Configuration Manual

MSc Research Project
Data Analytics

KARTHIK RAMACHANDRAN
Student ID: x21234884

School of Computing
National College of Ireland

Supervisor: Taimur Hafeez

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	KARTHIK RAMACHANDRAN
Student ID:	x21234884
Programme:	Data Analytics
Year:	2023
Module:	MSc Research Project
Supervisor:	Taimur Hafeez
Submission Due Date:	14/08/2023
Project Title:	Configuration Manual
Word Count:	590
Page Count:	10

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:	KARTHIK RAMACHANDRAN
Date:	18th September 2023

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

KARTHIK RAMACHANDRAN
x21234884

1 Introduction

This paper give an idea about the implementation of the proposed system *1* (n.d.). The code execution and their corresponding output is shared in this configuration manual. The system configuration is also shared in this paper.

2 System Configuration of the proposed system

Google colab platform is used to execute the proposed system. The detailed configuration list is shown in Figure 1

Platform	Google colab pro +
GPU driver	Nvidia V100
RAM	52 GB
Storage	Google Cloud Platform

Figure 1: Configuration

3 Importing packages

The main packages that used for the proposed study were tensorflow and python. The study was built using tensorflow 12.0 framework. The list of packages imported were displayed in Figure 2

4 Data loading

The data is loaded from google cloud platform which is mounted to google colab. Figure 3

5 Exploratory data analysis

Exploratory data analysis were performed on amazon review dataset. The distribution of target variable is ploted in Figure 4

```
[2] import tensorflow as tf
    import pandas as pd
    import numpy as np
    from keras.layers import IntegerLookup
    from matplotlib import pyplot as plt
    import seaborn as sns
    from keras import backend as K
    import math
    from keras.models import Model
    import scann
```

Figure 2: Libraries

```
from google.colab import auth
auth.authenticate_user()
```

```
!echo "deb http://packages.cloud.google.com/apt gcsfuse-bionic main" > /etc/apt/sources.list.d/gcsfuse.list
!curl https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key add -
!apt -qq update
!apt -qq install gcsfuse
```

```
!mkdir myfolder
!gcsfuse --implicit-dirs datarepo123 myfolder
```

```
!ls
```

```
print("TensorFlow version:", tf.__version__)
```

```
import pandas as pd
import io
pd.set_option('display.max_colwidth', -1)

df = pd.read_csv('myfolder/amazon_reviews.txt', sep=" ")
```

Figure 3: Data loading

```
fig, axes = plt.subplots(ncols=2, figsize=(17, 4), dpi=100)
plt.tight_layout()

df.groupby('RATING').count()['DOC_ID'].plot(kind='pie', ax=axes[0],
                                             labels=['Rating 1(8%)', 'Rating 2 (6%)',
                                                     'Rating 3 (9%)','Rating 4 (19%)','Rating 5 (58%)'])
sns.countplot(x=df['RATING'], hue=df['RATING'], ax=axes[1])

axes[0].set_ylabel('')
axes[1].set_ylabel('')
axes[1].set_xticklabels(['Rating 1', 'Rating 2','Rating 3','Rating 4','Rating 5'])
axes[0].tick_params(axis='x', labelsize=15)
axes[0].tick_params(axis='y', labelsize=15)
axes[1].tick_params(axis='x', labelsize=15)
axes[1].tick_params(axis='y', labelsize=15)

axes[0].set_title('Target Distribution', fontsize=13)
axes[1].set_title('Target Count', fontsize=13)

plt.show()
```

Figure 4: Data exploration

```
plt.show()
```

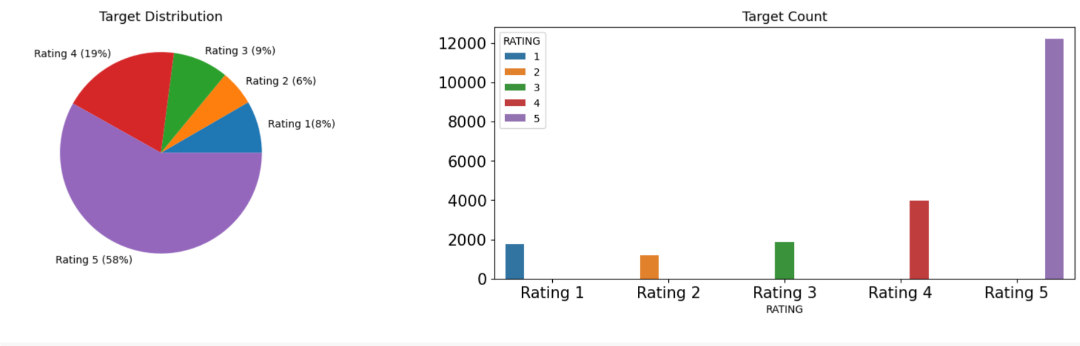

Figure 5: Plot

The length of reviews were analyzed. Data exploration illustrate count of reviews exceeding length 1500 in Figure 6

```
[12] review=df_1['REVIEW_TEXT'].values
     result = [len(sentence.split()) for sentence in review]

[13] j2 = [i for i in result if i >= 1500]
     j2

     [1632, 1614, 2805]
```

Figure 6: Data exploration

6 Data cleaning

The data was cleaned after removal of punctuation's and HTML tags in Figure 7

```
[10] df_1=df[['RATING','REVIEW_TEXT']]

[11] df_1['REVIEW_TEXT'] = df_1['REVIEW_TEXT'].str.replace(r'<[^<>]*>', '', regex=True)

     <ipython-input-11-df37f71cb972>:1: SettingWithCopyWarning:
     A value is trying to be set on a copy of a slice from a DataFrame.
     Try using .loc[row_indexer,col_indexer] = value instead

     See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ret
       df_1['REVIEW_TEXT'] = df_1['REVIEW_TEXT'].str.replace(r'<[^<>]*>', '', regex=True)
```

Figure 7: Data cleaning

7 Label encoder and text vectorization

The model parameters were assigned with corresponding value. The dataframe was converted into dataset and split into training and test dataset. The training dataset were used for encoding target variable in Figure 8

```
[16] ds=tf.data.Dataset.from_tensor_slices((dict(df_1[['REVIEW_TEXT']]),dict(df_1[['RATING']])))
```

```
[17] lim=df.shape[0]
     lim_bound=(lim/32)*0.8
     print(lim_bound)

     525.0
```

```
[18] train_ds=ds.batch(32).take(525)
     test_ds=ds.batch(32).skip(525)
```

```
[19] len(test_ds)

     132
```

```
[20] score_onehot=IntegerLookup(output_mode="one_hot")
     score_ds=train_ds.map(lambda x,y: y['RATING'])
     score_onehot.adapt(score_ds)

     TARGET=len(score_onehot.get_vocabulary())
     print(TARGET)
```

Figure 8: Label encoding

The dataframe was converted into dataset using preprocessing function which also create dataset in batches and shuffled it and shown in in Figure 9

```
[21] def preprocess_dataset(series,batch_size,shuffle_buffer_size):
       dataset=tf.data.Dataset.from_tensor_slices((dict(df_1[['REVIEW_TEXT']]),dict(df_1[['RATING']])))
       dataset=dataset.shuffle(shuffle_buffer_size,seed=1234)
       dataset=dataset.map(lambda x,y:(
           {"inputs":x['REVIEW_TEXT']},
           {"rating":score_onehot(y['RATING'])}
       ))
       dataset=dataset.batch(batch_size)
       return dataset
```

```
[22] train_ds=preprocess_dataset(train,batch_size=32,shuffle_buffer_size=512)
     test_ds=preprocess_dataset(test,batch_size=32,shuffle_buffer_size=512)
```

Figure 9: Dataset transformation

Now, text vectorization function was built for transformation of comments into its tokens as shown in Figure 10

8 Model creation and metrics

Model template was created and is shown in Figure 11

Training of model was achieved using fit method and displayed in Figure 12

Accuracy and loss were ploted and displayed in Figure 13 and Figure 14 respectively.

9 Transforming reviews into vectors

Embedding layer was extracted from the model, and this layer was used to convert reviews into embedding vectors in Figure 15

```
text_layer=tf.keras.layers.TextVectorization(max_tokens=VOCABULARY,output_sequence_length=MAXLEN,standardize='lower_and_strip_punctuation',
                                             split='whitespace',output_mode='int')
train_text=train_ds.map(lambda x, y:x['inputs'])
text_layer.adapt(train_text)
```

Figure 10: Text vectorization

```
def create_model():
  raw_str=tf.keras.Input(shape=(1,),dtype=tf.string,name='inputs')
  vectorize_layer=text_layer(raw_str)
  concat_feature=tf.keras.layers.Embedding(VOCABULARY,EMBEDDING_OUT,mask_zero=True,
                                         name='sending_embedding_layer')(vectorize_layer)
  concat_1_layer=tf.keras.layers.LSTM(units=180,activation='tanh',return_sequences=True)(concat_feature)
  concat_2_layer=tf.keras.layers.LSTM(units=280,activation='tanh')(concat_1_layer)
  op=tf.keras.layers.Dense(TARGET,activation='softmax',name='rating')(concat_2_layer)
  model=tf.keras.Model(inputs=[raw_str],outputs=[op])
  model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001,clipvalue=0.5),
              loss={'rating':tf.keras.losses.categorical_crossentropy},metrics='accuracy')
  return model
```

Figure 11: LSTM model using embedding layer

```
model=create_model()
history=model.fit(train_ds,epochs=5,validation_data=test_ds)

Epoch 1/5
657/657 [==============================] - 149s 211ms/step - loss: 0.8742 - accuracy: 0.6789 - val_loss: 0.9887
Epoch 2/5
657/657 [==============================] - 90s 137ms/step - loss: 0.7132 - accuracy: 0.7177 - val_loss: 0.7893
Epoch 3/5
657/657 [==============================] - 76s 116ms/step - loss: 0.5959 - accuracy: 0.7643 - val_loss: 0.7350
Epoch 4/5
657/657 [==============================] - 73s 111ms/step - loss: 0.5063 - accuracy: 0.8020 - val_loss: 0.6783
Epoch 5/5
657/657 [==============================] - 64s 97ms/step - loss: 0.5111 - accuracy: 0.8066 - val_loss: 0.6560 -
```

Figure 12: Model Fit

```
[ ]  plt.plot(history.history['accuracy'])
     plt.plot(history.history['val_accuracy'])
     plt.title('Accuracy of the model')
     plt.ylabel('accuracy')
     plt.xlabel('epoch')
     plt.legend(['train', 'val'], loc='upper left')
     plt.show()
```

Figure 13: Accuracy plot

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Loss value of the model')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```

Figure 14: Loss plot

```
embed = Model(inputs=model.get_layer(name="inputs").input,
              outputs=model.get_layer(name="sending_embedding_layer").output)


def encoded_data(ip):
    r = embed.predict([ip])
    #encode1 = tf.nn.l2_normalize( r)
    return r

df['embeded']=df['REVIEW_TEXT'].apply(lambda x:encoded_data(str(x)))

df.to_json("encodings.json", orient='records')
```

Figure 15: Embedding vectors

10 Implementation of scann

This phase was executed in different script called Scann.ipynb The json stored from above script was loaded and the packages were imported initially in Figure 16

```
pip install scann
```

```
import tensorflow as tf
import scann
import pandas as pd
import scann
import numpy as np
```

```
from google.colab import auth
auth.authenticate_user()
```

```
!echo "deb http://packages.cloud.google.com/apt gcsfuse-bionic main" > /etc/apt/sources.list.d/gcsfuse.list
!curl https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key add -
!apt -qq update
!apt -qq install gcsfuse
```

```
!mkdir myfolder
!gcsfuse --implicit-dirs datarepo123 myfolder
```

Figure 16: Loading packages

The loaded file was read using pandas package and the DOC_ID acts as identifier was used to identify the reviews. The list of DOC_ID was collected in target variable in Figure 17

```
import pandas as pd
import io
pd.set_option('display.max_colwidth', -1)

encodings = pd.read_json('myfolder/encodings.json')
```

```
targets = encodings['DOC_ID'].values.tolist()
targets = list(set(targets))
```

Figure 17: Reading encodings

11 Scann Model

Encodings of each review were collected in an array in Figure 18

Scann model was built using the dot product parameter in Figure 19

12 Search Function

Filter_neigbor function was created to calculate the distance between each reviews and the comments having distance greater than 200 were captured in Figure 20

```
e_arr = []
x, y = len(encodings.embeded[0]), len(encodings.embeded[0][0])
for i in range(encodings.shape[0]):
    e_arr.append(encodings.embeded[i] )
```

Figure 18: Encodings

```
# simpler tensor
embedding_raw = tf.convert_to_tensor(e_arr, name = "embedding_raw", dtype=tf.float32)

embedding = tf.reshape(embedding_raw, name = "embedding", shape=(encodings.shape[0], x*y*128))

searcher = scann.scann_ops.builder(embedding, 10, "dot_product").score_brute_force(True).build()

# save searcher
scann_module = searcher.serialize_to_module()
```

Figure 19: Scann model

```
def filter_neighbours(t):
    try:
        compare_with = t
        query = encodings[encodings.DOC_ID == compare_with].index[0]
        query_vector = embedding[query]
        neighbors, distances = searcher.search(query_vector, final_num_neighbors = 20)
        neighbors = neighbors.numpy().tolist()
        distances = distances.numpy().tolist()
        r = []
        if len(distances) > 0:
            for n, d in zip (neighbors, distances):
                v = {}
                v['target'] = t
                v['neighbor'] = encodings.iloc[n]['DOC_ID']
                v['distance'] = d
                r.append(v)
        r = list(filter(lambda d: d['neighbor'] != t, r))
        r = list(filter(lambda d: d['distance'] > 200 , r))
        #r = list(filter(lambda d: d['neighbor'] not in targets, r))
        return r
    except Exception as eroare:
        print(eroare)
```

Figure 20: Filter function

Embedding vector of each comment was passed to the filter_neigbor function in Figure 22

```
ra = []
for x in targets:
    ra.append(filter_neighbours(x))

rb = []
for y in ra:
    if isinstance(y, list):
        for x in y:
            rb.append(x)

rdf = pd.DataFrame.from_dict(rb)
rdf_filtered=rdf[rdf['distance']>220].copy()

rdf_filtered.to_parquet('scanned.parquet')
```

Figure 21: Passing each vector

Raw data were loaded further to merge back with comments using DOCcolumn in Figure 22


```
ra = []
for x in targets:
    ra.append(filter_neighbours(x))

rb = []
for y in ra:
    if isinstance(y, list):
        for x in y:
            rb.append(x)

rdf = pd.DataFrame.from_dict(rb)
rdf_filtered=rdf[rdf['distance']>220].copy()

rdf_filtered.to_parquet('scanned.parquet')
```


Figure 22: Loading raw data

13 Results

The target column indicated the reviews and neighbor column captured similar reviews. The result is displayed in Figure 23

B	C	D	E	F
REVIEW_TEXT_NEIGHBOR	REVIEW_TEXT_TARGET	target	neighbor	distance
I purchased this TV last month as a	I purchased this TV last month as a bedr	3933	1746	243.5477
I purchased this TV last month as a	I purchased this TV last month as a bedr	4021	1746	243.5565
I am reviewing the Wubble ball on	I am reviewing the Wubble ball on two l	3416	1973	217.7109
It's a piece of junk! My son had see	It's a piece of junk! My son had see	3879	2335	212.9282
Just like everyone else, my bulb bu	Just like everyone else, my bulb burnt ou	3007	2470	210.7244
Just like everyone else, my bulb bu	Just like everyone else, my bulb burnt ou	3057	2470	211.8583
Just like everyone else, my bulb bu	Just like everyone else, my bulb burnt ou	2470	3007	210.725
Just like everyone else, my bulb bu	Just like everyone else, my bulb burnt ou	3057	3007	210.725
Just like everyone else, my bulb bu	Just like everyone else, my bulb burnt ou	2470	3057	211.8583
Just like everyone else, my bulb bu	Just like everyone else, my bulb burnt ou	3007	3057	210.7244
I am reviewing the Wubble ball on	I am reviewing the Wubble ball on two l	1973	3416	217.7109
It's a piece of junk! My son ha	It's a piece of junk! My son had seen the	2335	3879	212.928
I purchased this TV last month as a	I purchased this TV last month as a bedr	1746	3933	243.5478
I purchased this TV last month as a	I purchased this TV last month as a bedr	4021	3933	243.5847
I purchased this TV last month as a	I purchased this TV last month as a bedr	1746	4021	243.557
I purchased this TV last month as a	I purchased this TV last month as a bedr	3933	4021	243.5852

Figure 23: Results

References

1 (n.d.).
URL: *https://www.tensorflow.org/text/tutorials/text_classification_rnn*