National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Shriya Gandhi
Student ID: x19218079

School of Computing
National College of Ireland

Supervisor: Jorge Basilio

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Shriya Gandhi

Student ID: X19218079

Programme: Data Analytics **Year:** 2021

Module: MSc Research Project

Lecturer: Jorge Basilio

Submission Due Date: 16-08-2021

Project Title: Classifying the Insincere Questions using Transfer Learning

Word Count: ..... **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: .....

Date: .....

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

Shriya Gandhi
Student ID: x19218079

1 Introduction

This research project aims to automatically classify the insincere questions on Quora using the transformer-based models. In this configuration manual, we have outlined the steps to be followed for smooth replication of research project. The different steps of a project life cycle, starting from data collection until model testing and evaluation, are explained. Reference to code snippets is provided wherever required.

2 System Configuration

The entire project has been implemented on Google Colaboratory Pro with GPU (Tesla P100-PCIE-16GB) enabled. Google Colaboratory Pro version ensures that the model training session does not get disconnected because of RAM outage.

3 Dataset

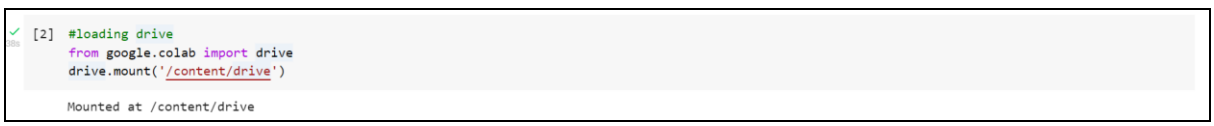
The dataset for our research work is extracted from Kaggle.

4 Google Colaboratory Setup

The data extracted from Kaggle is uploaded to Google drive as in google colab we can access the data by simply mounting the drive. The code snippet for mounting of drive-in google colab can be found below.

Authorization for mounting google drive onto colab.

Google Drive successfully mounted.



Also, before proceeding towards coding, we first checked the GPU availability.

```
[1]  # If there's a GPU available...
     import torch
     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 P100-PCIE-16GB
```

5 Installing Python Libraries for Data Pre-processing

The required libraries for data pre-processing are installed and imported.

```
[3]  #Importing libraries
     import re
     import nltk
     import pandas as pd
     import io
     import matplotlib.pyplot as plt
     import seaborn as sns
     from tqdm.notebook import tqdm
     from tqdm import tqdm
     from wordcloud import WordCloud, STOPWORDS
     from nltk.tokenize import word_tokenize
     nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
True
```

```
[4]  !pip install contractions
     import contractions #importing python library to expand the contractions
```

```
Requirement already satisfied: contractions in /usr/local/lib/python3.7/dist-packages (0.0.52)
Requirement already satisfied: textsearch>=0.0.21 in /usr/local/lib/python3.7/dist-packages (from contractions) (0.0.21)
Requirement already satisfied: anyascii in /usr/local/lib/python3.7/dist-packages (from textsearch>=0.0.21->contractions) (0.2.0)
Requirement already satisfied: pyahocorasick in /usr/local/lib/python3.7/dist-packages (from textsearch>=0.0.21->contractions) (1.4.2)
```

The uploaded dataset is read and stored inside a pandas dataframe.

```
[ ] train_df = pd.read_csv("/content/drive/MyDrive/Shriya/train.csv") #Read dataset
```

```
train_df.head()
```

	qid	question_text	target
0	00002165364db923c7e6	How did Quebec nationalists see their province...	0
1	000032939017120e6e44	Do you have an adopted dog, how would you enco...	0
2	0000412ca6e4628ce2cf	Why does velocity affect time? Does velocity a...	0
3	000042bf85aa498cd78e	How did Otto von Guericke used the Magdeburg h...	0
4	0000455dfa3e01eae3af	Can I convert montra helicon D to a mountain b...	0

```
[ ] train_df.shape
```

```
(1306122, 3)
```

6 Exploratory data analysis

```
#Checking for presence of NA values
(train_df.isna()).sum()
```

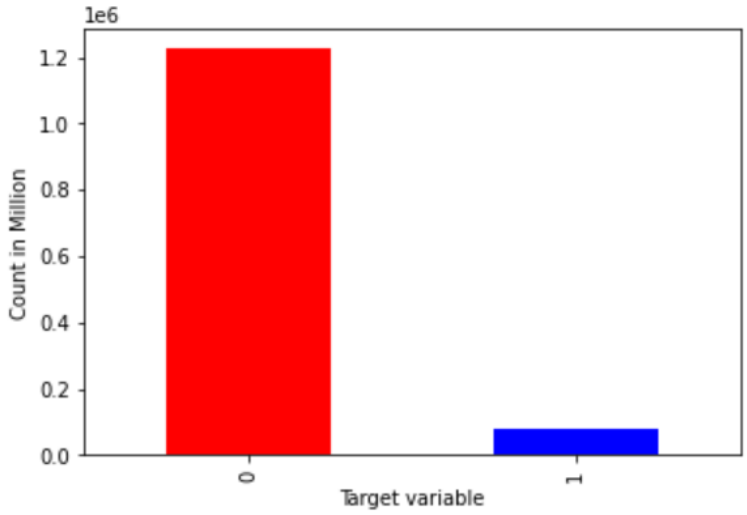
```
qid              0
question_text    0
target           0
dtype: int64
```

```
[ ] train_df['target'].value_counts()
```

```
0    1225312
1      80810
Name: target, dtype: int64
```

```
[ ]  #Bar plot for Target variable distribution
     train_df.target.value_counts().plot(kind='bar',color=["red","blue"])
     plt.xlabel("Target variable")
     plt.ylabel("Count in Million")
```

```
Text(0, 0.5, 'Count in Million')
```



Word cloud

```
#Wordcloud with commonly used words
comment_words = ''
stopwords = set(STOPWORDS)
# iterate through the csv file
for val in train_df.question_text:
    # typecaste each val to string
    val = str(val)
    # split the value
    tokens = val.split()
    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

    comment_words += " ".join(tokens)+" "

wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(comment_words)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```

An additional column, quest_len, is added to python data frame. It stores the length of each question. The maximum, mean, median question lengths are identified.

```
[ ]  #Max length of questions - sincere and insincere
     train_df['quest_len'] = train_df['question_text'].apply(lambda x: len(str(x).split()))
     print("Maximum length of a sincere question:", max(train_df[train_df['target']==0]['quest_len']))
     print("Maximum length of a insincere question:", max(train_df[train_df['target']==1]['quest_len']))

     Maximum length of a sincere question: 134
     Maximum length of a insincere question: 64

[ ]  #Mean length of questions
     res = train_df['quest_len'].mean()

     print("The mean length in words are : " +  str(res))

     The mean length in words are : 12.803609463740754

[ ]  #Median length of questions
     res = train_df['quest_len'].median()

     print("The median length in words are : " +  str(res))

     The median length in words are : 11.0
```

Using quest_len, plots to visualize the sincere and insincere question lengths are plotted.

```
#Plot for finding length of question tokens - sincere
sincere = train_df[train_df["target"] == 0]

plt.figure(figsize = (15, 8))
sns.distplot(sincere["quest_len"], hist = True, label = "sincere",color='blue')
plt.legend(fontsize = 10)
plt.title("Questions Length Distribution by Class", fontsize = 12)
plt.xlabel('Sincere Question Length')
plt.show()
```

```
#Plot for finding length of question tokens - insincere
insincere = train_df[train_df["target"] == 1]

plt.figure(figsize = (15, 8))
sns.distplot(insincere["quest_len"], hist = True, label = "insincere", color='red',norm_hist=True)
plt.legend(fontsize = 10)
plt.title("Questions Length Distribution by Class", fontsize = 12)
plt.xlabel('Insincere Question Length')
plt.show()
```

7 Data Pre-processing

The contractions such as didn't, couldn't present in the questions are expanded. Using lambda expressions, the steps of clean_text function are iterated over all the questions of the dataset. The pre-processed data is stored inside a new column, cleaned_questions.

```
[ ]  #Function for expanding the contractions
     def clean_text(txt):
         cleaned_text= contractions.fix(txt)
         return cleaned_text

[ ]  #Applying clean_text function on our data
     tqdm.pandas()
     train_df['cleaned_questions']  = train_df['question_text'].progress_apply(lambda txt: clean_text(txt))

     100%|██████████| 1306122/1306122 [00:11<00:00, 114692.11it/s]
```

Functions for removing punctuations and numbers are written and applied over the cleaned_questions text.

```
     #Function for removing punctuations
     def remove_punch(text):
         # Removing punctuations in string
         # Using regex
         res = re.sub(r'[^a-zA-Z]', ' ', text)
         return res

[ ]  train_df['cleaned_questions']  = train_df['cleaned_questions'].progress_apply(lambda text: remove_punch(text)) #applying remove_punct on our data

     100%|██████████| 1306122/1306122 [00:07<00:00, 182714.10it/s]

[ ]  #Function for removing numbers
     def remove_num(text):
       # using filter and lambda
       response = "".join(filter(lambda x: not x.isdigit(), text))
       return response

[ ]  train_df['cleaned_questions']  = train_df['cleaned_questions'].progress_apply(lambda text: remove_num(text)) #applying remove_num on our data

     100%|██████████| 1306122/1306122 [00:17<00:00, 76575.70it/s]
```

Finally, the data is validated for the presence of any duplicate records. The pre-processed file is then stored as a CSV file.

```
     train_df.shape

     (1306122, 5)

[ ]  train_df.drop_duplicates('cleaned_questions', keep='first',inplace=True) #dropping duplicates if any

[ ]  train_df.shape #df size reduced after dropping duplicates

     (1305328, 5)

[ ]  train_df['target'].value_counts()

     0    1224569
     1      80759
     Name: target, dtype: int64

[ ]  #Writing the pre-processed dataframe to a csv file
     train_df.to_csv('clean_data.csv')
```

8 Model Implementation

We have implemented BERT and XLNET models for three samples of data. The steps for implementation of both models are similar. The below section briefs about the steps followed for XLNET with 10% data.

Installing the modeling dependencies

```
#Installing the necessary dependencies
!pip uninstall torch torchvision -y
!pip install torch==1.6.0+cu101 torchvision==0.7.0+cu101 -f https://download.pytorch.org/whl/torch_stable.html
!pip install -U transformers
!pip install -U simpletransformers
```

```
[ ]  #importing libraries
     import pandas as pd
     import numpy as np
     from sklearn.model_selection import train_test_split
```

```
[ ]  #Importing required libraries for model training
     from simpletransformers.classification import ClassificationModel
     import pandas as pd
     import logging
     import sklearn

     2021-08-14 21:12:31.452135: I tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully opened dynamic library libcudart.so.11.0
```

The cleaned data is uploaded to google drive. The google drive is mounted in colaboratory, and data is read in pandas dataframe.

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

```
Mounted at /content/drive
```

```
[ ]  df=pd.read_csv("/content/drive/MyDrive/Shriya/clean_data.csv") #reading csv file
```

```
[ ]  df=df.iloc[:,3:] #selecting the required column from dataframe
```

```
[ ]  df.head()
```

	target	cleaned_questions
0	0	How did Quebec nationalists see their province...
1	0	Do you have an adopted dog how would you enco...
2	0	Why does velocity affect time Does velocity a...
3	0	How did Otto von Guericke used the Magdeburg h...
4	0	Can I convert montra helicon D to a mountain b...

We have used the sample() function to extract 10% data from the original dataframe.

Note- For 50% and 70% data, the argument inside sample() changes to 0.5 and 0.7, respectively.

```
[ ]  df=df.sample(frac=0.1) #sampling the data to select 10% records
```

```
[ ]  df.shape
```

```
(130533, 2)
```

The data is split into train-val-test at a ratio of 80-10-10 using the train_test_split function.

```
[ ]  # splitting the data into training, test and eval dataset
     X = df['cleaned_questions']
     y = df['target']
```

```
[ ]  #splitting in 80-10-10 ratio
     X_train, X_rem, y_train, y_rem = train_test_split(X,y, train_size=0.8)
     test_size = 0.5
     X_valid, X_test, y_valid, y_test = train_test_split(X_rem,y_rem, test_size=0.5)
```

```
[ ]  print(X_train.shape), print(y_train.shape)
     print(X_valid.shape), print(y_valid.shape)
     print(X_test.shape), print(y_test.shape)
```

```
(104426,)
(104426,)
(13053,)
(13053,)
(13054,)
(13054,)
(None, None)
```

Python's logging module is used to track the execution of model commands and errors if any.

```
[ ]  logging.basicConfig(level=logging.INFO)
     transformers_logger = logging.getLogger("transformers")
     transformers_logger.setLevel(logging.WARNING)
```

After running multiple iterations, an optimum set of hyperparameters are chosen for our proposed models. We have used the Classification module provided by Simple Transformer distribution. The model, its tokenizer, maximum length of sequence, epoch, batch size,

learning rate, optimizer, dropout and more are given as arguments to the ClassificationModel function.

We have chosen evaluate_during_training to be True, thus in the train_model function, both training and evaluation data are passed.

The wandb_project stores the runtime data of the model. We have extracted the GPU utilization plot from wandb projects created for each run.

Note – For BERT models, the ClassificationModel arguments are bert and bert-base-uncased.

```
[ ]  results=[]
     # Create a ClassificationModel
     model = ClassificationModel('xlnet', 'xlnet-base-cased', args={
        'max_seq_length': 134,
        'train_batch_size': 16,
        'eval_batch_size': 32,
        'evaluate_during_training': True,
        'evaluate_during_training_steps': 30000,
        'num_train_epochs': 2,
        'weight_decay': 0,
        'learning_rate': 1e-5,
        'adam_epsilon': 1e-8,
        'logging_steps': 50,
        'save_steps': 2000,
        'eval_all_checkpoints': True,
        'overwrite_output_dir': True,
        'optimizer' : 'AdamW',
        'manual_seed' : 42,
        'no_cache' : True,
        'wandb_project' :"XLNET10", #The wandb XLNET10 project keeps a log of all the actions and outputs of model training and eval
        'dropout': 0.3,
        'output_dir': 'outputs/'})# All the selected parameters are passed as an argument to the Classification Model

     # Train the model
     model.train_model(train_df,eval_df=eval_df) #Model training
```

The eval_model is used to obtain the model performance results on evaluation data.

```
[ ]  # Evaluate the model
     result, model_outputs, wrong_predictions = model.eval_model(eval_df, acc=sklearn.metrics.classification_report)

     print(result['acc'])
     # append model score
     results.append(result['acc'])
```

9 Predictions

Once the evaluation results look fine, predict function is used to test the model with test data. A confusion matrix and classification report are printed for each model.

```
[ ]  from sklearn.metrics import confusion_matrix
     from sklearn.metrics import classification_report

     predictions, raw_outputs = model.predict(test_df.cleaned_questions.tolist())

     # confusion matrix
     matrix = confusion_matrix(y_test,predictions, labels=[0,1])
     print('Confusion matrix : \n',matrix)
```

```
INFO:simpletransformers.classification.classification_utils: Converting to features started. Cache is not used.
0%  27/13054 [00:04<34:27, 6.30it/s]
100% 408/408 [01:33<00:00, 4.46it/s]
Confusion matrix :
 [[12043   179]
 [  320   512]]
```

```
[ ]  matrix = classification_report(y_test,predictions,labels=[0,1])
     print('Classification report : \n',matrix)
```

The model is tested on few raw samples as well.

```
[ ]  predictions, raw_outputs = model.predict(['Is asia over populated?'])
```

```
INFO:simpletransformers.classification.classification_utils: Converting to features started. Cache is not used.
100% 1/1 [00:00<00:00, 16.00it/s]
100% 1/1 [00:00<00:00, 19.54it/s]
```

```
[ ]  predictions
```

```
[0]
```

```
[ ]  predictions, raw_outputs = model.predict(['Is Donald Trump going to eat my family'])
```

```
INFO:simpletransformers.classification.classification_utils: Converting to features started. Cache is not used.
100% 1/1 [00:00<00:00, 15.69it/s]
100% 1/1 [00:00<00:00, 19.24it/s]
```

```
[ ]  predictions
```

```
[1]
```