Configuration Manual

MSc Research Project
Data Analytics

Rohit Kumar Shrivas
Student ID: x19226403

School of Computing
National College of Ireland

Supervisor: Hicham Rifai

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Rohit Kumar Shrivas
Student ID:	x19226403
Programme:	Data Analytics
Year:	2021
Module:	MSc Research Project
Supervisor:	Hicham Rifai
Submission Due Date:	16/08/2021
Project Title:	Configuration Manual
Word Count:	621
Page Count:	8

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:	Rohit Kumar Shrivas
Date:	16th August 2021

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

Rohit Kumar Shrivas
x19226403

1 Introduction

The main objective of this manual is to document the procedure and configuration of hardware utilized to provide a guideline to any user willing to produce the desired outcomes. This manual consists of code snippets used for exploratory data analysis, data preparation, model creation etc. alongside hardware and software specifications of the machine utilized for the implementation of this research project. The structure of this technical manual is as follows: Chapter 2, highlighting the hardware and software configuration requirements, Chapter 3, discusses the data collection technique adopted, chapter 4, sheds light on exploratory data analysis conducted over the data, and then chapter 5 discusses the implementation aspect of all the models.

2 Environment

2.1 Configuration of Hardware and Software

The hardware configuration of the machine utilized for the implementation of this research project is shown in Figure 1. It has Intel's i5 10th generation core with 2.11 GHz clock, 64-bit Windows 10 OS installed with 8 GB of onboard RAM.

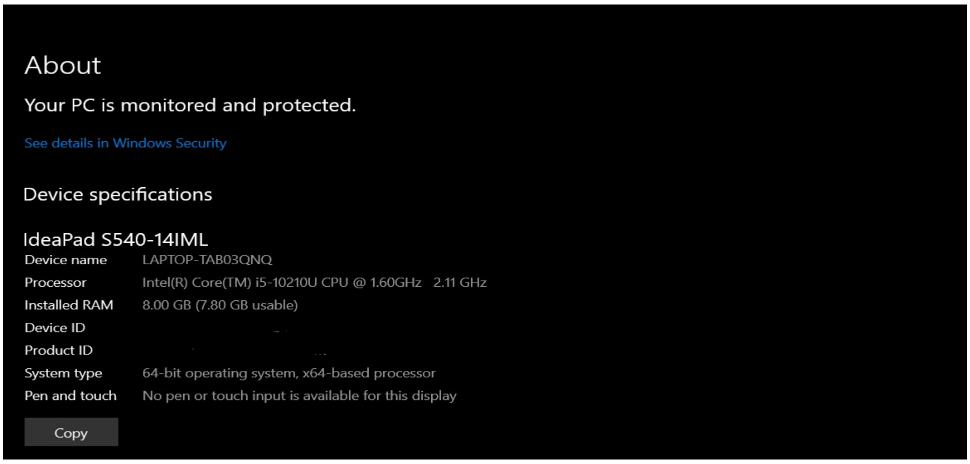

Figure 1: Hardware configuration of systemutilized for implementation of this research project

For the implementation of this project, Google Colaboratory'cloud based Jupyter notebooks have been utilized. For EDA purposes the default resource i.e., 12.69 GB of

RAM and 107.72 GB of Disk space is utilized. However, for RNN based Deep Learning Models, Graphics Processing Unit (GPU) accelerators have been utilized and for BERT based models, Tensor Processing Unit (TPU) have been utilized.

3 Collection of Data

The dataset has been downloaded from Kaggle which is an open platform. The link to the dataset is `https://www.kaggle.com/c/quora-insincere-questions-classification/data?select=train.csv`. This data is then uploaded into drive and the code is shown in Figure 2is utilized to upload it into Google's Colab instance.

```
# Authenticate and create the PyDrive client.
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)
```

```
link_train = 'https://drive.google.com/file/d/1ktAT2mqPsRe2WmImqMIfFJkIvUwe_bMN/view?usp=sharing' # The shareable link
```

```
train_id = '1ktAT2mqPsRe2WmImqMIfFJkIvUwe_bMN'
```

```
downloaded = drive.CreateFile({'id':train_id})
downloaded.GetContentFile('train.csv')
train_df = pd.read_csv('train.csv')
```

Figure 2: Code for uploading data into Google Colaboratory instance from Drive

4 Exploratory Data Analysis

EDA was carried out to understand the flow of data. A pie chart has been plotted based on the classification category to understand the structure of data.

```
train_df['target'].value_counts().plot(kind = 'pie', labels = ['Sincere', 'Insincere'],
    startangle = 90, autopct = '%1.0f%%')
```

Figure 3: Classification of data based on categories

Most frequent words found in both categories of data are then plotted along with wordcloud.

```python
# convert huge strings to lists
Sincere = Sincere.split(' ')
Insincere = Insincere.split(' ')

# get the 15 most common words and their frequency values of each group using Counter
Sincere_count = Counter(Sincere).most_common(15)
Insincere_count = Counter(Insincere).most_common(15)

# function to generate input for plotting
def get_words_and_values(counter):
    words = []
    values = []
    for i in range(0, len(counter)):
        words.append(counter[i][0])
        values.append(counter[i][1])
    return words, values

# simple plot function
def freq_plot(words, values, tit = ''):
    plt.bar(words, values)
    plt.ylabel('Absolute Frequency')
    plt.title(tit)
    plt.xticks(rotation = 45)

# call the functions for SIncere questions first
words, values = get_words_and_values(Sincere_count)
freq_plot(words, values, tit = 'Frequency of the 15 most common words in Sincere questions')
```

Figure 4: Most frequent words appearing in sincere category

```python
Sincere = train_df.loc[train_df['target'] == 0]
Insincere = train_df.loc[train_df['target'] == 1]

Sincere = ' '.join(question for question in Sincere['question_text'].astype(str))
Insincere = ' '.join(question for question in Insincere['question_text'].astype(str))

wordcloud = wc.WordCloud(max_font_size = 160, max_words = 150,
                         background_color = 'black', width = 800,
                         height = 500).generate(Sincere)
# plot it
plt.figure(figsize = (15, 10))
plt.imshow(wordcloud, interpolation = 'bilinear')
plt.axis("off")
plt.show()
```

Figure 5: Generating WordCloud of most frequent words appearing in sincere category

5 Implementation of Models

5.1 RNN based Deep Learning Models

Figure 6 shows the import of required libraries.

```
import re
import numpy as np
import pandas as pd

from collections import Counter
from afinn import Afinn
from nltk.corpus import sentiwordnet as swn
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer

from tqdm import tqdm
import math
from sklearn.model_selection import train_test_split
from sklearn import metrics

from keras.preprocessing.text import Tokenizer
from tensorflow.keras.utils import to_categorical
from keras.preprocessing.sequence import pad_sequences
from keras.layers import CuDNNLSTM, CuDNNGRU
from keras.layers import Bidirectional, GlobalMaxPool1D, GlobalAveragePooling1D
from keras.layers import Input, Embedding, Dense, Conv1D, Conv2D, MaxPool2D, concatenate, Reshape, Flatten, Concatenate, Dropout
from keras.models import Model, Sequential
from keras import activations, initializers, regularizers, constraints, optimizers, layers
from keras.optimizers import Adam
from keras import backend as K
from keras.engine.topology import Layer
from keras.callbacks import *
```

Figure 6: Libraries required by RNN based Deep Learning Models

Figure 7 shows the code utilized in splitting the data into train and validation set in the ratio of 80:20.

```
## Splitting the data into training and validation set in 80:20 ratio.
train_set,validation_set=train_test_split(data_train,test_size=0.2,random_state=2018)
print(train_set.shape)
print(validation_set.shape)
```

```
(1044897, 3)
(261225, 3)
```

Figure 7: Splitting the data into train and validation

Figure 8 shows the code utilized for filling of '_NA_' values.

```
## fill up the missing values
train_X = data_train["question_text"].fillna("_na_").values
validation_X = data_validation["question_text"].fillna("_na_").values
```

Figure 8: Filling '_NA_' values

Figure 9 shows the code utilized for tokenization of sentences.

```
## Tokenize the sentences
tokenizer = Tokenizer(num_words=max_features)
tokenizer.fit_on_texts(list(train_X))
train_X = tokenizer.texts_to_sequences(train_X)
validation_X = tokenizer.texts_to_sequences(validation_X)
```

Figure 9: Tokenizing the sentences

Figure 10 shows the code utilized for padding of sentences.

```
## Pad the sentences
train_X = pad_sequences(train_X, maxlen=maxlen)
validation_X = pad_sequences(validation_X, maxlen=maxlen)
```

Figure 10: Padding the sentences

Figure 11 shows the code utilized for prediction of RNN based Deep learning Models.

```
predict_validation_y = model_4.predict([validation_X], batch_size=256, verbose=1)
for thresh in np.arange(0.1, 0.501, 0.01):
    thresh = np.round(thresh, 2)
    print("F1 score at threshold {0} is {1}".format(thresh, metrics.f1_score(validation_y, (predict_validation_y>thresh).astype(int))))
```

Figure 11: fnal classification prediction through RNN based Deep Learning models

5.2 Transformers based Models

Figure 12 shows the import of required libraries.

```
import re
import numpy as np
import pandas as pd
from collections import Counter
from afinn import Afinn
from nltk.corpus import sentiwordnet as swn
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer

from tokenizers import BertWordPieceTokenizer
from sklearn.model_selection import train_test_split
from sklearn import metrics

import tensorflow as tf
from tensorflow.keras.layers import Dense, Input
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.models import Model
from tensorflow.keras.callbacks import ModelCheckpoint
import transformers
from tqdm.notebook import tqdm
from tokenizers import BertWordPieceTokenizer
import sentencepiece
```

Figure 12: Libraries required by Transformers based Models

Figure 13 shows the function created to fast tokenize the data.

```
maxlen=192
chunk_size=256
def fast_encode(texts, tokenizer, chunk_size=chunk_size, max_length=maxlen):
    tokenizer.enable_truncation(max_length=maxlen)
    tokenizer.enable_padding(length=maxlen)
    all_ids = []
    #sliding window methodology
    for i in tqdm(range(0, len(texts), chunk_size)):
        text_chunk = texts[i:i+chunk_size].tolist()
        encs = tokenizer.encode_batch(text_chunk)
        all_ids.extend([enc.ids for enc in encs])

    return np.array(all_ids)
```

Figure 13: Function for fast tokenization of data

Figure 14 shows the function created to build the transformer model.

```
def build_model(transformer, max_len=512):
    input_word_ids = Input(shape=(max_len,), dtype=tf.int32, name="input_word_ids")
    #Replaced from the Embedding+LSTM/CoNN layers
    sequence_output = transformer(input_word_ids)[0]
    cls_token = sequence_output[:, 0, :]
    out = Dense(1, activation='sigmoid')(cls_token)

    model = Model(inputs=input_word_ids, outputs=out)
    model.compile(Adam(learning_rate=1e-5), loss='binary_crossentropy', metrics=['accuracy'])
    model.summary()
    return model
```

Figure 14: Function for building transformer model

Figure 15 shows the code to detecting the no. of TPU clusters available.

```
try:
    tpu = tf.distribute.cluster_resolver.TPUClusterResolver()
    print('Running on TPU ', tpu.master())
except ValueError:
    tpu = None

if tpu:
    tf.config.experimental_connect_to_cluster(tpu)
    tf.tpu.experimental.initialize_tpu_system(tpu)
    strategy = tf.distribute.experimental.TPUStrategy(tpu)
else:
    strategy = tf.distribute.get_strategy()

print("REPLICAS: ", strategy.num_replicas_in_sync)
```

Figure 15: Detecting th no. of TPU clusters available

Figure 16 shows the code which Tokenize the data using fast tokenizer function.

```
[ ]
#Tokenizing the samples

train_x = fast_encode(train_set['question_text'].astype(str), fast_tokenizer_distilbert, max_length=MAX_LEN)
val_x = fast_encode(validation_set['question_text'].astype(str), fast_tokenizer_distilbert, max_length=MAX_LEN)
train_y = train_set['target'].values
val_y = validation_set['target'].values
print(train_x.shape)
print(train_y.shape)
print(val_x.shape)
print(val_y.shape)
```

Figure 16: Function for fast tokenization of data

Figure 17 shows the code utilized in converting data into tensorflow compatible format.

```
#Converting datasets in order to make it compataible with Tensorflow

train_dataset = (
    tf.data.Dataset
    .from_tensor_slices((train_x, train_y))
    .repeat()
    .shuffle(2048)
    .batch(BATCH_SIZE)
    .prefetch(AUTO)
)

valid_dataset = (
    tf.data.Dataset
    .from_tensor_slices((val_x, val_y))
    .batch(BATCH_SIZE)
    .cache()
    .prefetch(AUTO)
)
print(train_dataset)
print(valid_dataset)
```

Figure 17: Converting the data into tensorflow compatible format

Figure 18 shows the code utilized for prediction of Transformers based Models.

```
pred_val_y = DistilBERT_model.predict(val_x, batch_size=BATCH_SIZE)
for thresh in np.arange(0.1, .701, 0.01):
    thresh = np.round(thresh, 2)
    print("F1 score at threshold {0} is {1}".format(thresh, metrics.f1_score(val_y, (pred_val_y>thresh).astype(int))))
```

Figure 18: Final classification prediction through Transformers based models