Configuration Manual

MSc Research Project
MSc in Data Analytics

Sayali Kule
Student ID: 21101205

School of Computing
National College of Ireland

Supervisor: Abubakr Siddig

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Sayali Kule
Student ID:	21101205
Programme:	MSc in Data Analytics
Year:	2022
Module:	MSc Research Project
Supervisor:	Abubakr Siddig
Submission Due Date:	15/08/2022
Project Title:	Configuration Manual
Word Count:	1079
Page Count:	26

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:	Sayali Kule
Date:	18th September 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

Sayali Kule
21101205

1 Introduction

This configuration document provides all the necessary information used in the research study of Intent classification. The implementation was carried out on a standard laptop. This manual provides the instructions to replicate the thesis work step by step. All the artifacts attached along with the report are explained here. This manual covers code snippets from data collection, model building, experiments, and result evaluation.

2 Specification Details

2.1 Hardware Specification

The research was performed on a standard personal laptop, MacBook Air. The device specifications are given below in figure Figure 1.

Figure 1: Device Specification

2.2 Software Specification

All the code artefacts were written in python language and run using Google Collab. The data collected was stored on Gdrive. The collab uses its own cloud server to execute the code and the collab notebooks are stored on Gdrive. The collab specifications are given below in the figure.

```
Filesystem      Size  Used Avail Use% Mounted on
overlay         108G   38G   71G  35% /
tmpfs            64M     0   64M   0% /dev
shm             5.8G     0  5.8G   0% /dev/shm
/dev/root       2.0G  1.2G  812M  59% /sbin/docker-init
tmpfs           6.4G   24K  6.4G   1% /var/colab
/dev/sda1        81G   41G   41G  51% /etc/hosts
tmpfs           6.4G     0  6.4G   0% /proc/acpi
tmpfs           6.4G     0  6.4G   0% /proc/scsi
tmpfs           6.4G     0  6.4G   0% /sys/firmware
```

```
!cat /proc/meminfo

MemTotal:        13298580 kB
MemFree:         10893700 kB
MemAvailable:    12511516 kB
Buffers:           104112 kB
Cached:           1680664 kB
SwapCached:             0 kB
Active:            927412 kB
Inactive:         1288644 kB
Active(anon):      398368 kB
Inactive(anon):       460 kB
Active(file):      529044 kB
Inactive(file):   1288184 kB
Unevictable:            0 kB
Mlocked:                0 kB
SwapTotal:              0 kB
SwapFree:               0 kB
Dirty:                652 kB
Writeback:              0 kB
AnonPages:         431276 kB
Mapped:            226500 kB
Shmem:               1200 kB
KReclaimable:       84076 kB
Slab:              125884 kB
SReclaimable:       84076 kB
SUnreclaim:         41808 kB
KernelStack:         4896 kB
PageTables:          5988 kB
NFS_Unstable:           0 kB
Bounce:                 0 kB
WritebackTmp:           0 kB
CommitLimit:      6649288 kB
Committed_AS:     2986040 kB
```

Figure 2: Google Collabe Specification

3 Data Collection

This research made use of an intent classification dataset created by Larson et al. (2019). This dataset contains data .JSON file. The dataset consists of user text queries and intents. The dataset is available at the below-mentioned link. The size of the dataset is 2.5MB.

CLINC Dataset : https://aclanthology.org/D19-1131/

The dataset was uploaded on Gdrive to be used in code. To mount the GDrive on collab following code was used.

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

Mounted at /content/gdrive
```

4 Exploratory Data Analysis

The code for exploratory Data Analysis is given in artefact 'RIC_DATA_EDA_Preprocessing.ipynb'.

The .Json file was imported into the Collab notebook. Required python libraries to perform the exploratory data analysis were loaded. The JSON data was stored in train and test dataframes. The below figure shows the required python library.

```
#Import required libraries
import pandas as pd
import numpy as np
import json
import plotly.express as px
import string
import nltk
import matplotlib.pyplot as plt
import pandas as pd
from wordcloud import WordCloud, STOPWORDS
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.feature_extraction.text import CountVectorizer
```

Figure 3: Required Libraries

The number of records in each dataframe are printed below.

```
[10] #Print the number of record
     print('train data:' ,len(data['train']))
     print('test data:' ,len(data['test']))

     train data: 15000
     test data: 4500
```

There were 150 unique intents in the train and test dataframe. The training dataset had 100 records per intent wherein the test data had 30 records.

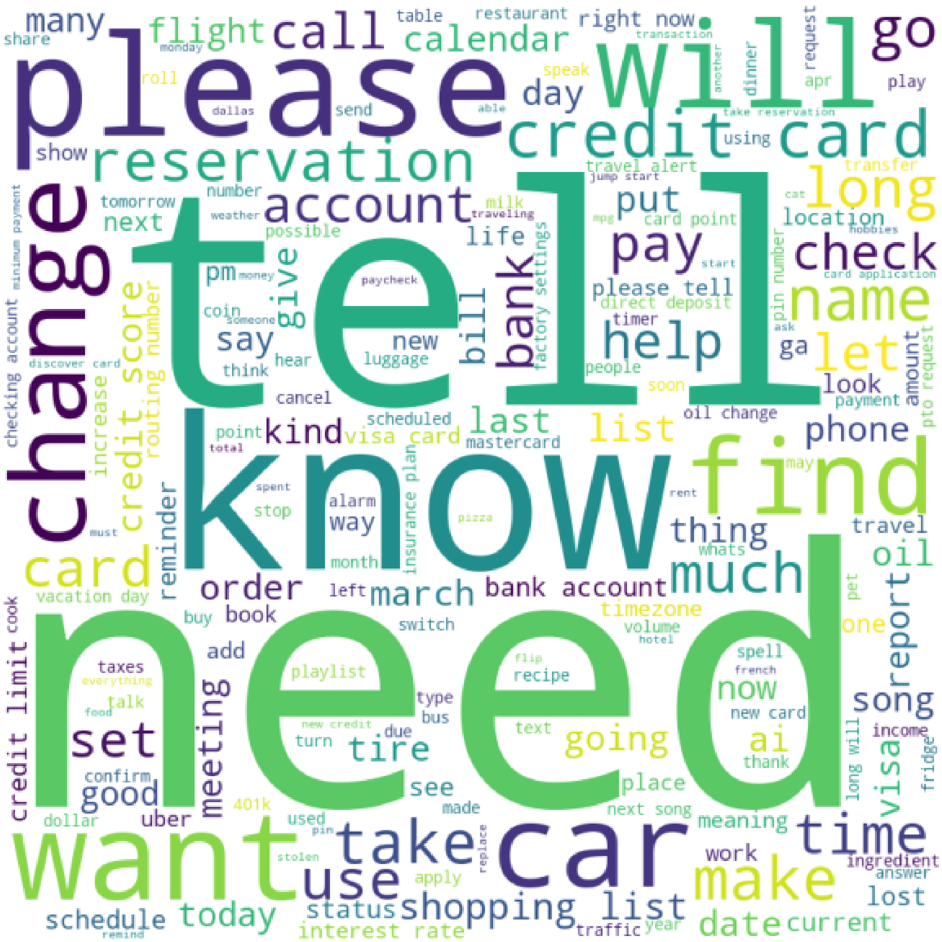
```
#Unique intent list and count
df_train['Intent'].value_counts()

translate          100
order_status       100
goodbye            100
account_blocked    100
what_song          100
                  ...
reminder           100
change_speed       100
tire_pressure      100
no                 100
card_declined      100
Name: Intent, Length: 150, dtype: int64
```

(a) Train Data

```
df_test['Intent'].value_counts()

translate          30
order_status       30
goodbye            30
account_blocked    30
what_song          30
                   ..
reminder           30
change_speed       30
tire_pressure      30
no                 30
card_declined      30
Name: Intent, Length: 150, dtype: int64
```

(b) Test Data

A wordcloud of the input text was plotted using Wordcloud library. It highlighted significant data points from text. The plotted wordcloud is given below.



The dataset was checked for occurrence of Email , URL, any mentions, any special character. The results are given below.

```
text_at_a_glance(df_train.Text)

Data has 0 rows with emails
Data has 0 rows with urls
Data has 0 rows with mentions
Data has 1 rows with hashtags
```

Bigrams are two words when used together create a distinct meaning. Similarly, trigrams are three words that create a distinct meaning when used together. The below two figures shows the Bigrams and Trigrams calculated from the training dataset.

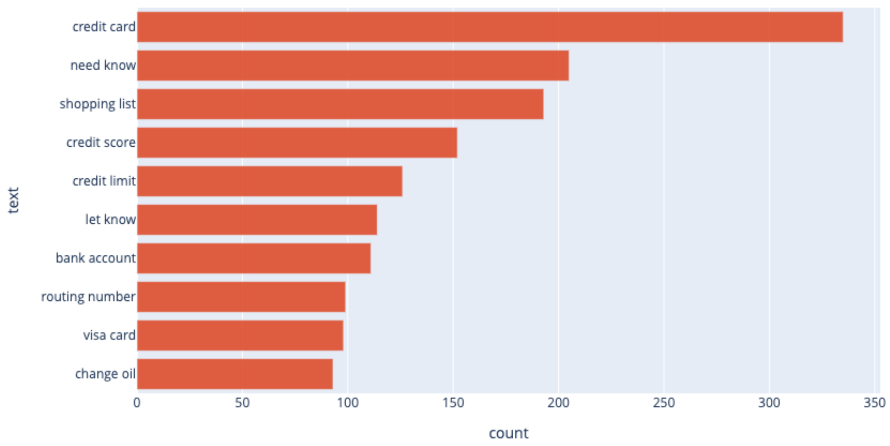


Figure 5: Bigrams

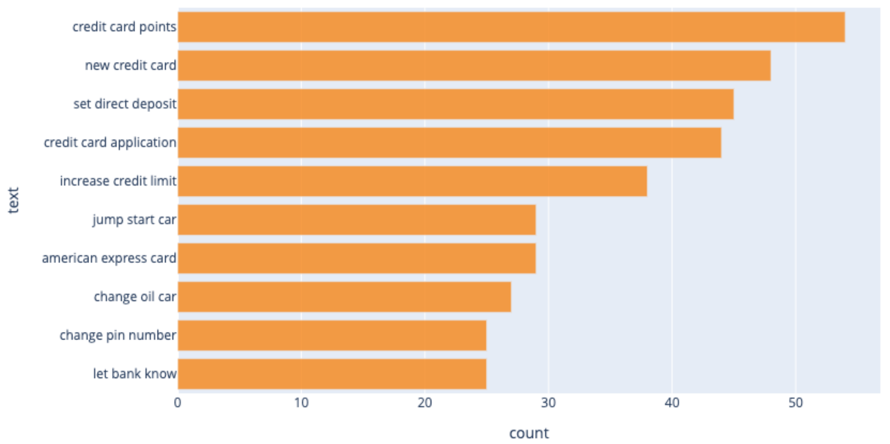


Figure 6: Trigrams

5 Data Preprocessing

The code for data preprocessing is given in artefact 'RIC_DATA_EDA_Preprocessing.ipynb'.

Text pre-processing was done using existing python libraries. The text data was clean, processed, and made ready for modeling. The steps followed and the code required to perform the steps are mentioned below. The output of these steps is given at the end.

5.1 Punctuation Removal

As observed in text analysis, one of the records contains a hashtag. To get rid of the symbols following code was executed.

Punctuation Removal

```
def remove_punctuation(text):
    punctuationfree="".join([i for i in text if i not in string.punctuation])
    return punctuationfree
```

```
[25] df_train['clean_text']= df_train['Text'].apply(lambda x:remove_punctuation(x))
     df_train.head()
```

Figure 7: Code for Punctuation Removal

5.2 Lower Casing

The user query text for correctly classifying intent was first pre-processed by converting it in lower case. The function used here was str.lower()

Lower Case the text

```
[27] df_train['clean_text']= df_train['clean_text'].str.lower()
     df_train.head()
```

Figure 8: Code to Lower Case the text

5.3 Stop Word Removal

The stop words are low information words e,g ‘a’, ’an’, ‘the’. These are commonly used words in a language. The removal of additional low information words allows focusing on important words. A predefined list of stopwords is available or it can also be customized. The function to remove the stop word is given below.

Remove the stop words

```
[33] import nltk
     nltk.download('stopwords')
     stopwords = nltk.corpus.stopwords.words('english')

     [nltk_data] Downloading package stopwords to /root/nltk_data...
     [nltk_data]   Unzipping corpora/stopwords.zip.
```

```
def remove_stopwords(text):
    output= [i for i in text if i not in stopwords]
    return output
```

```
[35] df_train['text_tokenised_no_stop']= df_train['text_tokenised'].apply(lambda x:remove_stopwords(x))
     df_train.head()
```

Figure 9: Code to Remove the stop word

5.4 Tokenisation

This process splits the text into tokens by a set of rules. The BERT tokenizer was downloaded from the library and further used to tokenize, mask and padding. The code and the results are mentioned below. This was taken care before training the model

```
#Download BERT and tokenizer
# Hugging-face tokenizer library.
transformer = AutoModel.from_pretrained(model_name)
tokenizer = AutoTokenizer.from_pretrained(model_name)
```

```
Downloading: 100% 570/570 [00:00<00:00, 16.7kB/s]
Downloading: 100% 436M/436M [00:16<00:00, 28.5MB/s]
Downloading: 100% 213k/213k [00:00<00:00, 663kB/s]
Downloading: 100% 436k/436k [00:00<00:00, 679kB/s]
```

```
# Tokenization : Hugginface BERT tokenizer.encode()
for (text, label_mask) in examples:
  encoded_sent = tokenizer.encode(text[0], add_special_tokens=True, max_length=max_seq_length, padding="max_length", truncation=True)
  input_ids.append(encoded_sent)
  #print(label_map[text[1]])
  label_id_array.append(label_map[text[1]])
  label_mask_array.append(label_mask)

# Attention to token (to ignore padded input wordpieces)
for sent in input_ids:
  att_mask = [int(token_id > 0) for token_id in sent]
  input_mask_array.append(att_mask)
# Convertion to Tensor
input_ids = torch.tensor(input_ids)
input_mask_array = torch.tensor(input_mask_array)
label_id_array = torch.tensor(label_id_array, dtype=torch.long)
label_mask_array = torch.tensor(label_mask_array)

# Building the TensorDataset
dataset = TensorDataset(input_ids, input_mask_array, label_id_array, label_mask_array)

if do_shuffle:
  sampler = RandomSampler
else:
  sampler = SequentialSampler

# Building the DataLoader
return DataLoader(
```

Figure 10: Code for Tokeniser

5.5 Data Split

The training data was split into different files based on the number of records. The below function was created to split the data as per choice and save the data in a different folder on GDrive. This would be used while implementation. Test file was added to all the folders.

Split the data

```
def split_labeled(labeled_prop,unlabeled_prop):
    test_size = unlabeled_prop/ (labeled_prop + unlabeled_prop)
    df_labeled, df_unlabeled = train_test_split(df_train,stratify=df_train['Intent'], test_size=test_size, random_state=42)

    df_labeled = df_labeled.reset_index(drop=True)
    df_unlabeled = df_unlabeled.reset_index(drop=True)
                                                (parameter) stratify: Series | Any | DataFrame | None
    with open(f'/content/gdrive/MyDrive/RIC DATA/{labeled_prop}_{unlabeled_prop}/labeled.tsv','a+') as f_out:
        f_out.write('fine_label utterance'+'\n')
    with open(f'{labeled_prop}_{unlabeled_prop}/labeled.tsv','a+') as f_out:
        for i in range(len(df_labeled)):
            line = ' '.join([df_labeled.loc[i,'Intent'], df_labeled.loc[i,'Text']])
            f_out.write(line+'\n')

    with open(f'/content/gdrive/MyDrive/RIC DATA/{labeled_prop}_{unlabeled_prop}/unlabeled.tsv','a+') as f_out:
        f_out.write('fine_label utterance'+'\n')
    with open(f'/content/gdrive/MyDrive/RIC DATA/{labeled_prop}_{unlabeled_prop}/unlabeled.tsv','a+') as f_out:
        for i in range(len(df_unlabeled)):
            line = ' '.join(['UNK_UNK', df_unlabeled.loc[i,'Text']])
            f_out.write(line+'\n')
```

```
[ ] #1st Variation - 10 records each intent labelled data, 90 records each intent unlabelled data
    split_labeled(labeled_prop=10,unlabeled_prop=90)
```

Figure 11: Function to split the data

RIC_DATA_EDA_Preprocessing.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment

+ Code + Text RAM Disk

```
#2nd Variation - 20 records each intent labelled data, 80 records each intent unlabelled data
split_labeled(labeled_prop=20,unlabeled_prop=80)
```

+ Code + Text

```
[ ] #3rd Variation - 30 records each intent labelled data, 70 records each intent unlabelled data
    split_labeled(labeled_prop=30,unlabeled_prop=70)

[ ] #4th Variation - 40 records each intent labelled data, 60 records each intent unlabelled data
    split_labeled(labeled_prop=40,unlabeled_prop=60)

[ ] #5th Variation - 50 records each intent labelled data, 50 records each intent unlabelled data
    split_labeled(labeled_prop=50,unlabeled_prop=50)

[ ] #6th Variation - 60 records each intent labelled data, 40 records each intent unlabelled data
    split_labeled(labeled_prop=60,unlabeled_prop=40)

[ ] #7th Variation - 70 records each intent labelled data, 30 records each intent unlabelled data
    split_labeled(labeled_prop=70,unlabeled_prop=30)

[ ] #8th Variation - 80 records each intent labelled data, 20 records each intent unlabelled data
    split_labeled(labeled_prop=80,unlabeled_prop=20)

#9th Variation - 90 records each intent labelled data, 10 records each intent unlabelled data
split_labeled(labeled_prop=90,unlabeled_prop=10)
```

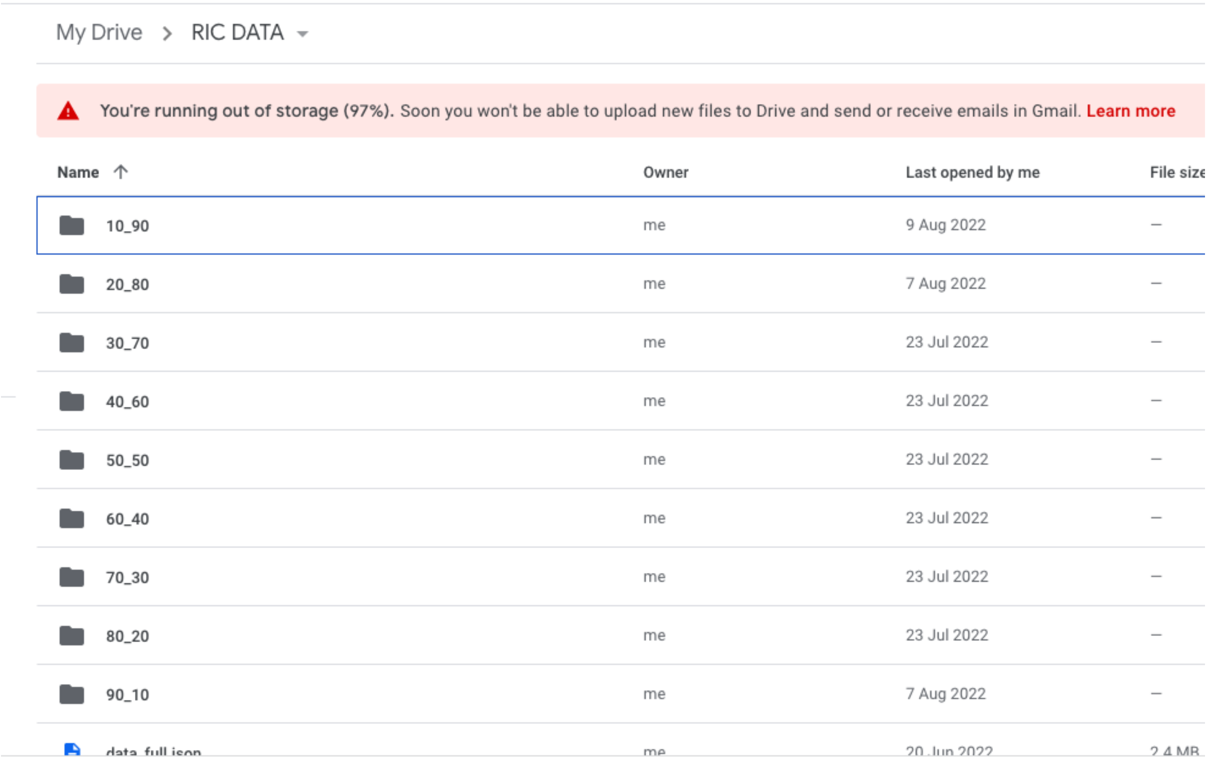


Figure 12: New data folders

6 Model Implementation

This section refers the artefact 'RIC_BERT_GAN_Model.ipynb'

The intent classification model was built using the following steps. The code used the Pytorch library to implement the BERT and GAN model. The "bert-base-cased" model was used in the implementation. It uses the data from the different GDrive folders to carry out different experiments which were mentioned in an earlier section. The training data was built using data loader defined here which takes care of tokenization. Once the model is trained, evaluation steps were executed to learn about the model's performance. The training step was repeated for every experiment and test data was evaluated.

1. Import all the required libraries. Required version of transformers==4.3.2

```python
!pip install transformers==4.3.2
import torch
import io
import torch.nn.functional as F
import random
import numpy as np
import time
import math
import datetime
import torch.nn as nn
from transformers import *
from torch.utils.data import TensorDataset, DataLoader, RandomSampler, SequentialSampler
#!pip install torch==1.7.1+cu101 torchvision==0.8.2+cu101 -f https://download.pytorch.org/whl/torch_stable.html
#!pip install sentencepiece

##Set random values
seed_val = 42
random.seed(seed_val)
np.random.seed(seed_val)
torch.manual_seed(seed_val)
if torch.cuda.is_available():
  torch.cuda.manual_seed_all(seed_val)
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting transformers==4.3.2
  Downloading transformers-4.3.2-py3-none-any.whl (1.8 MB)
     |████████████████████████████████| 1.8 MB 4.7 MB/s
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting transformers==4.3.2
  Downloading transformers-4.3.2-py3-none-any.whl (1.8 MB)
     |████████████████████████████████| 1.8 MB 4.7 MB/s
Collecting sacremoses
  Downloading sacremoses-0.0.53.tar.gz (880 kB)
     |████████████████████████████████| 880 kB 45.8 MB/s
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.7/dist-packages (from transformers==4.3.2) (4.64.0)
Requirement already satisfied: importlib-metadata in /usr/local/lib/python3.7/dist-packages (from transformers==4.3.2) (4.12.0)
Requirement already satisfied: filelock in /usr/local/lib/python3.7/dist-packages (from transformers==4.3.2) (3.7.1)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.7/dist-packages (from transformers==4.3.2) (2022.6.2)
Collecting tokenizers<0.11,>=0.10.1
  Downloading tokenizers-0.10.3-cp37-cp37m-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_12_x86_64.manylinux2010_x86_64.whl (3.3 MB)
     |████████████████████████████████| 3.3 MB 30.7 MB/s
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (from transformers==4.3.2) (2.23.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.7/dist-packages (from transformers==4.3.2) (21.3)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.7/dist-packages (from transformers==4.3.2) (1.21.6)
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-packages (from importlib-metadata->transformers==4.3.2) (3.8.1)
Requirement already satisfied: typing-extensions>=3.6.4 in /usr/local/lib/python3.7/dist-packages (from importlib-metadata->transformers==4.3.2) (4.1.1)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /usr/local/lib/python3.7/dist-packages (from packaging->transformers==4.3.2) (3.0.9)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.7/dist-packages (from requests->transformers==4.3.2) (1.24.3
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests->transformers==4.3.2) (2.10)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from requests->transformers==4.3.2) (3.0.4)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages (from requests->transformers==4.3.2) (2022.6.15)
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from sacremoses->transformers==4.3.2) (1.15.0)
Requirement already satisfied: click in /usr/local/lib/python3.7/dist-packages (from sacremoses->transformers==4.3.2) (7.1.2)
Requirement already satisfied: joblib in /usr/local/lib/python3.7/dist-packages (from sacremoses->transformers==4.3.2) (1.1.0)
Building wheels for collected packages: sacremoses
  Building wheel for sacremoses (setup.py) ... done
  Created wheel for sacremoses: filename=sacremoses-0.0.53-py3-none-any.whl size=895260 sha256=64c2d5b555e60b6e22c7185f0a914d24515741309d615b828c95fb9212d6cd
```

2. Check if the GPU is available

```python
# If there's a GPU available...
if torch.cuda.is_available():
    # Tell PyTorch to use the GPU.
    device = torch.device("cuda")
    print('There are %d GPU(s) available.' % torch.cuda.device_count())
    print('We will use the GPU:', torch.cuda.get_device_name(0))
# If not...
else:
    print('No GPU available, using the CPU instead.')
    device = torch.device("cpu")
```

```
There are 1 GPU(s) available.
We will use the GPU: Tesla T4
```

3. Below code set the parameter values required for a model such as the number of hidden layers in the generator, and discriminator, dropout rate. It also sets the path for input data.

```
#--------------------------------
#  Transformer parameters
#--------------------------------
max_seq_length = 64
batch_size = 64

#--------------------------------
#  GAN-BERT specific parameters
#--------------------------------
# number of hidden layers in the generator
num_hidden_layers_g = 1;
# number of hidden layers in the discriminator
num_hidden_layers_d = 1;
# size of the generator's input noisy vectors
noise_size = 100
# dropout to be applied to discriminator's input vectors
out_dropout_rate = 0.2

# Replicate labeled data to balance poorly represented datasets
apply_balance = True

model_name = "bert-base-cased"

labeled_file = "/content/drive/MyDrive/RIC DATA/60_40/labeled.tsv"
unlabeled_file = "/content/drive/MyDrive/RIC DATA/60_40/unlabeled.tsv"
test_filename = "/content/drive/MyDrive/RIC DATA/60_40/test.tsv"
```

4. Import the training data (labeled and unlabeled) and test data and convert them into examples using the below-mentioned function.

```
#Function to convert the example
def get_examples(input_file):
  """Creates examples for the training and dev sets."""
  examples = []

  with open(input_file, 'r') as f:
      contents = f.read()
      file_as_list = contents.splitlines()

      for line in file_as_list[1:]:
          split = line.split(" ")
          question = ' '.join(split[1:])

          text_a = question
          inn_split = split[0].split(":")
          label = inn_split[0]
          examples.append((text_a, label))
      f.close()

  return examples
```

```
[ ] #Load the examples
labeled_examples = get_examples(labeled_file)
unlabeled_examples = get_examples(unlabeled_file)
test_examples = get_examples(test_filename)
```

5. Create a list of unique intents from the labeled examples and intent 'UNK_UNK' at the end to represent unlabeled examples

```python
#find the unique labels from the intent
label_list = []
for item in labeled_examples:
    #print(str(item[1]))
      if item[1] in label_list:
        continue
        #print('exist')
      else:
        #print(str(item[1]))
        label_list.append(item[1])
```

```python
[ ] label_list.append('UNK_UNK')
```

```python
[ ] label_list
```

```
[ 'w2',
 'transactions',
 'alarm',
 'roll_dice',
 'taxes',
 'what_are_your_hobbies',
 'uber',
 'pto_balance',
```

6. Create a label map where each intent is numbered.

+ Code + Text

```
#convert the input examples into dataloader
label_map = {}
for (i, label) in enumerate(label_list):
  print
  label_map[label] = i
#------------------------------
#   Load the train dataset
#------------------------------
train_examples = labeled_examples
#The labeled (train) dataset is assigned with a mask set to True
train_label_masks = np.ones(len(labeled_examples), dtype=bool)
#If unlabel examples are available
if unlabeled_examples:
  train_examples = train_examples + unlabeled_examples
  #The unlabeled (train) dataset is assigned with a mask set to False
  tmp_masks = np.zeros(len(unlabeled_examples), dtype=bool)
  train_label_masks = np.concatenate([train_label_masks,tmp_masks])
```

```
[ ] label_map
```

```
{'accept_reservations': 50,
 'account_blocked': 31,
 'alarm': 2,
 'application_status': 54,
 'apr': 105,
 'are_you_a_bot': 109,
 'balance': 25,
 'bill_balance': 43
```

7. Download the pre-trained BERT model and its tokenizer

```
[ ] #Download BERT and tokenizer
    # Hugging-face tokenizer library.
    transformer = AutoModel.from_pretrained(model_name)
    tokenizer = AutoTokenizer.from_pretrained(model_name)
```

```
Downloading: 100% 570/570 [00:00<00:00, 16.7kB/s]
Downloading: 100% 436M/436M [00:16<00:00, 28.5MB/s]
Downloading: 100% 213k/213k [00:00<00:00, 663kB/s]
Downloading: 100% 436k/436k [00:00<00:00, 679kB/s]
```

8. Create a dataloader function that takes training and test data as input and intent map. This dataloader creates a vector representation of all the records making them ready for training and testing

```
def generate_data_loader(input_examples, label_masks, label_map, do_shuffle = False, balance_label_examples = False):
  '''
  Generate a Dataloader given the input examples, eventually masked if they are
  to be considered NOT labeled.
  '''
  examples = []

  # Count the percentage of labeled examples
  num_labeled_examples = 0
  for label_mask in label_masks:
    if label_mask:
      num_labeled_examples += 1
  label_mask_rate = num_labeled_examples/len(input_examples)
  print(label_mask_rate)

  # if required it applies the balance
  for index, ex in enumerate(input_examples):
    if label_mask_rate == 1 or not balance_label_examples:
      examples.append((ex, label_masks[index]))
    else:
      # IT SIMULATE A LABELED EXAMPLE
      if label_masks[index]:
        balance = int(1/label_mask_rate)
        balance = int(math.log(balance,2))
        if balance < 1:
          balance = 1
        for b in range(0, int(balance)):
          examples.append((ex, label_masks[index]))
      else:
        examples.append((ex, label_masks[index]))

  #-----------------------------------------------
```

```
  # Tokenization : Hugginface BERT tokenizer.encode()
  for (text, label_mask) in examples:
    encoded_sent = tokenizer.encode(text[0], add_special_tokens=True, max_length=max_seq_length, padding="max_length", truncation=True)
    input_ids.append(encoded_sent)
    #print(label_map[text[1]])
    label_id_array.append(label_map[text[1]])
    label_mask_array.append(label_mask)

  # Attention to token (to ignore padded input wordpieces)
  for sent in input_ids:
    att_mask = [int(token_id > 0) for token_id in sent]
    input_mask_array.append(att_mask)
  # Convertion to Tensor
  input_ids = torch.tensor(input_ids)
  input_mask_array = torch.tensor(input_mask_array)
  label_id_array = torch.tensor(label_id_array, dtype=torch.long)
  label_mask_array = torch.tensor(label_mask_array)

  # Building the TensorDataset
  dataset = TensorDataset(input_ids, input_mask_array, label_id_array, label_mask_array)

  if do_shuffle:
    sampler = RandomSampler
  else:
    sampler = SequentialSampler

  # Building the DataLoader
  return DataLoader(
```

9. Below code snipped converts training and test dataset to tensor dataset.

```
[ ] train_dataloader = generate_data_loader(train_examples, train_label_masks, label_map, do_shuffle = True, balance_label_examples = apply_balance)
```

```
0.6
```

```
[ ] for w, x, y, z in train_dataloader:
      print(w,x, y,z)
```

```
Streaming output truncated to the last 5000 lines.
        [1, 1, 1,  ..., 0, 0, 0],
        [1, 1, 1,  ..., 0, 0, 0],
        ...,
        [1, 1, 1,  ..., 0, 0, 0],
        [1, 1, 1,  ..., 0, 0, 0],
        [1, 1, 1,  ..., 0, 0, 0]]) tensor([ 29, 130,  97, 150, 150, 150, 150, 150, 112, 150, 150,  36, 114, 137,
         32, 143,   7, 150, 117, 150, 136, 141, 150, 150, 150,   2, 136,  30,
        150, 150, 114, 150, 150, 118, 150, 118, 150,  55,  13, 150,  48, 150,
         84, 150,  59,  47, 104,  78, 150,  15, 150, 150, 149,  15,  97, 150,
        140,  37,  62, 150, 150, 122, 150, 150]) tensor([ True,  True,  True, False, False, False, False, False,  True, False,
        False,  True,  True,  True,  True,  True,  True, False,  True, False,
         True,  True, False, False, False,  True,  True,  True, False, False,
         True, False, False,  True, False,  True, False,  True,  True, False,
         True, False,  True, False,  True,  True,  True,  True, False,  True,
        False, False,  True,  True,  True, False,  True,  True,  True, False,
        False,  True, False, False])
tensor([[ 101, 1202, 1128,  ...,    0,    0,    0],
        [ 101, 1156, 1128,  ...,    0,    0,    0],
        [ 101, 1169, 1128,  ...,    0,    0,    0],
        ...,
```

```
[ ] test_label_masks = np.ones(len(test_examples), dtype=bool)
    test_dataloader = generate_data_loader(test_examples, test_label_masks, label_map, do_shuffle = False, balance_label_examples = False)
```

```
1.0
```

```
for w, x, y, z in test_dataloader:
  print(w,x, y,z)
```

```
tensor([[ 101, 1293, 1156,  ...,    0,    0,    0],
        [ 101, 1184,  112,  ...,    0,    0,    0],
        [ 101, 1293, 1156,  ...,    0,    0,    0],
        ...,
        [ 101, 1383,  170,  ...,    0,    0,    0],
        [ 101, 1383,  170,  ...,    0,    0,    0],
        [ 101, 1383,  170,  ...,    0,    0,    0]]) tensor([[1, 1, 1,  ..., 0, 0, 0],
        [1, 1, 1,  ..., 0, 0, 0],
        [1, 1, 1,  ..., 0, 0, 0],
        ...,
        [1, 1, 1,  ..., 0, 0, 0],
        [1, 1, 1,  ..., 0, 0, 0],
        [1, 1, 1,  ..., 0, 0, 0]]) tensor([118, 118, 118, 118, 118, 118, 118, 118, 118, 118, 118, 118, 118, 118,
        118, 118, 118, 118, 118, 118, 118, 118, 118, 118, 118, 118, 118, 118,
        118, 118, 120, 120, 120, 120, 120, 120, 120, 120, 120, 120, 120, 120,
        120, 120, 120, 120, 120, 120, 120, 120, 120, 120, 120, 120, 120, 120,
        120, 120, 120, 120,  98,  98,  98,  98]) tensor([True, True, True, True, True, True, True, True, True, True, True, True,
        True, True, True, True, True, True, True, True, True, True, True, True,
        True, True, True, True, True, True, True, True, True, True, True, True,
        True, True, True, True, True, True, True, True, True, True, True, True,
```

10. Define the generator and discriminator. The generator creates fake data and discriminator classifies the intent.

```python
class Generator(nn.Module):
    def __init__(self, noise_size=100, output_size=512, hidden_sizes=[512], dropout_rate=0.1):
        super(Generator, self).__init__()
        layers = []
        hidden_sizes = [noise_size] + hidden_sizes
        for i in range(len(hidden_sizes)-1):
            layers.extend([nn.Linear(hidden_sizes[i], hidden_sizes[i+1]), nn.LeakyReLU(0.2, inplace=True), nn.Dropout(dropout_rate)])

        layers.append(nn.Linear(hidden_sizes[-1],output_size))
        self.layers = nn.Sequential(*layers)

    def forward(self, noise):
        output_rep = self.layers(noise)
        return output_rep

class Discriminator(nn.Module):
    def __init__(self, input_size=512, hidden_sizes=[512], num_labels=2, dropout_rate=0.1):
        super(Discriminator, self).__init__()
        self.input_dropout = nn.Dropout(p=dropout_rate)
        layers = []
        hidden_sizes = [input_size] + hidden_sizes
        for i in range(len(hidden_sizes)-1):
            layers.extend([nn.Linear(hidden_sizes[i], hidden_sizes[i+1]), nn.LeakyReLU(0.2, inplace=True), nn.Dropout(dropout_rate)])

        self.layers = nn.Sequential(*layers) #per il flatten
        self.logit = nn.Linear(hidden_sizes[-1],num_labels+1) # +1 for the probability of this sample being fake/real.
        self.softmax = nn.Softmax(dim=-1)
```

11. Below step imports the config file which was required to get the vector dimension. It also initialised the generator and discriminator class

```python
# The config file is required to get the dimension of the vector produced by
# the underlying transformer
config = AutoConfig.from_pretrained(model_name)
hidden_size = int(config.hidden_size)
# Define the number and width of hidden layers
hidden_levels_g = [hidden_size for i in range(0, num_hidden_layers_g)]
hidden_levels_d = [hidden_size for i in range(0, num_hidden_layers_d)]

#------------------------------------------------
#   Instantiate the Generator and Discriminator
#------------------------------------------------
generator = Generator(noise_size=noise_size, output_size=hidden_size, hidden_sizes=hidden_levels_g, dropout_rate=out_dropout_rate)
discriminator = Discriminator(input_size=hidden_size, hidden_sizes=hidden_levels_d, num_labels=len(label_list), dropout_rate=out_dropout_rate)

# Put everything in the GPU if available
if torch.cuda.is_available():
  generator.cuda()
  discriminator.cuda()
  transformer.cuda()
  transformer = torch.nn.DataParallel(transformer)     #Implements data parallelism at the module level.

  print(config)
```

12. The configured components are given below.

```
BertConfig {
  "architectures": [
    "BertForMaskedLM"
  ],
  "attention_probs_dropout_prob": 0.1,
  "gradient_checkpointing": false,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 768,
  "initializer_range": 0.02,
  "intermediate_size": 3072,
  "layer_norm_eps": 1e-12,
  "max_position_embeddings": 512,
  "model_type": "bert",
  "num_attention_heads": 12,
  "num_hidden_layers": 12,
  "pad_token_id": 0,
  "position_embedding_type": "absolute",
  "transformers_version": "4.3.2",
  "type_vocab_size": 2,
  "use_cache": true,
  "vocab_size": 28996
}
```

```
[ ] hidden_size
```

```
768
```

```
[ ] generator
```

```
Generator(
  (layers): Sequential(
    (0): Linear(in_features=100, out_features=768, bias=True)
    (1): LeakyReLU(negative_slope=0.2, inplace=True)
    (2): Dropout(p=0.2, inplace=False)
    (3): Linear(in_features=768, out_features=768, bias=True)
  )
)
```

```
discriminator
```

```
Discriminator(
  (input_dropout): Dropout(p=0.2, inplace=False)
  (layers): Sequential(
    (0): Linear(in_features=768, out_features=768, bias=True)
    (1): LeakyReLU(negative_slope=0.2, inplace=True)
    (2): Dropout(p=0.2, inplace=False)
  )
  (logit): Linear(in_features=768, out_features=152, bias=True)
  (softmax): Softmax(dim=-1)
)
```

13. Define the optimizer and the scheduler which helps to improve the performance of the model.

```
#optimizer
learning_rate_discriminator = 5e-5
learning_rate_generator = 5e-5
dis_optimizer = torch.optim.AdamW(d_vars, lr=learning_rate_discriminator)
gen_optimizer = torch.optim.AdamW(g_vars, lr=learning_rate_generator)
```

```
[ ] dis_optimizer
```

```
AdamW (
Parameter Group 0
    amsgrad: False
    betas: (0.9, 0.999)
    capturable: False
    eps: 1e-08
    foreach: None
    lr: 5e-05
    maximize: False
    weight_decay: 0.01
)
```

```
#scheduler
num_train_epochs = 10
warmup_proportion = 0.1
num_train_examples = len(train_examples)
num_train_steps = int(num_train_examples / batch_size * num_train_epochs)
num_warmup_steps = int(num_train_steps * warmup_proportion)

scheduler_d = get_constant_schedule_with_warmup(dis_optimizer,
                                           num_warmup_steps = num_warmup_steps)
scheduler_g = get_constant_schedule_with_warmup(gen_optimizer,
                                           num_warmup_steps = num_warmup_steps)
```

+ Code + Te

14. The below code block trains the model and performs model evaluation on test dataset.

```
for epoch_i in range(0, num_train_epochs):
#for epoch_i in range(0, 1):
    # ========================================
    #               Training
    # ========================================
    # Perform one full pass over the training set.
    print("")
    print('======== Epoch {:} / {:} ========'.format(epoch_i + 1, num_train_epochs))
    print('Training...')

    # Measure how long the training epoch takes.
    t0 = time.time()

    # Reset the total loss for this epoch.
    tr_g_loss = 0
    tr_d_loss = 0

    # Put the model into training mode.
    transformer.train()
    generator.train()
    discriminator.train()

    # For each batch of training data...
    for step, batch in enumerate(train_dataloader):

        # Progress update every print_each_n_step batches.
        if step % print_each_n_step == 0 and not step == 0:
```

```
        # Progress update every print_each_n_step batches.
        if step % print_each_n_step == 0 and not step == 0:
            # Calculate elapsed time in minutes.
            elapsed = format_time(time.time() - t0)

            # Report progress.
            print('  Batch {:>5,}  of  {:>5,}.    Elapsed: {:}.'.format(step, len(train_dataloader), elapsed))

        # Unpack this training batch from our dataloader.
        b_input_ids = batch[0].to(device)
        b_input_mask = batch[1].to(device)
        b_labels = batch[2].to(device)
        b_label_mask = batch[3].to(device)

        real_batch_size = b_input_ids.shape[0]

        # Encode real data in the Transformer
        model_outputs = transformer(b_input_ids, attention_mask=b_input_mask)
        hidden_states = model_outputs[-1]

        # Generate fake data
        noise = torch.zeros(real_batch_size, noise_size, device=device).uniform_(0, 1)
        gen_rep = generator(noise)

        # Generate the output of the Discriminator for real and fake data.
        disciminator_input = torch.cat([hidden_states, gen_rep], dim=0)
        # Then, we select the output of the discriminator
        features, logits, probs = discriminator(disciminator_input)
```

```
# Finally, we separate the discriminator's output for the real and fake
# data
features_list = torch.split(features, real_batch_size)
D_real_features = features_list[0]
D_fake_features = features_list[1]

logits_list = torch.split(logits, real_batch_size)
D_real_logits = logits_list[0]
D_fake_logits = logits_list[1]

probs_list = torch.split(probs, real_batch_size)
D_real_probs = probs_list[0]
D_fake_probs = probs_list[1]

#---------------------------------
#  LOSS evaluation
#---------------------------------
# Generator's LOSS estimation
g_loss_d = -1 * torch.mean(torch.log(1 - D_fake_probs[:,-1] + epsilon))
g_feat_reg = torch.mean(torch.pow(torch.mean(D_real_features, dim=0) - torch.mean(D_fake_features, dim=0), 2))
g_loss = g_loss_d + g_feat_reg

# Discriminator's LOSS estimation
logits = D_real_logits[:,0:-1]
log_probs = F.log_softmax(logits, dim=-1)

label2one_hot = torch.nn.functional.one_hot(b_labels, len(label_list))  #create one hot embeddings, which is ver
per_example_loss = -torch.sum(label2one_hot * log_probs, dim=-1)
```

```
label2one_hot = torch.nn.functional.one_hot(b_labels, len(label_list))  #create one hot embeddings, which is
per_example_loss = -torch.sum(label2one_hot * log_probs, dim=-1)
per_example_loss = torch.masked_select(per_example_loss, b_label_mask.to(device))
labeled_example_count = per_example_loss.type(torch.float32).numel()

# It may be the case that a batch does not contain labeled examples,
# so the "supervised loss" in this case is not evaluated
if labeled_example_count == 0:
  D_L_Supervised = 0
else:
  D_L_Supervised = torch.div(torch.sum(per_example_loss.to(device)), labeled_example_count)

D_L_unsupervised1U = -1 * torch.mean(torch.log(1 - D_real_probs[:, -1] + epsilon))
D_L_unsupervised2U = -1 * torch.mean(torch.log(D_fake_probs[:, -1] + epsilon))
d_loss = D_L_Supervised + D_L_unsupervised1U + D_L_unsupervised2U

#---------------------------------
#  OPTIMIZATION
#---------------------------------

gen_optimizer.zero_grad()
dis_optimizer.zero_grad()

# Calculate weigth updates
# retain_graph=True is required since the underlying graph will be deleted after backward
g_loss.backward(retain_graph=True)
d_loss.backward()

# Apply modifications
```

```
        # Apply modifications
        gen_optimizer.step()
        dis_optimizer.step()

        # A detail log of the individual losses
        #print("{0:.4f}\t{1:.4f}\t{2:.4f}\t{3:.4f}\t{4:.4f}".
        #       format(D_L_Supervised, D_L_unsupervised1U, D_L_unsupervised2U,
        #              g_loss_d, g_feat_reg))

        # Save the losses to print them later
        tr_g_loss += g_loss.item()
        tr_d_loss += d_loss.item()

        # Update the learning rate with the scheduler
        #if apply_scheduler:
        scheduler_d.step()
        scheduler_g.step()

    # Calculate the average loss over all of the batches.
    avg_train_loss_g = tr_g_loss / len(train_dataloader)
    avg_train_loss_d = tr_d_loss / len(train_dataloader)

    # Measure how long this epoch took.
    training_time = format_time(time.time() - t0)

    print("")
    print("  Average training loss generetor: {0:.3f}".format(avg_train_loss_g))
    print("  Average training loss discriminator: {0:.3f}".format(avg_train_loss_d))
```

+ Code + Text

```
    print("")
    print("  Average training loss generetor: {0:.3f}".format(avg_train_loss_g))
    print("  Average training loss discriminator: {0:.3f}".format(avg_train_loss_d))
    print("  Training epcoh took: {:}".format(training_time))

    # ========================================
    #     TEST ON THE EVALUATION DATASET
    # ========================================
    # After the completion of each training epoch, measure our performance on
    # our test set.
    print("")
    print("Running Test...")

    t0 = time.time()

    # Put the model in evaluation mode--the dropout layers behave differently
    # during evaluation.
    transformer.eval() #maybe redundant
    discriminator.eval()
    generator.eval()

    # Tracking variables
    total_test_accuracy = 0

    total_test_loss = 0
    nb_test_steps = 0

    all_preds = []
```

```
        # Unpack this training batch from our dataloader.
        b_input_ids = batch[0].to(device)
        b_input_mask = batch[1].to(device)
        b_labels = batch[2].to(device)



        with torch.no_grad():
            model_outputs = transformer(b_input_ids, attention_mask=b_input_mask)
            hidden_states = model_outputs[-1]
            _, logits, probs = discriminator(hidden_states)
            ###log_probs = F.log_softmax(probs[:,1:], dim=-1)
            filtered_logits = logits[:,0:-1]
            # Accumulate the test loss.
            total_test_loss += nll_loss(filtered_logits, b_labels)

        # Accumulate the predictions and the input labels
        _, preds = torch.max(filtered_logits, 1)
        all_preds += preds.detach().cpu()
        all_labels_ids += b_labels.detach().cpu()

    # Report the final accuracy for this validation run.
    all_preds = torch.stack(all_preds).numpy()
    all_labels_ids = torch.stack(all_labels_ids).numpy()
    test_accuracy = np.sum(all_preds == all_labels_ids) / len(all_preds)
    print("  Accuracy: {0:.3f}".format(test_accuracy))

    #calculate classification matrix
```

```
    #calculate classification matrix
    num_correct = 0
    num_incorrect = 0
    for i in all_preds:
      if all_preds[i] == all_labels_ids[i]:
        #print('Identified Correctly')
        num_correct += 1
      else: #print('Identified Incorrectly')
        num_incorrect += 1

    print("  Correctly Identified Labels: {0:.3f}".format(num_correct))
    print("  Incorrectly Identified Labels: {0:.3f}".format(num_incorrect))

    # Calculate the average loss over all of the batches.
    avg_test_loss = total_test_loss / len(test_dataloader)
    avg_test_loss = avg_test_loss.item()

    # Measure how long the validation run took.
    test_time = format_time(time.time() - t0)

    print("  Test Loss: {0:.3f}".format(avg_test_loss))
    print("  Test took: {:}".format(test_time))

    # Record all statistics from this epoch.
    training_stats.append(
        {
            'epoch': epoch_i + 1,
            'Training Loss generator': avg_train_loss_g,
```

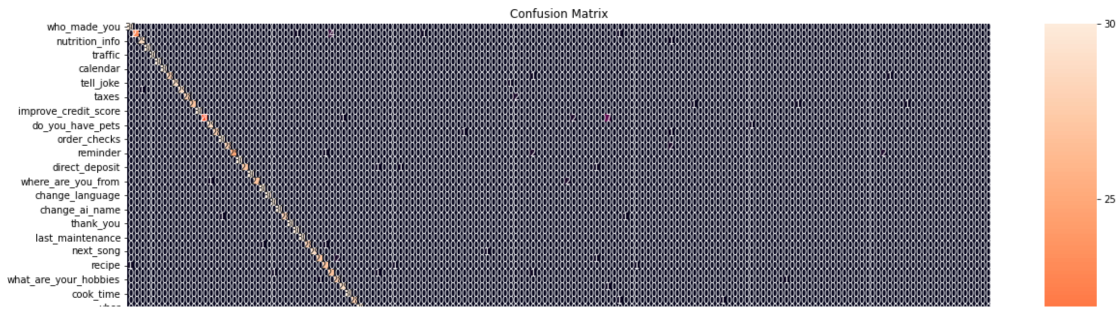
15. The below code prints the performance achieved after each epoch.

```
#Print the Training statistics
for stat in training_stats:
  print(stat)

print("\nTraining complete!")

print("Total training took {:} (h:mm:ss)".format(format_time(time.time()-total_t0)))
```

```
{'epoch': 1, 'Training Loss generator': 0.42101408326403894, 'Training Loss discriminator': 7.36140989952899, 'Valid. Loss': 4.115070343017578, 'Valid. Accur
{'epoch': 2, 'Training Loss generator': 0.7616998406166726, 'Training Loss discriminator': 3.7616554534181637, 'Valid. Loss': 1.8845173120498657, 'Valid. Acc
{'epoch': 3, 'Training Loss generator': 0.7610380177802228, 'Training Loss discriminator': 1.9445677143462161, 'Valid. Loss': 0.7157779932022095, 'Valid. Acc
{'epoch': 4, 'Training Loss generator': 0.7408130381969695, 'Training Loss discriminator': 1.1575902867824472, 'Valid. Loss': 0.4146716296672821, 'Valid. Acc
{'epoch': 5, 'Training Loss generator': 0.7324782016429495, 'Training Loss discriminator': 0.8922758429608446, 'Valid. Loss': 0.2921811044216156, 'Valid. Acc
{'epoch': 6, 'Training Loss generator': 0.7298689144722958, 'Training Loss discriminator': 0.8085053169980962, 'Valid. Loss': 0.28152650594711304, 'Valid. Ac
{'epoch': 7, 'Training Loss generator': 0.7279369262938804, 'Training Loss discriminator': 0.7711162770048101, 'Valid. Loss': 0.2782150208950043, 'Valid. Acc

Training complete!
Total training took 0:28:54 (h:mm:ss)
```

16. Classification matrix was calculated with below code.

```
plt.figure(figsize=(20,20))

sns.heatmap(cm_df, annot=True)

plt.title('Confusion Matrix')

plt.ylabel('Actal Values')

plt.xlabel('Predicted Values')

plt.show()
```



17. Classification report was given by below code.

```python
#Classification Report
from sklearn.metrics import plot_confusion_matrix, classification_report
print(classification_report(all_labels_ids, all_preds))
```

```
              precision    recall  f1-score   support

           0       0.97      1.00      0.98        30
           1       1.00      0.77      0.87        30
           2       0.97      0.97      0.97        30
           3       0.97      1.00      0.98        30
           4       1.00      1.00      1.00        30
           5       1.00      1.00      1.00        30
           6       1.00      1.00      1.00        30
           7       0.90      0.93      0.92        30
           8       0.94      0.97      0.95        30
           9       1.00      0.97      0.98        30
          10       1.00      0.93      0.97        30
          11       0.96      0.90      0.93        30
          12       0.81      1.00      0.90        30
          13       1.00      0.67      0.80        30
          14       0.88      0.97      0.92        30
          15       0.97      0.93      0.95        30
          16       0.91      1.00      0.95        30
          17       0.97      0.93      0.95        30
          18       1.00      0.83      0.91        30
          19       1.00      1.00      1.00        30
          20       1.00      0.90      0.95        30
          21       1.00      1.00      1.00        30
          22       0.96      0.90      0.93        30
          23       0.83      1.00      0.91        30
```

References

Larson, S., Mahendran, A., Peper, J. J., Clarke, C., Lee, A., Hill, P., Kummerfeld, J. K., Leach, K., Laurenzano, M. A., Tang, L. and Mars, J. (2019). An evaluation dataset for intent classification and out-of-scope prediction, pp. 1311–1316.
URL: *https://aclanthology.org/D19-1131*