MSIN0221 Natural Language Processing 2020/21

Group Coursework Report

> Please find the Google Drive Link to our presentation at the end of this report

Word count: 4392

Introduction

Problem Statement

An increase in digitalisation has led to an explosive growth of emails in our inbox. This is becoming particularly troubling as an average worker spends approximately 28% of their working hours reading and replying to emails (McKinsey Global Institute, 2012). For this project scope, we will be analysing emails from Enron Corporation to classify the sentiment in an email and then summarise it through the use of natural language processing. This project relates to a proposal submitted earlier and addresses all ideas outlined there. Our sections for sentiment analysis and text summarisation will each consist of a description of the model architecture, as well as a discussion of the results and an error analysis. All of these will be briefly evaluated and compared in the Summary.

Pre-Processing

We will go through the following steps: lowercasing all characters, removing non-value-adding text including URLs, replies, forwards and links, removing some punctuations and stopwords before finally normalising our data with lemmatisation.

Data Annotation

As we have limited resources, we annotate only 100 emails by the six contributors. Annotation includes sentiment classification and extractive summary. We used Snorkel for further labelling of the sentiment. Snorkel is used for weak labeling. While not optimal, Snorkel still provides annotation that matches our hand-labelled data to a satisfactory level.

Sentiment Analysis

We use two methods to calculate the sentiment classification. First, we build an RNN model architecture with LSTM cells from scratch. For our second model, we will use a pre-trained model from the HuggingFace Transformers library. As mentioned in our literature review, BERT is suitable for our project scope due to its flexibility. As BERT is computationally expensive and because of limited available resources, we will use the smaller 12 encoder layer, BERT base, for our project.

Text Summarisation

Text summarisation can be extractive or abstractive. We identify essential excerpts from the text in extractive summarisation, which are used as part of our summary. Abstractive text summarisation employs more complex and powerful NLP methods to interpret a text and create summaries of it. As this method is difficult and computationally expensive, we opted to use extractive. We mentioned in our literature review that the use of pre-trained transformer models gives a satisfying performance. However, because we lack labellings and human annotators, we changed course. Instead, we will use BERT, GPT-2 and XLNet transformer models from the Hugginface library to compare with hand-annotated labels.

```
#To support python 2 & 3
from __future__ import division, print_function, unicode_literals

# Common imports
import numpy as np
import os
import pandas as pd
#For consistent output
np.random.seed(42)

# To plot pretty figures
import seaborn as sns
%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt

# Ignore unnecessary warnings
import warnings
warnings.filterwarnings(action="ignore", message="^internal gelsd")

#display all cells
from IPython.core.interactiveshell import InteractiveShell


try:
  import transformers
except  ModuleNotFoundError:
  !pip install transformers;
  !pip install torch;


from google.colab import drive

drive.mount('/content/gdrive')
root_path = 'gdrive/MyDrive/Enron notebooks/'
```

Loading the data and creating labeling sets

The dataset for this project are the emails originally made available by US authorities in the aftermath of the Enron scandal in the early 2000s. The full dataset has been downloaded from the CMU website (Cohen, 2015), where it has been pseudonomized and is still maintained today.

The data consists of several files and has to be extracted using the Python email library and the get_payload method.

```
import email
import transformers
from transformers import BertModel, BertTokenizer, AdamW, get_linear_schedule_wi
import torch
import numpy as np
import pandas as pd
import seaborn as sns
from pylab import rcParams
import matplotlib.pyplot as plt
from matplotlib import rc
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
from collections import defaultdict
from textwrap import wrap
from torch import nn, optim
from torch.utils.data import Dataset, DataLoader
import transformers
from transformers import BertModel, BertTokenizer, AdamW, get_linear_schedule_wi
import torch
%matplotlib inline
%config InlineBackend.figure_format='retina'
sns.set(style='whitegrid', palette='muted', font_scale=1.2)
HAPPY_COLORS_PALETTE = ["#01BEFE", "#FFDD00", "#FF7D00", "#FF006D", "#ADFF02", "
sns.set_palette(sns.color_palette(HAPPY_COLORS_PALETTE))
rcParams['figure.figsize'] = 12, 8
RANDOM_SEED = 42
np.random.seed(RANDOM_SEED)
torch.manual_seed(RANDOM_SEED)
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
```

```
df = pd.read_csv("/content/gdrive/MyDrive/Enron notebooks/Separate notebooks/ema
df.head()
```

```
print(df.loc[4]['message'])
```

```
message = df.loc[4]['message']
mail = email.message_from_string(message)
mail.get_payload()


def get_field(field, messages):
    column = []
    for message in messages:
        e = email.message_from_string(message)
        column.append(e.get(field))
    return column

def body(messages):
    column = []
    for message in messages:
        e = email.message_from_string(message)
        column.append(e.get_payload())
    return column


df['date'] = get_field("Date", df['message'])
df['subject'] = get_field("Subject", df['message'])
df['from'] = get_field("X-From", df['message'])
df['to'] = get_field("X-To", df['message'])
df['body'] = body(df['message'])

# create nan columns for human labeling
df["sentiment_label"] = np.nan
df["summary_label"] = np.nan

df.shape


df.head()


filter_df = df[(df.subject.str.contains("Re:|Fwd:|Fw:|RE:|FW:")==False)&(df.body
filter_df.shape
```

We can see that the resulting dataset contained many replies and forwards of messages, which we want to extract here for two reasons: Replies and forwards often contain no or only very little new text (e.g. forwarding an email to a supervisor often just includes the original text) and also, the results might be biased by the fact that the text in replies and forwarded emails appears more than once in the training data, meaning that the model could see the text of the original email more often.

Thus, after filtering out replies and forwards, we can see that roughly 300,000 emails remain, which is sufficient for training and testing data.

```
filter_df.head()
```

```
#df.to_csv("gdrive/My Drive/MyDrive/Enron notebooks/emails_prepared.csv")

labeling_df = filter_df.sample(n=100)
labeling_df.head()
```

The dataframe is saved as CSV and a sample of 100 randomly selected instances is divided up into small chunks for the 6 members to label manually with a binary sentiment label and an extractive text summary.

```
#labeling_df.iloc[:16, :].to_csv("gdrive/My Drive/MyDrive/Enron notebooks/carina
#labeling_df.iloc[16:32, :].to_csv("gdrive/My Drive/MyDrive/Enron notebooks/fred
#labeling_df.iloc[32:49, :].to_csv("gdrive/My Drive/MyDrive/Enron notebooks/luke
#labeling_df.iloc[49:66, :].to_csv("gdrive/My Drive/MyDrive/Enron notebooks/jaso
#labeling_df.iloc[66:83, :].to_csv("gdrive/My Drive/MyDrive/Enron notebooks/neil
#labeling_df.iloc[83:101, :].to_csv("gdrive/My Drive/MyDrive/Enron notebooks/nik
```

Pre-Processing

Lowercasing

We need to lowercase all words as the NLTK tokeniser is case sensitive.

We also need to substitute words likely to be removed during data cleaning, such as ‘t with not.

Removing non-value-adding text

URLs, links, numbers and non-value-adding texts can be removed from our corpus. We also pulled emails and strings between “cc:” and “subject”.

Punctuation was also removed, except for ., ? and !.

Stopwords should also be removed, which was done using the NLTK stopword list.

Numbers are also not meaningful in this context and were removed. Finally, we remove any spaces and special characters.

Normalisation

Text normalisation is converting text into its simplest word representation. By simplifying the word representation, we reduce the number of tokens and also the possibility of confusing our models.

The main methods of normalisation are stemming and lemmatisation. While lemmatisation is a complex process, stemming reduces the words to their stem and is used here for its good results and computational ease.

```
!pip install PorterStemmer
!pip install wordnet
!nltk.download('wordnet')
```

```
    Requirement already satisfied: PorterStemmer in /usr/local/lib/python3.7/di
    Requirement already satisfied: wordnet in /usr/local/lib/python3.7/dist-pac
    Requirement already satisfied: colorama==0.3.9 in /usr/local/lib/python3.7/
    /bin/bash: -c: line 0: syntax error near unexpected token `'wordnet''
    /bin/bash: -c: line 0: `nltk.download('wordnet')'
```

```
# !pip install PorterStemmer
# !pip install wordnet
# nltk.download('wordnet')

#Preprocessing
import os
import re
from tqdm import tqdm
import nltk

from nltk.corpus import stopwords
```

```
from nltk.stem import WordNetLemmatizer,PorterStemmer
from nltk.tokenize import RegexpTokenizer
lemmatizer = WordNetLemmatizer()

def text_preprocessing(s):
    """
    - Lowercase the sentence
    - Change "'t" to "not"
    - Remove "name@email.com"
    - Remove links & URLs
    - Remove Strings between two delimiters
    - Isolate and remove punctuations except "?", "!", "."
    - Remove other special characters & Numbers
    - Remove stop words except "not" and "can"
    - Remove trailing whitespace
    """
    s = s.lower()  #Changing all words to lowercase
    # Change 't to 'not'
    s = re.sub(r"\'t", " not", s)
    #Removing URL and Links
    url_pattern = re.compile(r'https?://\S+|www\.\S+')
    s= url_pattern.sub(r' ', s)

    html_pattern = re.compile('<.*?>')
    s= html_pattern.sub(r' ', s)
    #Removing strings between two delimiters
    firstDelPos=s.find("cc:")
    secondDelPos=s.find("subject")
    s = s.replace(s[firstDelPos:secondDelPos], "")
    #Removing Number
    s = re.sub('[0-9]+', ' ', s)
    # Remove name@email.com
    s = re.sub(r'([a-z0-9._]+@.*?)[\s]', '', s)
    # Isolate and remove punctuations except '?'
    s = re.sub(r'([\'\"\(\)\?\\\/\,])', r' \1 ', s)
    s = re.sub(r'[^\w\s\?!.]', ' ', s)
   # Remove some special characters
    s = re.sub(r'([\_\;\:\|•«\n])', ' ', s)
    # Remove stopwords except 'not' and 'can'
    s =" ".join([word for word in s.split()
                  if word not in stopwords.words('english')
                  or word in ['not', 'can']])
    s=" ".join([lemmatizer.lemmatize(w) for w in s.split()])
    # Remove trailing whitespace
    s = " ".join([word for word in s.split() if len(word)>2])
    s = re.sub(r'\s+', ' ', s.strip())
    return s

#df["processed"] =df["body"].apply(lambda body:text_preprocessing(body))
#df["processed_length"] = df["processed"].apply(lambda processed: len([word for 
# df["body_length"] = df["body"].apply(lambda body: len([word for word in body.s
```

Creating Sentiment Labels using Snorkel

In order to create sentiment labels for our dataset, we will use Snorkel's labelling functions.

We will create two sets of labels. The first classifies sentences as either positive, negative or neutral. The second as only positive or negative.

In order to benchmark the performance of the labels we created, we manually labelled 100 randomly selected emails as either positive, negative or neutral.

Once we achieve satisfactory performance, we will apply our labelling model to the rest of the data and use the created labels for our subsequent analysis.

```
!pip install snorkel
!pip install textblob
```

```
Requirement already satisfied: snorkel in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: tensorboard<2.0.0,>=1.14.0 in /usr/local/lib
Requirement already satisfied: scipy<2.0.0,>=1.2.0 in /usr/local/lib/python
Requirement already satisfied: numpy<1.20.0,>=1.16.5 in /usr/local/lib/pyth
Requirement already satisfied: scikit-learn<0.25.0,>=0.20.2 in /usr/local/l
Requirement already satisfied: networkx<2.4,>=2.2 in /usr/local/lib/python3
Requirement already satisfied: torch<2.0.0,>=1.2.0 in /usr/local/lib/python
Requirement already satisfied: pandas<2.0.0,>=1.0.0 in /usr/local/lib/pytho
Requirement already satisfied: munkres>=1.0.6 in /usr/local/lib/python3.7/d
Requirement already satisfied: tqdm<5.0.0,>=4.33.0 in /usr/local/lib/python
Requirement already satisfied: wheel>=0.26; python_version >= "3" in /usr/l
Requirement already satisfied: setuptools>=41.0.0 in /usr/local/lib/python3
Requirement already satisfied: six>=1.10.0 in /usr/local/lib/python3.7/dist
Requirement already satisfied: protobuf>=3.6.0 in /usr/local/lib/python3.7/
Requirement already satisfied: grpcio>=1.6.3 in /usr/local/lib/python3.7/di
Requirement already satisfied: absl-py>=0.4 in /usr/local/lib/python3.7/dis
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.7/
Requirement already satisfied: werkzeug>=0.11.15 in /usr/local/lib/python3.
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dis
Requirement already satisfied: decorator>=4.3.0 in /usr/local/lib/python3.7
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.7/dis
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/pyt
Requirement already satisfied: importlib-metadata; python_version < "3.8" i
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: textblob in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: nltk>=3.1 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-package
```

```
#load manually labelled emails
import nltk
import pandas as pd
nltk.download('wordnet')

emails_labelled = pd.read_csv("/content/gdrive/MyDrive/Enron notebooks/Separate

# run labelled data through cleaning

#emails_labelled = text_preprocessing(emails_labelled['body'])
emails_labelled['body'] = emails_labelled['body'].apply(lambda body:text_preproc


# create y-label array

y_train = emails_labelled.sentiment_label.values


# see distribution of how manually labelled data was labelled

emails_labelled.groupby("sentiment_label").size()
```

Creating Labelling Functions for Multiclass Sentiment Classification

```
from snorkel.labeling import labeling_function

NEUTRAL = 0
NEG = 1
POS = 2
ABSTAIN = -1


#load list of positive and negative words from Github. Orginal creators of this

positive_words = pd.read_csv('https://gist.githubusercontent.com/mkulakowski2/42
positive_words = list(positive_words.iloc[10:,0].values)

negative_words =  pd.read_csv('https://gist.githubusercontent.com/mkulakowski2/4
negative_words = list(negative_words.iloc[10:,0].values)
```

```
# from the list of around 6000 positive and negative words to create our first s

@labeling_function()
def neg_words(x):
    neg_count = 0
    for j in negative_words:
        counting = x.body.split().count(j)
        neg_count = neg_count + counting
    return NEG if neg_count >=3 else ABSTAIN


@labeling_function()
def pos_words(x):
    pos_count = 0
    for i in positive_words:
        counting = x.body.split().count(i)
        pos_count = pos_count + counting
    return POS if pos_count >=4 else ABSTAIN

@labeling_function()
def neutral_words(x):
    neg_count = 0
    for j in negative_words:
        counting = x.body.split().count(j)
        neg_count = neg_count + counting
    pos_count = 0
    for i in positive_words:
        counting = x.body.split().count(i)
        pos_count = pos_count + counting
    return NEUTRAL if pos_count <4 and neg_count <3 else ABSTAIN
```

For our first set of labelling functions for multiclass classification we use the negative and positve word repository by Minqing Hu and Bing Liu (2004) and Bing Liu, Minqing Hu and Junsheng Cheng (2005).

The first function labels the email as negative if more than 3 negative words are included in the email, otherwise it abstains. The second classifies an email as postive if more than 4 positive words occur in the email, otherwise it abstains. The third classifies the email as neutral there are fewer than 4 positive and fewer than 3 negative words otherwise it abstains.

Our selection of the thresholds was based on inspecting the manually labelled data. The threshold number of words to label an email as either positive or negative may seem high at first glance. However, after looking into a sample of emails, we found that the majority of short emails had a neutral sentiment, whereas the positive and negative labelled emails were more verbose.

```
#use textblob pretrained sentiment classifier

from snorkel.preprocess import preprocessor
from textblob import TextBlob


@preprocessor(memoize=True)
def textblob_polarity(x):
    scores = TextBlob(x.body)
    x.polarity = scores.polarity
    return x


# Label high polarity emails as positive.
@labeling_function(pre=[textblob_polarity])
def polarity_positive(x):
    return POS if x.polarity > 0.15 else ABSTAIN


@labeling_function(pre=[textblob_polarity])
def polarity_negative(x):
    return NEG if x.polarity < -0.15 else ABSTAIN

@labeling_function(pre=[textblob_polarity])
def polarity_neutral(x):
    return NEUTRAL if x.polarity >= -0.15 and x.polarity <= 0.15 else ABSTAIN
```

Our next set of labelling functions builds on Snorkel's ability to utilise third party models. Here, we use Textblob's sentiment sentiment score to classify emails as either positive, negative or neutral.

The textblob functions label a sentence as positive if its sentiment score is above 0.15, negative if below -0.15 and neutral if in between. In order to get the correct thresholds, we looked at individually labelled emails and their scores. The thresholds are relatively low due as the majority of emails are written in a professional, neutral tone.

```
#apply our labelling functions on the labelled data to see how Snorkel performs

from snorkel.labeling import PandasLFApplier
from snorkel.labeling import LFAnalysis

# set labelling functions

lfs = [polarity_positive, polarity_negative, polarity_neutral, neg_words, pos_wo

# apply labelling functions to manually labelled data

applier = PandasLFApplier(lfs)
L_train = applier.apply(emails_labelled)

# display analysis

LFAnalysis(L=L_train, lfs=lfs).lf_summary()
```

```
/usr/local/lib/python3.7/dist-packages/tqdm/std.py:658: FutureWarning: The
  from pandas import Panel
100%|██████████| 98/98 [00:31<00:00,  3.07it/s]
```

	j	Polarity	Coverage	Overlaps	Conflicts
polarity_positive	0	[2]	0.275510	0.275510	0.204082
polarity_negative	1	[1]	0.061224	0.061224	0.051020
polarity_neutral	2	[0]	0.663265	0.663265	0.214286
neg_words	3	[1]	0.214286	0.214286	0.204082
pos_words	4	[2]	0.285714	0.285714	0.214286
neutral_words	5	[0]	0.653061	0.653061	0.204082

Snorkel's analyis gives us a coverage, overlap and conflict score for each labelling function.

The coverage reflects the percentage of emails the function was able to label. We see a high percentage for the neutral functions (polarity_neutral, neutral_words) which cover 66% and 65% respectively. This is a good score as it approximately matches the percentage of sentiment scores we manually labelled (60%). Furthermore, the positive functions cover 27% and 28% repectively, comapred to the 25%, we manually labelled. The negative functions cover 21% and 6% compared to the 13% we manually labelled. These levels are all very satisfactory.

There are some conflicts, but this is expected as we are dealing with a multiclass problem. One reason for this could be due to long emails containing both multiple positive and negative words and therefore being classifies as both by our heuristic classifiers.

Nevertheless, the point of Snorkel is that the labelling functions do not need to be perfect as the label model is able to generalise beyond the functions we created.

```
# use Snorkel's label model to classify instances based on our labelling functio

from snorkel.labeling.model import LabelModel

label_model = LabelModel(cardinality=4, verbose=True)
label_model.fit(L_train, n_epochs=100, seed=123, log_freq=20, l2=0.1, lr=0.01)

#create predicted values based on Snorkel model

preds = label_model.predict(L_train)

#apply Snorkel labels to manually labelled data

emails_labelled["label_snorkel"] = label_model.predict(L=L_train, tie_break_poli
```

Snorkel's label model estimates the accuracies of the labelling functions we created and reweights them to create lables as accurately as possible.

```
# see distributin of Snorkel labels

emails_labelled["label_snorkel"].value_counts()

#1 is negative, 2 positive, 0 neutral, 3 Abstain
```

```
0    49
2    44
1     5
Name: label_snorkel, dtype: int64
```

```
preds
```

```
# check accuracy of Snorkel labels

from snorkel.analysis import metric_score


acc = metric_score(y_train, preds, probs=None, metric="accuracy")
print(f"LabelModel Accuracy: {acc:.3f}")
```

```
LabelModel Accuracy: 0.520
```

We get an accuracy score of 52%, compared to a baseline score of 33% if we were to randomly select labels. After looking at where the model classified labels incorrectly, we found that especially longer emails were classified as either positive or negative where we manually labelled them neutral.

However, further finetuning would end up overfitting our manually labelled data. Not having perfect labels is one of the drawbacks of using Snorkel. However, the cost and time benefit of being able to label an entire dataset outweigh this drawback. Therefore, we will apply our labelling model onto our dataset to create the sentiment labels.

```
#load data

emails = pd.read_csv('/content/gdrive/MyDrive/Enron notebooks/Separate notebooks

L_train_b = applier_b.apply(emails_processed)


label_model_b = LabelModel(cardinality=3, verbose=True)
label_model_b.fit(L_train_b, n_epochs=100, seed=123, log_freq=20, l2=0.1, lr=0.0

#We then see how well that model performs on the development set(the one we labe


preds_b = label_model_b.predict(L_train_b)

emails_processed["label_snorkel_binary"] = label_model_b.predict(L=L_train_b, ti


# check distribution of sentiment labels

emails["label_snorkel"].value_counts()
```

```
0    19731
2    11484
1     6012
Name: label_snorkel, dtype: int64
```

Snorkel Labels for Binary Sentiment Classification

In addition, to our multiclass labels, we want to add binary sentiment labels (positive, negative). We want this as sentiment models prefer binary output.

We will follow the same procedure as above but alter the labelling functions to create a binary output.

```
from snorkel.labeling import labeling_function

NEG = 1
POS = 2
ABSTAIN = -1
```

```
# We will use the emails we labelled as either positive or negative in the initi

emails_labelled_b = emails_labelled[(emails_labelled["sentiment_label"] == 1.0)
emails_labelled_b.shape
emails_labelled_b.head()
```

	Unnamed: 0	Unnamed: 0.1	date	subject	fr
5	5	140495	Wed, 31 Jan 2001 00:57:00 -0800 (PST)	(no subject)	MELF116@aol.c
9	9	446135	Wed, 6 Dec 2000 00:33:00 -0800 (PST)	EECC HOLIDAY PARTY INVITATION (PRINTABLE COPY)...	Gayla E Sei
14	14	323054	Wed, 23 Jan 2002 14:39:35 -0800 (PST)	Enron Mentions -- 01/23/02	Palmer, Sa </O=ENRON/OU=NA/CN=RECIPIENTS/C
15	15	173277	Tue, 3 Apr 2001 07:04:00 -0700 (PDT)	Broadband Business	Stanley Hor
18	2	434704	Thu, 22 Feb 2001 04:36:00 -0800 (PST)	El Paso Victory and Opportunity	Rebecca W Cant

```
#create y_train

y_train_b = emails_labelled_b["sentiment_label"]

#get an idea of how the binary sentiment scores are distributed

emails_labelled_b.groupby("sentiment_label").size()

    sentiment_label
    1.0    13
    2.0    25
    dtype: int64


# create labelling fucntions for binary labelling using positive and negative wo

@labeling_function()
def neg_words_b(x):
    neg_count = 0
    for j in negative_words:
        counting = x.body.split().count(j)
        neg_count = neg_count + counting
    return NEG if neg_count >=3 else ABSTAIN


@labeling_function()
def pos_words_b(x):
    pos_count = 0
    for i in positive_words:
        counting = x.body.split().count(i)
        pos_count = pos_count + counting
    return POS if pos_count >=3 else ABSTAIN
```

We use the same labelling functions but without a neutral function.

```
# use textblob to create binary sentiment labelling functions

@preprocessor(memoize=True)
def textblob_polarity_b(x):
    scores = TextBlob(x.body)
    x.polarity = scores.polarity
    return x


# Label high polarity emails as positive.
@labeling_function(pre=[textblob_polarity])
def polarity_positive_b(x):
    return POS if x.polarity > 0.05 else ABSTAIN


@labeling_function(pre=[textblob_polarity])
def polarity_negative_b(x):
    return NEG if x.polarity < 0.05 else ABSTAIN


# apply labbeling classifies and apply them to emails

lfs_b = [polarity_positive_b, polarity_negative_b, neg_words_b, pos_words_b]

applier_b = PandasLFApplier(lfs_b)
L_train_b = applier_b.apply(emails_labelled_b)

LFAnalysis(L=L_train_b, lfs=lfs_b).lf_summary()
```

```
/usr/local/lib/python3.7/dist-packages/tqdm/std.py:658: FutureWarning: The 
  from pandas import Panel
100%|██████████| 38/38 [00:10<00:00,  3.74it/s]
```

	j	Polarity	Coverage	Overlaps	Conflicts
polarity_positive_b	0	[2]	0.684211	0.500000	0.236842
polarity_negative_b	1	[1]	0.315789	0.052632	0.026316
neg_words_b	2	[1]	0.289474	0.289474	0.263158
pos_words_b	3	[2]	0.526316	0.526316	0.263158

Looking at the analysis, we can see good coverage rates which match our manually labelled data. However, with only 38 data points to compare to we should not give too much importance to this or we risk overfitting. Again there are some conflicts, which most likely arrise due to both negative and positive words being in long emails.

```
#create a label model to classify instances based on labelling functions


label_model_b = LabelModel(cardinality=3, verbose=True)
label_model_b.fit(L_train_b, n_epochs=100, seed=123, log_freq=20, l2=0.1, lr=0.0

#create predicted values based on Snorkel model

preds_b = label_model_b.predict(L_train_b)

#apply Snorkel labels to manually labelled data

emails_labelled_b["label_snorkel_binary"] = label_model_b.predict(L=L_train_b, t
```

```
    /usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:13: SettingWit
    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
      del sys.path[0]
```

```
# check performance of Snorkel labels vs manually labelled data for the 38 data

acc = metric_score(y_train_b, preds_b, probs=None, metric="accuracy")
print(f"LabelModel Accuracy: {acc:.3f}")
```

```
    LabelModel Accuracy: 0.763
```

We get an accuracy of 76% which is pretty good. Looking at where the model had different predictions, we see that it struggled with the longer emails.

Next, we will apply our model onto the preprocessed dataset as before. Then, we will randomly select 5000 rows on which we will train our models on.

```
L_train_b = applier_b.apply(emails)


label_model_b = LabelModel(cardinality=3, verbose=True)
label_model_b.fit(L_train_b, n_epochs=100, seed=123, log_freq=20, l2=0.1, lr=0.0

#We then see how well that model performs on the development set(the one we labe


preds_b = label_model_b.predict(L_train_b)

emails["label_snorkel_binary"] = label_model_b.predict(L=L_train_b, tie_break_po
```

```
    /usr/local/lib/python3.7/dist-packages/tqdm/std.py:658: FutureWarning: The
      from pandas import Panel
    100%|██████████| 37227/37227 [44:46<00:00, 13.86it/s]
```

```
# from the 37 emails we, want to select 5000 for our modelling

#select rows where Snorkel classified them as either positive or negative (not a

emails = emails[(emails["label_snorkel_binary"] == 1) | (emails["label_snorkel_b
emails.groupby("label_snorkel_binary").size()
```

```
    label_snorkel_binary
    1    16628
    2    20555
    dtype: int64
```

```
# choose a random sample of 5000 and check distribution

emails_5k = emails.sample(5000, random_state=42)

emails_5k.groupby("label_snorkel_binary").size()

#save file for future use

#emails.to_csv('emails_5k.csv')
```

```
    label_snorkel_binary
    1    2271
    2    2729
    dtype: int64
```

Double-click (or enter) to edit

Data Understanding

```
torch.cuda.is_available()
```

```
True
```

```
df = pd.read_csv("/content/gdrive/MyDrive/Enron notebooks/Separate notebooks/ema
df= df.drop(["Unnamed: 0.1","Unnamed: 0.1.1","Unnamed: 0.1","Unnamed: 0"],axis=1
```

```
df.head()
```

	index	date	subject	from	to	
0	26974	Mon, 4 Jun 2001 02:12:00 -0700 (PDT)	final version - Harper Agreement	Cheryl Nelson	Sheila Glover	m
1	3749	Mon, 17 Apr 2000 08:23:00 -0700 (PDT)	Memorandum on Peregrine Hearing	DOCUMENTS <DOCUMENTS@isda.org>	Mark Taylor <Mark.Taylor@enron.com>	
2	23082	Wed, 25 Apr 2001 09:57:00 -0700 (PDT)	Constellation	Debra Perlingiere	Veronica Espinoza	i
3	20561	Tue, 3 Apr 2001 06:31:00 -0700 (PDT)	Request Submitted: Access Request for gautam.g...	Sally Beck	Patti Thompson	
4	9831	Fri, 10 Nov 2000 01:57:00 -0800 (PST)	Hiring Aram at a VP level	Vince J Kaminski	Rick Buy	

```
df['label_snorkel_binary'] = df['label_snorkel_binary'].replace([1],0)
df['label_snorkel_binary'] = df['label_snorkel_binary'].replace([2],1)
```

```
df.head()
```

	index	date	subject	from	to	
0	26974	Mon, 4 Jun 2001 02:12:00 -0700 (PDT)	final version - Harper Agreement	Cheryl Nelson	Sheila Glover	m
1	3749	Mon, 17 Apr 2000 08:23:00 -0700 (PDT)	Memorandum on Peregrine Hearing	DOCUMENTS <DOCUMENTS@isda.org>	Mark Taylor <Mark.Taylor@enron.com>	
2	23082	Wed, 25 Apr 2001 09:57:00 -0700 (PDT)	Constellation	Debra Perlingiere	Veronica Espinoza	i
3	20561	Tue, 3 Apr 2001 06:31:00 -0700 (PDT)	Request Submitted: Access Request for gautam.g...	Sally Beck	Patti Thompson	
4	9831	Fri, 10 Nov 2000 01:57:00 -0800 (PST)	Hiring Aram at a VP level	Vince J Kaminski	Rick Buy	

```
df.label_snorkel_binary.unique()
```

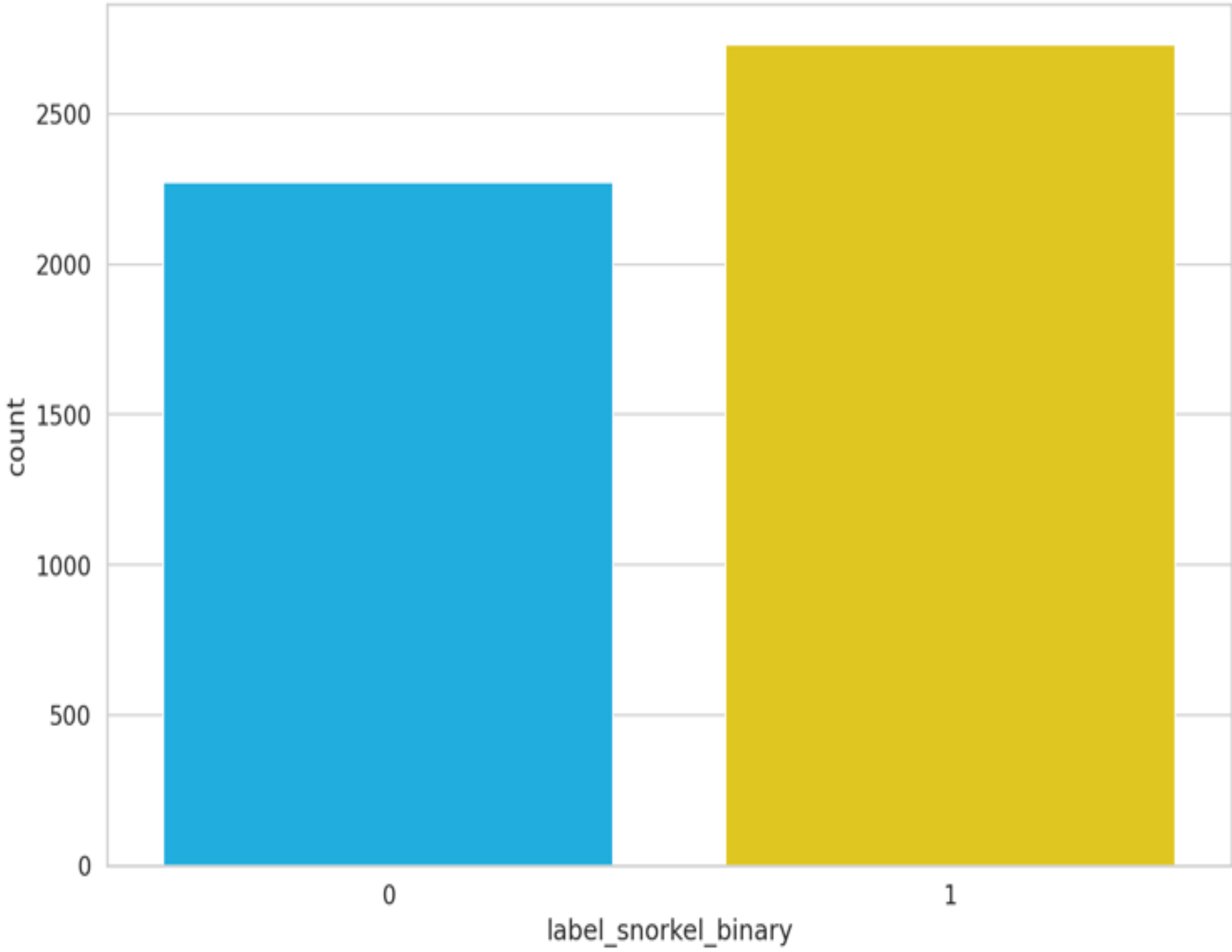
```
array([0, 1])
```

```
df.label_snorkel_binary.value_counts(normalize=True)
```

```
1    0.5458
0    0.4542
Name: label_snorkel_binary, dtype: float64
```

```
sns.countplot(df.label_snorkel_binary)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWar
  FutureWarning
<matplotlib.axes._subplots.AxesSubplot at 0x7f9e38ec2b50>
```



As we can see the labelling is quite balanced so won't need to adjust the class balance

LSTM from Scratch

As outlined in our project proposal, recurrent neural networks (RNNs) can show good performance in sentiment classification when built and specifically for a dataset. We thus trained an RNN with LSTM cells as baseline model and to compare performance against popular transformer-based pretrained models later on.

Due to the aforementioned limitations of data labeling, the sentiment labels created using Snorkel are applied to evaluate performance.

Inspirations for the baseline LSTM architecture used were taken from the course notebook 16 on advanced RNNs.

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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, ca
```

```
import tensorflow as tf
import tensorflow.keras as keras
from tensorflow.keras import layers
from tensorflow.keras import backend as K

device_name = tf.test.gpu_device_name()
if device_name != '/device:GPU:0':
  raise SystemError('GPU device not found')
print('Found GPU at: {}'.format(device_name))
```

```
Found GPU at: /device:GPU:0
```

Due to the size of the model and the dataset, the model was trained using a GPU-supported server instance on Google Colab. The use of GPUs can significantly speed up training when the types of operations are suited for GPU processing, which is the case in RNNs, where tensor multiplications are common (Appleyard et al., 2016).

```
from sklearn.model_selection import train_test_split
import pandas as pd
import torch
import numpy as np

df = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/emails_5k.csv")

df["label_snorkel_binary"] = np.where(df["label_snorkel_binary"] == 1, 0, 1)

df_train, df_dev_orig = train_test_split(df, test_size=0.2, shuffle=True)

prev_dev_len = len(df_dev_orig)
df_dev, df_test = train_test_split(df_dev_orig, test_size=0.5, shuffle=True)
print("Previous dev set ({} examples) split into {} dev and {} test set examples
```

```
    Previous dev set (1000 examples) split into 500 dev and 500 test set exampl
```

The 5000 random samples from the preprocessed emails dataset are the output of the preprocessing steps performed before and will also be used for other models. They are first relabeled by changing the positive "2" labels to 1 and the negative "1" labels to 0, which is easier and faster to process for a binary classification model. Then, the data is split into 20% testing and 80% training data, followed by another 50% split of the 20% into a dev/validation and test set.

```
len((df[df["processed_length"]>200])+df[df["processed_length"]<2])/len(df["proce
```

```
    0.0112
```

```
indexNames=df[(df["processed_length"]>200)].index
indexNames2=df[(df["processed_length"]<2)].index
# Delete these row indexes from dataFrame
df.drop(indexNames , inplace=True)
df.drop(indexNames2 , inplace=True)

df.reset_index(inplace=True)
```

To reduce the memory usage and in order to achieve comparability in performance with models used later, emails with more than 200 characters after preprocessing, (1% of instances), are dropped from the dataset. The problem with longer instances is that in a later step, we will have to pad the text bodies which are shorter to the length of the longest sequence, thus keeping the very few longer instances would slow down training significantly and use up more memory space since all instances would become larger due to the padding.

```
df = df[["processed", "label_snorkel_binary"]]
```

```
df.head()
```

	processed	label_snorkel_binary
0	sheila per phone message also fyi completed re...	0
1	attached please find memorandum prepared allen...	0
2	sent draft gisb sample master is. debra perlin...	1
3	please ask sheri thomas person need access eol...	0
4	rick want bring aram sogomonian back enron lev...	0

```
df.shape
```

```
(4944, 2)
```

```
try:
    from flair.embeddings import WordEmbeddings
except ModuleNotFoundError:
    !pip install flair
    from flair.embeddings import WordEmbeddings

# Load the glove embeddings
glove_embedding = WordEmbeddings('glove')
embedding_size = glove_embedding.embedding_length
```

```
Collecting flair
  Downloading https://files.pythonhosted.org/packages/f0/3a/1b46a0220d6176b
     |████████████████████████████████| 286kB 8.4MB/s
Collecting konoha<5.0.0,>=4.0.0
  Downloading https://files.pythonhosted.org/packages/02/be/4dd30d56a0a1961
Requirement already satisfied: numpy<1.20.0 in /usr/local/lib/python3.7/dis
Collecting ftfy
  Downloading https://files.pythonhosted.org/packages/04/06/e5c80e2e0f97962
     |████████████████████████████████| 71kB 8.2MB/s
Collecting deprecated>=1.2.4
  Downloading https://files.pythonhosted.org/packages/fb/73/994edfcba744431
Collecting transformers>=4.0.0
  Downloading https://files.pythonhosted.org/packages/ed/d5/f4157a376b8a794
     |████████████████████████████████| 2.0MB 14.1MB/s
Requirement already satisfied: regex in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: scikit-learn>=0.21.3 in /usr/local/lib/pytho
Requirement already satisfied: hyperopt>=0.1.1 in /usr/local/lib/python3.7/
Requirement already satisfied: python-dateutil>=2.6.1 in /usr/local/lib/pyt
Collecting gdown==3.12.2
  Downloading https://files.pythonhosted.org/packages/50/21/92c3cfe56f5c064
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
    Preparing wheel metadata ... done
```

```
Collecting huggingface-hub
  Downloading https://files.pythonhosted.org/packages/af/07/bf95f398e659820
Collecting sqlitedict>=1.6.0
  Downloading https://files.pythonhosted.org/packages/5c/2d/b1d99e9ad157dd7
Collecting janome
  Downloading https://files.pythonhosted.org/packages/a8/63/98858cbead27df7
     |████████████████████████████████| 19.7MB 74.4MB/s
Collecting langdetect
  Downloading https://files.pythonhosted.org/packages/56/a3/8407c1e62d59801
     |████████████████████████████████| 983kB 52.6MB/s
Collecting sentencepiece==0.1.95
  Downloading https://files.pythonhosted.org/packages/f5/99/e0808cb947ba10f
     |████████████████████████████████| 1.2MB 52.0MB/s
Requirement already satisfied: tqdm>=4.26.0 in /usr/local/lib/python3.7/dis
Collecting mpld3==0.3
  Downloading https://files.pythonhosted.org/packages/91/95/a52d3a83d0a29ba
     |████████████████████████████████| 798kB 51.5MB/s
Requirement already satisfied: gensim<=3.8.3,>=3.4.0 in /usr/local/lib/pyth
Collecting torch<=1.7.1,>=1.5.0
  Downloading https://files.pythonhosted.org/packages/90/5d/095ddddc91c8a76
     |████████████████████████████████| 776.8MB 23kB/s
Requirement already satisfied: tabulate in /usr/local/lib/python3.7/dist-pa
Collecting bpemb>=0.3.2
  Downloading https://files.pythonhosted.org/packages/91/77/3f0f53856e86af3
Collecting segtok>=1.5.7
  Downloading https://files.pythonhosted.org/packages/41/08/582dab5f4b1d5ca
Requirement already satisfied: lxml in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: matplotlib>=2.2.3 in /usr/local/lib/python3.
Collecting overrides<4.0.0,>=3.0.0
  Downloading https://files.pythonhosted.org/packages/ff/b1/10f69c00947518e
Collecting requests<3.0.0,>=2.25.1
  Downloading https://files.pythonhosted.org/packages/29/c1/24814557f1d22c5
     |████████████████████████████████| 61kB 9.6MB/s
Requirement already satisfied: importlib-metadata<4.0.0,>=3.7.0 in /usr/loc
Requirement already satisfied: wcwidth in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: wrapt<2,>=1.10 in /usr/local/lib/python3.7/d
```

```
df_train['processed'] = df_train['processed'].astype(str)
df_dev['processed'] = df_dev['processed'].astype(str)
df_test['processed'] = df_test['processed'].astype(str)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: SettingWith
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

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: SettingWith
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
  This is separate from the ipykernel package so we can avoid doing imports
```

```
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/pyth
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dis
```

```
training_sentences = df_train['processed'].values
PAD_token = '[PAD]'
OOV_token = '[OOV]'
vocab = {
    PAD_token: 0,
    OOV_token: 1
}

# Loop through all the training sentences
for sentence in training_sentences:
    tokens = sentence.split()  # word tokenise
    for word in tokens:
        # If we haven't already added the word to the vocab and we have an embed
        # We could alternatively add all the words, and learn the embeddings for
        if word not in vocab and word in glove_embedding.precomputed_word_embedd
            vocab[word] = len(vocab)  # add the word to the vocab, using the nex

vocab_size = len(vocab)  # store the vocab size
print('There are {} tokens in the vocab'.format(vocab_size))
print(list(vocab.items())[:20])  # show the first items in the vocab
```

```
There are 13140 tokens in the vocab
[('[PAD]', 0), ('[OOV]', 1), ('yes', 2), ('even', 3), ('better', 4), ('mich
```

```
  Created wheel for overrides: filename=overrides-3.1.0-py37-none-any.whl s
```

The GloVe embeddings are loaded from the flair python library to represent the word tokens in the email body text. Next, the word embeddings are represented as indeces, since this numeric representation uses less memory and is easier to learn for the LSTM RNN we will be using.

```
ERROR: torchtext 0.9.0 has requirement torch==1.8.0, but you'll have torch
ERROR: google-colab 1.0.0 has requirement requests~=2.23.0, but you'll have
ERROR: datascience 0.10.6 has requirement folium==0.2.1, but you'll have fo
Installing collected packages: overrides, requests, konoha, ftfy, deprecate
  Found existing installation: requests 2.23.0
    Uninstalling requests-2.23.0:
      Successfully uninstalled requests-2.23.0
  Found existing installation: gdown 3.6.4
    Uninstalling gdown-3.6.4:
      Successfully uninstalled gdown-3.6.4
  Found existing installation: torch 1.8.0+cu101
    Uninstalling torch-1.8.0+cu101:
      Successfully uninstalled torch-1.8.0+cu101
Successfully installed bpemb-0.3.2 deprecated-1.2.12 flair-0.8.0.post1 ftfy
2021-03-21 21:00:51,312 https://flair.informatik.hu-berlin.de/resources/emb
100%|██████████| 160000128/160000128 [00:09<00:00, 16756018.40B/s]2021-03-2

2021-03-21 21:01:01,966 removing temp file /tmp/tmp4o6i951t
2021-03-21 21:01:03,378 https://flair.informatik.hu-berlin.de/resources/emb
100%|██████████| 21494764/21494764 [00:02<00:00, 9871863.65B/s]2021-03-21 2
2021-03-21 21:01:06,075 removing temp file /tmp/tmpuwl2m388
```

```
import numpy as np

# Initialise the embedding matrix with zeros
embedding_matrix = np.zeros(shape=(vocab_size, embedding_size))

# Now, for every word in the vocab, we will update the corresponding row in the
# pre-loaded GloVe vector
for word, word_index in vocab.items():
    if word in glove_embedding.precomputed_word_embeddings.vocab:  # if the word
        word_embedding = glove_embedding.precomputed_word_embeddings.word_vec(wo
        # Save in the word_embedding matrix
        embedding_matrix[word_index, :] = word_embedding

# Show the word embeddings for the word "loves" at index 17
num_dimensions_to_show = 20
print(glove_embedding.precomputed_word_embeddings.word_vec("loves")[:num_dimensi
print(embedding_matrix[17, :num_dimensions_to_show])

mean_embedding = np.mean(embedding_matrix, axis=0)  # calculate the mean across
embedding_matrix[vocab[OOV_token], :] = mean_embedding
```

```
[ 0.17488    0.72361    0.74495   -0.27366   -0.18477    0.51367
 -0.55783   -0.022889   0.48081   -0.49067    0.24341   -0.16199
 -0.0029404  0.19032   -0.34576    0.18186    0.28025    0.33089
  0.36676    1.8785   ]
[-0.05534    -0.069753    0.020788   -0.72575998 -0.09101    -0.24574
  0.58590001 -0.35335001 -0.70144999 -0.91051    -0.52392     0.02143
  0.048701    0.1768     -0.025951    0.16628     0.59886998 -0.0033538
 -0.90547001  0.93190002]
```

```
def convert_words_to_indices(sentence: str) -> list:
    tokens = sentence.split()
    return [vocab[token] if token in vocab else vocab[OOV_token] for token in to

df_train['sentence_indices'] = df_train['processed'].apply(convert_words_to_indi
df_dev['sentence_indices'] = df_dev['processed'].apply(convert_words_to_indices)
df_test['sentence_indices'] = df_test['processed'].apply(convert_words_to_indice

df_train[['processed', 'sentence_indices', 'label_snorkel_binary']].head()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:6: SettingWith
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

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:7: SettingWith
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
  import sys
```

	processed	sentence_indices	label_snorkel_binary
2676	yes even better that!	[2, 3, 4, 1]	1
1080	michelle sheehan lead executive assistant mshe...	[5, 6, 7, 8, 9, 1, 10, 11, 12, 13, 14, 15, 1, 1]	0
3091	forwarded hunter shively hou ect enron technol...	[16, 17, 18, 19, 20, 21, 22, 23, 24, 17, 18, 1...	0
300	attached esp definition taken commodity regula...	[86, 87, 88, 89, 90, 1]	0

```
longest_sequence = max(df_train['sentence_indices'].apply(lambda x: len(x)))

def pad_sequence(tokens: list, maxlen: int = longest_sequence) -> list:
    pre = [0]*(maxlen - len(tokens))  # make a list of 0s the size that we need
    return pre + tokens

# Pad the training, validation and test data
df_train['sentence_indices'] = df_train['sentence_indices'].apply(pad_sequence)


df_dev['sentence_indices'] = df_dev['sentence_indices'].apply(pad_sequence)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: SettingWith
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
  """Entry point for launching an IPython kernel.
```

```
df_test['sentence_indices'] = df_test['sentence_indices'].apply(pad_sequence)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: SettingWith
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
  """Entry point for launching an IPython kernel.
```

```
X_train, y_train = np.vstack(df_train['sentence_indices'].values), df_train['lab
X_dev, y_dev = np.vstack(df_dev['sentence_indices'].values), df_dev['label_snork
X_test, y_test = np.vstack(df_test['sentence_indices'].values), df_test['label_s

with tf.device("/gpu:0"):
  model = keras.Sequential()  # the keras Sequential class groups a linear stack

  # Add an Embedding layer expecting input vocab, and output dimenstion the size
  # mask_zero = True tells the Embedding layer that we use index 0 for padded to
  model.add(layers.Embedding(input_shape=(longest_sequence,), input_dim=vocab_si
                             weights=[embedding_matrix], mask_zero=True, trainable

  # Create an LSTM layer
  model.add(layers.Bidirectional(layers.LSTM(256, return_sequences=True)))
  #model.add(layers.Bidirectional(layers.LSTM(256, return_sequences=True)))  # a
  model.add(layers.Bidirectional(layers.LSTM(128, return_sequences=True, recurre
  model.add(layers.Bidirectional(layers.LSTM(128, return_sequences=True)))  # an
  model.add(layers.Bidirectional(layers.LSTM(64, return_sequences=False)))  # an

  # Add another Dense layer (with relu activation) and apply dropout
  model.add(keras.layers.Dense(32, activation='relu'))
  model.add(keras.layers.Dropout(rate=0.2))  # we will drop 40% of the input uni

  # Add a Dense layer with a single unit and sigmoid activation.
  model.add(layers.Dense(1, activation='sigmoid'))

  # Compile the model
  model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy
```

```
WARNING:tensorflow:Layer lstm_15 will not use cuDNN kernel since it doesn't
WARNING:tensorflow:Layer lstm_15 will not use cuDNN kernel since it doesn't
WARNING:tensorflow:Layer lstm_15 will not use cuDNN kernel since it doesn't
```

The model compiled above shows the result of some iterative tuning for this dataset that was made on the basis of the RNN with LSTM cells and dropout presented in the NLP notebook 16 on advanced RNNs. Adjustments and evaluation were made on the basis of validation/dev set accuracy.

Compared to the original model, it was found that one of the originally five LSTM layers could be dropped for improved performance (at approximately 1 million fewer trainable parameters and faster training times). Also, reducing the dropout rate subsequently to only 20% after the 31 neuron dense layer instead of the previous 40% seemed to slightly improve the validation accuracy, possibly due to the now reduced regularization being at lower risk of underfitting the data.

The output of the model is a single neuron with sigmoid activation to predict the binary class of the sentiment label. The resulting model can be seen below.

```
with tf.device("/gpu:0"):
 model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_3 (Embedding)      (None, 255, 100)          1314000
_________________________________________________________________
bidirectional_14 (Bidirectio (None, 255, 512)          731136
_________________________________________________________________
bidirectional_15 (Bidirectio (None, 255, 256)          656384
_________________________________________________________________
bidirectional_16 (Bidirectio (None, 255, 256)          394240
_________________________________________________________________
bidirectional_17 (Bidirectio (None, 128)               164352
_________________________________________________________________
dense_6 (Dense)              (None, 32)                4128
_________________________________________________________________
dropout_3 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_7 (Dense)              (None, 1)                 33
=================================================================
Total params: 3,264,273
Trainable params: 1,950,273
Non-trainable params: 1,314,000
_________________________________________________________________
```

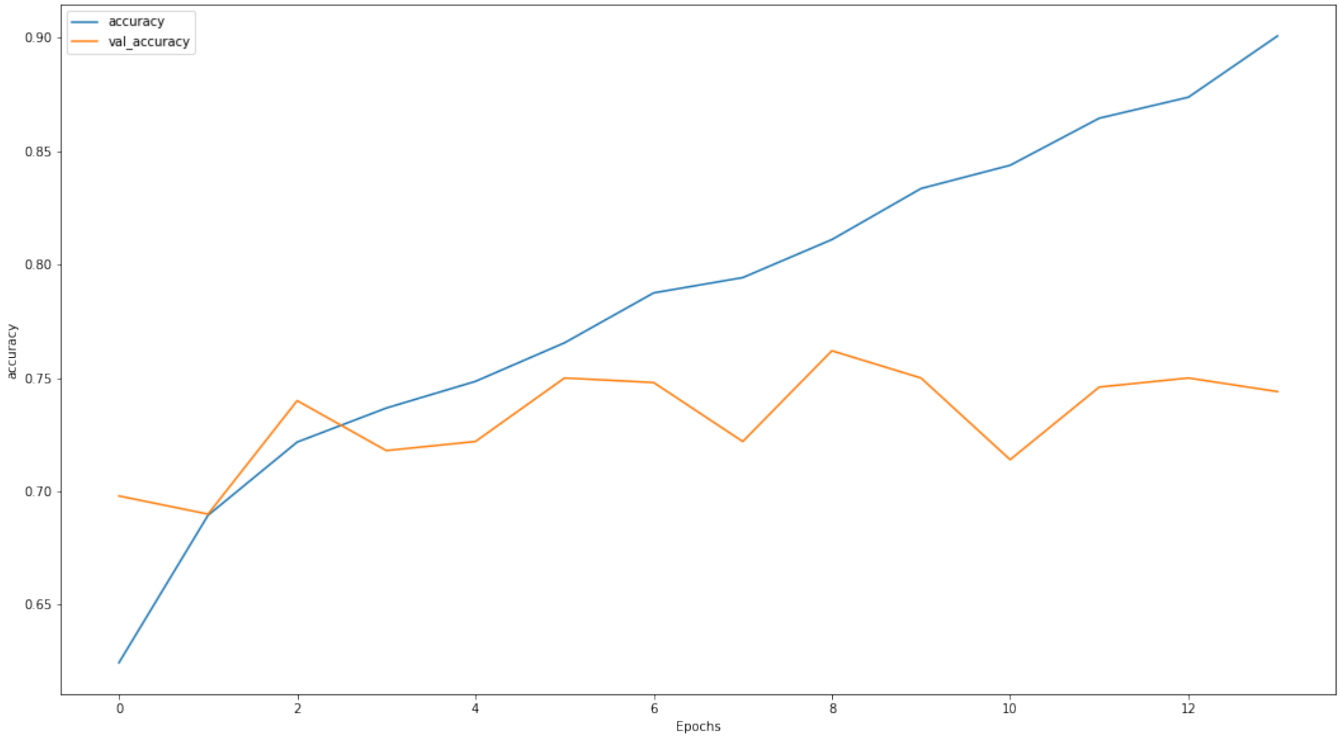
Model training was adjusted several times during finetuning. While the early stopping callback received an increase patience of 5 epochs based on validation accuracy improvement to not stop too early, the number of epochs was reduced multiple times to 20 since further training would only lead to overfitting to the training data. Also, batch size was changed several times (and the neurons per layer in model compilation), but a batch size of 256 proved to be optimal for model performance and training speed.

```
NUM_EPOCHS = 20
BATCH_SIZE = 256
early_stopping = keras.callbacks.EarlyStopping(monitor='val_accuracy', patience=
with tf.device("/gpu:0"):
 history = model.fit(X_train,  y_train,
                     batch_size=BATCH_SIZE, epochs=NUM_EPOCHS,
                     validation_data=(X_dev, y_dev), callbacks=[early_stopping])
```

```
Epoch 1/20
16/16 [==============================] - 102s 5s/step - loss: 0.6692 - accu
Epoch 2/20
16/16 [==============================] - 74s 5s/step - loss: 0.6024 - accur
Epoch 3/20
16/16 [==============================] - 75s 5s/step - loss: 0.5802 - accur
Epoch 4/20
16/16 [==============================] - 74s 5s/step - loss: 0.5506 - accur
Epoch 5/20
16/16 [==============================] - 75s 5s/step - loss: 0.5246 - accur
Epoch 6/20
16/16 [==============================] - 75s 5s/step - loss: 0.5039 - accur
Epoch 7/20
16/16 [==============================] - 75s 5s/step - loss: 0.4583 - accur
Epoch 8/20
16/16 [==============================] - 75s 5s/step - loss: 0.4376 - accur
Epoch 9/20
16/16 [==============================] - 74s 5s/step - loss: 0.4365 - accur
Epoch 10/20
16/16 [==============================] - 74s 5s/step - loss: 0.3924 - accur
Epoch 11/20
16/16 [==============================] - 74s 5s/step - loss: 0.3734 - accur
Epoch 12/20
16/16 [==============================] - 73s 5s/step - loss: 0.3318 - accur
Epoch 13/20
16/16 [==============================] - 73s 5s/step - loss: 0.2971 - accur
Epoch 14/20
16/16 [==============================] - 73s 5s/step - loss: 0.2455 - accur
```

Results

```
import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline
plt.rcParams['figure.figsize'] = (18, 10)  # set default figure size

def plot_graphs(history, metric):
    plt.plot(history.history[metric])
    plt.plot(history.history['val_'+metric], '')
    plt.xlabel("Epochs")
    plt.ylabel(metric)
    plt.legend([metric, 'val_'+metric])
    plt.show()

plot_graphs(history, 'accuracy')
```



As can be seen from the training and validation accuracy graph above, while training accuracy increases continuously during the training epochs and reaches more than 90% accuracy after 13 epochs, there is no continuous increase in validation accuracy. This means that further training would potentially overfit the model to the training data and risk worse generalization. Early stopping on validation accuracy stops the training after 13 epochs instead of the maximuim 20. In previous versions of the model, the validation accuracy was often less stable and started to decrease even earlier, which is why the number of layers was slightly reduced in the model.

The prediction on the development set below shows that the model seems to generalize well with early stopping and restoration of the best weights for maximum validation accuracy.

Error Analysis

```
def predict_sentiment(X, threshold=0.5):
    probabilities = model.predict(X)
    predictions = [1 if prob >= threshold else 0 for prob in probabilities]
    return predictions
```

```
df_dev['prediction'] = predict_sentiment(X_dev)
df_dev[['processed', 'label_snorkel_binary', 'prediction']][:20]
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: SettingWith
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
  """Entry point for launching an IPython kernel.
```

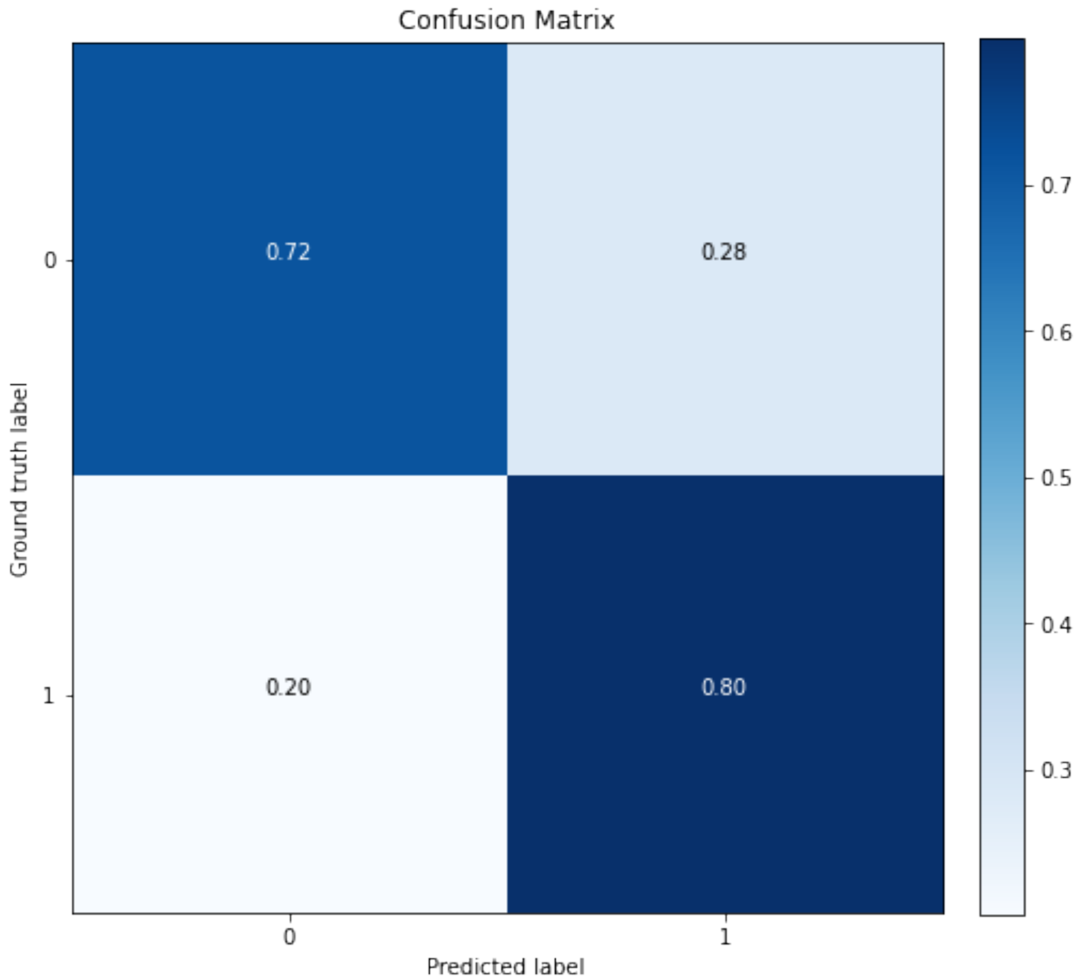
	processed	label_snorkel_binary	prediction
3074	original message donoho lindy sent wednesday o...	1	1
3028	baby step forwarded sara shackleton hou ect en...	1	0
4162	document setting mlokay local setting temporar...	0	0
4758	confirmed date prc rod hayslett gretchen jenni...	1	1
3143	karen lisa bill rose engeldorf town week mean ...	1	0
4667	good work move flight stairs. move pay someone...	1	1
2681	susan attached description gallup facility att...	1	1
1831	hey rob try call questions. thanks. john origi...	1	1
4069	hello look like alot mark fischer west deal co...	0	1
1967	sold july north atlantic sithe disregard previ...	0	0
4870	yahoo global exchange service perfect commerce...	1	1
1020	sara following item outstanding issue respect ...	0	1
2852	office wednesday can ask jessica presas laura ...	1	1
1016	joe attached clean blacklined copy partial ter...	1	1
2794	adam johnson	0	0

```
from sklearn.metrics import accuracy_score
model_val_acc = 100 * accuracy_score(df_dev['label_snorkel_binary'].values, df_d
print("The model's validation accuracy score is {:.2f}%".format(model_val_acc))
```

```
The model's validation accuracy score is 76.20%
```

```
import itertools
def plot_confusion_matrix(cm, classes, normalize=False, title='Confusion Matrix'
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
    fig, ax = plt.subplots(1, 1, figsize=(8, 8))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.colorbar(fraction=0.046, pad=0.04)

    # Add the labels
    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    ax.set(yticks=[-0.5, 1.5],
           xticks=[0, 1],
           yticklabels=classes,
           xticklabels=classes)
    ax.yaxis.set_major_locator(matplotlib.ticker.IndexLocator(base=1, offset=0.5
    if title:
        plt.title(title)
    plt.ylabel('Ground truth label')
    plt.xlabel('Predicted label')
    plt.show()
```

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(df_dev['label_snorkel_binary'].values, df_dev['prediction'
plot_confusion_matrix(cm, ("0", "1"), normalize=True)
```



In the first 20 lines of predicted data and in the confusion matrix, we can see that the model seems to be almost balanced between false positive and false negative errors, which is hinting at good decision boundaries. We see that those cases in which the model made a wrong prediction were mostly edge cases where sentiment seems neutral. Thus, false positives and false negatives don't occur extensively, while precision could be improved with training on more data and more finetuning.

BERT Model for Sentiment Classification

BERT was trained on English Wikipedia (2,500M words) and BooksCorpus (800M words), achieving the best accuracies for some NLP tasks in 2018 (Devlin et al., 2018). We have chosen to use BERT as a sentiment classifier as it is one of the most popular pre-trained model and produces state-of-the-art performances and was discussed during lecture 20 - 'The Transformers'. It was easily accessible via the HuggingFace library.

In addition, we will use PyTorch to fine-tune the model.

Model preprocessing

To feed the data to the model, we need to preprocess it as we did for the LSTM model. However, BERT requires different preprocessing steps compared to LSTM. Indeed, for BERT we need to add special tokens to separate sentences and do classification. Furthermore, similarly to what we have done in LSTM, we will introduce padding. Finally, we will create arrays of 0s for pad tokens and 1s for real tokens called attention mask.

For this tasked we have chosen to use the pre-buid bert base uncased tokenizer as will preprocess the data for us and works very fast. Furthermore we chose the uncased version as we lowered the sentences and therefore there were no cased words.

```
import sys
import numpy
numpy.set_printoptions(threshold=sys.maxsize)
```

```
# df.processed.unique()
```

```
df["processed"][0]
# df.head()
```

```
'sheila per phone message also fyi completed review csfb agreement last ni
ght faxed comment attorney. cheryl nelson senior counsel'
```

```
df["body"][0]
```

```
'sheila, as per your phone message:\n\n\n\nalso, fyi, i completed review o
f the csfb agreement last night and faxed \ncomments to their attorney.\n\
ncheryl nelson\nsenior counsel\neb3816\n(713) 345-4693\nhttp://gss.enron.c
```

```
from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer()
# X = vectorizer.fit_transform(df["processed"][:5])
# print(vectorizer.get_feature_names(), '\n')

# # Show featurised representations
# for i, sentence in enumerate(df["processed"][:5]):
#     print("Email {}:\n{}".format(i+1, X.toarray()[i,:]))


PRE_TRAINED_MODEL_NAME = 'bert-base-uncased'
tokenizer = BertTokenizer.from_pretrained(PRE_TRAINED_MODEL_NAME)
```

Downloading: 100% 232k/232k [00:00<00:00, 836kB/s]

Downloading: 100% 28.0/28.0 [00:00<00:00, 179B/s]

Downloading: 100% 466k/466k [00:00<00:00, 6.22MB/s]

```
encoding = tokenizer.encode_plus(
  df["processed"][10],
  max_length=128,
  add_special_tokens=True, # Add '[CLS]' and '[SEP]' to show begining and end of
    truncation = True,
  return_token_type_ids=False,
  pad_to_max_length=True,
  return_attention_mask=True,
  return_tensors='pt',  # Return PyTorch tensors
)
encoding.keys()
```

```
/usr/local/lib/python3.7/dist-packages/transformers/tokenization_utils_base
  FutureWarning,
dict_keys(['input_ids', 'attention_mask'])
```

```
encoding['input_ids'][0]
```

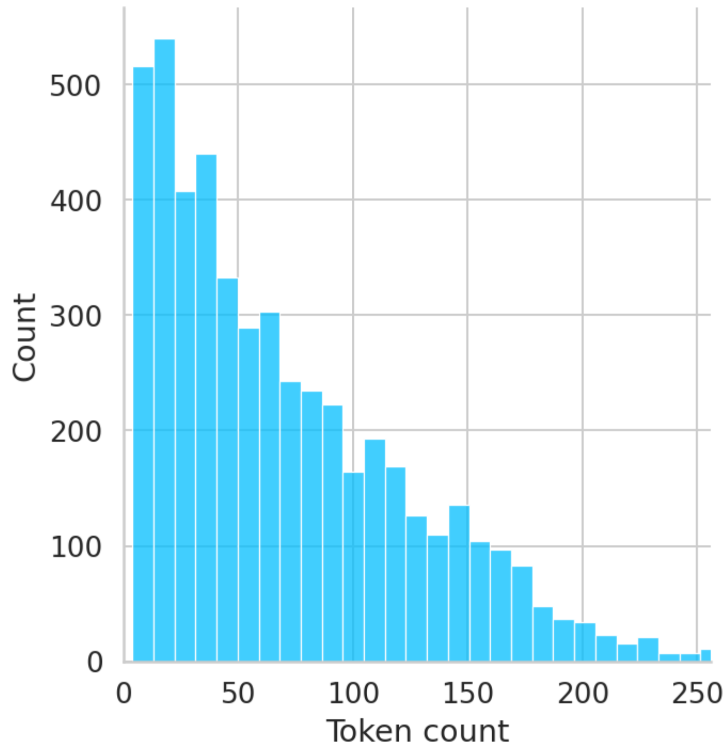
```
tensor([  101,  5763,  6128, 26947, 22942, 22476,  2075,  8001,  4433,  455
         2304, 18194,  4372,  4948,  4762, 29109,  2140,  5309,  3820,  101
         3531,  9449,  6016,  2051,  6016,  7396,  3914,  2157,  2191,  317
         7615,  2241,  7396,  3319,  1012,  4762, 29109,  7770,  4948,  530
         3820,  1012,  9986, 19818,  9080,  2089, 18777,  4905,  7396,  480
         2089,  4728, 21598, 18777,  1012,  2025,  3154,  3832,  7799,  218
         3762, 19488,  2363, 19818,  9080,  7561,  3319, 28170,  4353, 2473
        19818,  9080,  9975, 10890,  1012,  8343,  2363,  4807,  7561,  353
         2025,  8757,  3202,  7026,  5653,  3202,  3972, 12870,  4471, 1444
         2015,  1012, 27830,  2098,  4762, 29109,  2140,  4372,  4948,  530
         3820,  1012, 19387,  2546,  4762, 29109,  7770,  4948,  5309,  382
         1012,  9986,   102,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0])
```

```
print(tokenizer.convert_ids_to_tokens(encoding['input_ids'][0]))
```

```
['[CLS]', 'promised', 'jerry', 'lei', '##tman', 'attach', '##ing', 'revised
```

This shows an example the tokenzier tokenzises the sentences

```
class_names = ['negative', 'positive']
token_lens = []
for txt in df.processed:
    tokens = tokenizer.encode(txt, max_length=512,truncation=True)
    token_lens.append(len(tokens))

sns.displot(token_lens)
plt.xlim([0, 256]);
plt.xlabel('Token count');
```



In order to feed data to BERT we need to choose fixed length sentences. We have used the token count to determine the length of the sentence. As we can see practically all of the emails have less than 250 tokens. We will therefore set the MAX LENGTH of 256 for our model.

BERT - Sentiment Classifier

Now that all the data has been processed we can start by building the model. We have chosen to use the basic BERT model rather than 'BERTforSentenceClassification' to create a new class so we can specify our own choice of classifiers. Furthermore, we will fine-tune the model by adding one output layer in order to get back the classification token.

```
class GPEmailsDataset(Dataset):
    def __init__(self, emails, targets, tokenizer, max_len):
        self.emails = emails
        self.targets = targets
        self.tokenizer = tokenizer
        self.max_len = max_len
    def __len__(self):
        return len(self.emails)

    def __getitem__(self, item):
        emails = str(self.emails[item])
        target = self.targets[item]

        encoding = self.tokenizer.encode_plus(
          emails,
            truncation=True,
          add_special_tokens=True,
          max_length=self.max_len,
          return_token_type_ids=False,
          padding="max_length",
          return_attention_mask=True,
          return_tensors='pt',
    )
        return {
          'emails_text': emails,
          'input_ids': encoding['input_ids'].flatten(),
          'attention_mask': encoding['attention_mask'].flatten(),
          'targets': torch.tensor(target, dtype=torch.long)
    }


df_train, df_test = train_test_split( df,test_size=0.2,random_state=RANDOM_SEED)
df_val, df_test = train_test_split(df_test,test_size=0.5,random_state=RANDOM_SEE


MAX_LEN = 256
```

We will create an iterator for our dataset using the torch DataLoader class. This will help us save on memory durnig training.

```
def create_data_loader(df, tokenizer, max_len, batch_size):
    ds = GPEmailsDataset(emails=df.processed.to_numpy(),
                         targets=df.label_snorkel_binary.to_numpy(),
                         tokenizer=tokenizer, max_len=max_len
  )
    return DataLoader (ds, batch_size=batch_size,
                       num_workers=2
  )
BATCH_SIZE = 20
train_data_loader = create_data_loader(df_train, tokenizer, MAX_LEN, BATCH_SIZE)
val_data_loader = create_data_loader(df_val, tokenizer, MAX_LEN, BATCH_SIZE)
test_data_loader = create_data_loader(df_test, tokenizer, MAX_LEN, BATCH_SIZE)


data = next(iter(train_data_loader))
data.keys()

print(data['input_ids'].shape)
print(data['attention_mask'].shape)
print(data['targets'].shape)
```

```
torch.Size([20, 256])
torch.Size([20, 256])
torch.Size([20])
```

```
bert_model = BertModel.from_pretrained(PRE_TRAINED_MODEL_NAME)
```

```
Downloading: 100%    433/433 [00:00<00:00, 2.28kB/s]

Downloading: 100%    440M/440M [00:11<00:00, 37.7MB/s]
```

```
class SentimentClassifier(nn.Module):

    def __init__(self, n_classes):
        super(SentimentClassifier, self).__init__()
        self.bert = BertModel.from_pretrained(PRE_TRAINED_MODEL_NAME)
        self.drop = nn.Dropout(p=0.3)
        self.out = nn.Linear(self.bert.config.hidden_size, n_classes)
        # print((self.bert.config.hidden_size))
    def forward(self, input_ids, attention_mask):
        returned = self.bert(
        input_ids=input_ids,
        attention_mask=attention_mask
        )
        pooled_output = returned["pooler_output"]
        output = self.drop(pooled_output)
        return self.out(output)
```

We added a dropout layer for regularization of our model, to prevent to much overfitting.

```
model = SentimentClassifier(2)
model = model.to(device)
```

```
input_ids = data['input_ids'].to(device)
attention_mask = data['attention_mask'].to(device)
print(input_ids.shape) # batch size x seq length
print(attention_mask.shape) # batch size x seq length
```

```
    torch.Size([20, 256])
    torch.Size([20, 256])
```

```
import torch.nn.functional as F
```

```
F.softmax(model(input_ids, attention_mask), dim=1) #output of the model

    tensor([[0.6896, 0.3104],
            [0.6615, 0.3385],
            [0.5556, 0.4444],
            [0.6976, 0.3024],
            [0.4613, 0.5387],
            [0.6034, 0.3966],
            [0.5613, 0.4387],
            [0.6263, 0.3737],
            [0.7275, 0.2725],
            [0.5174, 0.4826],
            [0.5154, 0.4846],
            [0.6161, 0.3839],
            [0.4858, 0.5142],
            [0.6799, 0.3201],
            [0.5222, 0.4778],
            [0.6927, 0.3073],
            [0.5332, 0.4668],
            [0.5848, 0.4152],
            [0.5606, 0.4394],
            [0.6464, 0.3536]], device='cuda:0', grad_fn=<SoftmaxBackward>)
```

Training

We will use AdamW for the optimazation which is provided by Hugging Face as it corrects the weight decay. Furthermore, we used a learning rate at 2e-5 which is one of the recommended learning rate in BERT's documentation.

```
EPOCHS = 10
optimizer = AdamW(model.parameters(), lr=2e-5, correct_bias=False)
total_steps = len(train_data_loader) * EPOCHS
scheduler = get_linear_schedule_with_warmup(
  optimizer,
  num_warmup_steps=0,
  num_training_steps=total_steps
)
loss_fn = nn.CrossEntropyLoss().to(device)
```

We then defined two helper functions to train our model for one epoch, and evaluate the results

```
#Go through our training data in one epoch
def train_epoch(model,data_loader,loss_fn,optimizer,device, scheduler, n_example

    model = model.train()
    losses = []
    correct_predictions = 0
    for d in data_loader:
        input_ids = d["input_ids"].to(device)
        attention_mask = d["attention_mask"].to(device)
        targets = d["targets"].to(device)
        outputs = model(
          input_ids=input_ids,
          attention_mask=attention_mask
    )
        _, preds = torch.max(outputs, dim=1)
        loss = loss_fn(outputs, targets)
        correct_predictions += torch.sum(preds == targets) # get all predictions
        losses.append(loss.item())
        loss.backward()
        nn.utils.clip_grad_norm_(model.parameters(), max_norm=1.0)
        optimizer.step()
        scheduler.step()
        optimizer.zero_grad()
    return correct_predictions.double() / n_examples, np.mean(losses)


def eval_model(model, data_loader, loss_fn, device, n_examples):
    model = model.eval()
    losses = []
    correct_predictions = 0
    with torch.no_grad():
        for d in data_loader:
            input_ids = d["input_ids"].to(device)
            attention_mask = d["attention_mask"].to(device)
            targets = d["targets"].to(device)
            outputs = model(
                input_ids=input_ids,
                attention_mask=attention_mask
      )
            _, preds = torch.max(outputs, dim=1)
            loss = loss_fn(outputs, targets)
            correct_predictions += torch.sum(preds == targets)
            losses.append(loss.item())
    return correct_predictions.double() / n_examples, np.mean(losses)
```

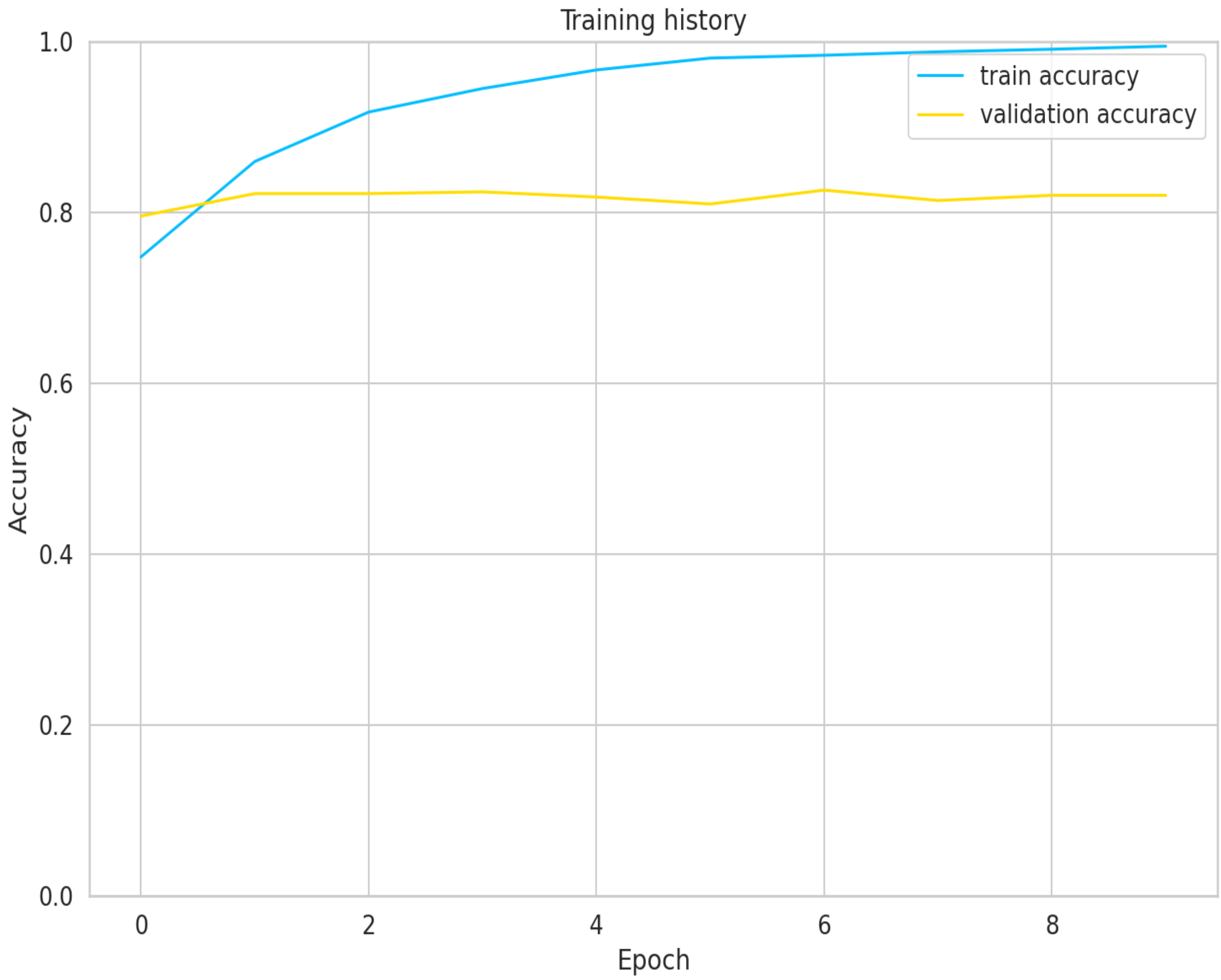
With the two helper functions, we defined a training loop for 10 epochs

```
%%time
```

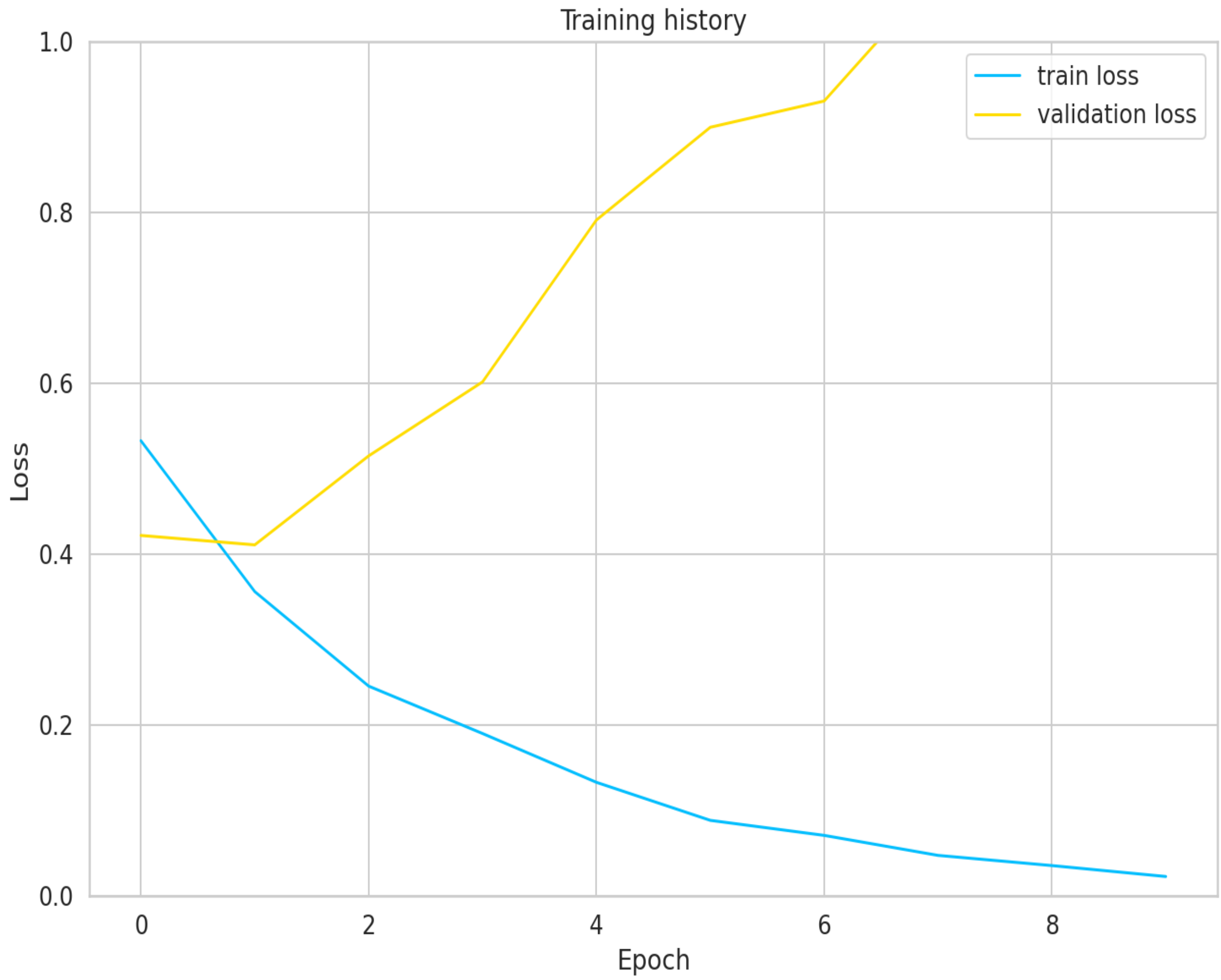
```
history = defaultdict(list)
best_accuracy = 0

for epoch in range(EPOCHS):
    print(f'Epoch {epoch + 1}/{EPOCHS}')
    print('-' * 10)
    train_acc, train_loss = train_epoch(
    model,
    train_data_loader,
    loss_fn,
    optimizer,
    device,
    scheduler,
    len(df_train)
  )
    print(f'Train loss {train_loss} accuracy {train_acc}')

    val_acc, val_loss = eval_model( model,val_data_loader,loss_fn,device,len(df_
  )
    print(f'Val   loss {val_loss} accuracy {val_acc}')
    print()

    history['train_acc'].append(train_acc)
    history['train_loss'].append(train_loss)
    history['val_acc'].append(val_acc)
    history['val_loss'].append(val_loss)

    if val_acc > best_accuracy:
        torch.save(model.state_dict(), 'best_model_state.bin')
        best_accuracy = val_acc
```

```
Epoch 1/10
----------
Train loss 0.5325414388619288 accuracy 0.747661188369153
Val   loss 0.4215835547447205 accuracy 0.7955465587044535

Epoch 2/10
----------
Train loss 0.3558491629670666 accuracy 0.8594184576485461
Val   loss 0.4105792248249054 accuracy 0.8218623481781376

Epoch 3/10
----------
Train loss 0.24519805714600917 accuracy 0.9173198482932996
Val   loss 0.5147284209728241 accuracy 0.8218623481781376

Epoch 4/10
----------
Train loss 0.18969706014842924 accuracy 0.9448798988621997
Val   loss 0.6013904649019242 accuracy 0.8238866396761134

Epoch 5/10
----------
Train loss 0.13254661899944298 accuracy 0.9666245259165612
Val   loss 0.7909202051162719 accuracy 0.8178137651821863

Epoch 6/10
----------
Train loss 0.08801498077574628 accuracy 0.9805309734513273
Val   loss 0.8994998586177826 accuracy 0.8097165991902834

Epoch 7/10
----------
Train loss 0.07041132965447343 accuracy 0.9838179519595448
Val   loss 0.9302435219287872 accuracy 0.8259109311740891

Epoch 8/10
----------
Train loss 0.046987214030443945 accuracy 0.9878634639696586
Val   loss 1.0792480552196502 accuracy 0.8137651821862348

Epoch 9/10
----------
Train loss 0.03505636012532062 accuracy 0.990897597977244
Val   loss 1.0916883897781373 accuracy 0.819838056680162

Epoch 10/10
----------
Train loss 0.02225782785293025 accuracy 0.9944374209860936
Val   loss 1.1140643894672393 accuracy 0.819838056680162

CPU times: user 15min 55s, sys: 14min 53s, total: 30min 49s
Wall time: 31min
```

```
plt.plot(history['train_acc'], label='train accuracy')
plt.plot(history['val_acc'], label='validation accuracy')
plt.title('Training history')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend()
plt.ylim([0, 1]);
```



```
plt.plot(history['train_loss'], label='train loss')
plt.plot(history['val_loss'], label='validation loss')
plt.title('Training history')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend()
plt.ylim([0, 1]);
```



We can observe that the training accuracy and the validation accuracy intersect after 1 epoch. The training accuracy approaches 100% after 7 epochs, however, the validation accuracy apears to stay stable over training instances.

As further exploration, we could try to fine-tune further the model by changing some parameters such as the learning rate or the dropout probability.

For the purpose of the project, we will keep the results and evaluate this model on our test set.

```
#Evaluation
len(df_test)

df_test["processed"]
```

```
642              get free download msn explorer report.xls
315     dan good morning ref. tel. friday bunkering ho...
561     staff meeting monday dennis vega gave brief ma...
2770    peppermint missed drew birthday mom miss much ...
4231    auto generated mail. training service auto gen...
                                ...
2988             game. win fred right after. fuck eating.
2373                        image image image image image
4503    looking copy final cpuc order. please email go...
3129    mr. christman thank invitation dr. lay speak u...
2776    forwarded mike grigsby hou ect enron north ame...
Name: processed, Length: 495, dtype: object
```

```
test_acc, _ = eval_model(
  model,
  test_data_loader,
  loss_fn,
  device,
  len(df_test)
)
test_acc.item()
```

```
0.8282828282828283
```

The accuracy on the test set is smilar to the validation set, the model seems to generalise well.

Error Analysis

```
def get_predictions(model, data_loader):
    model = model.eval()
    emails_texts = []
    predictions = []
    prediction_probs = []
    real_values = []
    with torch.no_grad():
        for d in data_loader:
            texts = d["emails_text"]
            input_ids = d["input_ids"].to(device)
            attention_mask = d["attention_mask"].to(device)
            targets = d["targets"].to(device)
            outputs = model(
              input_ids=input_ids,
              attention_mask=attention_mask
          )
            _, preds = torch.max(outputs, dim=1)
            emails_texts.extend(texts)
            predictions.extend(preds)
            prediction_probs.extend(outputs)
            real_values.extend(targets)
        predictions = torch.stack(predictions).cpu()
        prediction_probs = torch.stack(prediction_probs).cpu()
        real_values = torch.stack(real_values).cpu()
        return emails_texts, predictions, prediction_probs, real_values


y_emails_texts, y_pred, y_pred_probs, y_test = get_predictions(
  model,
  test_data_loader
)
```

The first step in our error analysis is to look at what the model predicted for the first 20 instances. This will allows us to understand where the model has some difficulies in predicting the correct sentiment.

```
import pandas as pd
pd.set_option('max_colwidth', 500)

df_test['predictions'] = y_pred
df_test[['processed', 'label_snorkel_binary', 'predictions']].head(20)
```

	processed	label_snorkel_binary	predictions
642	get free download msn explorer report.xls	1	1
	dan good morning ref. tel. friday bunkering hoegh galleon lake charles redused quantity f.o. recommended bunker prosedure posted onb. not fill tank capasity port. max capasity		

315	bunkerbarge mt. revert question regarding demurrage. please note following bunker nomination vessel lng hoegh galleon port lake charles date eta june quantity grade fuel cst price usd mtw delivery barge usd min usd overtime buyer account seller matrix marine fuel llc supplier matrix marine fuel llc buyer leif egh oslo ...	1	1
561	staff meeting monday dennis vega gave brief marketing presentation. copy presentation attached not attendance anyone would like electronic copy. please feel free contact dennis would like discus content require support. thank you. cathy phillips	1	1
2770	peppermint missed drew birthday mom miss much except chris oscar got mad not include little oscar drew birthday party. pissed mom dad. thing party greg tia celia party not talk other. also super ramiro band little while. not forget get drew birthday present please send cash only!!	0	0
4231	auto generated mail. training service auto generated sent tuesday march jason wolfe subject investinme.enron.com login information dear jason wolfe log investinme. enron.com following login password login jason.wolfe password wolfej first step get started login password noted above. web site demonstration held march a.m. every half hour please plan attend one session. question regarding login information can contact development center investinme. enron.com new tool help manage professional d...	0	0
2053	sara home today..if get chance please call thank	0	0
2876	would much. lisa mellencamp kay mann corp enron subject draft enron cpcn application richmond project vision complete panic sweet child. least saved that. lisa mellencamp enron north america corp. smith st. houston tel fax kay mann lisa mellencamp hou ect subject draft enron cpcn application richmond project yes it. thought heartburn michael. summer. lisa mellencamp kay mann corp enron subject draft enron cpcn application richmond project hopefully not	0	0

Secondly, we will calculate the precision, recall and F1-score in order to understand precisly where the model is failling and where it performs well.

continuing end june prc discus finalize pacific basin lng strategy organization issue

```
print(classification_report(y_test, y_pred, target_names=class_names))

              precision    recall  f1-score   support

    negative       0.81      0.78      0.79       210
    positive       0.84      0.86      0.85       285

    accuracy                           0.83       495
   macro avg       0.83      0.82      0.82       495
weighted avg       0.83      0.83      0.83       495
```

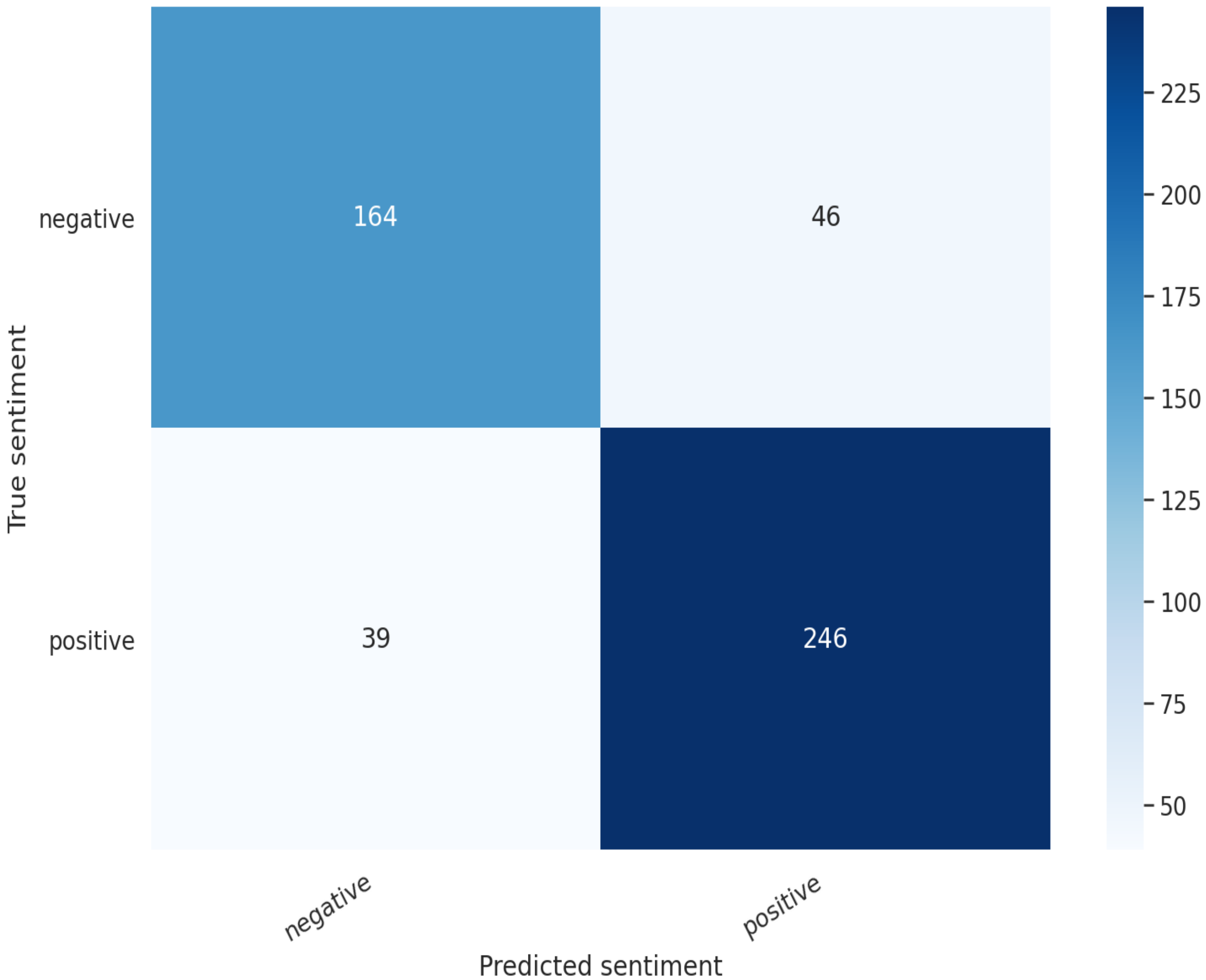
help !!!!!!!!!!!!!!!!!! gotta see man! see

From the classification report, we can observe that the model predicts a lot better positive sentiment compared to negative. Indeed, the F1 score in predicting positive sentiment is 85% compared to 79% in predicting negative. Furthermore, we can observe that has high precision for both sentiment but is lacking recall in predicting negative sentiment. Meaning that the model classifies more false negative.

From the sample predictions we can observe that where the model has difficulties in predicing are for instances where even humans would have difficulties in classifying as a binary classification.

As further exploration, we could try a multi-class classification and evaluate how the model performs.

```
def show_confusion_matrix(confusion_matrix):
    hmap = sns.heatmap(confusion_matrix, annot=True, fmt="d", cmap="Blues")
    hmap.yaxis.set_ticklabels(hmap.yaxis.get_ticklabels(), rotation=0, ha='right
    hmap.xaxis.set_ticklabels(hmap.xaxis.get_ticklabels(), rotation=30, ha='righ
    plt.ylabel('True sentiment')
    plt.xlabel('Predicted sentiment');
cm = confusion_matrix(y_test, y_pred)
df_cm = pd.DataFrame(cm, index=class_names, columns=class_names)
show_confusion_matrix(df_cm)
```



Finally from the confusion matrix allows a visual representation of the scores calculated above. This confirms our previous analysis as we observe a higher number of false negative.

The results shows that the model is performing relatively well on the test set with only a few instances misclassified

Extractive Summarisation with BERT, GPT-2, Text Rank

To summarise the incoming emails, we will employ extractive summarisation, which seeks to find the most informative sentences within a large body of text and then forms them to a summary.

We will compare various techniques using 100 manually summarised emails, such that we can compare our models with the manual summarisations.

Importing Libraries

```
# Installing relevant libraries
!pip install langdetect
!pip install bert-extractive-summarizer
!pip install torch
!pip install rouge_score
!pip install transformers==2.2.0
!pip install spacy==2.0.12
```

```
         .:::.     .::.
        ....yy:    .yy.
        :.  .yy.    y.
             :y:   .:
             .yy  .:
              yy..:
              :y:.
              .y.
             .:.
        ....:.
        :::.

• Project files and data should be stored in /project. This is shared among
  in the project.
• Personal files and configuration should be stored in /home/faculty.
• Files outside /project and /home/faculty will be lost when this server is
• Create custom environments to setup your servers reproducibly.

Requirement already satisfied: langdetect in /opt/anaconda/envs/Python3/lib
Requirement already satisfied: six in /opt/anaconda/envs/Python3/lib/python

         .:::.     .::.
        ....yy:    .yy.
        :.  .yy.    y.
             :y:   .:
             .yy  .:
              yy..:
              :y:.
              .y.
```

```
            .:.
        ....:.
        :::.

  • Project files and data should be stored in /project. This is shared among
    in the project.
  • Personal files and configuration should be stored in /home/faculty.
  • Files outside /project and /home/faculty will be lost when this server is
  • Create custom environments to setup your servers reproducibly.

Requirement already satisfied: bert-extractive-summarizer in /opt/anaconda/
Requirement already satisfied: transformers in /opt/anaconda/envs/Python3/l
Requirement already satisfied: spacy in /opt/anaconda/envs/Python3/lib/pyth
Requirement already satisfied: scikit-learn in /opt/anaconda/envs/Python3/l
Requirement already satisfied: requests in /opt/anaconda/envs/Python3/lib/p
Requirement already satisfied: tqdm in /opt/anaconda/envs/Python3/lib/pytho
Requirement already satisfied: sacremoses in /opt/anaconda/envs/Python3/lib
Requirement already satisfied: sentencepiece in /opt/anaconda/envs/Python3/
Requirement already satisfied: regex in /opt/anaconda/envs/Python3/lib/pyth
Requirement already satisfied: boto3 in /opt/anaconda/envs/Python3/lib/pyth
Requirement already satisfied: numpy in /opt/anaconda/envs/Python3/lib/pyth
Requirement already satisfied: cymem<1.32,>=1.30 in /opt/anaconda/envs/Pyth
Requirement already satisfied: murmurhash<0.29,>=0.28 in /opt/anaconda/envs
Requirement already satisfied: plac<1.0.0,>=0.9.6 in /opt/anaconda/envs/Pyt
Requirement already satisfied: ujson>=1.35 in /opt/anaconda/envs/Python3/li
Requirement already satisfied: preshed<2.0.0,>=1.0.0 in /opt/anaconda/envs/
Requirement already satisfied: thinc<6.11.0,>=6.10.3 in /opt/anaconda/envs/
Requirement already satisfied: dill<0.3,>=0.2 in /opt/anaconda/envs/Python3
Requirement already satisfied: scipy>=0.19.1 in /opt/anaconda/envs/Python3/
```

```
# Importing necessary libraries
import pandas as pd
import numpy as np
import torch
import langdetect
import re
import math
from rouge_score import rouge_scorer

# for BERT
from summarizer import Summarizer

# for GPT-2
from summarizer import TransformerSummarizer

# for Text Rank
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize, sent_tokenize, RegexpTokenizer
from nltk.stem import WordNetLemmatizer
from nltk.stem.porter import PorterStemmer
from sklearn.metrics.pairwise import cosine_similarity
import networkx as nx
from nltk import sent_tokenize, word_tokenize, PorterStemmer

# Ignoring Warnings
pd.set_option('mode.chained_assignment', None)


# Read manually labelled data
emails_labelled = pd.read_csv('/project/emails_labelled.csv',index_col=[0])

# Formatting the dataframe
emails_labelled = emails_labelled.reset_index()
emails_labelled = emails_labelled.drop(["index","from","to"], axis = 1)


# Assigning text body to BERT and GPT-2 columns for further analysis, later we c
emails_labelled["Bert"] = emails_labelled["body"]
emails_labelled["GPT2"] = emails_labelled["body"]
```

```
emails_labelled.head()
```

	date	subject	body	sentiment_label	summary_label	
0	Fri, 14 Jul 2000 08:44:00 -0700 (PDT)	New Gas Transportation Product on EnronOnline	The Global Gas Pipeline group is looking to tr...	0.0	Global Gas Pipeline group is looking to trade ...	The Gl Pipeline look
1	Wed, 6 Dec 2000 07:26:00 -0800 (PST)	Revised CalJournal Ad	IEP Team,\nAttached is a revised January CalJo...	0.0	Revised January CalJournal ad for review	Team,\n is Januar
	Thu, 13 Dec		\n\tAt your			\

Pre-trained BERT

We make use of bert-extractive-summarizer, a pre-trained BERT python package, finetuned for extractive summarisation. This tool utilizes the HuggingFace library. It works by first embedding the sentences, then running a clustering algorithm, finding the sentences that are closest to the cluster's centroids.

```
# Setting up BERT model
bert_model = Summarizer()

# Defining function to apply to emails
def bert_summary(text):
    bert_text = ''.join(bert_model(text))
    return bert_text

# Applying function to emails
emails_labelled["Bert"] = emails_labelled["Bert"].apply(lambda x: bert_summary(x
```

Pre-trained GPT-2

Within the TransformerSummarizer() wrapper function, we simply configure 'GPT2' as paramenter.

```
# Setting up GPT-2 model
GPT2_model = TransformerSummarizer(transformer_type="GPT2",transformer_model_key

# Defining function to apply to emails
def GPT2_summary(text):
    GPT2_text = ''.join(GPT2_model(text, min_length=50))
    return GPT2_text

# Applying function to emails
emails_labelled["GPT2"] = emails_labelled["GPT2"].apply(lambda x: GPT2_summary(x
```

Data Cleaning for Text Rank

With the pre-trained BERT and GPT-2 models, no major data cleaning and pre-processing was necessary. Thus these steps are performed now, to prepare the data for further extraction summarisation models, namely Text Rank.

```
# Defining function to remove special characters
def clean(email):
    return re.sub(r"^.*:\s.*|^-.*\n?.*-$|\n|^>", '', email, 0, re.MULTILINE)

# Defining function to remove stop words
def removeStopwords(sentence):
    newSentence = " ".join([i for i in sentence if i not in stopWords])
    return newSentence

# Defining function to present each sentence separately
def prettySentences(sentence):
    for s in sentence:
        print(s)
        print()


# Applying functions defined above to data and appending it to dataframe

# 1. Applying clean function
mydata = []
for sentence in emails_labelled.body:
    mydata.append(clean(sentence))

# 2. Applying Stopwords and prettySentences functions
sentences = []
for sentence in mydata:
    sentences.append(sent_tokenize(sentence))
```

```
# Using 100 dimension version of Glove Embedding
wordEmbeddings = {}
with open ("/project/glove.6B.100d.txt", encoding = 'utf - 8') as f:
    for line in f:
        values = line.split()
        key = values[0]
        wordEmbeddings[key] = np.asarray(values[1:], dtype = 'float32')

cleanSentences = []

# sentence formatting and removal of stop words
for email in sentences:
        email = [re.sub(r"[^a-zA-Z]", " ", s, 0, re.MULTILINE) for s in email]
        email = [re.sub(r"\s+", " ", s, 0, re.MULTILINE) for s in email]
        cleanSentences.append([s.lower() for s in email])

stopWords = stopwords.words('english')

for i in range(len(cleanSentences)):
    cleanSentences[i] = [removeStopwords(r.split()) for r in cleanSentences[i]]
```

Text Rank

TextRank is an extractive and unsupervised text summarization technique. It ranks sentences along their importance, by assigning them a similiarity score, which are then stored in a square matrix. Put differently, a vector representation is established for each sentence. Similarities between sentence vectors are then calculated and stored in a matrix. The similarity matrix is converted into a graph, with sentences as vertices and similarity scores as edges, for sentence ranking. A certain number of top-ranked sentences form the final summary.

```
# Creating vector representations
sentenceVectors = []
for email in cleanSentences:
    temp = []
    for s in email:
        if len(s) != 0:
            v = sum([ wordEmbeddings.get(w, np.zeros((100,))) for w in s.split()
        else:
            v = np.zeros((100,))
        temp.append(v)
    sentenceVectors.append(temp)
```

```
# Calculating similarity stores and storing them to matrix
similarityMatrix = []
for i in range(len(cleanSentences)):
    email = cleanSentences[i]
    temp = np.zeros((len(email), len(email)))
    j_range = temp.shape[0]
    k_range = temp.shape[1]
    for j in range(j_range):
        for k in range(k_range):
            if j != k:
                temp[j][k] = cosine_similarity(sentenceVectors[i][j].reshape(1,
                                                                  sentenceVectors[i][
    similarityMatrix.append(temp)


# Similarity matrix converted into a graph, with sentences as vertices and simil
scores = []
for i in similarityMatrix:
    nxGraph= nx.from_numpy_array(i)
    scores.append(nx.pagerank_numpy(nxGraph))


# Ranking top sentences
rankedSentences = []

for i in range(len(scores)):
    rankedSentences.append(sorted(((scores[i][j],s) for j,s in enumerate(sentenc


# Out of the number of sentences extracted, we only want to take 2 sentence from
textrank_summarised = []
for i in range(0,95):
    sentence = rankedSentences[i]
    sentence_1 = sentence[0]
    if len(rankedSentences[i]) > 1:
            sentence_2 = rankedSentences[i]
            sentence_2 = sentence_2[1]
    else:
        sentence_2 = ("0","0")
    textrank_summarised.append([sentence_1,sentence_2])
    textrank_summarised

textrank_table = pd.DataFrame(textrank_summarised)
```

Rouge Score

ROUGE is a set of metrics for evaluating automatic summarization of texts as well as machine translations. We use it by comparing the automatically produced summaries from BERT and GPT-2 against our set manually produced reference summaries. Recall (in the context of ROUGE) refers to how much of the reference summary the system summary is recovering or capturing (freeCodeCamp,2017).

Due to the shortness of our emails, we make use of ROUGE-1, which refers to the overlap of unigrams, single words, between the system summary and reference summary.

In the context of ROUGE, while precision measures how much of the system summary was needed, recall refers to how much of the reference summary the system summary is recovering or capturing.

```
# Setting up ROUGE scorer
scorer = rouge_scorer.RougeScorer(['rouge1'], use_stemmer=True)

# Defining function to implement on entire body
def rouge_score(text,text2):
    GPT_scores = scorer.score(text1,text2)
    rouge_s = GPT_scores.get("rouge1")
    return rouge_s
```

```
# Applying ROUGE score to BERT
bert_score_list = []
for x in range(0,len(emails_labelled)):
    bert_scores = scorer.score(emails_labelled["Bert"][x],emails_labelled["summa
    rouge_s = bert_scores.get("rouge1")
    bert_score_list.append(rouge_s)


# create a data frame with all the rouge score generated
bert_score_list = pd.DataFrame(bert_score_list)
bert_score_list = bert_score_list.rename(columns = {"precision":"bert precision"

bert_score_list = bert_score_list.join(emails_labelled["body"])
bert_score_list = bert_score_list.join(emails_labelled["Bert"])

bert_score_list = bert_score_list[["body","Bert","bert precision","bert recall",
bert_score_list.head(10)
```

	body	Bert	bert precision	bert recall	bert fmeasure
0	The Global Gas Pipeline group is looking to tr...	The Global Gas Pipeline group is looking to tr...	0.583333	0.482759	0.528302
1	IEP Team,\nAttached is a revised January CalJo...	IEP Team,\nAttached is a revised January CalJo...	1.000000	0.206897	0.342857
2	\n\tAt your earliest convenience, please send ...	At your earliest convenience, please send me c...	1.000000	0.333333	0.500000
3	I ran a redline from the last version I had el...	I ran a redline from the last version I had el...	1.000000	0.458333	0.628571
4	---------------------- Forwarded by Daren J Fa...	---------------------- Forwarded by Daren J Fa...	1.000000	0.120690	0.215385
5	Hey John & Angie, Have	Hey John & Angie, Have	0.000000	0.000000	0.000000

```
# count the number of GPT2 summary with more than 0.6 fmeasure
bert_count = bert_score_list[bert_score_list["bert fmeasure"]>0.6].count()
bert_accept = bert_count["bert fmeasure"]
bert_reject = len(emails_labelled) - bert_count["bert fmeasure"]
```

```
# Applying ROUGE score to GPT-2
GPT2_score_list = []
for x in range(0,len(emails_labelled)):
    GPT_scores = scorer.score(emails_labelled["GPT2"][x],emails_labelled["summar
    rouge_s = GPT_scores.get("rouge1")
    GPT2_score_list.append(rouge_s)

# create a data frame with all the rouge score generated
GPT2_score_list = pd.DataFrame(GPT2_score_list)
GPT2_score_list = GPT2_score_list.rename(columns = {"precision":"GPT2 precision"

GPT2_score_list = GPT2_score_list.join(emails_labelled["body"])
GPT2_score_list = GPT2_score_list.join(emails_labelled["GPT2"])
GPT2_score_list = GPT2_score_list[["body","GPT2","GPT2 precision","GPT2 recall",
GPT2_score_list.head(10)
```

	body	GPT2	GPT2 precision	GPT2 recall	GPT2 fmeasure
0	The Global Gas Pipeline group is looking to tr...	The Global Gas Pipeline group is looking to tr...	0.583333	0.500000	0.538462
1	IEP Team,\nAttached is a revised January CalJo...	IEP Team,\nAttached is a revised January CalJo...	1.000000	0.214286	0.352941
2	\n\tAt your earliest convenience, please send ...	At your earliest convenience, please send me c...	1.000000	0.333333	0.500000
3	I ran a redline from the last version I had el...	I ran a redline from the last version I had el...	1.000000	0.458333	0.628571
4	---------------------- Forwarded by Daren J Fa...	---------------------- Forwarded by Daren J Fa...	1.000000	0.120690	0.215385
5	Hey John & Angie, Have	Hey John & Angie, Have	0.000000	0.000000	0.000000

```
# count the number of GPT2 summary with more than 0.6 fmeasure
GPT2_count = GPT2_score_list[GPT2_score_list["GPT2 fmeasure"]>0.6].count()
GPT2_accept = GPT2_count["GPT2 fmeasure"]
GPT2_reject = len(emails_labelled) - GPT2_count["GPT2 fmeasure"]
```

```
# Applying ROUGE score to TextRank
# Since sentence is separated into numbers of list, we need to combine the 2 sen
textrank_table["textrank_summary"] = ""
for i in range(0,len(textrank_table)):
    list_sentence = []
    x = textrank_table[0][i][1]
    y = textrank_table[1][i][1]
    sentence_combined = x + y
    list_sentence.append(sentence_combined)
    textrank_table["textrank_summary"][i]=sentence_combined

#text rank rouge score
textrank_score_list = []
for x in range(0,len(emails_labelled)):
    textrank_scores = scorer.score(textrank_table["textrank_summary"][x],emails_
    rouge_s = textrank_scores.get("rouge1")
    textrank_score_list.append(rouge_s)

textrank_score_list = pd.DataFrame(textrank_score_list)
textrank_score_list = textrank_score_list.rename(columns = {"precision":"textran
textrank_score_list = textrank_score_list.join(emails_labelled["body"])
textrank_score_list = textrank_score_list.join(textrank_table["textrank_summary"
textrank_score_list = textrank_score_list[["body","textrank_summary","textrank p
textrank_score_list.head(10)
```

	body	textrank_summary	textrank precision	textrank recall	textrank fmeasure
0	The Global Gas Pipeline group is looking to tr...	The Initial products are expected to be day ah...	0.583333	0.341463	0.430769
1	IEP Team,\nAttached is a revised January CalJo...	However, since the deadline has not changed (a...	0.166667	0.033333	0.055556
2	\n\tAt your earliest convenience, please send ...	\tAt your earliest convenience, please send me...	1.000000	0.312500	0.476190
3	I ran a redline from the last version I had el...	I've forwarded the current version to Herman f...	1.000000	0.407407	0.578947
4	--------------------- Forwarded by Daren J Fa...	Thanks............Mary Solmonson12/15/99 06:55...	0.857143	0.133333	0.230769

```
# count the number of text rank summary with more than 0.5 fmeasure
textrank_count = textrank_score_list[textrank_score_list["textrank fmeasure"]>0.
textrank_accept = textrank_count["textrank fmeasure"]
textrank_reject = len(emails_labelled) - textrank_count["textrank fmeasure"]
```

Extractive Summarisation Overview and Limitation

```
print("Average GPT-2 precision: {:.2f}%".format((bert_score_list["bert precision
print("Average GPT-2 recall: {:.2f}%".format((bert_score_list["bert recall"].mea
print("Average GPT-2 f-measure: {:.2f}%".format((bert_score_list["bert fmeasure"
print("\n")
print("Average GPT-2 precision: {:.2f}%".format((GPT2_score_list["GPT2 precision
print("Average GPT-2 recall: {:.2f}%".format((GPT2_score_list["GPT2 recall"].mea
print("Average GPT-2 f-measure: {:.2f}%".format((GPT2_score_list["GPT2 fmeasure"
print("\n")
print("Average TextRank precision: {:.2f}%".format((textrank_score_list["textran
print("Average TextRank recall: {:.2f}%".format((textrank_score_list["textrank r
print("Average TextRank f-measure: {:.2f}%".format((textrank_score_list["textran
```

```
Average GPT-2 precision: 72.48%
Average GPT-2 recall: 26.11%
Average GPT-2 f-measure: 32.77%


Average GPT-2 precision: 71.50%
Average GPT-2 recall: 26.69%
Average GPT-2 f-measure: 33.31%


Average TextRank precision: 64.81%
Average TextRank recall: 20.84%
Average TextRank f-measure: 27.77%
```

In the outputs above, precision is quite high for all models, specifically BERT and GPT-2. Recall and overall accuracy, f-measure, show models are not doing a sufficient job in summarization. As pre-trained models such as the bert-extractive-summarizer do not allow for much finetuning, we recognise the limitations of the pre-trained extractive models. However, the models' lack of overlap with the manually generated summaries, does not imply the summaries generated are inferior.

```
summary_table = pd.DataFrame()
bert_list = [bert_accept, bert_reject]
GPT2_list = [GPT2_accept, GPT2_reject]
textrank_list = [textrank_accept, textrank_reject]


summary_table["Bert"] = ""
summary_table["GPT2"] = ""
summary_table["Textrank"] = ""
```

```
summary_table["Bert"] = bert_list
summary_table["GPT2"] = GPT2_list
summary_table["Textrank"] = textrank_list


summary_table.rename(index = {0:"Overlap > 60%", 1:"Overlap < 60%"})
```

	Bert	GPT2	Textrank
Overlap > 60%	15	18	12
Overlap < 60%	80	77	83

We can see that GPT-2 is doing the best job in summarising, with regards to overlap with the manually generated summaries.
Looking at the scorings with regards to overlap only, however, fails to take into account the length of summaries, as for example, including the entire email body could simply result in a precision score of 1.

Hence, we next compare the original body of the first email with each summary generated by the respective model.

```
print("Original Sentence: {}".format(emails_labelled["body"][0]))
print("\n")
print("Manual Summary: {}".format(emails_labelled["summary_label"][0]))
print("\n")
print("BERT Summary: {}".format(emails_labelled["Bert"][0]))
print("\n")
print("GPT2 Summary: {}".format(emails_labelled["GPT2"][0]))
print("\n")
print("Textrank Summary: {}".format(textrank_table["textrank_summary"][0]))
```

```
Original Sentence: The Global Gas Pipeline group is looking to trade gas tr
EnronOnline.  The Initial products are expected to be day ahead and rest of
the month transportation. Houston PipeLine and Northern Natural Gas Pipelin
will be the initial pipelines posting bids and offers. The expected date of
launch  is 8/23/00. We do not have the GTC or Product descriptions yet.  I
will forward those to you'll as soon as I have them. In the meantime if
you'll could please let me know if there are any legal, credit , tax, risk,
and other concerns we should be addressing. Thanks a lot.

Savita


Manual Summary: Global Gas Pipeline group is looking to trade gas transport
EnronOnline, any legal, credit , tax, risk,
and other concerns we should be addressing


BERT Summary: The Global Gas Pipeline group is looking to trade gas transpo
EnronOnline. Houston PipeLine and Northern Natural Gas Pipeline
will be the initial pipelines posting bids and offers.


GPT2 Summary: The Global Gas Pipeline group is looking to trade gas transpo
EnronOnline. The Initial products are expected to be day ahead and rest of
the month transportation.


Textrank Summary: The Initial products are expected to be day ahead and res
```

Upon inspection of the summary example above, we recognise that despite the ROUGE score being rather low, the summarizations extracted seem adequate in terms of context and length. While BERT and GPT-2 do a good job at giving context, TextRank captures the key action point the message conveys, by specifying that input on legal, credit, tax, risk and other concerns is required.

Looking at the performance of each pre-trained model and TextRank, ROUGE score was able to provide insight into some of the limitations of extractive summarisation. All models are able to extract information relevant to the manual summary (high precision). However, the majority of the sentences summarised are longer in comparison to the manual summarised sentences. Hence, increasing length of sentence will provide a better score in precision, yet this lowers the recall score. Despite the precision-and-recall-trade-off, we believe that improving on the recall score would be an outlook for future modelling. Not only this, having a lower number of sentences will prove efficient in Enron's case, to speed up communication. Another improvement for the pre-trained model would be recognising the irrelevance of shorter sentences or excertps such as "Thanks a lot" from the example. These phrases not only increase sentence length, but also reduce summarisation accuracy. To solve this problem, we should look into abstractive summarisation, as this approach would allow for re-phrasing of email content, further improving the performance of our summarization efforts.

Summary

This project aims to analyse natural language patterns in Emails by classifying the sentiment and generating a summary using original emails by the Enron Corporation. As per the original proposal submitted, different methods, deploying both model architectures designed from scratch, as well as pretrained models, were used to complete this task and achieve better performance. Labeling was performed on the originally unlabeled dataset, exploiting both human annotation and weak labeling using Snorkel. Preparation and tokenisation partly varied between models due to the model requirements but were kept comparable to be able to evaluate the performance across models.

Sentiment Classification

As a baseline model, an RNN with LSTM cells and dropout was compiled, which achieved a good performance on the validation set after some fine-tuning. The error analysis shows that the model was relatively balanced in false positive and false negative predictions, while the biggest concern for improved model performance is the implementation of early stopping and the number of epochs as to not overfit the data with this rather complex model.

A pretrained BERT transformer model was then fit on the email data, surpassing the RNN performance. The errors here were also balanced between false positive and false negative predictions, as visible in the confusion matrix. Another risk is the tendency to overfit if too many epochs of training are used. Thus, the models behave very similarly in the resulting predictions, although the BERT pretrained model is preferred for future use due to its performance on validation data.

The evaluation of both sentiment models was done using the previously generated Snorkel labels, of which only 5000 instances were used due to model size and training on limited infrastructure.

Text Summarisation

This section deviated from the original plan outlined in our proposal to fine-tune a pre-trained transformer model. The reasons for this were the small number of labels we could generate with human annotation (approximately 100), as well as the computationally expensive fine-tuning, while pre-trained models often deliver good performance out-of-the-box.

Extractive summarisation is used, deploying models from the HuggingFace library: BERT, TextRank and GPT-2. The performance of the models was compared using the ROUGE-1 score, which assesses the 1-gram-overlap between the manually generated summary and the respective model performance. The results leave room for improvement, although GPT-2 performed very well, especially given the out-of-the-box application. Manually comparing the summaries shows that predictions were often long and included information that was unnecessary. However, all models succeeded in capturing the core information of each email, implying there is not one right way of summarising. Future finetuning could make summaries more concise, especially for short emails. It is advised to look into abstractive summarisation next, to overcome the limitations of extractive summarisation.

Outlook

The next steps in bringing the proposed system into production would be to develop general categories for forwarding the emails, improving on the existing model errors as outlined above in the individual error analyses and the summary, as well as deploying the model for use as a web or mobile application. This could become a valuable tool to make corporate communication more efficient and should be explored in further research.

References

Appleyard, J., Kocisky, T. & Blunsom, P., 2016. Optimizing Performance of Recurrent Neural Networks on GPUs. [Online] Available at: https://arxiv.org/abs/1604.01946 [Accessed 20 March 2021].

Ganesan, K., 2017. An intro to ROUGE, and how to use it to evaluate summaries. [Online] Available at: https://www.freecodecamp.org/news/what-is-rouge-and-how-it-works-for-evaluation-of-summaries-e059fb8ac840/#:~:text=ROUGE%20stands%20for%20Recall%2DOriented,as%20well%20as%20machine%20translations.&text=To%20get%20a%20good%20quantitative,and%20recall%20usi [Accessed 21 March 2021].

Devlin, J. & Chang, M. & Lee, K. & Toutanova, K., 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. [Online] Available at: https://arxiv.org/abs/1810.04805 [Accessed 20 March 2021].

McKinsey Global Institute, 2012. The social economy: Unlocking value and productivity through social technologies. [Online] Available at: https://www.mckinsey.com/industries/technology-media-and-telecommunications/our-insights/the-social-economy [Accessed 23 March 2021].

Cohen, W. W., 2015. Enron Email Dataset. [Online] Available at: https://www.cs.cmu.edu/~./enron/ [Accessed 15 March 2021].

Hu, M. & Liu, B., 2004. Opinion Lexicon: Positive. [Online] Available at: https://gist.githubusercontent.com/mkulakowski2/4289437/raw/1bb4d7f9ee82150f339f09b5b1a0e6823d633958/positive-words.txt [Accessed 15 March 2021].

Liu, B., Hu, M. & Cheng, J., 2005. Opinion Lexicon: Negative. [Online] Available at: https://gist.githubusercontent.com/mkulakowski2/4289441/raw/dad8b64b307cd6df8068a379079becbb3f91101a/negative-words.txt [Accessed 15 March 2021].

Code References

Bartolo, M., 2021. Advanced RNNs. Lecture Slides [Online]

Bartolo, M., 2021. The Transformer. Lecture Slides [Online]

Cheng, R., 2021. BERT Text Classification Using Pytorch. [online] Medium. Available at: https://towardsdatascience.com/bert-text-classification-using-pytorch-723dfb8b6b5b [Accessed 25 March 2021].

Joshi, P., 2021. Transfer Learning NLP|Fine Tune Bert For Text Classification. [online] Analytics Vidhya. Available at: https://www.analyticsvidhya.com/blog/2020/07/transfer-learning-for-nlp-fine-tuning-bert-for-text-classification/ [Accessed 25 March 2021].

Tran, C., 2021. Tutorial: Fine-tuning BERT for Sentiment Analysis - by Skim AI. [online] Skim AI. Available at: https://skimai.com/fine-tuning-bert-for-sentiment-analysis/ [Accessed 25 March 2021].

Presentation Video and Proposal

Since the upload size of our presentation video was too large, kindly use the following link to a Google Drive containing the video in MP4 format. The total video length is 3:16 min.

https://drive.google.com/file/d/1A40Dgti2sDmDOr_Re87HYp0UnzmR3gCb/view?usp=sharing

The presentation used can be found here:

https://drive.google.com/file/d/1CtSdj_l_SfSniFG7fQPMz1iL3OBa3UyZ/view?usp=sharing

For reference, our original proposal is also available on Google Drive via the following link:
https://drive.google.com/file/d/1RLiRBTHfOYdfpFTiE435fVsYgSAfPnwY/view?usp=sharing