Text Classification 2 - Priyesh Patel

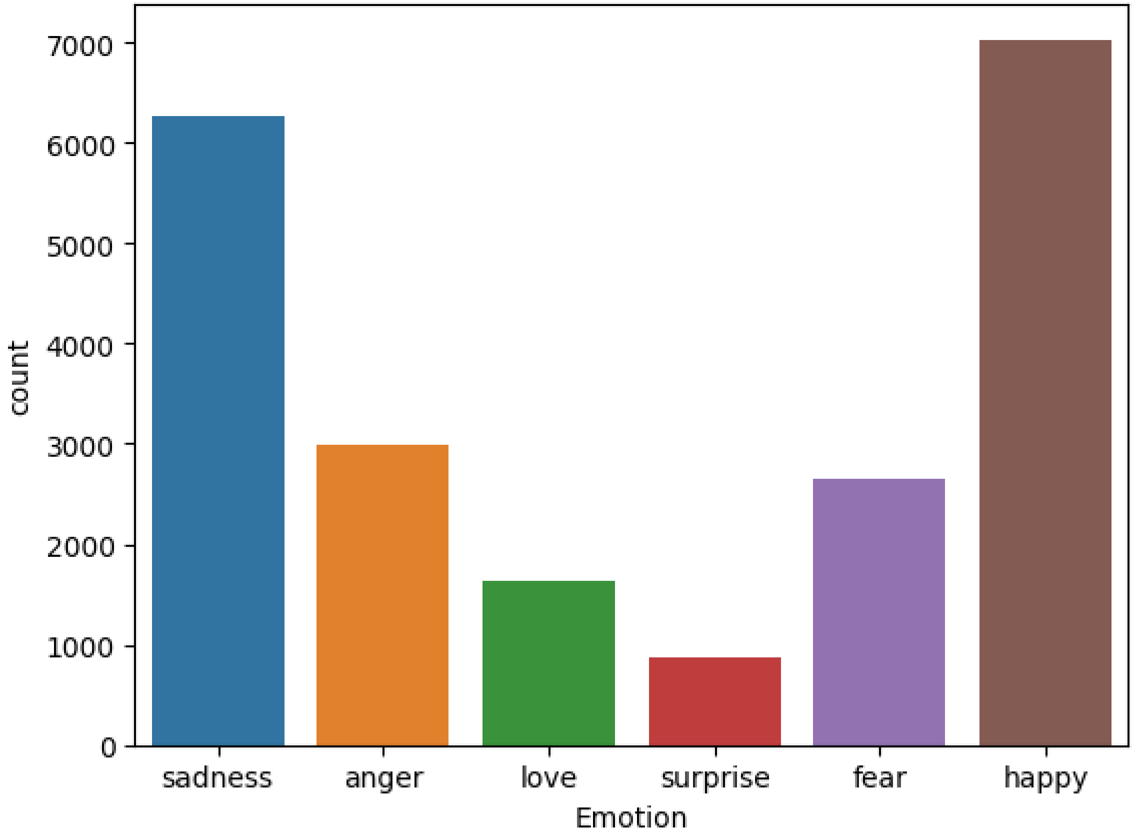
Emotions in Text Dataset

The dataset I chose was a multiclassification dataset for emotion in text. The Dataset contains a total of 21405 data entries with texas classification categories such as sadness, anger, love, surprise, fear, and happy. The distribution varies with sadness and happy having the largest amount of data entires. So I decided to condense the classification to happy and sad. Using Deep learning models given some text the models should predict weather it convays a happy or sad emotion.

```
import seaborn as sb
import pandas as pd
import numpy as np
import tensorflow as tf

df = pd.read_csv('emotion.csv')
sb.countplot( x = "Emotion", data = df)
```

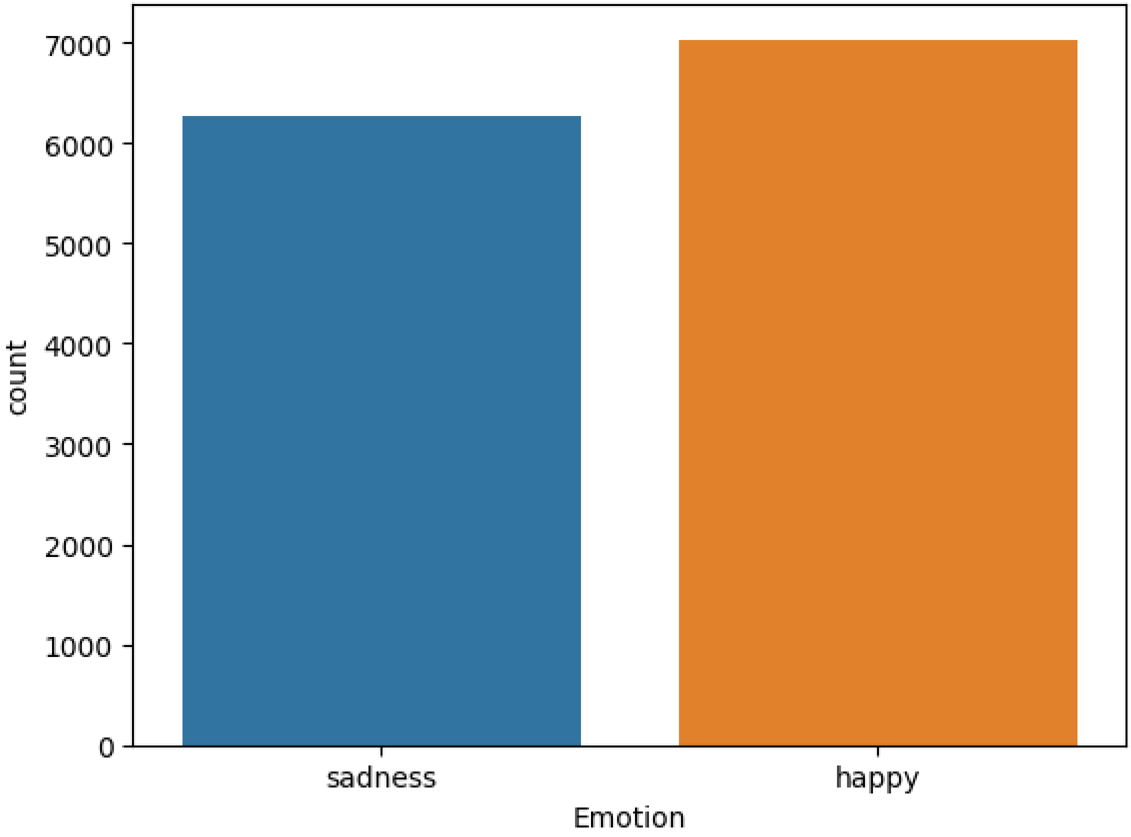
```
<Axes: xlabel='Emotion', ylabel='count'>
```



```
#Remove anger, love, surprise, and fear entires
mask = (df['Emotion'] != 'anger') & (df['Emotion'] != 'love') & (df['Emotion'] != 'su
df = df[mask]

sb.countplot( x = "Emotion", data = df)
```

```
    <Axes: xlabel='Emotion', ylabel='count'>
```



Get Training and test set

```
print('rows and columns:', df.shape)
print(df.head())
```

```
    rows and columns: (13294, 2)
                                                     Text  Emotion
    0                             i didnt feel humiliated  sadness
    1   i can go from feeling so hopeless to so damned...  sadness
    5    ive been feeling a little burdened lately wasn...  sadness
    8   i have been with petronas for years i feel tha...    happy
    10  i feel like i have to make the suffering i m s...  sadness
```

```
# split df into train and test
i = np.random.rand(len(df)) < 0.8
train = df[i]
test = df[~i]
print("train data size: ", train.shape)
print("test data size: ", test.shape)
```

```
    train data size:  (10626, 2)
    test data size:  (2668, 2)
```

```
from tensorflow.keras.preprocessing.text import Tokenizer
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras import datasets, layers, models, preprocessing

# # set up X and Y
num_labels = 2
vocab_size = 25000
batch_size = 32
maxlen = 500

# fit the tokenizer on the training data
tokenizer = Tokenizer(num_words=vocab_size)
tokenizer.fit_on_texts(train.Text)

x_train = tokenizer.texts_to_matrix(train.Text, mode='tfidf')
x_test = tokenizer.texts_to_matrix(test.Text, mode='tfidf')

encoder = LabelEncoder()
encoder.fit(train.Emotion)
y_train = encoder.transform(train.Emotion)
y_test = encoder.transform(test.Emotion)

# check shape
print("train shapes:", x_train.shape, y_train.shape)
print("test shapes:", x_test.shape, y_test.shape)
print("test first five labels:",x_test, y_test[:5])
```

```
    train shapes: (10626, 25000) (10626,)
    test shapes: (2668, 25000) (2668,)
    test first five labels: [[0.         1.27070438 0.93516194 ... 0.         0.
     [0.         0.75049848 0.93516194 ... 0.         0.         0.        ]
     [0.         1.95837918 0.93516194 ... 0.         0.         0.        ]
     ...
     [0.         0.         0.         ... 0.         0.         0.        ]
     [0.         0.         0.         ... 0.         0.         0.        ]
     [0.         0.         0.         ... 0.         0.         0.        ]] [1 0 0
```

Sequential Model

```
from tensorflow.keras import layers, models

# build a Sequential model
seq_model = models.Sequential()
seq_model.add(layers.Dense(32, activation='relu'))
seq_model.add(layers.Dense(1, activation='sigmoid'))

#compile
seq_model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

```
history = seq_model.fit(x_train, y_train,
                    epochs=10,
                    verbose=1,
                    validation_split=0.1)
```

```
    Epoch 1/10
    299/299 [==============================] - 8s 5ms/step - loss: 0.3076 - accuracy
    Epoch 2/10
    299/299 [==============================] - 1s 4ms/step - loss: 0.0388 - accuracy
    Epoch 3/10
    299/299 [==============================] - 1s 4ms/step - loss: 0.0118 - accuracy
    Epoch 4/10
    299/299 [==============================] - 1s 4ms/step - loss: 0.0048 - accuracy
    Epoch 5/10
    299/299 [==============================] - 1s 4ms/step - loss: 0.0027 - accuracy
    Epoch 6/10
    299/299 [==============================] - 1s 4ms/step - loss: 0.0018 - accuracy
    Epoch 7/10
    299/299 [==============================] - 1s 4ms/step - loss: 0.0013 - accuracy
    Epoch 8/10
    299/299 [==============================] - 1s 4ms/step - loss: 9.4930e-04 - accu
    Epoch 9/10
    299/299 [==============================] - 1s 4ms/step - loss: 7.2251e-04 - accu
    Epoch 10/10
    299/299 [==============================] - 1s 4ms/step - loss: 5.5912e-04 - accu
```

```
# evaluate
score = seq_model.evaluate(x_test, y_test, batch_size=batch_size, verbose=1)
print('Accuracy: ', score[1])

# get predictions so we can calculate more metrics
pred = seq_model.predict(x_test)
pred_labels = [1 if p>0.5 else 0 for p in pred]
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
print()
print('accuracy score: ', accuracy_score(y_test, pred_labels))
print('precision score: ', precision_score(y_test, pred_labels))
print('recall score: ', recall_score(y_test, pred_labels))
print('f1 score: ', f1_score(y_test, pred_labels))
```

```
84/84 [==============================] - 0s 3ms/step - loss: 0.1988 - accuracy:
Accuracy:  0.9449025392532349
84/84 [==============================] - 0s 2ms/step

accuracy score:  0.9449025487256372
precision score:  0.9531502423263328
recall score:  0.929866036249015
f1 score:  0.9413641802951735
```

Simple RNN Model

```
from tensorflow.keras import layers, models

# build a Sequential model with Embedding and SimpleRNN layers
max_features = 10000

RNN_model = models.Sequential()
RNN_model.add(layers.Embedding(max_features, 32))
RNN_model.add(layers.SimpleRNN(32))
RNN_model.add(layers.Dense(1, activation='sigmoid'))
```

```
RNN_model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding (Embedding)       (None, None, 32)          320000    
                                                                 
 simple_rnn (SimpleRNN)      (None, 32)                2080      
                                                                 
 dense_2 (Dense)             (None, 1)                 33        
                                                                 
=================================================================
Total params: 322,113
Trainable params: 322,113
Non-trainable params: 0
_________________________________________________________________
```

```
# compile
RNN_model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

```
# train
history = RNN_model.fit(x_train[:500], y_train[:500],
                        epochs=2,
                        batch_size=32,
                        validation_split=0.1)
```

```
Epoch 1/2
15/15 [==============================] - 889s 59s/step - loss: 0.6856 - accuracy
Epoch 2/2
15/15 [==============================] - 884s 59s/step - loss: 0.6822 - accuracy
```

```
# evaluate
score = RNN_model.evaluate(x_test, y_test, batch_size=batch_size, verbose=1)
print('Accuracy: ', score[1])
```

```
84/84 [==============================] - 152s 2s/step - loss: 0.6940 - accuracy:
Accuracy:  0.5243628025054932
```

CNN Model

```
from tensorflow.keras import layers, models
max_features = 10000
maxlen = 25000
batch_size = 32

CNN_model = models.Sequential()
CNN_model.add(layers.Embedding(max_features, 128, input_length=maxlen))
CNN_model.add(layers.Conv1D(32, 7, activation='relu'))
CNN_model.add(layers.MaxPooling1D(5))
CNN_model.add(layers.Conv1D(32, 7, activation='relu'))
CNN_model.add(layers.GlobalMaxPooling1D())
CNN_model.add(layers.Dense(1))

CNN_model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_2 (Embedding)     (None, 25000, 128)        1280000
```

```
     conv1d_2 (Conv1D)           (None, 24994, 32)         28704

     max_pooling1d_1 (MaxPooling  (None, 4998, 32)         0
     1D)

     conv1d_3 (Conv1D)           (None, 4992, 32)          7200

     global_max_pooling1d_1 (Glo  (None, 32)               0
     balMaxPooling1D)

     dense_4 (Dense)             (None, 1)                 33

    =================================================================
    Total params: 1,315,937
    Trainable params: 1,315,937
    Non-trainable params: 0
    _________________________________________________________________
```

```
# compile

CNN_model.compile(optimizer=tf.keras.optimizers.RMSprop(learning_rate=1e-4),  # set l
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

```
# train
history = CNN_model.fit(x_train, y_train,
                        epochs=10,
                        batch_size=32,
                        validation_split=0.1)
```

```
    Epoch 1/10
    299/299 [==============================] - 7s 23ms/step - loss: 0.6909 - accurac
    Epoch 2/10
    299/299 [==============================] - 7s 22ms/step - loss: 0.6879 - accurac
    Epoch 3/10
    299/299 [==============================] - 6s 22ms/step - loss: 0.6860 - accurac
    Epoch 4/10
    299/299 [==============================] - 6s 22ms/step - loss: 0.6839 - accurac
    Epoch 5/10
    299/299 [==============================] - 6s 21ms/step - loss: 0.6821 - accurac
    Epoch 6/10
    299/299 [==============================] - 6s 22ms/step - loss: 0.6804 - accurac
    Epoch 7/10
    299/299 [==============================] - 6s 21ms/step - loss: 0.6785 - accurac
    Epoch 8/10
    299/299 [==============================] - 6s 22ms/step - loss: 0.6760 - accurac
    Epoch 9/10
    299/299 [==============================] - 6s 22ms/step - loss: 0.6755 - accurac
    Epoch 10/10
    299/299 [==============================] - 6s 21ms/step - loss: 0.6728 - accurac
```

```
from sklearn.metrics import classification_report

pred = CNN_model.predict(x_test)
pred = [1.0 if p>= 0.5 else 0.0 for p in pred]
print(classification_report(y_test, pred))
```

```
84/84 [==============================] - 1s 7ms/step
              precision    recall  f1-score   support

           0       0.56      0.61      0.58      1399
           1       0.53      0.48      0.50      1269

    accuracy                           0.55      2668
   macro avg       0.54      0.54      0.54      2668
weighted avg       0.54      0.55      0.54      2668
```

GRU Model

```
GRU_model = models.Sequential()
GRU_model.add(layers.Embedding(max_features, 32))
GRU_model.add(layers.GRU(32))
GRU_model.add(layers.Dense(1, activation='sigmoid'))

GRU_model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_3 (Embedding)     (None, None, 32)          320000

 gru (GRU)                   (None, 32)                6336

 dense_5 (Dense)             (None, 1)                 33

=================================================================
Total params: 326,369
Trainable params: 326,369
Non-trainable params: 0
_________________________________________________________________
```

```
# compile
GRU_model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])


# train
history = GRU_model.fit(x_train, y_train,
                        epochs=10,
                        batch_size=32,
                        validation_split=0.1)
```

```
Epoch 1/10
299/299 [==============================] - 165s 553ms/step - loss: 0.6909 - accu
Epoch 2/10
299/299 [==============================] - 165s 553ms/step - loss: 0.6909 - accu
Epoch 3/10
299/299 [==============================] - 166s 554ms/step - loss: 0.6906 - accu
Epoch 4/10
299/299 [==============================] - 165s 553ms/step - loss: 0.6907 - accu
Epoch 5/10
299/299 [==============================] - 166s 555ms/step - loss: 0.6909 - accu
Epoch 6/10
299/299 [==============================] - 166s 554ms/step - loss: 0.6908 - accu
Epoch 7/10
299/299 [==============================] - 166s 555ms/step - loss: 0.6906 - accu
Epoch 8/10
299/299 [==============================] - 166s 554ms/step - loss: 0.6907 - accu
Epoch 9/10
299/299 [==============================] - 166s 554ms/step - loss: 0.6905 - accu
Epoch 10/10
299/299 [==============================] - 166s 555ms/step - loss: 0.6906 - accu
```

```
from sklearn.metrics import classification_report

pred = GRU_model.predict(x_test)
pred = [1.0 if p>= 0.5 else 0.0 for p in pred]
print(classification_report(y_test, pred))
```

```
84/84 [==============================] - 20s 230ms/step
              precision    recall  f1-score   support

           0       0.52      1.00      0.69      1399
           1       0.00      0.00      0.00      1269

    accuracy                           0.52      2668
   macro avg       0.26      0.50      0.34      2668
weighted avg       0.27      0.52      0.36      2668

/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classification.py:1344:
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classification.py:1344:
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classification.py:1344:
  _warn_prf(average, modifier, msg_start, len(result))
```

Analysis

I decided to use a fairly large dataset that classified text into types of emotions such as happy, sad, fear, disgust, love, and surprised. Unfortunately, the dataset was uneven, and the majority of the classification were happy and sad. So, I decided to do a binary classification between those two emotions. I started by preprocessing and vectorizing the data. The first model I created was a simple sequential model using Keras models and layers functions. The sequential model did by far the best from all the other model I created. During the sequential models training phase the accuracy was very high which made me think that it might have overfitted. However during the testing phase it had a prediction accuracy of 94%. In addition, the training time for the sequential model was very fast. Next I created a simple RNN model that included embedding. For this model I had a lot of issues during train with the expect training time for an epoch was over five hours long. With these issues I had to significantly drop the training set size to only 500 entries which is very small. Even with the reduced dataset the training took over 30 minutes to compete and had a very poor performance with an accurate of 52%. This is expected with such a small pool of data for a deep learning process. At this point it would be better to flip a coin then waiting for the model to train. Finally, I created models for CNN and GRU they both ran relatively faster than the RNN model. However, their performance was like the RNN model with accuracies of 55% and 52% respectively. I think that since there are a lot of hyperparameter you need to tweak and fine tune it. This makes it hard to know what model will work best without extensive trial and error. Due to my runtime issues, I think that these model weren't able to perform as well as they possible could have. I think that for the particular dataset and classification I was trying to predict a sequential model is far superior in performance and training speed.