Configuration Manual

MSc Research Project
Data Analytics

Ermesa Pepe
Student ID: X20212887

School of Computing
National College of Ireland

Supervisor: Vladimir Milosavljevic

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: ……Ermesa Pepe………………………………………………………………………………

Student ID: …X20212887……………………………………………………………………………..……

Programme: ……… Master in Data Analytics …………….. **Year:** …2022….………..

Module: ……Research Project………………………………………………………………………..………

Lecturer: ……… Vladimir Milosavljevic …………………………………………………………..………

Submission Due Date: …15/08/2022………………………………………………………………………..………

Project Title: Human-centric Approach to Emails Phishing Detection ……………….………

Word Count:………1977…………………… **Page Count:** ……………..28……….……..………

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: ……*Ermesa Pepe*………………………………………………………………………………

Date: 15/082022……………………………………………………………………………………………

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

Ermesa Pepe
Student ID: X20212887

1 Introduction

The configuration manual document gives an overview and insights into the research carried out as part of the Industry Internship. This manual will provide the details of the system configuration and Tools utilised while performing the study and the project implementation. This project developed five deep learning models as part of the research. The implementation section will guide the process carried out in the development phase, along with the final results of the investigation.

2 System Configuration

The system used while performing the activity was personal as the internship was Remote. The configuration of the system is as follows.

2.1 Hardware Configuration

- Operating system: Windows 11
- Processor: Intel i7-12th gen
- GPU: Nvidia 3070 Ti
- System Compatibility: 64-bit
- Hard Disk: Hybrid (1T SSD)
- RAM: 16GB

2.2 Software Configurations

Before starting the model building phase, the following software, tools and libraries listed in Table 1 were installed in the system.

Table 1: System Configuration

Software/Tools	Version	Information
Python	3.9.7	To develop the Model, Python is used in this project.
Anaconda	4.13.0	It is windows suitable platform that allows users computations, package management and model deployments. (Anaconda, 2022)
Jupyter Lab	6.4.5	Notebook web-based interactive development environment for notebooks, code, and data (Jupyter, 2022)
TensorFlow	2.9.1	For running deep neural networks, TensorFlow is an important library. (TensorFlow, 2022)

Keras	2.9.0	It provides powerful deep learning APIs to boost performance and scale. (Keras, 2022)
Numpy	1.20.3	It is an open-source tool used to perform complex mathematical problems in data. (Numpy, 2022)
Sci-Kit Learn	0.24.2	It is the library for problems such as Classification, Regression, and data pre-processing. (scikit-learn: machine learning in Python — scikit-learn 0.24.2 documentation, 2021)
Matplotlib	3.4.3	Matplotlib is a Python library for creating static, animated, and interactive visualisations. (Matplolib, 2022)
Bokeh	2.4.3	Monitor GPU/Memory usage *python -m jupyterlab_nvdashboard.server <port-number>* (Bokeh, 2022)

To run the simulation, access the Simulation folder on Onedrive Simulation,

- open the file *Simulation.ipynb* file and
- run all the cells to load the five models
- input when requesting any email body to obtain the result on spam detection, a summarisation of the email, the email intent, and the possible principle of persuasion contained in the email.

3 Implementation

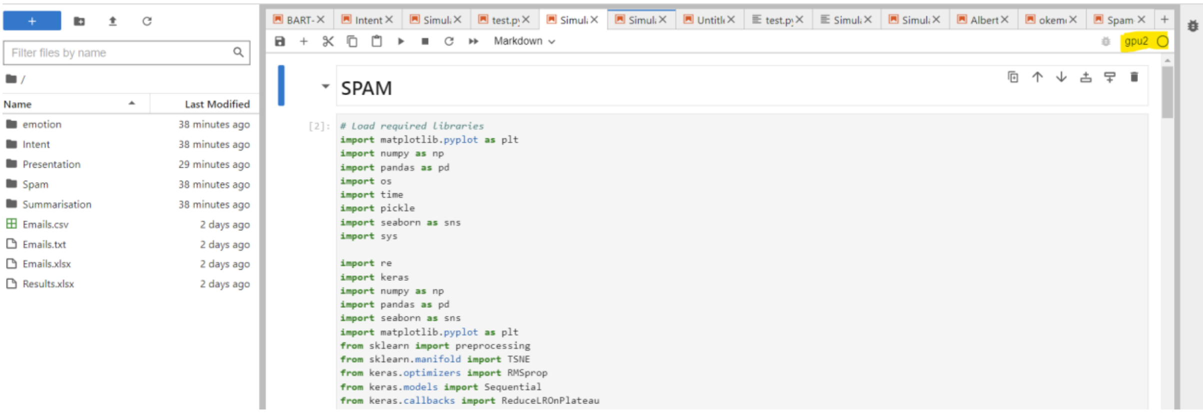
Jupyter ab has been configured to use GPU by following the tutorial available online
https://www.techentice.com/how-to-make-jupyter-notebook-to-run-on-gpu/
That also requires to install. In this section, the step-by-step guide is mentioned to run the project in any windows system. 1. Download and Install Anaconda Software in the windows system. (https://www.anaconda.com/products/individual)

Once the *Jupyter Lab* environment is set, we can open Jupyter lab from Anaconda prompt.

As we can see, Jupyter Lab is using the gpu2 environment created in Anaconda.

After opening jupyter Lab, click on the new notebook (gpu2) in which the development part for the Model will be covered.

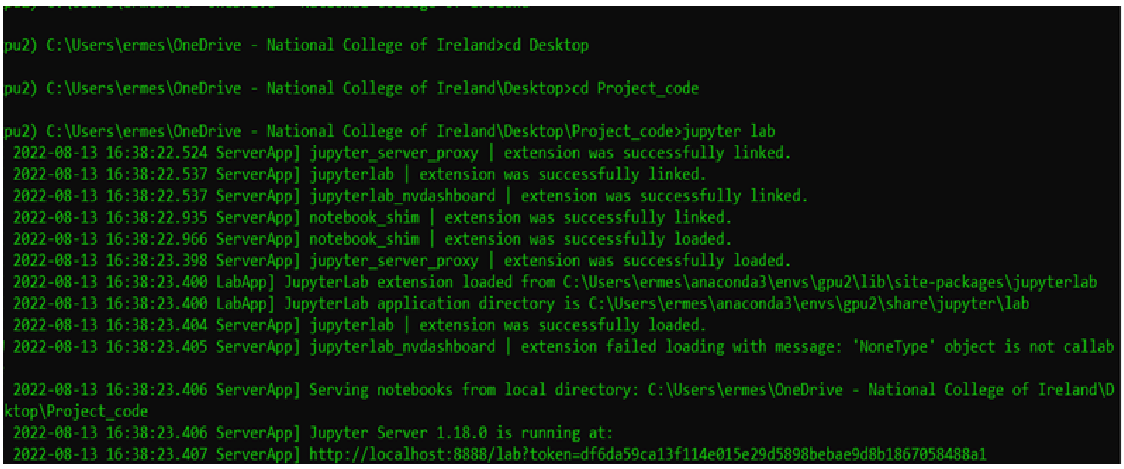
In the new notebook, first import all the required libraries.



3.1 Spam Detection

Import the necessary libraries:

```
# Load required libraries
import numpy as np
import pandas as pd
import os
import time
import pickle
import seaborn as sns
import sys
#sys.setrecursionlimit(1500)
%matplotlib inline
import re
import keras
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn.manifold import TSNE
from keras.optimizers import RMSprop
from keras.models import Sequential
from keras.callbacks import ReduceLROnPlateau
from sklearn.model_selection import train_test_split
from keras.layers import Dropout, Dense, BatchNormalization
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score, roc_auc_score
```



```
pu2) C:\Users\ermes\OneDrive - National College of Ireland>cd Desktop

pu2) C:\Users\ermes\OneDrive - National College of Ireland\Desktop>cd Project_code

pu2) C:\Users\ermes\OneDrive - National College of Ireland\Desktop\Project_code>jupyter lab
 2022-08-13 16:38:22.524 ServerApp] jupyter_server_proxy | extension was successfully linked.
 2022-08-13 16:38:22.537 ServerApp] jupyterlab | extension was successfully linked.
 2022-08-13 16:38:22.537 ServerApp] jupyterlab_nvdashboard | extension was successfully linked.
 2022-08-13 16:38:22.935 ServerApp] notebook_shim | extension was successfully linked.
 2022-08-13 16:38:22.966 ServerApp] notebook_shim | extension was successfully loaded.
 2022-08-13 16:38:23.398 ServerApp] jupyter_server_proxy | extension was successfully loaded.
 2022-08-13 16:38:23.400 LabApp] JupyterLab extension loaded from C:\Users\ermes\anaconda3\envs\gpu2\lib\site-packages\jupyterlab
 2022-08-13 16:38:23.400 LabApp] JupyterLab application directory is C:\Users\ermes\anaconda3\envs\gpu2\share\jupyter\lab
 2022-08-13 16:38:23.404 ServerApp] jupyterlab | extension was successfully loaded.
 2022-08-13 16:38:23.405 ServerApp] jupyterlab_nvdashboard | extension failed loading with message: 'NoneType' object is not callab

 2022-08-13 16:38:23.406 ServerApp] Serving notebooks from local directory: C:\Users\ermes\OneDrive - National College of Ireland\D
ktop\Project_code
 2022-08-13 16:38:23.406 ServerApp] Jupyter Server 1.18.0 is running at:
 2022-08-13 16:38:23.407 ServerApp] http://localhost:8888/lab?token=df6da59ca13f114e015e29d5898bebae9d8b1867058488a1
```

After that, import the provided dataset.

```
import pandas as pd
import numpy as np
import rouge
from sklearn.metrics.pairwise import cosine_similarity
import networkx as nx
import seaborn as sns
import matplotlib.pyplot as plt
from operator import itemgetter
from sentence_transformers import SentenceTransformer

from torch.utils.data import DataLoader
import math
from sentence_transformers import models, losses
from sentence_transformers import SentencesDataset, LoggingHandler, SentenceTransformer
from sentence_transformers.evaluation import EmbeddingSimilarityEvaluator
from sentence_transformers.readers import *
import logging
from datetime import datetime

from transformers import pipeline
```

Restrict the *TensorFlow* library to use as much memory as you wish to avoid OOM.

```
import gc
gc.collect()
```

```
gpus = tf.config.list_physical_devices('GPU')
if gpus:
  # Restrict TensorFlow to only allocate 1GB of memory on the first GPU
    try:
        tf.config.set_logical_device_configuration(
           gpus[0],
           [tf.config.LogicalDeviceConfiguration(memory_limit=6024)])
        logical_gpus = tf.config.list_logical_devices('GPU')
        print(len(gpus), "Physical GPUs,", len(logical_gpus), "Logical GPUs")
    except RuntimeError as e:
        # Virtual devices must be set before GPUs have been initialized
        print(e)
```

After that, import the provided dataset publicly available https://www.kaggle.com/datasets/wanderfj/enron-spam

```
HAM = 'ham'
SPAM = 'spam'

SOURCES = [
    ('C:\project\data\enron\beck-s',      HAM),
    ('C:\project\data\enron\farmer-d',    HAM),
    ('C:\project\data\enron\kaminski-v',  HAM),
    ('.C:\project\data\enron\kitchen-l',   HAM),
    ('C:\project\data\enron\lokay-m',     HAM),
    ('C:\project\data\enron\williams-w3', HAM),
    ('C:\project\data\enron\BG',          SPAM),
    ('C:\project\data\enron\GP',          SPAM),
    ('C:\project\data\enron\SH',          SPAM)
]

SKIP_FILES = {'cmds'}
NEWLINE="\n"
```

We load the data and tokenise it with the following code.

From this, the data pre-processing will be done using the following code.

```
def load_data():
    data = pd.DataFrame({'text': [], 'label': [],'file':[]})
    l = 0
    for path, classification in SOURCES:
        data_frame, nrows = build_data_frame(l, path, classification)
        data = data.append(data_frame)
        l += nrows
    data = data.reindex(np.random.permutation(data.index))
    return data
```

```
# We will load the Email spam dataset into Panadas dataframe here .
data=load_data()
```

```
Percent: [####################################              ] 72%
```

```
new_index=[x for x in range(len(data))]
data.index=new_index
```

```
def token_count(row):
    'returns token count'
    text=row['tokenized_text']
    length=len(text.split())
    return length

def tokenize(row):
    "tokenize the text using default space tokenizer"
    text=row['text']
    lines=(line for line in text.split(NEWLINE) )
    tokenized=""
    for sentence in lines:
        tokenized+= " ".join(tok for tok in sentence.split())
    return tokenized
```

```
data['tokenized_text']=data.apply(tokenize, axis=1)
```

```
data['token_count']=data.apply(token_count, axis=1)
```

```
data['lang']='en'
```

We explore the data with the following code:

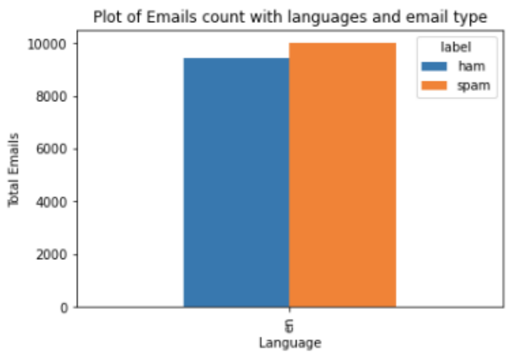
```
print("total emails : ", len(df))
print  ("total spam emails : ", len(df[df['label']=='spam']) )
print  ("total normal emails : ", len(df[df['label']=='ham']) )
```

```
total emails :  19441
total spam emails :  10000
total normal emails :  9441
```

```
df1 = df.groupby(['lang','label'])['label','lang'].size().unstack()

ax=df1.plot(kind='bar')
ax.set_ylabel("Total Emails")
ax.set_xlabel("Language")
ax.set_title("Plot of Emails count with languages and email type")
```

```
C:\Users\ermes\AppData\Local\Temp/ipykernel_12108/771743291.py:1: FutureWarning: Indexing with mu
  df1 = df.groupby(['lang','label'])['label','lang'].size().unstack()
```

```
Text(0.5, 1.0, 'Plot of Emails count with languages and email type')
```



We split the data into train, test and validation sets and replace the label value with numeric values spam=1 and ham=0 and split the dataset before training the Model.

```
df['spam']=df['label'].apply(lambda x: 1 if x=='spam' else 0)
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(df['text'],df['spam'], stratify=df['spam'])
```

We import the models Tensorflow Hub:

```
import os
os.environ['TFHUB_CACHE_DIR'] = r'C:\Users\ermes\OneDrive - National College of Ireland\Desktop\Project'

bert_preprocessor = hub.KerasLayer('https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3')
bert_encoder = hub.KerasLayer('https://tfhub.dev/google/universal-sentence-encoder-cmlm/en-base/1')
```

Train the Model with four layers, Dropout 0.1%, 256 inputs, and one final dense layer.

```
text_input = tf.keras.layers.Input(shape = (), dtype = tf.string, name = 'Inputs')
preprocessed_text = bert_preprocessor(text_input)
embeed = bert_encoder(preprocessed_text)
dropout = tf.keras.layers.Dropout(0.1, name = 'Dropout')(embeed['pooled_output'])
outputs = tf.keras.layers.Dense(256, activation='relu', name = 'Dense')(dropout)
dropout = tf.keras.layers.Dropout(0.1, name = 'Dropout2')(outputs)
outputs = tf.keras.layers.Dense(1, activation = 'sigmoid', name = 'Dense2')(dropout)
```

Create and Compile the final Model.

```
# creating final model
model_universal = tf.keras.Model(inputs = [text_input], outputs = [outputs])
```

```
# compiling our model
model_universal.compile(optimizer ='adam',
               loss = 'binary_crossentropy',
               metrics = Metrics)
```

```
#default batch size of 32
```

```
history = model_universal.fit(X_train, y_train, epochs = 6)
```

```
Epoch 1/6
456/456 [==============================] - 132s 255ms/step - loss: 0.0582 - accuracy: 0.9844 - precision: 0.9871 - recall: 0.9825
Epoch 2/6
456/456 [==============================] - 119s 261ms/step - loss: 0.0200 - accuracy: 0.9938 - precision: 0.9932 - recall: 0.9948
Epoch 3/6
456/456 [==============================] - 124s 273ms/step - loss: 0.0154 - accuracy: 0.9947 - precision: 0.9944 - recall: 0.9952
Epoch 4/6
456/456 [==============================] - 118s 259ms/step - loss: 0.0118 - accuracy: 0.9964 - precision: 0.9963 - recall: 0.9968
Epoch 5/6
456/456 [==============================] - 126s 275ms/step - loss: 0.0093 - accuracy: 0.9972 - precision: 0.9968 - recall: 0.9977
Epoch 6/6
456/456 [==============================] - 125s 274ms/step - loss: 0.0067 - accuracy: 0.9979 - precision: 0.9980 - recall: 0.9980
```

To Evaluate the Model use the evaluate() function:

```
# Evaluating performance
model_universal.evaluate(X_test,y_test)
```

```
152/152 [==============================] - 46s 294ms/step - loss: 0.0
[0.015627838671207428,
 0.9948570132255554,
 0.992439329624176,
 0.9976000189781189]
```

```
# getting y_pred by predicting over X_text and flattening it
y_pred = model_universal.predict(X_test)
```

```
152/152 [==============================] - 43s 281ms/step
```

```
y_pred = y_pred.flatten().round()
```

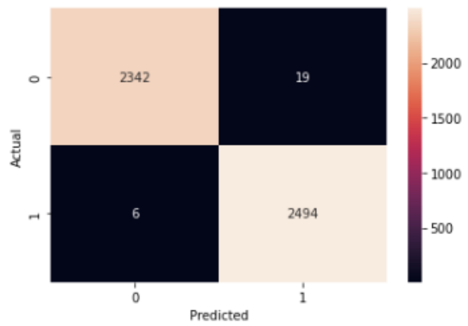
To Build the confusion matrix after getting the predicted values on the test set:

```python
# importing confusion maxtrix
from sklearn.metrics import confusion_matrix , classification_report
# creating confusion matrix
cm = confusion_matrix(y_test,y_pred)
```

```python
# plotting as a graph - importing seaborn
import seaborn as sns
```

```python
# creating a graph out of confusion matrix
sns.heatmap(cm, annot = True, fmt = 'd')
plt.xlabel('Predicted')
plt.ylabel('Actual')
```

```
Text(33.0, 0.5, 'Actual')
```




```python
# printing classification report
print(classification_report(y_test , y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.99      0.99      2361
           1       0.99      1.00      1.00      2500

    accuracy                           0.99      4861
   macro avg       0.99      0.99      0.99      4861
weighted avg       0.99      0.99      0.99      4861
```

Show the results from each Model in a table format

Results

```python
model_dict={}
model_dict['model']=model
model_dict['model_dist']=model_dist
model_dict['model_small_bert']=model_small_bert
model_dict['model_universal']=model_universal
model_dict['model_albert']=model_albert
model_dict['model_expert']=model_expert
model_dict['model_electra']=model_electra

def getResults(model_dict,sample_texts,sample_target):
    '''
    Get results from different models
    '''
    results=[]

    results_cm={}

    for name,model in model_dict.items():
#        print(name)
        tic1 = time.process_time()
        predicted_sample = model.predict(sample_texts)
        predicted_sample = predicted_sample.flatten().round()
        toc1 = time.process_time()
#        print(predicted_sample)

        cm=sklearn.metrics.confusion_matrix(sample_target, predicted_sample)
        results_cm[name]=cm

        total=len(predicted_sample)
        TP = cm[0][0]
        FP = cm[0][1]
        FN = cm[1][0]
        TN = cm[1][1]

        time_taken=round(toc1 - tic1,4)
        res=sklearn.metrics.precision_recall_fscore_support(sample_target, predicted_sample)
        acc=sklearn.metrics.accuracy_score(sample_target, predicted_sample)
        results.append([name,acc,np.mean(res[0]),np.mean(res[1]),np.mean(res[2]),total,TP,FP,FN,TN,str(time_taken)] )



    df_cols=['model','accuracy','precision','recall','f1_score','Total_samples','TP','FP','FN','TN','execution_time']
    result_df=pd.DataFrame(results,columns=df_cols)

    return result_df,results_cm
```

	model	accuracy	precision	recall	f1_score	Total_samples	TP	FP	FN	TN	execution_time
0	model	0.958033	0.957927	0.958234	0.958016	4861	2279	82	122	2378	46.0625
1	model_dist	0.954536	0.955159	0.955258	0.954536	4861	2315	46	175	2325	42.6406
2	model_small_bert	0.957416	0.957337	0.957446	0.957387	4861	2263	98	109	2391	34.3906
3	model_universal	0.990537	0.990529	0.990529	0.990529	4861	2338	23	23	2477	46.4062
4	model_albert	0.928204	0.932081	0.929694	0.928157	4861	2318	43	306	2194	38.5781
5	model_expert	0.921415	0.921385	0.921728	0.921397	4861	2202	159	223	2277	48.4844
6	model_electra	0.953713	0.953699	0.954070	0.953703	4861	2282	79	146	2354	34.7344

To save the Model:

SAVE THE MODEL

```
import pickle
#from sklearn import model_selection
#filename = 'SPAM.sav'
#pickle.dump(model_universal, open(filename, 'wb'))

import tensorflow as tf
path = r'C:\Users\ermes\OneDrive - National College of Ireland\Desktop\Project_code\Spam/SPAM.tf'
model_universal.save(path )
loaded_model= tf.keras.models.load_model(path)
```

To test the Model on an array of sentences:

```
predict_text1 = [
#ham
'Yes.  Thanks, gm\n\n\n\n\n\n   \n\n\t\n\n\t\n\n\tFrom:  Michelle Lokay                              07/05/2000 08:42 AM\n\n\t\n\n\n\nTo: Ginger McCain/HR/Corp/Enron@ENRON\n\ncc:  \n\n\n\nSubject: Re: Dental Coverage  \n\n\n\nYes, and as a matter of
'Sheila,\n\n\n\nNo, we have to go through the posting phase first.\n\nI shall ask Shirley to provide the job description.\n\n\n\n\n\nVince\n\n\n\n\n\n\n\n\n   \n\n\t\n\n\t\n\n\tFrom:  Sheila Walton                              08/04/2000 02:44 PM\r
#spam
'<html>\n\n<head>\n\n<meta http-equiv=3D"Content-Type" content=3D"text; charset=3Dus-ascii">\n\n</head>\n\n<body><font color=3DFBFBFB size=3D1>safety ill yet sea leave appreciate =\n\ntaking brown similar </font><br>\n\n<center>\n\n<table border=
'don\'t bburn yourself just to gget one brand name watch. pop into our\n\ncyberspace for top brands like Ro,lex, Cart.iers, Bvlgary, frank mullers,\n\nHarryVinstons, breguets, JaegerLecoultre, brietilings, tag heuers and\n\nTu.dors.\n\ndo you luv
]
test_results = loaded_model.predict(predict_text1)
output = np.where(test_results>0.50,'spam', 'ham')
output
```

```
WARNING:tensorflow:5 out of the last 159 calls to <function Model.make_predict_function.<locals>.predict_function at 0x0000023C4FEA1700> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to
(1) creating @tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of tensors. For (1), please define your @tf.function outside of the loop. For (2), @tf.function has reduce_retracing=Tru
e option that can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/guide/function#controlling_retracing and https://www.tensorflow.org/api_docs/python/tf/function for  more details.
WARNING:tensorflow:5 out of the last 159 calls to <function Model.make_predict_function.<locals>.predict_function at 0x0000023C4FEA1700> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to
(1) creating @tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of tensors. For (1), please define your @tf.function outside of the loop. For (2), @tf.function has reduce_retracing=Tru
e option that can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/guide/function#controlling_retracing and https://www.tensorflow.org/api_docs/python/tf/function for  more details.
1/1 [==============================] - 1s 917ms/step
array([['ham'],
       ['ham'],
       ['spam'],
       ['spam']], dtype='<U4')
```

3.2 Text summarisation

We will use Hugging Face's high-level Pipeline API in this part to create summaries with a pre-trained model. There are three main steps involved when you pass some text to a pipeline:

- The text is pre-processed into a format the Model can understand.
- The pre-processed inputs are given to the Model.
- The Model predictions are post-processed so that we can make sense of them.

Import The libraries:

```
import pandas as pd
import numpy as np
import rouge
from sklearn.metrics.pairwise import cosine_similarity
import networkx as nx
import seaborn as sns
import matplotlib.pyplot as plt
from operator import itemgetter
from sentence_transformers import SentenceTransformer

from torch.utils.data import DataLoader
import math
from sentence_transformers import models, losses
from sentence_transformers import SentencesDataset, LoggingHandler, SentenceTransformer
from sentence_transformers.evaluation import EmbeddingSimilarityEvaluator
from sentence_transformers.readers import *
import logging
from datetime import datetime

from transformers import pipeline
```

Import the processed data from Datasets

```
ENRON_PICKLE_LOC = "../Summarisation/dataframes\wrangled_enron_full_df.pkl"
BC3_PICKLE_LOC  = "../Summarisation/dataframes/wrangled_BC3_df.pkl"

enron_df = pd.read_pickle(ENRON_PICKLE_LOC)
BC3_df = pd.read_pickle(BC3_PICKLE_LOC)
```

Define the Rouge metric to evaluate the model

```
evaluator = rouge.Rouge(metrics=['rouge-n'],
                        max_n=1,
                        limit_length=True,
                        length_limit=100,
                        length_limit_type='words',
                        alpha=0.5, # Default F1_score
                        weight_factor=1.2,
                        stemming=True)
```

Select top 50 emails:

```
BC3_df_sel=BC3_df[:50]
```

```
ref_summaries = list(BC3_df_sel['summary'])
```

Load the BART model Using a Hugging face

```
summarizer = pipeline("summarization", model="facebook/bart-large-cnn")
```

BART model is pre-trained on the English language and fine-tuned on CNN Daily Mail. After that, it was introduced in the paper BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension by Lewis et al. and first released in [this repository (https://github.com/pytorch/fairseq/tree/master/examples/bart).

BART is a transformer encoder-encoder (seq2seq) model with a bidirectional (BERT-like) encoder and an autoregressive (GPT-like) decoder. BART is pre-trained by (1) corrupting text with an arbitrary noising function and (2) learning a model to reconstruct the original text.

BART is particularly effective when fine-tuned for text generation (e.g. summarisation, translation) but also works well for comprehension tasks (e.g. text classification, question answering). This particular checkpoint has been fine-tuned on CNN Daily Mail, a large collection of text-summary pairs.

To Apply the Model to each email and evaluate using the ROUGE metric:

```
import progressbar
df_sum=BC3_df_sel
bar = progressbar.ProgressBar(maxval=len(df_sum)).start()
candidate_summaries = []
scores = []
scoresp = []
scoresr = []

full_scores = []
#for i in tqdm (range (1), desc="Loading..."):
for index, row in df_sum.iterrows():
    reference = row.summary
    data=row.body
    hypothesis = summarizer(data,min_length=int(0.1 * len(data)), max_length=int(0.2 * len(data)))
    try:
        full_score=evaluator.get_scores(hypothesis[0]['summary_text'], reference)
        score = full_score['rouge-1']['f']
        scorep= full_score['rouge-1']['p']
        scorer= full_score['rouge-1']['r']
    except:
        score = 0.0
    ref_summaries
    candidate_summaries.append(hypothesis[0]['summary_text'])
    scores.append((score, reference, hypothesis))
    scoresp.append((scorep, reference, hypothesis))
    scoresr.append((scorer, reference, hypothesis))
    full_scores.append((full_score, reference, hypothesis))
    bar.update(index)
```

```
98% (49 of 50) |####################### | Elapsed Time: 0:03:39 ETA:   0:00:05
```

To calculate the Precision and Recall average score:

```
# Precision
unzipped = list(zip(*scoresp))
best_scores = unzipped[0]
scoresp = [i for i in scoresp if i != 0.0]
print("Number of human summaries " + str(len(best_scores)))
print('p average =  ' + str(sum(best_scores) / len(best_scores)))
#print('r average =  ' + str(sum(scoresr) / len(scoresr)))
```

```
Number of human summaries 50
p average =  0.20176435536315285
```

```
#Recall
unzipped = list(zip(*scoresr))
best_scores = unzipped[0]
scoresr = [i for i in scoresr if i != 0.0]
print("Number of human summaries " + str(len(best_scores)))
print('r average =  ' + str(sum(best_scores) / len(best_scores)))
#print('r average =  ' + str(sum(scoresr) / len(scoresr)))
```

```
Number of human summaries 50
r average =  0.4196758925145629
```

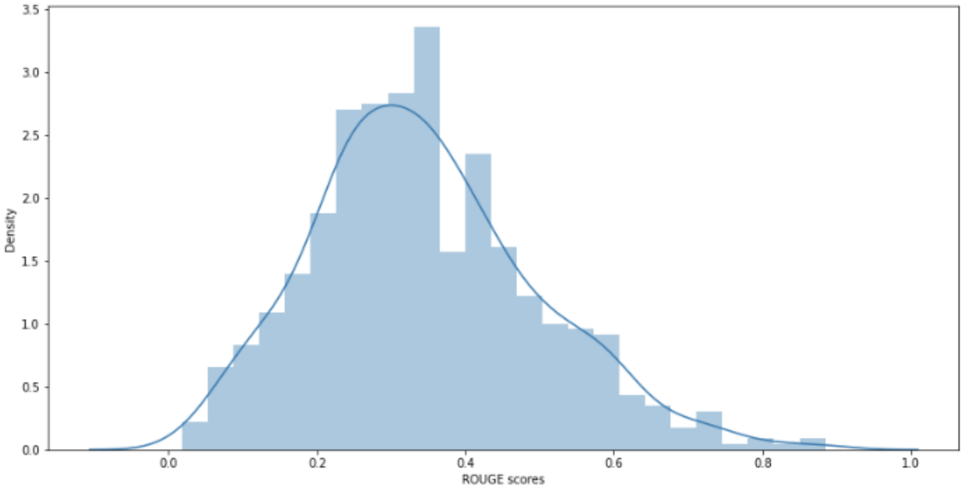
To plot the Rouge F1 score:

```
def plot_scores(scores):
    #removing 0 outliers since it meant page rank failed
    scores = [i for i in scores if i != 0.0]
    print("Number of human summaries " + str(len(scores)))
    print('F1 average =  ' + str(sum(scores) / len(scores)))

    _,ax = plt.subplots(figsize = (14,7))
    sns.distplot(scores, bins = 25)
    ax.set_xlabel('ROUGE scores')
```

```
unzipped = list(zip(*scores))
best_scores = unzipped[0]
plot_scores(best_scores)
```

```
Number of human summaries 662
F1 average =  0.3452468016613802
```

```
C:\Users\ermes\anaconda3\envs\gpu2\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will
n with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



To save the Model:

SAVE THE MODEL

```
import pickle
from sklearn import model_selection
filename = 'SUMMARISATION.sav'
pickle.dump(summarizer, open(filename, 'wb'))
```

```
# Load the model from disk
summarizer_load = pickle.load(open(filename, 'rb'))
test=(' Hello! My name is Shafaq. Your website or a website that your company hosts is infringing on a copyright-protected images owned by myself.'
      ' Take a look at this document with the links to my images you used at website.berkeley.edu and my earlier publications to get the evidence of my copyrights.'
      ' Download it right now and check this out for yourself: hxxps://sites.google.com/view/a0hf49gj29g-i4jb48n5/drive/folders/shared/1/download?ID=308682351554855915 '
      ' I believe you have willfully infringed my rights under 17 U.S.C. Section 101 et seq. and could be liable for statutory damages as high as $150,000 as set forth in Section 504(c)(2) '
      ' of the Digital Millennium Copyright Act ("DMCA") therein')
```

```
summarizer_load(test,min_length=int(0.1 * len(data)), max_length=int(0.2 * len(data)))
```

```
[{'summary_text': ' Your website or a website that your company hosts is infringing on a copyright-protected images owned by myself . Take a look at this document with the links to my images you used at website.berkeley.edu and my earlier publica
tions to get the evidence of my copyrights . I believe you have willfully infringed my rights under 17 U.S.C. Section 101 et seq. and could be liable for statutory damages as high as $150,000 .'}]
```

3.3 Intent Recognition

Import the necessary Libraries:

```
import tensorflow as tf
from tensorflow import keras

import os
import tempfile

import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns

import sklearn
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

```
mpl.rcParams['figure.figsize'] = (12, 10)
colors = plt.rcParams['axes.prop_cycle'].by_key()['color']
```

```
import tensorflow as tf
from tensorflow import keras
import os
import re
from collections import Counter
import numpy as np
import datetime

is_windows = (os.name == 'nt')
configproto = tf.compat.v1.ConfigProto()
configproto.gpu_options.allow_growth = True
sess = tf.compat.v1.Session(config=configproto)
tf.compat.v1.keras.backend.set_session(sess)
```

```
WARNING:tensorflow:From C:\Users\ermes\AppData\Local\Temp\ipykernel_10876\1855086169.py:13: The name tf.keras.backend.set_session is deprecat
ed.
```

```
import numpy as np
import matplotlib as plt
import pandas as pd
import xml.etree.ElementTree as ET
import io
```

Download the dataset from AnnotatedThreads and load it:

```
mypath_nontrain = 'PowerAnnotations_V1.0/AnnotatedThreads/non train/'
#run this for train in case you need it
mypath_train = 'PowerAnnotations_V1.0/AnnotatedThreads/train/'
```

```
# get all files in first folder.
from os import listdir
from os.path import isfile, join
filesInDir_nontrain = [f for f in listdir(mypath_nontrain) if (isfile(join(mypath_nontrain, f)) and f[-4:] == '.xml')]
```

```
# get all files in second folder.
from os import listdir
from os.path import isfile, join
filesInDir_train = [f for f in listdir(mypath_train) if (isfile(join(mypath_train, f)) and f[-4:] == '.xml')]
```

```
data = {}
for i, f in enumerate(filesInDir_nontrain):
    file = open(mypath_nontrain + f, 'r' , encoding="ISO-8859-1")
    data[f] = file.read()

for i, f in enumerate(filesInDir_train):
    file = open(mypath_train + f, 'r' , encoding="ISO-8859-1")
    data[f] = file.read()
```

Create a regular expression to select all the sentences tagged with “Inform” and “Request-Action” contained between []

```
import re
import os.path
df1 = pd.DataFrame()
for k, v in data.items():
    rgx = re.compile(r'[^?=\[\]]+(?=\])', re.MULTILINE)
    for match in rgx.finditer(v):
        searchObj = match.group(0)
        df1 = df1.append(pd.Series(searchObj),ignore_index=True)
```

```
C:\Users\ermes\AppData\Local\Temp\ipykernel_10876\3446751070.py:12: FutureWarning: The frame.append method is de
  df1 = df1.append(pd.Series(searchObj),ignore_index=True)
C:\Users\ermes\AppData\Local\Temp\ipykernel_10876\3446751070.py:12: FutureWarning: The frame.append method is de
  df1 = df1.append(pd.Series(searchObj),ignore_index=True)
```

```
df1
```

	0
0	Inform: time and place of party celebrating En...
1	Y
2	Inform: website where guests can RSVP
3	Conventional: Andy hopes to see recipients there
4	Inform: link for RSVP did not work
...	...
1139	Inform: Mertz has had a long-standing desire t...
1140	Inform: Erik and recipients need to kill some ...
1141	Inform: recipients should sight their guns and...
1142	Conventional: signature
1143	Inform: sarcastic response indicating strong s...

1144 rows × 1 columns

Filter all the rows with length <2:

```
df1 = df1[df1.length > 2]
```

```
df1['action'], df1['text'] = df1['sentences'].str.split(':', 1).str
```

```
C:\Users\ermes\AppData\Local\Temp\ipykernel_10876\2294770400.py:1: FutureWarning: Columnar iteration over characters will be deprecated in future releases.
  df1['action'], df1['text'] = df1['sentences'].str.split(':', 1).str
C:\Users\ermes\AppData\Local\Temp\ipykernel_10876\2294770400.py: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#returning-a-view-versus-a-copy
  df1['action'], df1['text'] = df1['sentences'].str.split(':', 1).str
C:\Users\ermes\AppData\Local\Temp\ipykernel_10876\2294770400.py: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#returning-a-view-versus-a-copy
  df1['action'], df1['text'] = df1['sentences'].str.split(':', 1).str
```

```
#labels=['Inform', 'Conventional','Request-Action', 'Commit','Inform-AnswerOffline']
labels=['Inform','Request-Action']
```

```
df1 = df1[~df1['action'].isin(labels)== False]
```

Split the data into test, train and validation sets.

```
traindf , validdf , testdf  = np.split(df_data.sample(frac=1), [int(.6*len(df_data)), int(.8*len(df_data))])
```

To balance the dataset, perform data augmentation with the following code on the minority class:

Data Augmentation

```
import nlpaug.augmenter.word as naw
```

```
TOPK=20 #default=100
ACT = 'insert' #"substitute"

aug_bert = naw.ContextualWordEmbsAug(
    model_path='distilbert-base-uncased',
    #device='cuda',
    action=ACT, top_k=TOPK)
#print("Original:")

#print("Augmented Text:")
for sentence in traindf.loc[traindf['action'] == 'Request-Action'].text:
    #print('Original :' + sentence)
    for ii in range(5):
        augmented_text = aug_bert.augment(sentence)
        new_raw={'action':'Request-Action','text': str(augmented_text[0])}
        traindf=traindf.append(new_raw, ignore_index=True)
        #print(augmented_text)

for sentence in traindf.loc[traindf['action'] == 'Request-Action'].text:
    #print('Original :' + sentence)
    for ii in range(3):
        augmented_text = aug_bert.augment(sentence)
        new_raw={'action':'Request-Action','text': str(augmented_text[0])}
        traindf=traindf.append(new_raw, ignore_index=True)
        #print(augmented_text)
```

Shuffle the Dataset.

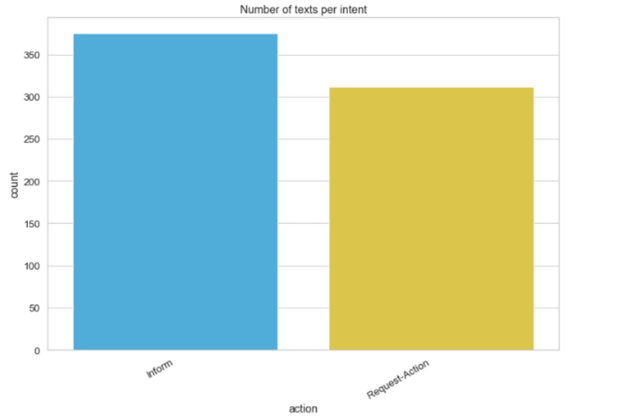
```
traindf.sample(frac=1)
```

	text	action
12	recipients should let Wade know of any change...	Inform
78	Mavis said she would call legislators such as...	Inform
370	deal ticket that volume is currently posted u...	Inform
420	respond by close of business meeting on thursd...	Request-Action
580	order chris buys for a power home rangers week...	Request-Action
...	...	...
133	Chris is available on Wednesday	Inform
675	airports arrange frequent shuttle charter flig...	Request-Action
391	gather monitoring team to review rotations ' I...	Request-Action
340	joke about Linda packing her bag and waiting ...	Inform
191	Michelle is not amused by the joke	Inform

687 rows × 2 columns

Plot the data after data augmentation:

```
#Distribution of class-labels in training-data:
chart = sns.countplot(trainlabels, palette=HAPPY_COLORS_PALETTE)
plt.title("Number of texts per intent")
chart.set_xticklabels(chart.get_xticklabels(), rotation=30, horizontalalignment='right');
```



You can load various Models from TensorFlow Hub¶
Here you can choose which BERT model you will load from TensorFlow Hub and fine-tune. There are multiple BERT models available.

- Small BERTs have the same general architecture but fewer and/or smaller Transformer blocks, which lets you explore tradeoffs between speed, size and quality.
- ALBERT: four different sizes of "A Lite BERT" that reduces the model size (but not computation time) by sharing parameters between layers.
- BERT Experts: eight models with the BERT-base architecture offer a choice between different pre-training domains to align more closely with the target task.
- Electra has the same architecture as BERT (in three different sizes) but gets pre-trained as a discriminator in a set-up that resembles a Generative Adversarial Network (GAN).
- BERT with Talking-Heads Attention and Gated GELU [base, large] has two improvements to the core of the Transformer architecture.

The model documentation on TensorFlow Hub https://www.tensorflow.org/hub has more details

The suggestion is to start with a Small BERT (with fewer parameters) since they are faster to fine-tune. If you like a small model with higher accuracy, ALBERT might be your next option. If you want even better accuracy, choose one of the classic BERT sizes or recent refinements like Electra, Talking Heads, or a BERT Expert.

Aside from the models available below, multiple versions of the models are larger and can yield even better accuracy, but they are too big to be fine-tuned on a single GPU.

Load BERT Expert by selecting the Model from the list:

```
#bert_model_name = 'small_bert/bert_en_uncased_L-8_H-512_A-8'
#bert_model_name = 'bert_en_uncased_L-12_H-768_A-12'
#bert_model_name = 'albert_en_base'
#bert_model_name = 'electra_small'
bert_model_name = 'experts_wiki_books'
#bert_model_name = 'talking-heads_base'

map_name_to_handle = {
    'bert_en_uncased_L-12_H-768_A-12':
        'https://tfhub.dev/tensorflow/bert_en_uncased_L-12_H-768_A-12/3',
    'bert_en_cased_L-12_H-768_A-12':
        'https://tfhub.dev/tensorflow/bert_en_cased_L-12_H-768_A-12/3',
    'bert_multi_cased_L-12_H-768_A-12':
        'https://tfhub.dev/tensorflow/bert_multi_cased_L-12_H-768_A-12/3',
    'small_bert/bert_en_uncased_L-2_H-128_A-2':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-2_H-128_A-2/1',
    'small_bert/bert_en_uncased_L-2_H-256_A-4':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-2_H-256_A-4/1',
    'small_bert/bert_en_uncased_L-2_H-512_A-8':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-2_H-512_A-8/1',
    'small_bert/bert_en_uncased_L-2_H-768_A-12':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-2_H-768_A-12/1',
    'small_bert/bert_en_uncased_L-4_H-128_A-2':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-4_H-128_A-2/1',
    'small_bert/bert_en_uncased_L-4_H-256_A-4':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-4_H-256_A-4/1',
    'small_bert/bert_en_uncased_L-4_H-512_A-8':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-4_H-512_A-8/1',
    'small_bert/bert_en_uncased_L-4_H-768_A-12':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-4_H-768_A-12/1',
    'small_bert/bert_en_uncased_L-6_H-128_A-2':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-6_H-128_A-2/1',
    'small_bert/bert_en_uncased_L-6_H-256_A-4':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-6_H-256_A-4/1',
    'small_bert/bert_en_uncased_L-6_H-512_A-8':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-6_H-512_A-8/1',
    'small_bert/bert_en_uncased_L-6_H-768_A-12':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-6_H-768_A-12/1',
    'small_bert/bert_en_uncased_L-8_H-128_A-2':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-8_H-128_A-2/1',
    'small_bert/bert_en_uncased_L-8_H-256_A-4':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-8_H-256_A-4/1',
    'small_bert/bert_en_uncased_L-8_H-512_A-8':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-8_H-512_A-8/1',
    'small_bert/bert_en_uncased_L-8_H-768_A-12':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-8_H-768_A-12/1',
    'small_bert/bert_en_uncased_L-10_H-128_A-2':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-10_H-128_A-2/1',
    'small_bert/bert_en_uncased_L-10_H-256_A-4':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-10_H-256_A-4/1',
    'small_bert/bert_en_uncased_L-10_H-512_A-8':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-10_H-512_A-8/1',
    'small_bert/bert_en_uncased_L-10_H-768_A-12':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-10_H-768_A-12/1',
    'small_bert/bert_en_uncased_L-12_H-128_A-2':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-12_H-128_A-2/1',
    'small_bert/bert_en_uncased_L-12_H-256_A-4':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-12_H-256_A-4/1',
    'small_bert/bert_en_uncased_L-12_H-512_A-8':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-12_H-512_A-8/1',
    'small_bert/bert_en_uncased_L-12_H-768_A-12':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-12_H-768_A-12/1',
    'albert_en_base':
        'https://tfhub.dev/tensorflow/albert_en_base/2',
    'electra_small':
```

To build the Model:

```
def build_classifier_model():
    text_input = tf.keras.layers.Input(shape=(), dtype=tf.string, name='text')
    preprocessing_layer = hub.KerasLayer(tfhub_handle_preprocess, name='preprocessing')
    encoder_inputs = preprocessing_layer(text_input)
    encoder = hub.KerasLayer(tfhub_handle_encoder, trainable=True, name='BERT_encoder')
    outputs = encoder(encoder_inputs)

    net = outputs['pooled_output']
    net = tf.keras.layers.Dropout(0.1, name = 'Dropout')(net)
    net = tf.keras.layers.Dense(1, activation = 'sigmoid', name = 'Dense')(net)
    return tf.keras.Model(text_input, net)
```

```
m_expert=build_classifier_model()
```

We now have all the pieces of training a model, including the pre-processing module, BERT encoder, data, and classifier.

Since this is a binary classification problem and the model outputs probabilities, we'll use *losses.BinaryCrossentropy* loss function.

To Loading the BERT model and training
Using the *classifier_model* you created earlier, you can compile the Model with the loss, metric and optimiser:

```
loss = tf.keras.losses.BinaryCrossentropy(from_logits=True)
metrics = tf.metrics.CategoricalAccuracy()
#loss=cross_entropy
```

```
Metrics = [tf.keras.metrics.BinaryAccuracy(name = 'accuracy'),
           tf.keras.metrics.Precision(name = 'precision'),
           tf.keras.metrics.Recall(name = 'recall')
           ]
```

```
epochs=5
optimizer=tf.keras.optimizers.Adam(1e-5)
m_expert.compile(optimizer = optimizer,
                        #loss=tf.keras.losses.kullback_leibler_divergence,
                         loss = 'binary_crossentropy',
                         metrics = Metrics)
```

```
print(f'Training model with {tfhub_handle_encoder}')
history = m_expert.fit(x=trainfeatures,y=trainlabels,
                               validation_data=(validfeatures,validlabels),
                               batch_size=8,
                               epochs=epochs)
```

```
Training model with https://tfhub.dev/google/experts/bert/wiki_books/2
Epoch 1/5
86/86 [==============================] - 23s 158ms/step - loss: 0.5959 - accuracy: 0.6856 - precision: 0.6633 -
ecall: 0.5455
Epoch 2/5
86/86 [==============================] - 13s 151ms/step - loss: 0.3258 - accuracy: 0.8646 - precision: 0.8544 -
ecall: 0.4545
Epoch 3/5
86/86 [==============================] - 13s 150ms/step - loss: 0.1381 - accuracy: 0.9549 - precision: 0.9432 -
ecall: 0.4545
Epoch 4/5
86/86 [==============================] - 13s 151ms/step - loss: 0.0695 - accuracy: 0.9854 - precision: 0.9871 -
ecall: 0.6364
Epoch 5/5
86/86 [==============================] - 13s 152ms/step - loss: 0.0457 - accuracy: 0.9854 - precision: 0.9840 -
ecall: 0.6364
```

Evaluate the model

Let's see how the model performs. Two values will be returned. Loss (a number which represents the error, lower values are better), and accuracy.

```
#testlabels
```

```
loss, accuracy, *is_anything_else_being_returned = m_electra.evaluate(testfeatures,testlabels)

print(f'Loss: {loss}')
print(f'Accuracy: {accuracy}')
```

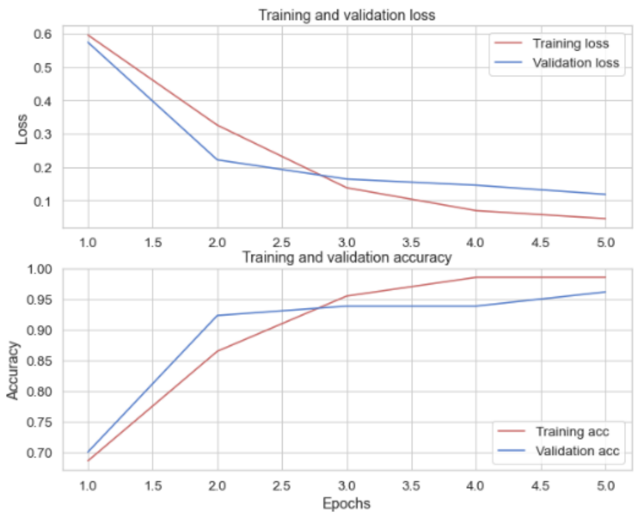
```
5/5 [==============================] - 1s 136ms/step - loss: 0.0523 - accuracy: 0.9846 - precision: 1.0000 - recall: 0.7143
Loss: 0.05227086320519447
Accuracy: 0.9846153855323792
```

Plot the accuracy and loss over time

Based on the History object returned by model.fit(). You can plot the training and validation loss for comparison, as well as the training and validation accuracy:

```
history_dict = history.history
print(history_dict.keys())

acc = history_dict['accuracy']
val_acc = history_dict['val_accuracy']
loss = history_dict['loss']
val_loss = history_dict['val_loss']

epochs = range(1, len(acc) + 1)
fig = plt.figure(figsize=(10, 8))
fig.tight_layout()

plt.subplot(2, 1, 1)
# "bo" is for "blue dot"
plt.plot(epochs, loss, 'r', label='Training loss')
# b is for "solid blue line"
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.grid(True)
# plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.subplot(2, 1, 2)
plt.plot(epochs, acc, 'r', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.grid(True)
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
```

```
dict_keys(['loss', 'accuracy', 'precision', 'recall', 'val_loss', 'val_accuracy', 'val_precision', 'val_recall'])
<matplotlib.legend.Legend at 0x11aacfd46d0>
```



Save The Model

```
## SAVE THE MODEL
import pickle
#from sklearn import model_selection
#filename = 'SPAM.sav'
#pickle.dump(model_universal, open(filename, 'wb'))

import tensorflow as tf
path = r'./INTENT.tf'
m_expert.save(path)
loaded_model= tf.keras.models.load_model(path)
```

```
WARNING:absl:Found untraced functions such as restored_function_body, restored_function_body, restored_function_body, restored_function_body, restored_function_body while saving (showing 5 o
ese functions will not be directly callable after loading.
```

```
examples = ['Hello! My name is Shafaq. Your website or a website that your company hosts is infringing on a copyright-protected images owned by myself.',
            'Take a look at this document with the links to my images you used at website.berkeley.edu and my earlier publications to get the evidence of my copyrights.',
            'Download it right now and check this out for yourself: hxxps://sites.google.com/view/a0hf49gj29g-i4jb48n5/drive/folders/shared/1/download?ID=308682351554855915',
            ' I believe you have willfully infringed my rights under 17 USC Section 101 et seq. and could be liable for statutory damages as high as $150,000 as set forth in Section 504(c)(2
```

```
results = tf.nn.relu(loaded_model(tf.constant(examples)))
actions=binarizer.inverse_transform(results.numpy())
actions
```

```
array(['Inform', 'Inform', 'Request-Action', 'Request-Action'],
      dtype='<U14')
```

3.4 Speech acts Tags

Import the necessary Libraries:

```
import os

#import shutil
import pandas as pd
import numpy as np

import tensorflow as tf
import tensorflow_hub as hub
import tensorflow_text as text
import seaborn as sns
from pylab import rcParams

import sklearn
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler


import matplotlib.pyplot as plt
tf.get_logger().setLevel('ERROR')

sns.set(style='whitegrid', palette='muted', font_scale=1.2)
HAPPY_COLORS_PALETTE = ["#01BEFE", "#FFDD00", "#FF7D00", "#FF006D", "#ADFF02", "#8F00FF"]
sns.set_palette(sns.color_palette(HAPPY_COLORS_PALETTE))
rcParams['figure.figsize'] = 12, 8
import warnings
warnings.filterwarnings("ignore")
```

Restrict the TensorFlow library to use as much memory as you wish to avoid OOM.

```
tf.keras.backend.clear_session()
```

```
import tensorflow as tf
print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))
```

```
Num GPUs Available:  1
```

```
gpus = tf.config.list_physical_devices('GPU')
if gpus:
  # Restrict TensorFlow to only use the first GPU
    try:
        tf.config.set_visible_devices(gpus[0], 'GPU')
        logical_gpus = tf.config.list_logical_devices('GPU')
        print(len(gpus), "Physical GPUs,", len(logical_gpus), "Logical GPU")
    except RuntimeError as e:
        # Visible devices must be set before GPUs have been initialized
        print(e)
```

```
1 Physical GPUs, 1 Logical GPU
```

Import the data from: https://data.world/brianray/enron-email-dataset/workspace/file?filename=enron_05_17_2015_with_labels_v2

Load the data:

```
enron = pd.read_csv("C:\\Users\\ermes\\OneDrive - National College of Ireland\\Desktop\\Project\\Intent\\enron_with_labels\\enron_05_17_2015_with_labels_v2.csv")
```

Add tags “click-link”, “download”, and “neutral” to each sentence in the dataset:

```
enron_query=enron.loc[enron['labeled']==True]
```

```
df=enron_query
```

```
# filtering the rows where Credit-Rating is Fair
df_link = df[df['content'].astype(str).str.contains('click')]
#print(df_link)
```

```
# filtering the rows where Credit-Rating is Fair
df_download = df[df['content'].astype(str).str.contains('download')]
#print(df_download)
```

```
# filtering the rows where Credit-Rating is Fair
df_attach = df[df['content'].astype(str).str.contains('click|download')==False]
#print(df_attach)
```

```
df_link_data = df_link.sample(n = 300, replace=True)
df_download_data = df_download.sample(n = 300, replace=True)
df_attach_data = df_attach.sample(n = 300, replace=True)
```

```
df_link_data['intent']='clicklink'
df_download_data['intent']='downloadfile'
df_attach_data['intent']='neutral'
```

```
df_data=df_link_data.append(df_download_data)
df_data=df_data.append(df_attach_data)
```

```
df_data = df_data[['content','intent']]
```

Shuffle and split the data into train, test and validation datasets:

```
df_data = df_data.sample(frac=1).reset_index(drop=True)
```

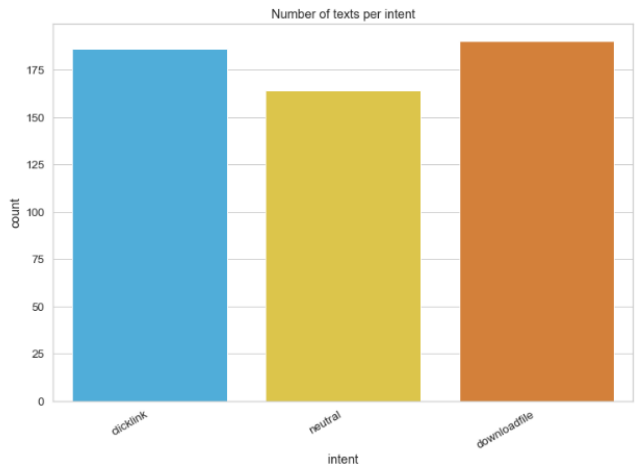
```
traindf , validdf , testdf  = np.split(df_data.sample(frac=1), [int(.6*len(df_data)), int(.8*len(df_data))])
```

Plot the data to check if balanced:

```
trainfeatures=traindf.copy()
trainlabels=trainfeatures.pop("intent")
```

```
trainfeatures=trainfeatures.values
```

```
#Distribution of class-labels in training-data:
chart = sns.countplot(trainlabels, palette=HAPPY_COLORS_PALETTE)
plt.title("Number of texts per intent")
chart.set_xticklabels(chart.get_xticklabels(), rotation=30, horizontalalignment='right');
```



Load the desired Model from TensorFlow Hub. Here you can choose which BERT model you will load from TensorFlow Hub and fine-tune. There are multiple BERT models available.

- Small BERTs have the same general architecture but fewer and/or smaller Transformer blocks, which lets you explore tradeoffs between speed, size and quality.
- ALBERT: four different sizes of “A Lite BERT” that reduces the model size (but not computation time) by sharing parameters between layers.
- BERT Experts: eight models with the BERT-base architecture offer a choice between different pre-training domains to align more closely with the target task.
- Electra has the same architecture as BERT (in three different sizes) but gets pre-trained as a discriminator in a set-up that resembles a Generative Adversarial Network (GAN).
- BERT with Talking-Heads Attention and Gated GELU [base, large] has two improvements to the core of the Transformer architecture.

```
bert_model_name = 'albert_en_base'
map_name_to_handle = {
    'bert_en_uncased_L-12_H-768_A-12':
        'https://tfhub.dev/tensorflow/bert_en_uncased_L-12_H-768_A-12/3',
    'bert_en_cased_L-12_H-768_A-12':
        'https://tfhub.dev/tensorflow/bert_en_cased_L-12_H-768_A-12/3',
    'bert_multi_cased_L-12_H-768_A-12':
        'https://tfhub.dev/tensorflow/bert_multi_cased_L-12_H-768_A-12/3',
    'small_bert/bert_en_uncased_L-2_H-128_A-2':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-2_H-128_A-2/1',
    'small_bert/bert_en_uncased_L-2_H-256_A-4':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-2_H-256_A-4/1',
    'small_bert/bert_en_uncased_L-2_H-512_A-8':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-2_H-512_A-8/1',
    'small_bert/bert_en_uncased_L-2_H-768_A-12':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-2_H-768_A-12/1',
    'small_bert/bert_en_uncased_L-4_H-128_A-2':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-4_H-128_A-2/1',
    'small_bert/bert_en_uncased_L-4_H-256_A-4':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-4_H-256_A-4/1',
    'small_bert/bert_en_uncased_L-4_H-512_A-8':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-4_H-512_A-8/1',
    'small_bert/bert_en_uncased_L-4_H-768_A-12':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-4_H-768_A-12/1',
    'small_bert/bert_en_uncased_L-6_H-128_A-2':
        'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-6_H-128_A-2/1',
    'small_bert/bert_en_uncased_L-6_H-256_A-4':
```

Define the Model. Create a very simple fine-tuned model with the pre-processing Model, the selected BERT model, one Dense and a Dropout layer.

```
def build_classifier_model():
    text_input = tf.keras.layers.Input(shape=(), dtype=tf.string, name='text')
    preprocessing_layer = hub.KerasLayer(tfhub_handle_preprocess, name='preprocessing')
    encoder_inputs = preprocessing_layer(text_input)
    encoder = hub.KerasLayer(tfhub_handle_encoder, trainable=True, name='BERT_encoder')
    outputs = encoder(encoder_inputs)

    net = outputs['pooled_output']
    net = tf.keras.layers.Dropout(0.1)(net)
    net = tf.keras.layers.Dense(3, activation='sigmoid', name='classifier')(net)
    return tf.keras.Model(text_input, net)
```

```
#Let’s check that the model runs with the output of the preprocessing model.
```

```
classifier_model = build_classifier_model()
```

Since this is a non-binary classification problem and the model outputs probabilities, we use *losses.CategoricalCrossentropy* loss function.

```
loss = tf.keras.losses.CategoricalCrossentropy(from_logits=True)
metrics = tf.metrics.CategoricalAccuracy()
```

To train the Model:

```
epochs=5
optimizer=tf.keras.optimizers.Adam(1e-5)
classifier_model.compile(optimizer=optimizer,
                         loss=loss,
                         metrics=metrics)
```

Note: training time will vary depending on the complexity of the BERT model you have selected.

```
print(f'Training model with {tfhub_handle_encoder}')
history = classifier_model.fit(x=trainfeatures,y=trainlabels,
                               validation_data=(validfeatures,validlabels),
                               batch_size=16,
                               epochs=epochs)
```

```
Training model with https://tfhub.dev/tensorflow/albert_en_base/2
Epoch 1/5
34/34 [==============================] - 26s 387ms/step - loss: 0.8504 - categorical_accuracy: 0.6037 - val_loss: 0.4595 - val_categorical_accuracy: 0.8722
Epoch 2/5
34/34 [==============================] - 13s 374ms/step - loss: 0.3244 - categorical_accuracy: 0.9000 - val_loss: 0.2075 - val_categorical_accuracy: 0.9722
Epoch 3/5
34/34 [==============================] - 13s 374ms/step - loss: 0.1534 - categorical_accuracy: 0.9574 - val_loss: 0.1337 - val_categorical_accuracy: 0.9611
Epoch 4/5
34/34 [==============================] - 13s 374ms/step - loss: 0.0901 - categorical_accuracy: 0.9722 - val_loss: 0.1169 - val_categorical_accuracy: 0.9667
Epoch 5/5
34/34 [==============================] - 13s 377ms/step - loss: 0.0605 - categorical_accuracy: 0.9852 - val_loss: 0.1080 - val_categorical_accuracy: 0.9667
```

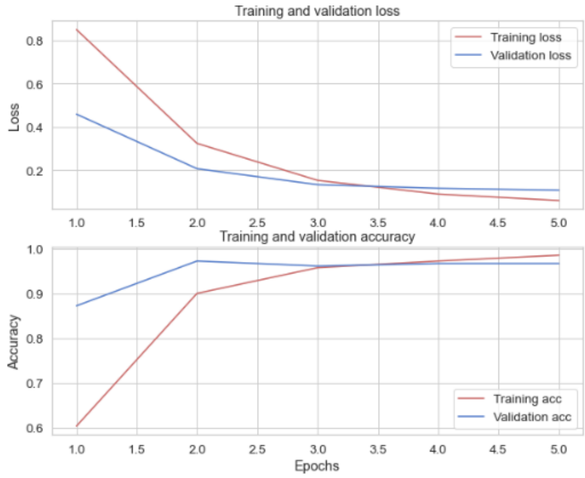
Let's see how the Model performs. Two values will be returned. Loss (a number representing the error, lower values are better) and accuracy. To Evaluate the Model:

```
loss, accuracy = classifier_model.evaluate(testfeatures,testlabels)

print(f'Loss: {loss}')
print(f'Accuracy: {accuracy}')
```

```
6/6 [==============================] - 2s 209ms/step - loss: 0.1760 - categorical_accuracy: 0.9333
Loss: 0.17596107721328735
Accuracy: 0.9333333373069763
```

Based on the History object returned by the Model. fit(). You can plot the training and validation loss for comparison and the training and validation accuracy. To plot:

```
history_dict = history.history
print(history_dict.keys())

acc = history_dict['categorical_accuracy']
val_acc = history_dict['val_categorical_accuracy']
loss = history_dict['loss']
val_loss = history_dict['val_loss']

epochs = range(1, len(acc) + 1)
fig = plt.figure(figsize=(10, 8))
fig.tight_layout()

plt.subplot(2, 1, 1)
# "bo" is for "blue dot"
plt.plot(epochs, loss, 'r', label='Training loss')
# b is for "solid blue line"
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.grid(True)
# plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.subplot(2, 1, 2)
plt.plot(epochs, acc, 'r', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.grid(True)
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
```



To save the Created Model:

```
## SAVE THE MODEL
import pickle
#from sklearn import model_selection
#filename = 'SPAM.sav'
#pickle.dump(model_universal, open(filename, 'wb'))

import tensorflow as tf
path = r'./TAGS.tf'
classifier_model.save(path)
loaded_model= tf.keras.models.load_model(path)
```

3.5 Emotion Recognition

Import the necessary libraries:

```
import tensorflow as tf
from tensorflow import keras
import os
import re
from collections import Counter
import numpy as np
import datetime

import warnings
warnings.filterwarnings('ignore')
#is_windows = (os.name == 'nt')
#configproto = tf.compat.v1.ConfigProto()
#configproto.gpu_options.allow_growth = True
#sess = tf.compat.v1.Session(config=configproto)
#tf.compat.v1.keras.backend.set_session(sess)
```

```
from numpy import array
from keras.preprocessing.text import one_hot
from keras_preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers.core import Activation, Dropout, Dense
from keras.layers import Flatten, LSTM
from keras.layers import GlobalMaxPooling1D
from keras.models import Model
from keras.layers import Embedding
from sklearn.model_selection import train_test_split
from keras.preprocessing.text import Tokenizer
from keras.layers import Input
from keras.layers import Concatenate
import sklearn

import pandas as pd
import numpy as np
import re

import matplotlib.pyplot as plt
```

Load the data from [dataset_finished.csv]

```
# Read the trainings data
text = (open(r'C:\Users\ermes\OneDrive - National College of Ireland\Desktop\Project\emotion\social-engineering-master\dataset_finished.csv', 'rb').read()).decode('utf-8', 'ignore')
```

```
df=df.drop(['spam','Label'], axis=1)
```

Perform an undersampling of the majority class:

```
excess = len(df[df['Reciprocity']==1]) - len(df[df['Scarcity']==1])
remove = np.random.choice(df[df['Reciprocity']==1].ID, excess, replace=False)
df = df[~df.ID.isin(remove)]
```

Perform Data augmentation on all the minority classes using *DistilBERT* and *nlpaug* library:

```
import nlpaug.augmenter.word as naw

TOPK=20 #default=100
ACT = 'insert' #"substitute"

aug_bert = naw.ContextualWordEmbsAug(
    model_path='distilbert-base-uncased',
    #device='cuda',
    action=ACT, top_k=TOPK)
#print("Original:")

#print("Augmented Text:")
for sentence in df.loc[df.Concistency == 1].sentence:
    #print('Original :' + sentence)
    for ii in range(6):
        augmented_text = aug_bert.augment(sentence)
        new_raw={'Reciprocity':0, 'Concistency':1, 'SocialProof':0, 'Authority':0,'Liking':0,'Scarcity':0,'sentence': str(augmented_text[0])}
        df=df.append(new_raw, ignore_index=True)
        #df=pd.concat(df, pd.DataFrame(new_raw))
        #print(new_raw)
```

```
import nlpaug.augmenter.word as naw

TOPK=20 #default=100
ACT = 'insert' #"substitute"

aug_bert = naw.ContextualWordEmbsAug(
    model_path='distilbert-base-uncased',
    #device='cuda',
    action=ACT, top_k=TOPK)
#print("Original:")

#print("Augmented Text:")
for sentence in df.loc[df.Scarcity == 1].sentence:
    #print('Original :' + sentence)
    for ii in range(2):
        augmented_text = aug_bert.augment(sentence)
        new_raw={'Reciprocity':0, 'Concistency':0, 'SocialProof':0, 'Authority':0,'Liking':0,'Scarcity':1,'sentence': str(augmented_text[0])}
        df=df.append(new_raw, ignore_index=True)
        #df=pd.concat(df, pd.DataFrame(new_raw))
        #print(new_raw)
```

```
import nlpaug.augmenter.word as naw

TOPK=20 #default=100
ACT = 'insert' #"substitute"

aug_bert = naw.ContextualWordEmbsAug(
    model_path='distilbert-base-uncased',
    #device='cuda',
    action=ACT, top_k=TOPK)
#print("Original:")

#print("Augmented Text:")
for sentence in df.loc[df.Liking == 1].sentence:
    #print('Original :' + sentence)
    for ii in range(2):
        augmented_text = aug_bert.augment(sentence)
        new_raw={'Reciprocity':0, 'Concistency':0, 'SocialProof':0, 'Authority':0,'Liking':1,'Scarcity':0,'sentence': str(augmented_text[0])}
        df=df.append(new_raw, ignore_index=True)
        #df=pd.concat(df, pd.DataFrame(new_raw))
        #print(new_raw)
```

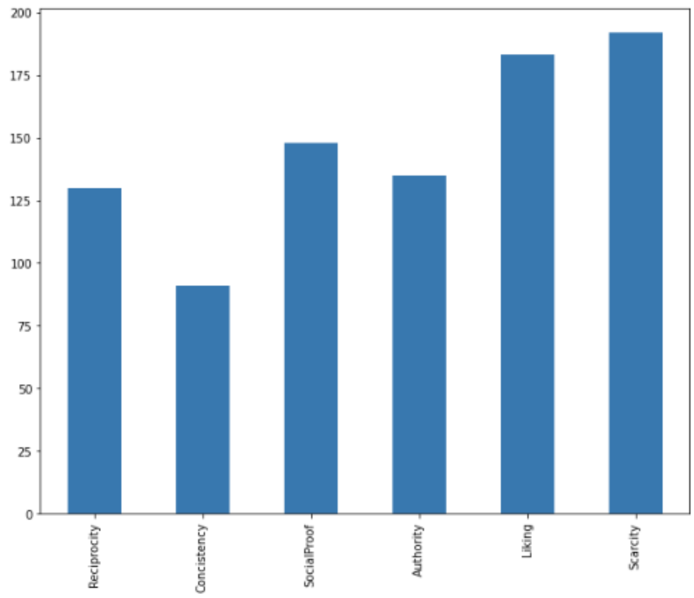
```
import nlpaug.augmenter.word as naw

TOPK=20 #default=100
ACT = 'insert' #"substitute"

aug_bert = naw.ContextualWordEmbsAug(
    model_path='distilbert-base-uncased',
    #device='cuda',
    action=ACT, top_k=TOPK)
#print("Original:")

#print("Augmented Text:")
for sentence in df.loc[df.Authority == 1].sentence:
    #print('Original :' + sentence)
    for ii in range(2):
        augmented_text = aug_bert.augment(sentence)
        new_raw={'Reciprocity':0, 'Concistency':0, 'SocialProof':0, 'Authority':1,'Liking':0,'Scarcity':0,'sentence': str(augmented_text[0])}
        df=df.append(new_raw, ignore_index=True)
        #df=pd.concat(df, pd.DataFrame(new_raw))
        #print(new_raw)
```

Save the balanced dataset in a CSV file and plot it to check the classes distribution:

```
df.to_csv(r'C:\Users\ermes\OneDrive - National College of Ireland\Desktop\Project\emotion\emotion.csv', encoding='utf-8')
```

	,ID,sentence,Reciprocity,Concistency,SocialProof,Authority,Liking,Scarcity
0	0,1,0,", papers , fruit , lollipops , and ciga...
1	1,2,0,", they will read a p .",0,0,0,0,0,0
2	2,3,0,1 write your sales letter with an indivi...
3	3,4,0,2 billion in commitments .,0,0,0,0,0,0
4	4,5,0,2 there were spelling errors in the text...

```
df=pd.read_csv(r'C:\Users\ermes\OneDrive - National College of Ireland\Desktop\Project\emotion\emotion.csv',sep = ',')
df_labels = df[["Reciprocity", "Concistency", "SocialProof", "Authority","Liking","Scarcity"]]
df_labels.head()

fig_size = plt.rcParams["figure.figsize"]
fig_size[0] = 10
fig_size[1] = 8
plt.rcParams["figure.figsize"] = fig_size

df_labels.sum(axis=0).plot.bar()
```



Remove punctuation and multiple spaces

```
def preprocess_text(sen):
    # Remove punctuations and numbers
    sentence = re.sub('[^a-zA-Z]', ' ', sen)

    # Single character removal
    sentence = re.sub(r"\s+[a-zA-Z]\s+", ' ', sentence)

    # Removing multiple spaces
    sentence = re.sub(r'\s+', ' ', sentence)

    return sentence
```

```
len(df)
```

1213

```
X = []
sentences = list(df["sentence"])
for sen in sentences:
    X.append(preprocess_text(sen))

y = df_labels.values
```

Split the dataset into train, test and validation datasets:

```
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)
```

To load the Small-BERT model using TensorFlow Hub:

```
import tensorflow_hub as hub
import tensorflow_text as text
import os
#import tensorflow_text as text
#hub.load()
os.environ['TFHUB_CACHE_DIR'] = r'C:\Users\ermes\OneDrive - National College of Ireland\Desktop\Project'

preprocessor = hub.KerasLayer('https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3')
encoder = hub.KerasLayer('https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-2_H-768_A-12/2')

tfhub_handle_preprocess='https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3'
tfhub_handle_encoder='https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-2_H-768_A-12/2'
```

To Build the Classifier Model:

```
def build_classifier_model():
    text_input = tf.keras.layers.Input(shape=(), dtype=tf.string, name='text')
    preprocessing_layer = hub.KerasLayer(tfhub_handle_preprocess, name='preprocessing')
    encoder_inputs = preprocessing_layer(text_input)
    encoder = hub.KerasLayer(tfhub_handle_encoder, trainable=True, name='BERT_encoder')
    outputs = encoder(encoder_inputs)

    net = outputs['pooled_output']
    net = tf.keras.layers.Dropout(0.1, name = 'Dropout')(net)
    net = tf.keras.layers.Dense(6, activation = 'sigmoid', name = 'Dense')(net)
    return tf.keras.Model(text_input, net)
```

```
model=build_classifier_model()
```

Then to fine Tune the Model on ten epochs :

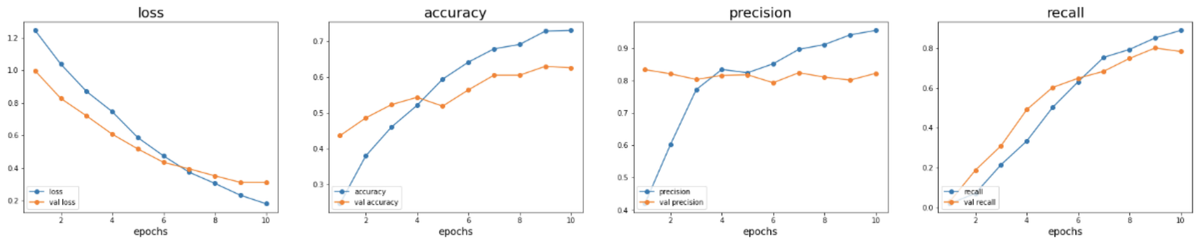
```
n_epochs = 10

METRICS = [
      tf.keras.metrics.CategoricalAccuracy(name="accuracy"),
      balanced_recall,
      balanced_precision,
      balanced_f1_score
]

Metrics = [tf.keras.metrics.CategoricalAccuracy(name = 'accuracy'),
           tf.keras.metrics.Precision(name = 'precision'),
           tf.keras.metrics.Recall(name = 'recall')
           ]

#earlystop_callback = tf.keras.callbacks.EarlyStopping(monitor = "val_loss",
#                                                      patience = 3,
#                                                      restore_best_weights = True)
#tf.keras.optimizers.Adam(1e-5)
model.compile(optimizer = tf.keras.optimizers.Adam(2e-5),
              #loss = "binary_crossentropy",
              loss = "CategoricalCrossentropy",
              metrics = Metrics)
```

```
model_fit = model.fit(np.array(x_train),
                      np.array(y_train),
                      epochs = n_epochs, batch_size=16,
                      validation_data = (np.array(x_test), np.array(y_test))
                      #callbacks = [earlystop_callback]
                     )
```

```
Epoch 1/10
61/61 [==============================] - 7s 76ms/step - loss: 1.2450 - accuracy: 0.2464 - precision: 0.4194 - recall: 0.0184
all: 0.0292
Epoch 2/10
61/61 [==============================] - 4s 70ms/step - loss: 1.0381 - accuracy: 0.3794 - precision: 0.6024 - recall: 0.0706
all: 0.1871
Epoch 3/10
61/61 [==============================] - 4s 72ms/step - loss: 0.8708 - accuracy: 0.4598 - precision: 0.7716 - recall: 0.2147
all: 0.3099
Epoch 4/10
61/61 [==============================] - 5s 77ms/step - loss: 0.7470 - accuracy: 0.5206 - precision: 0.8339 - recall: 0.3333
all: 0.4912
Epoch 5/10
61/61 [==============================] - 4s 73ms/step - loss: 0.5876 - accuracy: 0.5938 - precision: 0.8237 - recall: 0.5014
all: 0.6023
Epoch 6/10
61/61 [==============================] - 4s 73ms/step - loss: 0.4756 - accuracy: 0.6412 - precision: 0.8514 - recall: 0.6314
all: 0.6491
Epoch 7/10
61/61 [==============================] - 4s 73ms/step - loss: 0.3752 - accuracy: 0.6784 - precision: 0.8960 - recall: 0.7542
all: 0.6842
```

To plot the metrics and evaluate the Model:

```python
x = list(range(1, n_epochs+1))
metric_list = list(model_fit.history.keys())
num_metrics = int(len(metric_list)/2)

fig, ax = plt.subplots(nrows=1, ncols=num_metrics, figsize=(30, 5))

for i in range(0, num_metrics):
    ax[i].plot(x, model_fit.history[metric_list[i]], marker="o", label=metric_list[i].replace("_", " "))
    ax[i].plot(x, model_fit.history[metric_list[i+num_metrics]], marker="o", label=metric_list[i+num_metrics].replace("_", " "))
    ax[i].set_xlabel("epochs",fontsize=14)
    ax[i].set_title(metric_list[i].replace("_", " "),fontsize=20)
    ax[i].legend(loc="lower left")
```



To test the Model:

```python
examples = [' Hello! My name is Shafaq. Your website or a website that your company hosts is infringing on a copyright-protected images owned by myself.',
            ' Take a look at this document with the links to my images you used at website.berkeley.edu and my earlier publications to get the evidence of my copyrights.',
            ' Download it right now and check this out for yourself: hxxps://sites.google.com/view/a0hf49gj29g-i4jb48n5/drive/folders/shared/1/download?ID=308682351554855915',
            ' I believe you have willfully infringed my rights under 17 USC Section 101 et seq. and could be liable for statutory damages as high as $150,000 as set forth in Section 504(c)(2) of the D
```

```python
def predict_class2(reviews):
    index=[np.argmax(pred) for pred in model.predict(reviews)]
    return LABELS[index[0]]

def print_my_examples(inputs):
    result_for_printing = \
      [f'input: {inputs[i]:<30} : estimated intent: {predict_class2([inputs[i]])}'
                         for i in range(len(inputs))]
    print(*result_for_printing, sep='\n')
    print()
```

```python
print_my_examples(examples)
```

```
1/1 [==============================] - 0s 457ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 38ms/step
1/1 [==============================] - 0s 41ms/step
input:  Hello! My name is Shafaq. Your website or a website that your company hosts is infringing on a copyright-protected images owned by myself. : estimated intent: Liking
input:  Take a look at this document with the links to my images you used at website.berkeley.edu and my earlier publications to get the evidence of my copyrights. : estimated intent: Liking
input:  Download it right now and check this out for yourself: hxxps://sites.google.com/view/a0hf49gj29g-i4jb48n5/drive/folders/shared/1/download?ID=308682351554855915 : estimated intent: Scarcity
input:  I believe you have willfully infringed my rights under 17 USC Section 101 et seq. and could be liable for statutory damages as high as $150,000 as set forth in Section 504(c)(2) of the Digital
Millennium Copyright Act (“DMCA”) therein. : estimated intent: Reciprocity
```

To save The Model Created:

```python
## SAVE THE MODEL
import pickle

import tensorflow as tf
path = r'./EMOTION1.tf'
model.save(path)
loaded_model= tf.keras.models.load_model(path)
```

4 References

Anaconda. (2022). *anaconda*. Retrieved from https://www.anaconda.com/products/distribution

Bokeh. (2022). *Nvidia Developer*. Retrieved from https://developer.nvidia.com/blog/gpu-dashboards-in-jupyter-lab/

Jupyter. (2022). *Jupyter*. Retrieved from https://jupyter.org/

Keras. (2022). *Keras*. Retrieved from https://keras.io/

Matplolib. (2022). *Matplolib*. Retrieved from https://matplotlib.org/

Numpy. (2022). *Numpy*. Retrieved from https://numpy.org/

TensorFlow. (2022). *TensorFlow*. Retrieved from https://www.tensorflow.org/