EECS 487 Final Project: Stance Detection in Satire

Anurag Renduchintala, Yoojin Bae, Karl Yan

Run the following cell to mount the Google Drive

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

Run the following cell to import (and install) necessary modules

```
!pip install portalocker
!pip install transformers
```

```
Defaulting to user installation because normal site-packages is not
writeable
Requirement already satisfied: portalocker in c:\users\12484\appdata\
roaming\python\python310\site-packages (2.8.2)
Requirement already satisfied: pywin32>=226 in c:\users\12484\appdata\
roaming\python\python310\site-packages (from portalocker) (306)
Defaulting to user installation because normal site-packages is not
writeable
Requirement already satisfied: transformers in c:\users\12484\appdata\
roaming\python\python310\site-packages (4.35.2)
Requirement already satisfied: filelock in c:\users\12484\appdata\
roaming\python\python310\site-packages (from transformers) (3.13.1)
Requirement already satisfied: huggingface-hub<1.0,>=0.16.4 in c:\
users\12484\appdata\roaming\python\python310\site-packages (from
transformers) (0.19.4)
Requirement already satisfied: numpy>=1.17 in c:\users\12484\appdata\
roaming\python\python310\site-packages (from transformers) (1.26.2)
Requirement already satisfied: packaging>=20.0 in c:\users\12484\
appdata\roaming\python\python310\site-packages (from transformers)
(23.2)
Requirement already satisfied: pyyaml>=5.1 in c:\users\12484\appdata\
roaming\python\python310\site-packages (from transformers) (6.0.1)
Requirement already satisfied: regex!=2019.12.17 in c:\users\12484\
appdata\roaming\python\python310\site-packages (from transformers)
(2023.10.3)
Requirement already satisfied: requests in c:\users\12484\appdata\
roaming\python\python310\site-packages (from transformers) (2.31.0)
Requirement already satisfied: tokenizers<0.19,>=0.14 in c:\users\
12484\appdata\roaming\python\python310\site-packages (from
transformers) (0.15.0)
Requirement already satisfied: safetensors>=0.3.1 in c:\users\12484\
```

```
appdata\roaming\python\python310\site-packages (from transformers)
(0.4.1)
Requirement already satisfied: tqdm>=4.27 in c:\users\12484\appdata\
roaming\python\python310\site-packages (from transformers) (4.66.1)
Requirement already satisfied: fsspec>=2023.5.0 in c:\users\12484\
appdata\roaming\python\python310\site-packages (from huggingface-
hub<1.0,>=0.16.4->transformers) (2023.12.0)
Requirement already satisfied: typing-extensions>=3.7.4.3 in c:\users\
12484\appdata\roaming\python\python310\site-packages (from
huggingface-hub<1.0,>=0.16.4->transformers) (4.8.0)
Requirement already satisfied: colorama in c:\users\12484\appdata\
roaming\python\python310\site-packages (from tqdm>=4.27->transformers)
(0.4.6)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\
12484\appdata\roaming\python\python310\site-packages (from requests-
>transformers) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in c:\users\12484\appdata\
roaming\python\python310\site-packages (from requests->transformers)
(3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\12484\
appdata\roaming\python\python310\site-packages (from requests-
>transformers) (2.1.0)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\12484\
appdata\roaming\python\python310\site-packages (from requests-
>transformers) (2023.11.17)
```

Run the following if using Google Drive.

```
import os
import sys

'''# TODO: Change this to the path to your homework folder
GOOGLE_DRIVE_PATH = '/content/drive/MyDrive/EECS 487/Homework_3'
print(os.listdir(GOOGLE_DRIVE_PATH))
os.chdir(GOOGLE_DRIVE_PATH)'''

"# TODO: Change this to the path to your homework folder\
nGOOGLE_DRIVE_PATH = '/content/drive/MyDrive/EECS 487/Homework_3'\
nprint(os.listdir(GOOGLE_DRIVE_PATH))\nos.chdir(GOOGLE_DRIVE_PATH)"

import pandas as pd
import matplotlib.pyplot as plt
```

Colab GPU Resources

Check if GPU resources are available. If device = 'cpu', in the toolbar, click Runtime -> Change runtime type -> select GPU as the hardware accelerator.

Important:

Google Colab imposes a **dynamic GPU usage limit** that depends on how much/long you use Colab. This is to keep Colab free for everyone. You can read about it here. That being said, you should be able to complete this assignment without reaching your usage limit. You are **not** expected to spend your own money on Colab's paid GPU resources. In the event that you have run out of GPU resources, you would have to wait for resources or use a different Google account.

Here are some tips to conserve your GPU usage:

- Change your runtime to GPU only when are working on parts that require GPU
- When spending long intervals on coding/taking a break, remember to disconnect your runtime.

```
import torch

device = 'cuda' if torch.cuda.is_available() else 'cpu'
print(device)
```

```
cuda
```

```
# Install required packages
import nltk
nltk.download('punkt')
nltk.download('stopwords')
!pip install readability
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\12484\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\12484\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!

Defaulting to user installation because normal site-packages is not
writeable
Requirement already satisfied: readability in c:\users\12484\appdata\
roaming\python\python310\site-packages (0.3.1)
```

Run the following cell to load the autoreload extension so that functions in python files will be re-imported into the notebook every time we run them. We also need to import all necessary packages.

```
%load_ext autoreload
%autoreload 2

import os
import json

import numpy as np
from torch.utils.data import DataLoader
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

Main Task: Satire Detection

Prepare the data by importing the necessary modules. Get the compiled final data frame. Tokenize all reviews (lowercasing will be done later).

```
# clean the raw datasets, putting together only the important columns
(text and satire)
from prepare_data import ALL_DATA
# make a copy of this df. We don't want to modify the actual data.
satire_data = ALL_DATA.copy()
# lowercase all the headlines
satire_data['text'] = satire_data['text'].str.lower()
# tokenize all the text
satire_data['tokenized_text'] = satire_data['text'].apply(lambda x:
nltk.word_tokenize(x))
# verify that our data is balanced
print(len(satire_data[satire_data["satire"] == 0]))
print(len(satire_data[satire_data["satire"] == 1]))
```

```
2427
2427
```

Do Train, Test, Split; split the given data into training and testing sets.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test =
train_test_split(satire_data["text"], satire_data["satire"],
stratify=satire_data["satire"])
print(X_train.head(9))
print(len(X_train))
print(len(X_test))
```

```
2063    3 men indicted on murder charges in killing of...
925     poll finds more people would trust fauci if he...
2046    justice dept. announces first felony charges i...
2185    fully grown man much happier with new sonic th...
1877    johnson announces fiancã© pregnant with first ...
2379                                        cool lifehack!
1822    giancarlo esposito on gus fring, spike lee and...
2613    man ‘prorogues’ relationship with girlfriend w...
1777    is it safe to try on clothes at stores during ...
Name: text, dtype: object
3640
1214
```

Get the BERT model preprocessor and encoder. Import necessary packages.

```
import tensorflow_hub as hub
import tensorflow_text as text
import tensorflow as tf

preprocess_url =
'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3'
encoder_url = 'https://tfhub.dev/tensorflow/bert_en_uncased_L-12_H-
768_A-12/4'

bert_preprocess = hub.KerasLayer(preprocess_url)
bert_encoder = hub.KerasLayer(encoder_url)
```

Now create BERT and Neural Network Layers, and then, create the final model.

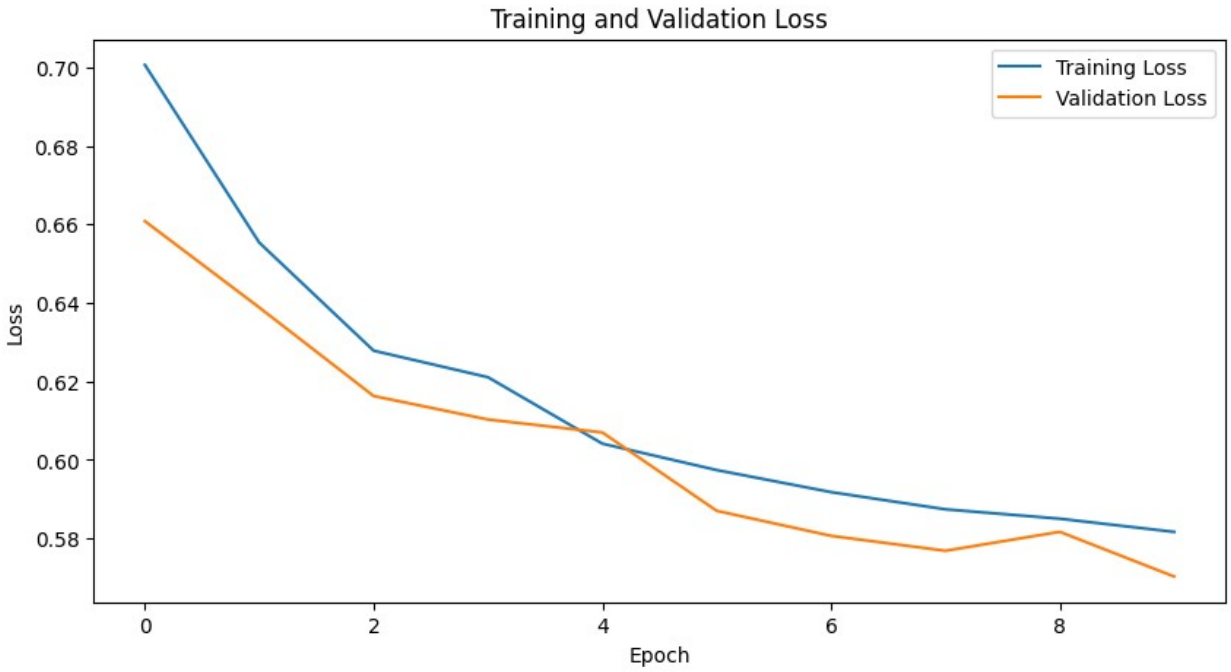
```
# Initialize some hyperparameters first. Mess with these to see what
you get.
learning_rate = 1e-3
weight_decay = 0.01
batch_size = 64
reg = "l2"

# BERT Layers
input_ = tf.keras.layers.Input(shape=(), dtype=tf.string, name='text')
preprocessed = bert_preprocess(input_)
output = bert_encoder(preprocessed)

# NN Layers
MODEL = tf.keras.Sequential([
    tf.keras.layers.Dropout(0.1, name='dropout',
input_shape=(output['pooled_output'].shape[1],)),
    tf.keras.layers.Dense(1, activation='sigmoid', name='output',
kernel_regularizer=tf.keras.regularizers.l2(l2=weight_decay) if
reg=="l2" else tf.keras.regularizers.l1(l1=weight_decay))
])

# Build the final model
model = tf.keras.Model(inputs=input_,
outputs=MODEL(output['pooled_output']))

# Compile the model
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=learnin
g_rate), loss="binary_crossentropy",
metrics=[tf.keras.metrics.BinaryAccuracy(name='accuracy'),
tf.keras.metrics.Precision(name="precision"),

tf.keras.metrics.Recall(name="recall")])
```

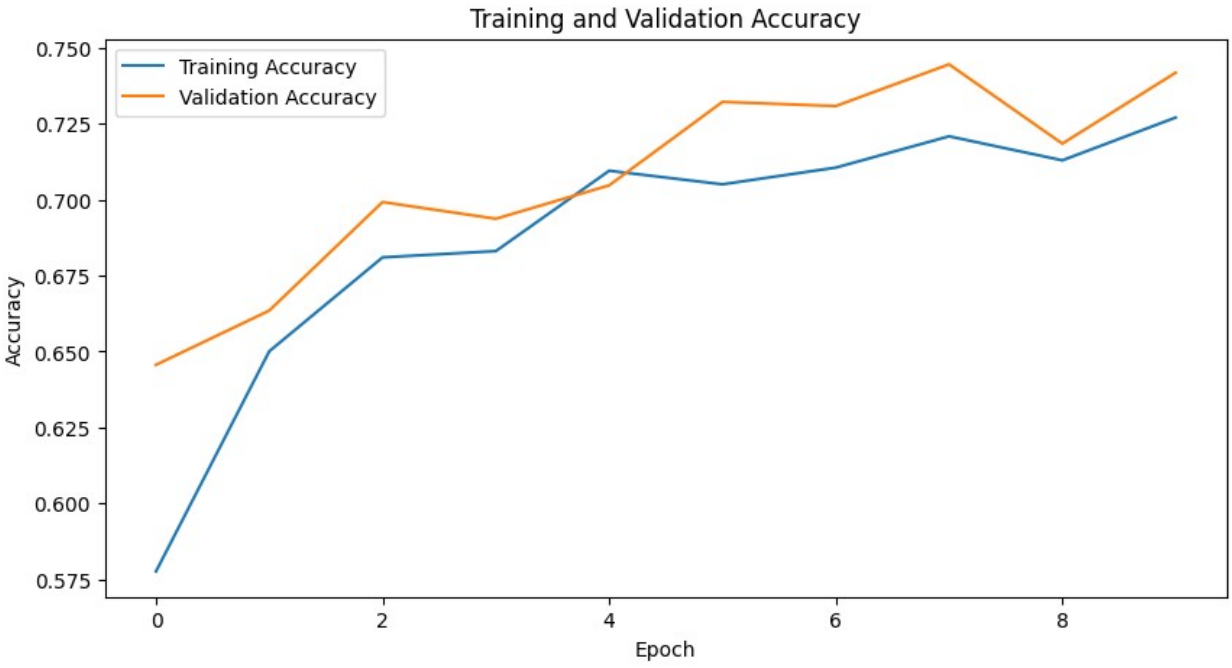
Train, Evaluate, Make Predictions.

```
# Train model. Graph validation loss and accuracy by epoch.
history = model.fit(X_train, y_train, epochs=10,
batch_size=batch_size, validation_split=.2)
# Plot training and validation loss
plt.figure(figsize=(10, 5))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()

# Plot training and validation accuracy
plt.figure(figsize=(10, 5))
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Evaluate model
test_loss, test_accuracy, test_precision, test_recall =
model.evaluate(X_test, y_test)
print(f'Test Loss: {test_loss}, Test Accuracy: {test_accuracy}')

# Make some predictions on new data (optional)
# TODO: Write some code for this (later on)
```

```
Epoch 1/10
46/46 [==============================] - 232s 5s/step - loss: 0.7006 -
accuracy: 0.5776 - precision: 0.5835 - recall: 0.5385 - val_loss:
0.6608 - val_accuracy: 0.6456 - val_precision: 0.6570 - val_recall:
0.6175
Epoch 2/10
46/46 [==============================] - 265s 6s/step - loss: 0.6554 -
accuracy: 0.6501 - precision: 0.6481 - recall: 0.6547 - val_loss:
0.6388 - val_accuracy: 0.6635 - val_precision: 0.6157 - val_recall:
0.8798
Epoch 3/10
46/46 [==============================] - 278s 6s/step - loss: 0.6278 -
accuracy: 0.6810 - precision: 0.6728 - recall: 0.7029 - val_loss:
0.6163 - val_accuracy: 0.6992 - val_precision: 0.6574 - val_recall:
0.8388
Epoch 4/10
46/46 [==============================] - 261s 6s/step - loss: 0.6211 -
accuracy: 0.6830 - precision: 0.6793 - recall: 0.6919 - val_loss:
0.6103 - val_accuracy: 0.6937 - val_precision: 0.6480 - val_recall:
```

```
0.8552
Epoch 5/10
46/46 [==============================] - 267s 6s/step - loss: 0.6042 -
accuracy: 0.7095 - precision: 0.7038 - recall: 0.7221 - val_loss:
0.6070 - val_accuracy: 0.7047 - val_precision: 0.7631 - val_recall:
0.5984
Epoch 6/10
46/46 [==============================] - 273s 6s/step - loss: 0.5974 -
accuracy: 0.7050 - precision: 0.7065 - recall: 0.7001 - val_loss:
0.5871 - val_accuracy: 0.7321 - val_precision: 0.7050 - val_recall:
0.8033
Epoch 7/10
46/46 [==============================] - 304s 7s/step - loss: 0.5918 -
accuracy: 0.7105 - precision: 0.7006 - recall: 0.7338 - val_loss:
0.5807 - val_accuracy: 0.7308 - val_precision: 0.7237 - val_recall:
0.7514
Epoch 8/10
46/46 [==============================] - 274s 6s/step - loss: 0.5874 -
accuracy: 0.7208 - precision: 0.7194 - recall: 0.7228 - val_loss:
0.5769 - val_accuracy: 0.7445 - val_precision: 0.7239 - val_recall:
0.7951
Epoch 9/10
46/46 [==============================] - 274s 6s/step - loss: 0.5850 -
accuracy: 0.7129 - precision: 0.7108 - recall: 0.7166 - val_loss:
0.5817 - val_accuracy: 0.7184 - val_precision: 0.6793 - val_recall:
0.8333
Epoch 10/10
46/46 [==============================] - 278s 6s/step - loss: 0.5817 -
accuracy: 0.7270 - precision: 0.7246 - recall: 0.7311 - val_loss:
0.5703 - val_accuracy: 0.7418 - val_precision: 0.7445 - val_recall:
0.7404
```






```
38/38 [==============================] - 78s 2s/step - loss: 0.5701 -
accuracy: 0.7455 - precision: 0.7551 - recall: 0.7265
Test Loss: 0.5700647830963135, Test Accuracy: 0.7454695105552673
```

Now, we will predict on unseen test data.

```
y_pred = model.predict(X_test)
y_pred = y_pred.flatten()
```

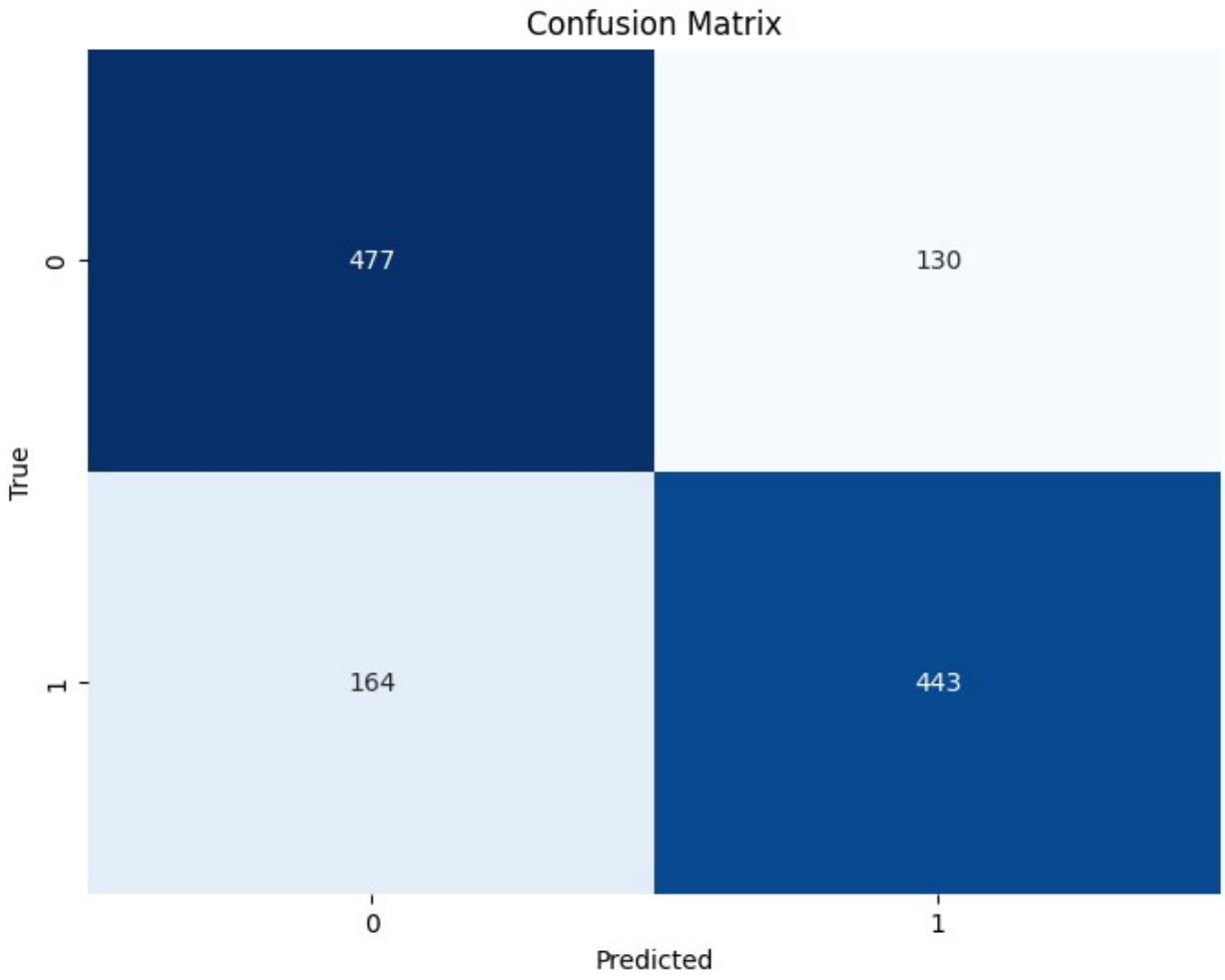
```
# our y-pred values are sigmoid values between 0 and 1. So, we need to convert them.
y_pred = [1 if score > 0.5 else 0 for score in y_pred]
y_test_ = list(y_test)
```

```
38/38 [==============================] - 64s 2s/step
```

For visualization purposes, we calculate confusion matrix to assess our model.

```
import seaborn as sns
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.metrics import classification_report

# Calculate confusion matrix and accuracy
cm = confusion_matrix(y_test_, y_pred)
acc = accuracy_score(y_test_, y_pred)
print("Accuracy: ", acc)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", cbar=False)
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()

# Print classification report
print("Classification Report:")
print(classification_report(y_test_, y_pred))
```

```
Accuracy:  0.7578253706754531
```




```
Classification Report:
              precision    recall  f1-score   support

           0       0.74      0.79      0.76       607
           1       0.77      0.73      0.75       607

    accuracy                           0.76      1214
   macro avg       0.76      0.76      0.76      1214
weighted avg       0.76      0.76      0.76      1214
```