================================
Total params: 194,180
Trainable params: 194,180
Non-trainable params: 0
_________________________________________________________________
```

Figure 13: Showing the model summary.

```
results = model.evaluate(X_test_seq_pad, y_test_oh)
print('/n')
print('Test accuracy of word embeddings model: {0:.2f}%'.format(results[1]*100))

3/3 [==============================] - 0s 11ms/step - loss: 1.6572 - accuracy: 0.6104
/n
Test accuracy of word embeddings model: 61.04%

df
```

Figure 14: Test accuracy of word embeddings model

```
[ ] df=pd.read_csv("/content/drive/MyDrive/emotions.csv")
    X = df['Sentences']
    y = df['Label']

[ ] from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=48)

    X_train

    129           यार कुछ पल्ले ही नहीं पर रहा
    151     रिचार्ज करने के लिए धन्यवाद मेरे हीरो
    373          ट्रेन क्या टाइम स्टेशन पहुंचेगी
    98            तुम कभी कुछ समझते ही नहीं हो
    482      मेरे को तुम्हारा सर्विस पसंद नहीं आया
                              ...
    347                        एक काम कर दो
    452            एप्प का परफॉर्मेंस बहुत खराब है
    337                          काम हो गया
    51              अबे चूतियों वाली बातें मत करो
    512            शिट यार फिर से चैनल्स बंद हो गए
    Name: Sentences, Length: 461, dtype: object
```

Figure 15: We may build our training data and test data with the aid of the Sklearn train test split function. This is so because the original dataset often serves as both the training data and the test data. Starting with a single dataset, we divide it into two datasets—train and test—in order to obtain the data needed to create a model.

```
] stop_words = ['',' ',' ','!','! ','!  ','! !','! ! ','! ! !','?','ही','तुमसे','बार','आप','तुम्हारे','तु','रहा','कुछ','कभी','एक','तु
  def my_tokenizer(s):
      return s.split(' ')

  from sklearn.feature_extraction.text import CountVectorizer
  vect = CountVectorizer(min_df=2, ngram_range=(1, 3), encoding='ISCII',tokenizer=my_tokenizer,stop_words=stop_words).fit(

  /usr/local/lib/python3.8/dist-packages/sklearn/feature_extraction/text.py:516: UserWarning: The parameter 'token_pattern'
    warnings.warn(
  /usr/local/lib/python3.8/dist-packages/sklearn/feature_extraction/text.py:396: UserWarning: Your stop_words may be incons
    warnings.warn(

] print(len(vect.get_feature_names()))

  479
```

```
] from sklearn.metrics import accuracy_score
  X_test_transformed = vect.transform(X_test)
  y_pred_train = model1.predict(X_train_vectorized)
  y_pred_test = model1.predict(X_test_transformed)
  print('Train accuracy = ', accuracy_score(y_train, y_pred_train))
  print('Test accuracy = ', accuracy_score(y_test, y_pred_test))

  Train accuracy =  0.7440347071583514
  Test accuracy =  0.6346153846153846
```

Figure 16: Showing the results of train accuracy and test accuracy.

```
[51] print('Final cross validation score = ', np.mean(c))

     Final cross validation score =  0.6140271493212669
```

Figure 17: Training on the whole data set and 10 fold cross validation core.

```
from sklearn import svm
svc = svm.SVC()
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV

# defining parameter range
param_grid = {'C': [0.1, 1, 10, 100, 1000],
              'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
              'kernel': ['rbf']}

grid = GridSearchCV(SVC(), param_grid, refit = True, verbose = 3)

# fitting the model for grid search
grid.fit(X_vectorized,y)
```

```
[56] grid_predictions = grid.predict(X_vectorized)

     # print classification report
     print(classification_report(y, grid_predictions))

              precision    recall  f1-score   support

       angry       0.74      0.96      0.84       130
       happy       0.91      0.89      0.90       151
     neutral       0.94      0.81      0.87       128
         sad       0.89      0.73      0.80       104

    accuracy                           0.86       513
   macro avg       0.87      0.85      0.85       513
weighted avg       0.87      0.86      0.86       513
```

Figure 18: Showing the SVM results

```
from sklearn.ensemble import RandomForestClassifier
classifier= RandomForestClassifier(n_estimators= 10, criterion="entropy")
classifier.fit(X_vectorized, y)
y_pred= classifier.predict(X_vectorized)
print(classification_report(y_pred, y))

              precision    recall  f1-score   support

       angry       0.99      0.94      0.97       137
       happy       0.97      0.99      0.98       148
     neutral       0.97      0.98      0.97       127
         sad       0.95      0.98      0.97       101

    accuracy                           0.97       513
   macro avg       0.97      0.97      0.97       513
weighted avg       0.97      0.97      0.97       513

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

Figure 19: Showing the Random forest result.