READ ME

Inspiration: https://towardsdatascience.com/capturing-context-in-emotion-ai-innovations-in-multimodal-video-sentiment-analysis-65e128ad8a1a

DATA Set: https://www.kaggle.com/datasets/zaber666/meld-dataset

- combined all datasets listed for our in models get_train.ipynb
- train_set.csv, contains all the text and audio data to build models

models.ipynb

- Text NLP preprocessing
  - lemmatization, removal of stopwords, lowercase text
- Audio feature embeddings from AST Transformer and audio resampler
- Bert feature Embeddings for text data
- Models Used
  - Logistic Regression
  - XG Boost
  - Bidirectional LSTM with stacking
  - Feedforward ANN
  - t-SNE and PCA to visualize highdimensional data boundaries

Used google colab+ to deal with computational constraints

Import Google Drive and Train Data

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

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

```
import re
import keras
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import torch
import librosa
import seaborn as sns

import transformers as ppb
from transformers import BertTokenizer
from transformers import ASTFeatureExtractor
from torchaudio.transforms import Resample

from keras.callbacks import EarlyStopping

import xgboost as xgb
from xgboost import XGBClassifier

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report,confusion_matrix
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import LabelEncoder,
from sklearn.linear_model import LogisticRegression
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE

from tensorflow.keras.optimizers import Adam
from tensorflow.keras import layers, models
from tensorflow.keras.layers import Flatten, Dense,Bidirectional,LSTM,BatchNormaliza
from tensorflow.keras.models import Sequential
from tensorflow.keras.preprocessing.sequence import pad_sequences

from nltk.corpus import stopwords
from wordcloud import WordCloud, STOPWORDS
import spacy
import nltk
nltk.download('stopwords')
```

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

```
train = pd.read_csv('/content/drive/My Drive/DS340 Final Project/Code/train_set.csv'
```

Get Training Data

```
train = pd.read_csv('/content/drive/My Drive/DS340 Final Project/Multimodal set/t
dev = pd.read_csv('/content/drive/My Drive/DS340 Final Project/Multimodal set/dev_
test = pd.read_csv('/content/drive/My Drive/DS340 Final Project/Multimodal set/test_
```

references: https://www.kaggle.com/code/abbeyra02/speech-emotion-recognition-using-xgboost#MELD

```
# get train audio
#for extracting audio, relied on code from the notebook attached
records = []
for dirname, _, filenames in os.walk('/content/drive/My Drive/DS340 Final Project/Mu
    for filename in filenames:
        records.append([filename, os.path.join(dirname,filename)])

paths = []
for i in train.index:
    dial_id = train['Dialogue_ID'][i]
    utt_id = train['Utterance_ID'][i]
    getPath = 'dia' + str(dial_id) + '_' + 'utt' + str(utt_id) + '.mp4'

    path = ''
    for index, (x, y) in enumerate(records):
        if getPath == x:
            path = y
            break

    paths.append(path)

train['path'] = paths
```

```
records = []
for dirname, _, filenames in os.walk('/content/drive/My Drive/DS340 Final Project
    for filename in filenames:
        records.append([filename, os.path.join(dirname,filename)])

paths = []
for i in test.index:
    dial_id = test['Dialogue_ID'][i]
    utt_id = test['Utterance_ID'][i]
    getPath = 'dia' + str(dial_id) + '_' + 'utt' + str(utt_id) + '.mp4'

    path = ''
    for index, (x, y) in enumerate(records):
        if getPath == x:
            path = y
            break

    paths.append(path)

test['path'] = paths


records = []
for dirname, _, filenames in os.walk('/content/drive/My Drive/DS340 Final Project/Mu
    for filename in filenames:
        records.append([filename, os.path.join(dirname,filename)])

paths = []
for i in dev.index:
    dial_id = dev['Dialogue_ID'][i]
    utt_id = dev['Utterance_ID'][i]
    getPath = 'dia' + str(dial_id) + '_' + 'utt' + str(utt_id) + '.mp4'

    path = ''
    for index, (x, y) in enumerate(records):
        if getPath == x:

            path = y
            break

    paths.append(path)

dev['path'] = paths


combined = pd.concat([train, test, dev], ignore_index=True)
combined.head()


for index, row in combined.iterrows():
    if row['path'] == '':
        combined.drop(index, inplace=True)
combined.to_csv('/content/drive/My Drive/DS340 Final Project/Multimodal set/train_
```

CLEAN UTTERANCES

Cleaning Text for NLP:

https://towardsdatascience.com/cleaning-preprocessing-text-data-for-sentiment-analysis-382a41f150d6

```
#relied on cleaning-preprocessing-text-data-for-sentiment-analysis article above
def clean(text):
    stop = stopwords.words('english')
    text = text.encode("ascii", "ignore").decode()
    text = text.lower()
    text = re.sub(r'[^A-Za-z\s\']', '', text) #remove all non alpha characters with
    filtered_words = [word for word in text if word not in stop]
    filtered_sentence = ' '.join(filtered_words)
    return text


cleaned = []
for text in train['Utterance']:
  cleaned.append(clean(text))

train['cleaned_text'] = cleaned

#lemmatization code for preprocessing
nlp = spacy.load('en_core_web_sm', disable=['parser', 'ner'])

doc = nlp(text)
def space(comment):
    doc = nlp(comment)
    return " ".join([token.lemma_ for token in doc])

train['cleaned_text']= train['cleaned_text'].apply(space)

train.head()
```

	Unnamed: 0	Sr No.	Utterance	Speaker	Emotion	Sentiment	Dialogue_ID	Utterance_
0	0	1	also I was the point person on my company□s tr...	Chandler	neutral	neutral	0	
1	1	2	You must□ve had your hands full.	The Interviewer	neutral	neutral	0	
2	2	3	That I did. That I did.	Chandler	neutral	neutral	0	
3	3	4	So let□s talk a little bit about your duties.	The Interviewer	neutral	neutral	0	

```
train.value_counts('Sentiment')
```

```
Sentiment
neutral     4906
negative    3265
positive    2318
Name: count, dtype: int64
```

```
#Subset for Sentiment data, get balanced set

df_neutral = train[train['Sentiment'] == 'neutral'].head(2310)
df_negative = train[train['Sentiment'] == 'negative'].head(2310)
df_positive = train[train['Sentiment'] == 'positive'].head(2310)

sentiment_sub = pd.concat([df_neutral, df_positive,df_negative])
```

Get Audio Data (Create NPY file)

https://github.com/NielsRogge/Transformers-Tutorials/blob/master/AST/Inference_with_the_Audio_Spectogram_Transformer_to_classify_audio.ipynb

https://pytorch.org/audio/stable/tutorials/audio_resampling_tutorial.html

Audio Features from transformer

```
#relied on pytorch audio resample tutorial
#ran this code twice to get two feature sets, one for sentiment and another for emot
target_sampling_rate = 16000
AST_extractor = ASTFeatureExtractor()
missed = []
resampler = Resample(orig_freq=48000, new_freq=target_sampling_rate)
audio_features = []
for i,path in enumerate(train['path']):

  try:
    waveform, sampling_rate = librosa.load(path)

  except Exception as e:
    print(f"Error loading file at index {i}: {e}") #chat GPT handle librosa.load err
    missed.append(i)
    audio_features.append('not found')
    continue

  if sampling_rate != target_sampling_rate:
      waveform = torch.tensor(waveform)
      waveform = resampler(waveform)

#relied on transfomers inference with AST github link above
  waveform = waveform.squeeze().numpy()
  print(i)
  inputs = AST_extractor(waveform, sampling_rate=target_sampling_rate, padding = 'ma
  input_values = inputs.input_values
  audio_features.append(input_values)

print(missed) # 3 audio files were unable to be found
```

```
torch.save(audio_features, '/content/drive/My Drive/DS340 Final Project/Multimodal s
#torch.save(audio_features, '/content/drive/My Drive/DS340 Final Project/Multimodal
```

```
audio_features_emotion = torch.load('/content/drive/My Drive/DS340 Final Project/Cod
```

```
audio_features_sentiment = torch.load('/content/drive/My Drive/DS340 Final Project/C
sentiment_dat=[]
missed_sentiment = []
for i,tensor in enumerate(audio_features_sentiment):
  if tensor == 'not found':
    missed_sentiment.append(i)
    continue
  sentiment_dat.append(tensor)
sentiment_dat = np.array(sentiment_dat)
missed_sentiment
```

```
[399, 3612, 4805]
```

```
#removed missing values for emotion
# indices to drop: [399, 2495, 5922]
emotion_sub_audio = train.drop([399, 2495, 5922], axis = 0)

#removed missing values for sentiment
#indices to drop: [399, 3612, 4805]
sentiment_sub_audio = sentiment_sub.drop([399, 3612, 4805], axis=0)
```

ANN for Sentiment features (positive, neutral, negative)

```
label_encoder = LabelEncoder()
integer_encoded = label_encoder.fit_transform(sentiment_sub_audio['Sentiment'])
n_classes = len(set(integer_encoded))
y_labels = keras.utils.to_categorical(integer_encoded, num_classes = n_classes)
```

```
#drop extra dimension
input_data_reshaped = np.reshape(sentiment_dat, (-1, 1024, 128))
input_data_reshaped.shape
```

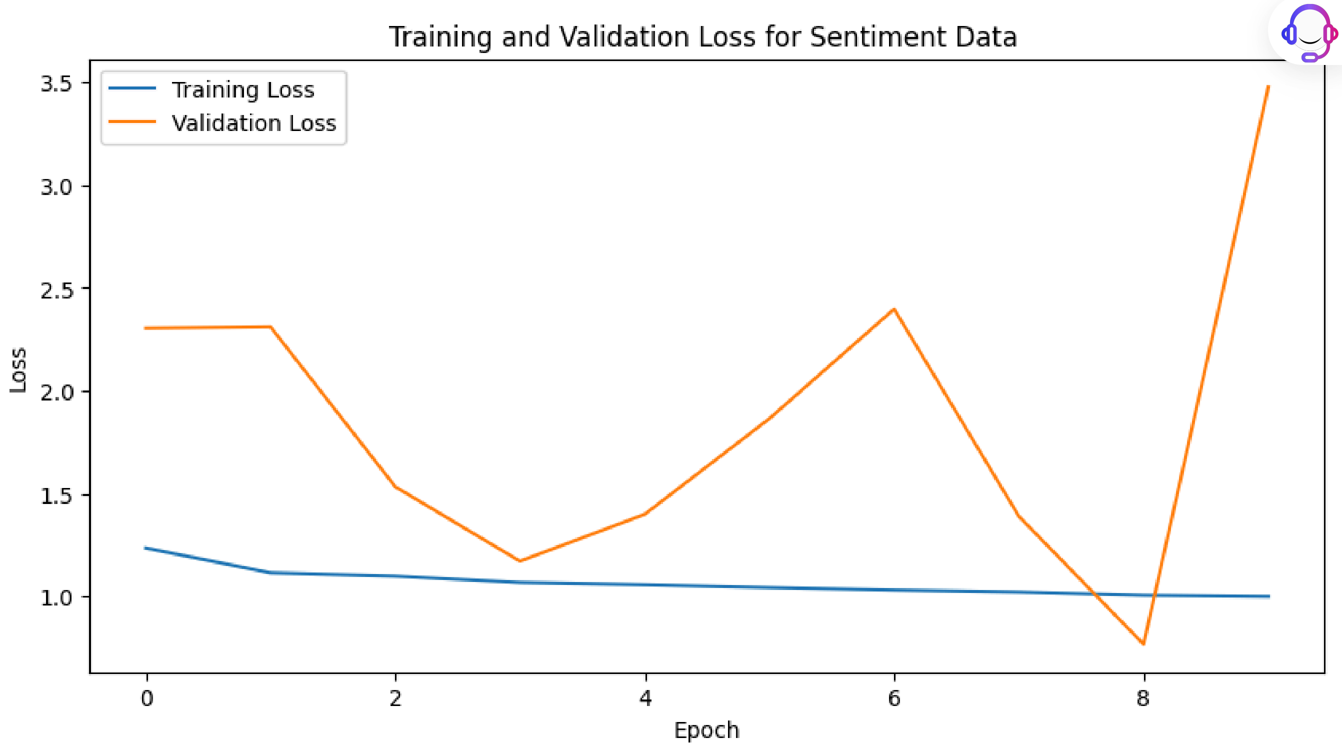
```
(6927, 1024, 128)
```

```
input_shape = (1024,128)
audio_model = models.Sequential()

audio_model.add(Flatten(input_shape=input_shape))
audio_model.add(Dense(units=256, activation='relu'))
audio_model.add(BatchNormalization())
audio_model.add(Dropout(.2))
audio_model.add(Dense(units=128, activation='relu'))
audio_model.add(BatchNormalization())
audio_model.add(Dropout(.2))

audio_model.add(Dense(units=n_classes, activation='softmax'))

audio_model.compile(optimizer = 'adam',
              loss='categorical_crossentropy',
             metrics=['accuracy'])
```

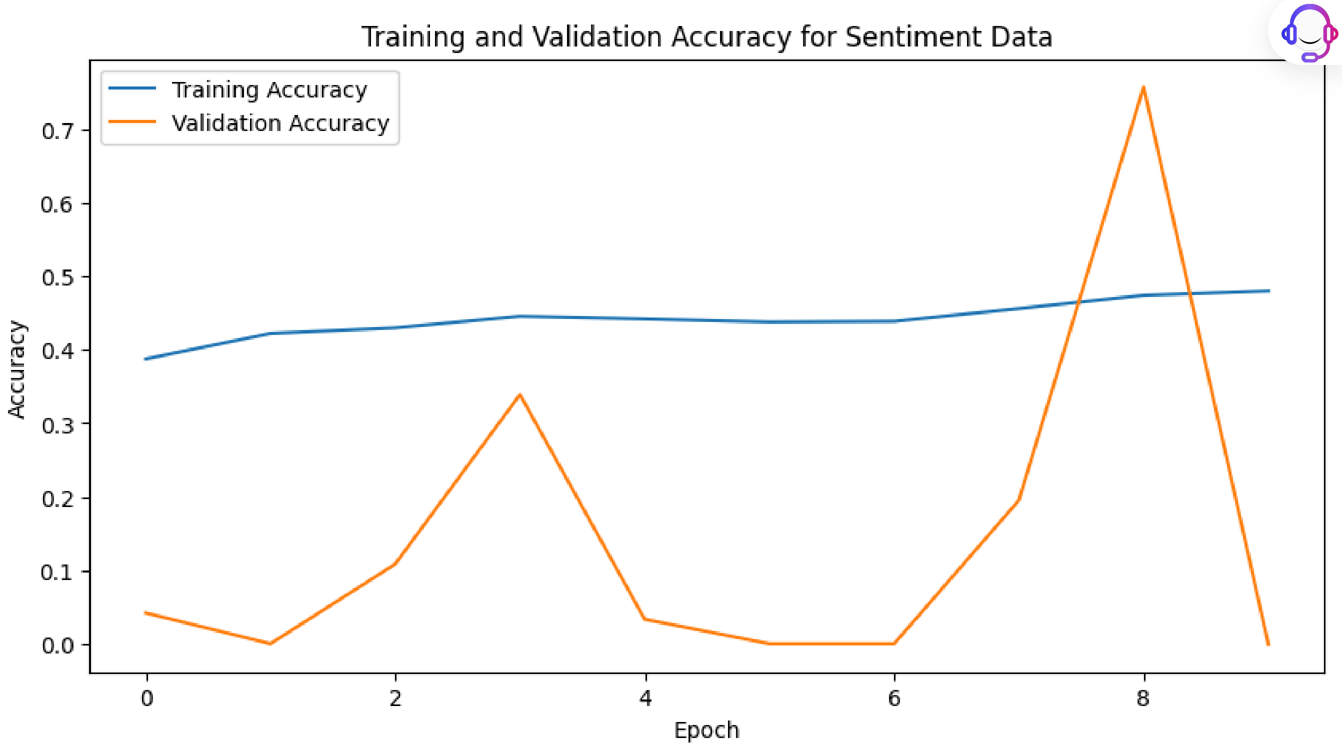
```
history_sentiment = audio_model.fit(input_data_reshaped, y_labels, epochs=10, batch_
```

```
Epoch 1/10
174/174 [==============================] - 9s 29ms/step - loss: 1.2735 - accurac
Epoch 2/10
174/174 [==============================] - 3s 19ms/step - loss: 1.1168 - accurac
Epoch 3/10
174/174 [==============================] - 3s 19ms/step - loss: 1.0964 - accurac
Epoch 4/10
174/174 [==============================] - 3s 19ms/step - loss: 1.0700 - accurac
Epoch 5/10
174/174 [==============================] - 3s 19ms/step - loss: 1.0578 - accurac
Epoch 6/10
174/174 [==============================] - 3s 19ms/step - loss: 1.0411 - accurac
Epoch 7/10
174/174 [==============================] - 3s 19ms/step - loss: 1.0275 - accurac
Epoch 8/10
174/174 [==============================] - 3s 19ms/step - loss: 1.0165 - accurac
Epoch 9/10
174/174 [==============================] - 3s 19ms/step - loss: 1.0063 - accurac
Epoch 10/10
174/174 [==============================] - 3s 19ms/step - loss: 1.0001 - accurac
```

```
plt.figure(figsize=(10, 5))
plt.plot(history_sentiment.history['loss'], label='Training Loss')
plt.plot(history_sentiment.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss for Sentiment Data ')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



```
plt.figure(figsize=(10, 5))
plt.plot(history_sentiment.history['accuracy'], label='Training Accuracy')
plt.plot(history_sentiment.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy for Sentiment Data')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



```
audio_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 flatten (Flatten)           (None, 131072)            0         
                                                                 
 dense (Dense)               (None, 256)               33554688  
                                                                 
 batch_normalization (Batch  (None, 256)               1024      
 Normalization)                                                  
                                                                 
 dropout (Dropout)           (None, 256)               0         
                                                                 
 dense_1 (Dense)             (None, 128)               32896     
                                                                 
 batch_normalization_1 (Bat  (None, 128)               512       
 chNormalization)                                                
                                                                 
 dropout_1 (Dropout)         (None, 128)               0         
                                                                 
 dense_2 (Dense)             (None, 3)                 387       
                                                                 
=================================================================
Total params: 33589507 (128.13 MB)
Trainable params: 33588739 (128.13 MB)
```

```
Non-trainable params: 768 (3.00 KB)
_________________________________________________________________
```

Audio Model for Emotion Features

```
emotion_sub_audio['Emotion'].value_counts()
```

```
Emotion
neutral     4904
joy         1721
surprise    1272
anger       1243
sadness      783
disgust      287
fear         276
Name: count, dtype: int64
```

```
label_encoder = LabelEncoder()
integer_encoded = label_encoder.fit_transform(emotion_sub_audio['Emotion'])
n_classes = len(set(integer_encoded))
y_labels = keras.utils.to_categorical(integer_encoded, num_classes = n_classes)
```

```
input_data_reshaped = np.reshape(audio_features_emotion, (-1, 1024, 128))
```

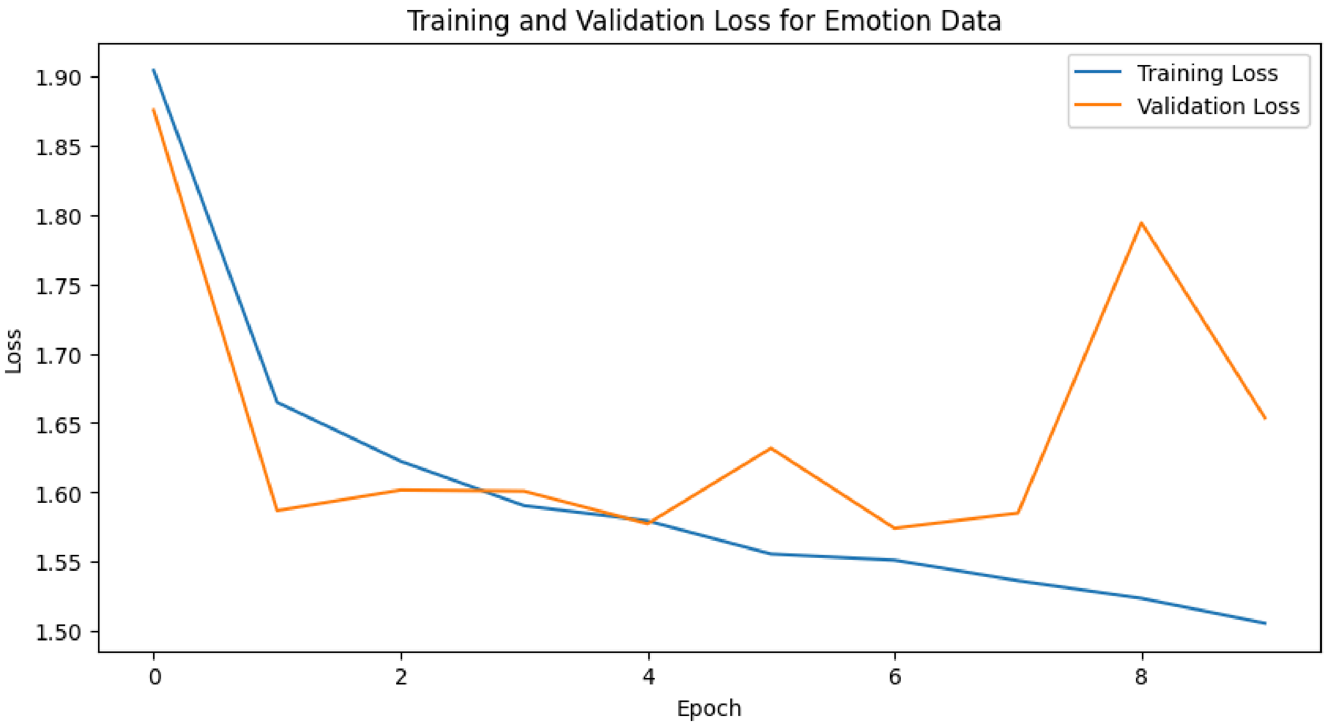
```
input_shape = (1024,128)
audio_model = models.Sequential()

audio_model.add(Flatten(input_shape=input_shape))
audio_model.add(Dense(units=256, activation='relu'))
audio_model.add(BatchNormalization())
audio_model.add(Dropout(.2))
audio_model.add(Dense(units=128, activation='relu'))
audio_model.add(BatchNormalization())
audio_model.add(Dropout(.2))

audio_model.add(Dense(units=n_classes, activation='softmax'))

audio_model.compile(optimizer = 'adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

```
history_emotion = audio_model.fit(input_data_reshaped, y_labels, epochs=10, batch_si
```

```
Epoch 1/10
263/263 [==============================] - 9s 28ms/step - loss: 1.9049 - accurac
Epoch 2/10
263/263 [==============================] - 5s 19ms/step - loss: 1.6579 - accurac
Epoch 3/10
```

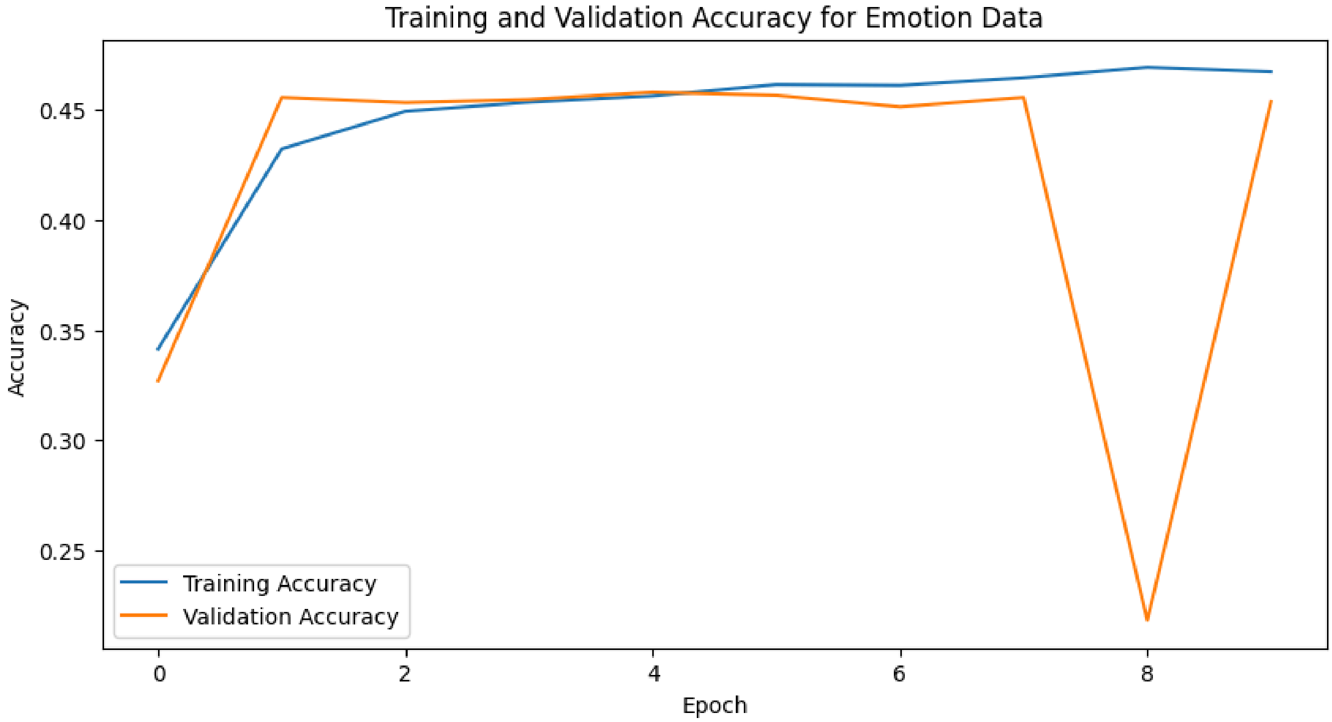
```
263/263 [==============================] - 5s 19ms/step - loss: 1.6228 - acc
Epoch 4/10
263/263 [==============================] - 5s 19ms/step - loss: 1.5951 - accurac
Epoch 5/10
263/263 [==============================] - 5s 19ms/step - loss: 1.5735 - accurac
Epoch 6/10
263/263 [==============================] - 5s 19ms/step - loss: 1.5570 - accurac
Epoch 7/10
263/263 [==============================] - 5s 19ms/step - loss: 1.5504 - accurac
Epoch 8/10
263/263 [==============================] - 5s 19ms/step - loss: 1.5369 - accurac
Epoch 9/10
263/263 [==============================] - 5s 19ms/step - loss: 1.5154 - accurac
Epoch 10/10
263/263 [==============================] - 5s 19ms/step - loss: 1.5069 - accurac
```

```
plt.figure(figsize=(10, 5))
plt.plot(history_emotion.history['loss'], label='Training Loss')
plt.plot(history_emotion.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss for Emotion Data ')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



```
plt.figure(figsize=(10, 5))
plt.plot(history_emotion.history['accuracy'], label='Training Accuracy')
plt.plot(history_emotion.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy for Emotion Data')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f7f74b8e590>
```



BERT TRANSFORMER Embeddings For Text

References: https://jalammar.github.io/a-visual-guide-to-using-bert-for-the-first-time/

```
#code for tokenizing, padding maskig from jalammar sentiment analysis notebook
model_class, tokenizer_class, pretrained_weights = (ppb.BertModel, ppb.BertTokenizer
tokenizer = tokenizer_class.from_pretrained(pretrained_weights)
bert_model = model_class.from_pretrained(pretrained_weights)
tokenized = sentiment_sub['cleaned_text'].apply((lambda x: tokenizer.encode(x, add_s
#relied on jalammar tutorial for masking/padding code
max_len = 0
for i in tokenized.values:
    if len(i) > max_len:
        max_len = len(i)

padded = np.array([i + [0]*(max_len-len(i)) for i in tokenized.values])
attention_mask = np.where(padded != 0, 1, 0)
input_ids = torch.tensor(np.array(padded))
attention_mask = torch.tensor(attention_mask)
```

```
#have to batch the data set or else feature extraction crashes session
def batch_array(arr, batch_size):
    batches = []
    for i in range(0, len(arr), batch_size):
        batches.append(arr[i:i+batch_size])
    return batches

attention_batches = batch_array(attention_mask,2000)
input_batches = batch_array(input_ids,2000)
```

```
#get embeddings
#ran this for both sentiment and emotion

last_hidden = []
text_embeddings = []

for i in range(len(attention_batches)):
  with torch.no_grad():
      last_hidden_states = bert_model(input_batches[i], attention_mask=attention_bat
      text_embeddings.append(last_hidden_states)
      sent_vec = last_hidden_states[0][:, 0, :]  # Assuming [CLS] is at position 0
      batch_embeddings = last_hidden.append(sent_vec.numpy())
```

```
torch.save(last_hidden, '/content/drive/My Drive/DS340 Final Project/Multimodal set/
torch.save(text_embeddings, '/content/drive/My Drive/DS340 Final Project/Multimodal
```

```
emotion_text_embeddings =torch.load('/content/drive/My Drive/DS340 Final Project/Cod
emotion_last_hidden_states = torch.load('/content/drive/My Drive/DS340 Final Project
```

```
sentiment_text_embeddings =torch.load('/content/drive/My Drive/DS340 Final Project/C
sentiment_last_hidden_states = torch.load('/content/drive/My Drive/DS340 Final Proje
sentiment_last = np.concatenate(sentiment_last_hidden_states, axis=0)
emotion_last = np.concatenate(emotion_last_hidden_states)
```

```
tensor_arrays = [output.last_hidden_state.numpy() for output in emotion_text_embeddi
emotion_features = np.concatenate(tensor_arrays, axis=0)
emotion_features.shape
```

```
(10489, 76, 768)
```

```
tensor_arrays = [output.last_hidden_state.numpy() for output in sentiment_text_embed
sentiment_features = np.concatenate(tensor_arrays, axis=0)
sentiment_features.shape
```

```
(6930, 76, 768)
```

```
#get encoded labels sentiment
label_encoder = LabelEncoder()
integer_encoded = label_encoder.fit_transform(sentiment_sub['Sentiment'])
```

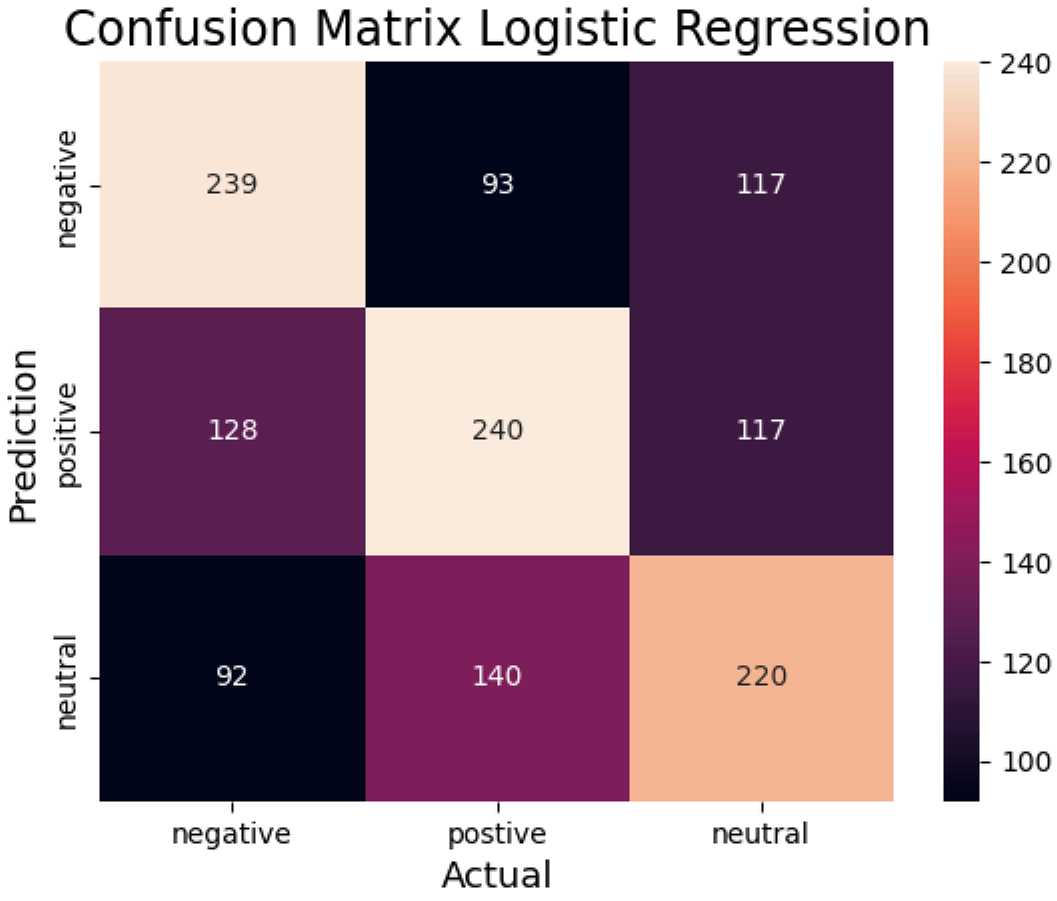
```
#logistic regression for sentiment
X_train, X_test, y_train, y_test = train_test_split(sentiment_last, integer_encoded,
model = LogisticRegression(multi_class='multinomial', solver='lbfgs', max_iter = 200
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.5043290043290043
```

```
sentiment_sub['integer_encoded'] = integer_encoded

ticks = {}
for index, row in sentiment_sub.iterrows():
  if row['Sentiment'] not in ticks:
    ticks[row['Sentiment']] = row['integer_encoded']

ticks
```

```
{'neutral': 1, 'positive': 2, 'negative': 0}
```

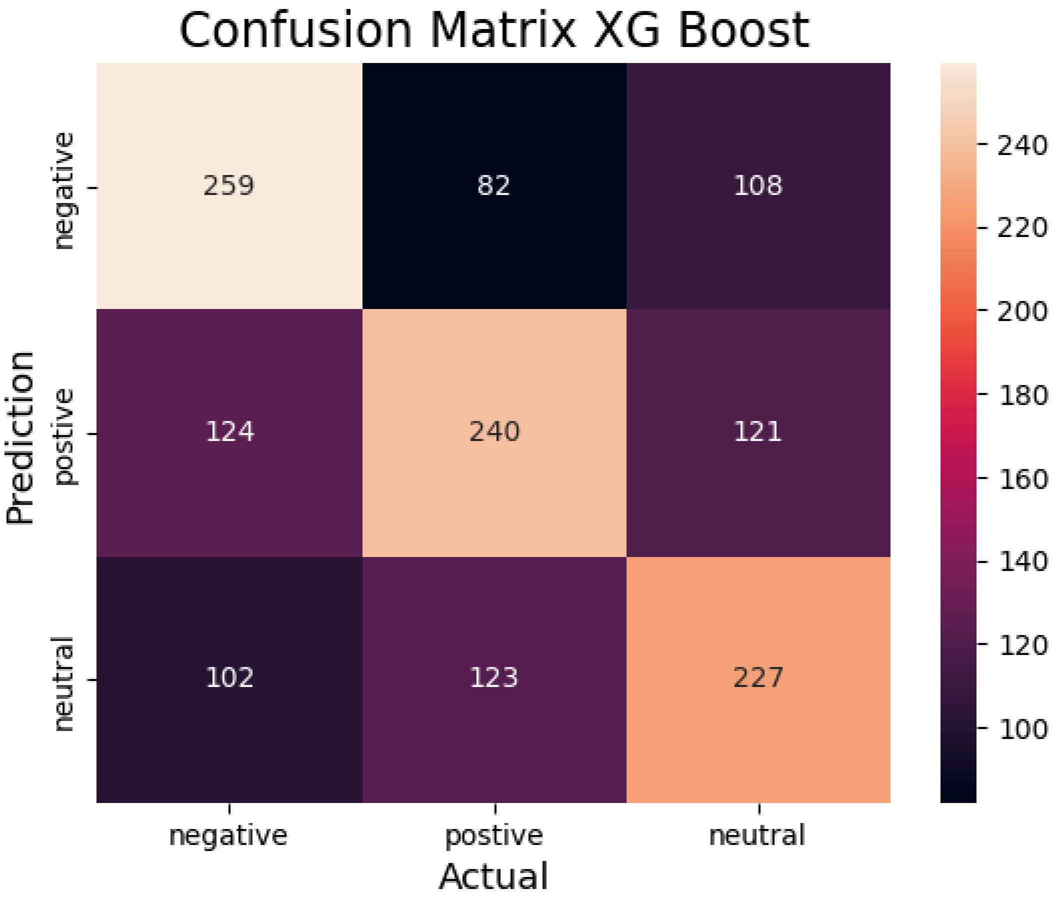
```
cm = confusion_matrix(y_test, y_pred)

sns.heatmap(cm,
            annot=True,
            fmt='g',
            xticklabels=['negative','postive','neutral'],
            yticklabels=['negative','positive','neutral']
            )
plt.ylabel('Prediction', fontsize=13)
plt.xlabel('Actual', fontsize=13)
plt.title('Confusion Matrix Logistic Regression', fontsize=17)
plt.show()
```



```
#XG boost for sentiment
xgb_classifier = xgb.XGBClassifier()

xgb_classifier.fit(X_train, y_train)

y_pred = xgb_classifier.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.5238095238095238
Classification Report:
              precision    recall  f1-score   support

           0       0.53      0.58      0.55       449
           1       0.54      0.49      0.52       485
           2       0.50      0.50      0.50       452

    accuracy                           0.52      1386
   macro avg       0.52      0.52      0.52      1386
weighted avg       0.52      0.52      0.52      1386
```

```
cm = confusion_matrix(y_test, y_pred)

sns.heatmap(cm,
            annot=True,
            fmt='g',
            xticklabels=['negative','postive','neutral'],
            yticklabels=['negative','postive','neutral'])
plt.ylabel('Prediction', fontsize=13)
plt.xlabel('Actual', fontsize=13)
plt.title('Confusion Matrix XG Boost', fontsize=17)
plt.show()
```



```
#Data for Emotion, 7 classes to predict

#run this to get correct y labels
label_encoder = LabelEncoder()
integer_encoded = label_encoder.fit_transform(train['Emotion'])
```

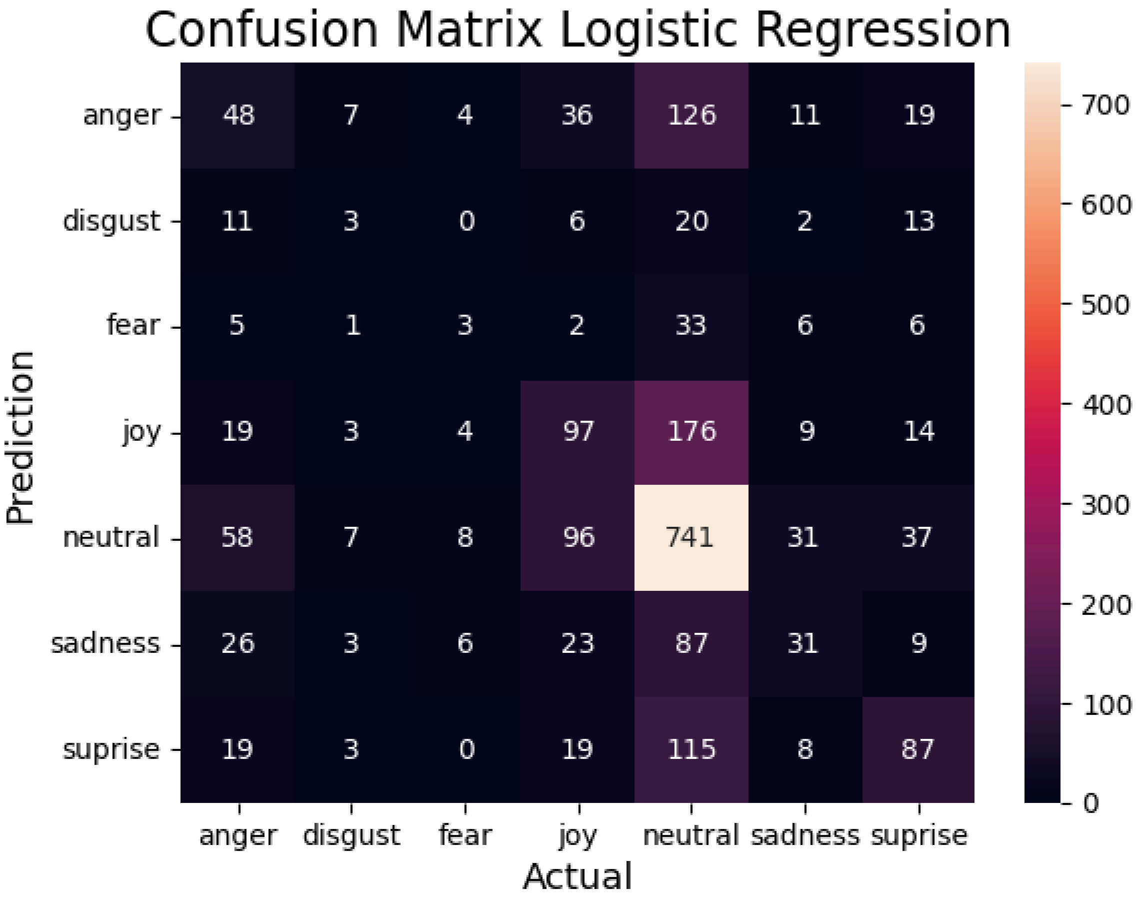
```
#Logistic Regression for Emotion
X_train, X_test, y_train, y_test = train_test_split(emotion_last, integer_encoded, t
model = LogisticRegression(multi_class='multinomial', solver='lbfgs', max_iter = 200
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.48141086749285034
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: C
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regressio
  n_iter_i = _check_optimize_result(
```

```
train['integer_encoded'] = integer_encoded
ticks = {}
for index, row in train.iterrows():
  if row['Emotion'] not in ticks:
    ticks[row['Emotion']] = row['integer_encoded']

ticks
```

```
{'neutral': 4,
 'surprise': 6,
 'fear': 2,
 'sadness': 5,
 'joy': 3,
 'disgust': 1,
 'anger': 0}
```

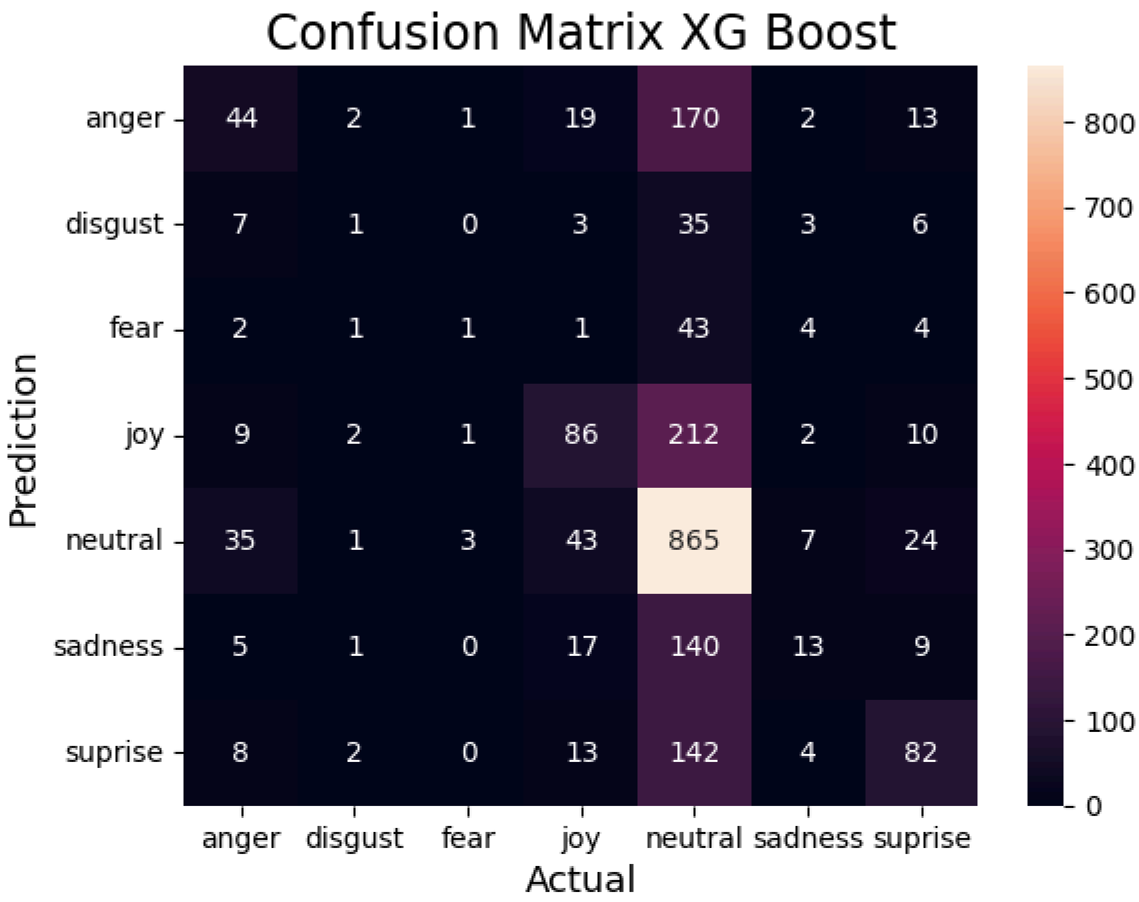
```
cm = confusion_matrix(y_test, y_pred)

sns.heatmap(cm,
            annot=True,
            fmt='g',
            xticklabels=['anger','disgust','fear', 'joy','neutral','sadness', 'supri
            yticklabels=['anger','disgust','fear', 'joy','neutral','sadness', 'supri
plt.ylabel('Prediction', fontsize=13)
plt.xlabel('Actual', fontsize=13)
plt.title('Confusion Matrix Logistic Regression', fontsize=17)
plt.show()
```



```
#XG boost for emotion
xgb_classifier = xgb.XGBClassifier()

xgb_classifier.fit(X_train, y_train)

y_pred = xgb_classifier.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.5204957102001907
Classification Report:
              precision    recall  f1-score   support

           0       0.40      0.18      0.24       251
           1       0.10      0.02      0.03        55
           2       0.17      0.02      0.03        56
           3       0.47      0.27      0.34       322
           4       0.54      0.88      0.67       978
           5       0.37      0.07      0.12       185
           6       0.55      0.33      0.41       251

    accuracy                           0.52      2098
   macro avg       0.37      0.25      0.26      2098
weighted avg       0.48      0.52      0.45      2098
```

```
cm = confusion_matrix(y_test, y_pred)

sns.heatmap(cm,
            annot=True,
            fmt='g',
            xticklabels=['anger','disgust','fear', 'joy','neutral','sadness', 'supri
            yticklabels=['anger','disgust','fear', 'joy','neutral','sadness', 'supri
plt.ylabel('Prediction', fontsize=13)
plt.xlabel('Actual', fontsize=13)
plt.title('Confusion Matrix XG Boost', fontsize=17)
plt.show()
```



	anger	disgust	fear	joy	neutral	sadness	suprise
anger	44	2	1	19	170	2	13
disgust	7	1	0	3	35	3	6
fear	2	1	1	1	43	4	4
joy	9	2	1	86	212	2	10
neutral	35	1	3	43	865	7	24
sadness	5	1	0	17	140	13	9
suprise	8	2	0	13	142	4	82

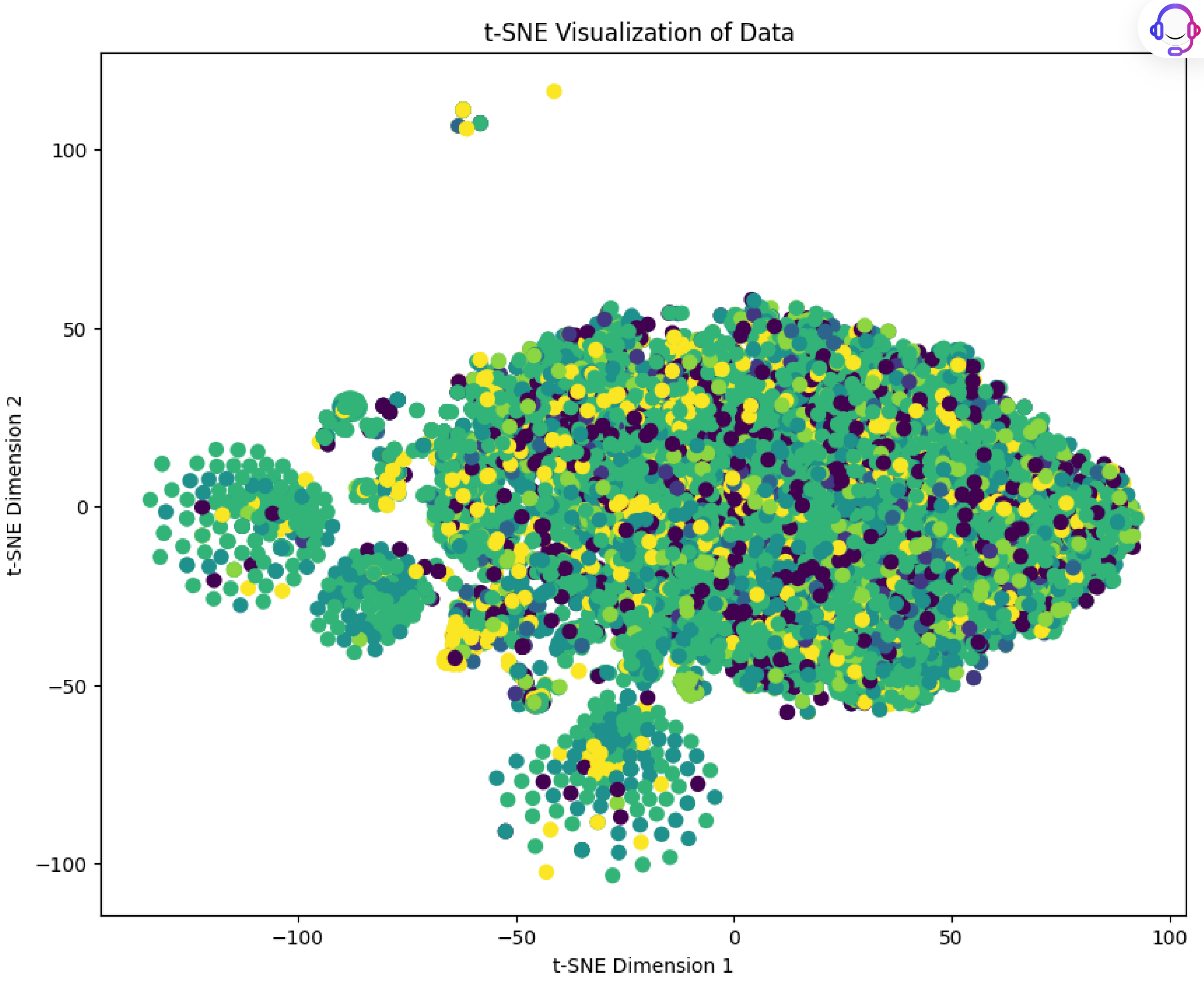
https://www.geeksforgeeks.org/confusion-matrix-machine-learning/

https://www.datacamp.com/tutorial/introduction-t-sne

```
#relied on t-sne tutorial link above
#t-sne for emotion, also run this for sentiment

pca = PCA(n_components=50)
data_pca = pca.fit_transform(emotion_last)
tsne = TSNE(n_components=2, random_state=0, n_iter=2000, perplexity=50)
data_tsne = tsne.fit_transform(data_pca)
tsne.kl_divergence_
```

```
2.360335350036621
```

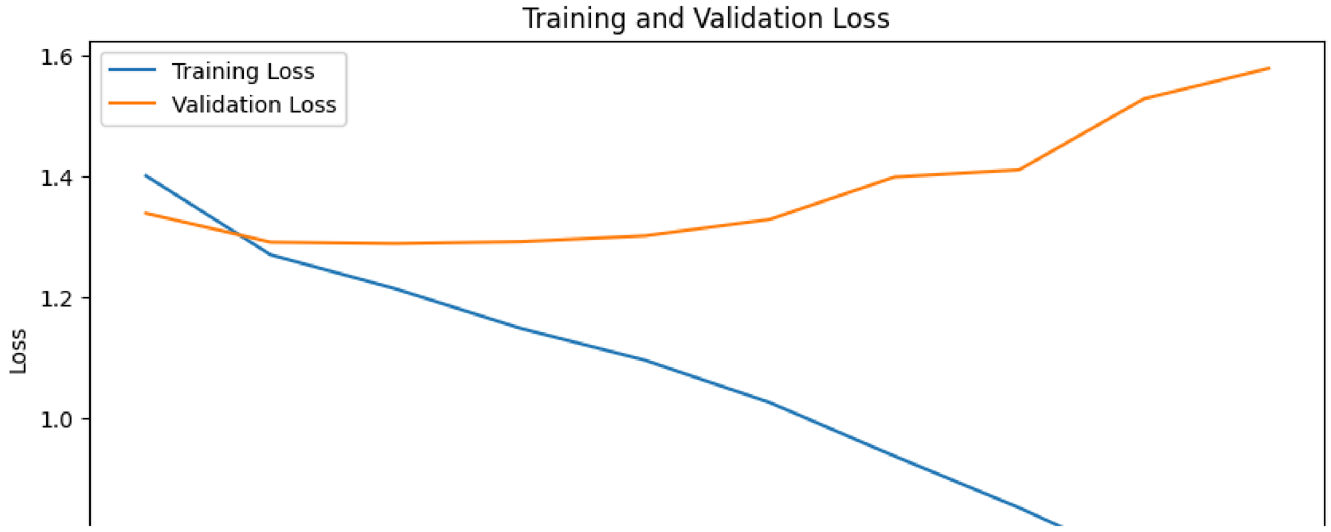
```
plt.figure(figsize=(10, 8))
plt.scatter(data_tsne[:, 0], data_tsne[:, 1], c=integer_encoded, marker='o', s=50, c
plt.title('t-SNE Visualization of Data ')
plt.xlabel('t-SNE Dimension 1')
plt.ylabel('t-SNE Dimension 2')
plt.show()
```



Bidirectional LSTM Model for Sentiment

```
#run this to get correct y labels
label_encoder = LabelEncoder()
integer_encoded = label_encoder.fit_transform(sentiment_sub['Sentiment'])
n_classes = len(set(integer_encoded))
y_labels = keras.utils.to_categorical(integer_encoded, num_classes = n_classes)
```

```
text_model = Sequential()
text_model.add(Bidirectional(LSTM(units=256, return_sequences = True), input_shap
text_model.add(BatchNormalization())
text_model.add(Dropout(.2))
text_model.add(LSTM(units=128, return_sequences=False))
text_model.add(BatchNormalization())
text_model.add(Dropout(.2))
text_model.add(Dense(n_classes, activation='softmax'))
optimizer = Adam(learning_rate = .0001)


sentiment_text_history = text_model.fit(sentiment_features, y_labels, epochs=10, bat


text_model.summary()


plt.figure(figsize=(10, 5))
plt.plot(sentiment_text_history.history['loss'], label='Training Loss')
plt.plot(sentiment_text_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(sentiment_text_history.history['accuracy'], label='Training Accuracy')
plt.plot(sentiment_text_history.history['val_accuracy'], label='Validation Accuracy'
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```


Training and Validation Loss
Training Loss
Validation Loss
Loss