Nick Johnson CS 4395.001 Text Classification 2

Find a text classification data set that interests you.

https://www.kaggle.com/datasets/tirendazacademy/fifa-world-cup-2022-tweets

Describe the data set and what the model should be able to predict.

I've chosen a dataset containing tweets that are about the 2022 FIFA World Cup. The model should be able to predict whether each tweet has a neutral, positive, or negative sentiment.

Create a graph showing the distribution of the target classes.

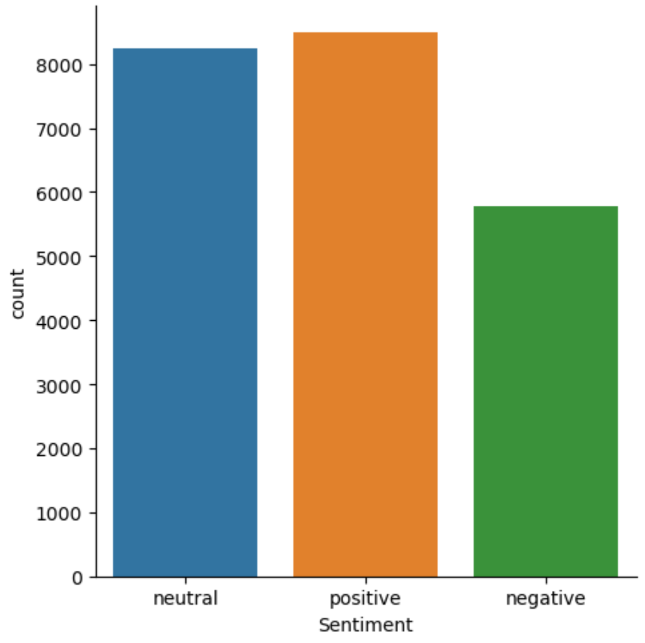
```
#Category Plot of target class

#showing how many neutral, positive, and negative tweets we have

import seaborn as sb

sb.catplot(x="Sentiment", kind="count", data=df)
```

```
    <seaborn.axisgrid.FacetGrid at 0x78c8d6bb3dd0>
```



Create a sequential model and evaluate on the test data

```
# necessary imports
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras import layers, models

from sklearn.preprocessing import LabelEncoder
import pickle
import numpy as np
import pandas as pd

# set seed for reproducibility
np.random.seed(1234)


#read in the csv
df = pd.read_csv('/kaggle/input/fifa-world-cup-2022-tweets/fifa_world_cup_2022_tweets.csv', header=0, usecols=[4,5], encoding='lat
#printing number of rows and columns in our dataframe
print('rows and columns:', df.shape)
```

```
#show the head of our dataframe
print(df.head())

     rows and columns: (22524, 2)
                                                    Tweet Sentiment
     0  What are we drinking today @TucanTribe \n@MadB...   neutral
     1  Amazing @CanadaSoccerEN  #WorldCup2022 launch ...  positive
     2  Worth reading while watching #WorldCup2022 htt...  positive
     3  Golden Maknae shinning bright\n\nhttps://t.co/...  positive
     4  If the BBC cares so much about human rights, h...  negative
```

Divide into train/test

```
# divide into train/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:  (18005, 2)
     test data size:  (4519, 2)
```

```
# set up X and Y
num_labels = 2
vocab_size = 25000
batch_size = 100

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

#our x will be the tweets
x_train = tokenizer.texts_to_matrix(train.Tweet, mode='tfidf')
x_test = tokenizer.texts_to_matrix(test.Tweet, mode='tfidf')

encoder = LabelEncoder()
encoder.fit(train.Sentiment)

#our y will be the sentiments
y_train = encoder.transform(train.Sentiment)
y_test = encoder.transform(test.Sentiment)

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

     train shapes: (18005, 25000) (18005,)
     test shapes: (4519, 25000) (4519,)
     test first five labels: [0 2 2 0 2]
```

```
# fit model
model = models.Sequential()
model.add(layers.Dense(32, input_dim=vocab_size, kernel_initializer='normal', activation='relu'))
model.add(layers.Dense(1, kernel_initializer='normal', activation='sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=10,
                    verbose=1,
                    validation_split=0.1)

     Epoch 1/10
     163/163 [==============================] - 4s 22ms/step - loss: -1.1626 - accuracy: 0.3636 - val_loss: -7.5650 - val_accuracy
     Epoch 2/10
     163/163 [==============================] - 2s 15ms/step - loss: -15.8673 - accuracy: 0.3976 - val_loss: -46.3624 - val_accura
     Epoch 3/10
     163/163 [==============================] - 2s 14ms/step - loss: -57.2973 - accuracy: 0.4182 - val_loss: -120.3263 - val_accu
     Epoch 4/10
     163/163 [==============================] - 2s 14ms/step - loss: -122.5447 - accuracy: 0.4225 - val_loss: -221.9988 - val_accu
     Epoch 5/10
```

```
163/163 [==============================] - 2s 14ms/step - loss: -207.8241 - accuracy: 0.4225 - val_loss: -349.4291 - val_accu
Epoch 6/10
163/163 [==============================] - 2s 14ms/step - loss: -314.2362 - accuracy: 0.4216 - val_loss: -504.4785 - val_accu
Epoch 7/10
163/163 [==============================] - 2s 14ms/step - loss: -438.8494 - accuracy: 0.4226 - val_loss: -681.8347 - val_accu
Epoch 8/10
163/163 [==============================] - 2s 14ms/step - loss: -579.0130 - accuracy: 0.4235 - val_loss: -876.3593 - val_accu
Epoch 9/10
163/163 [==============================] - 2s 14ms/step - loss: -735.8018 - accuracy: 0.4248 - val_loss: -1094.8547 - val_acc
Epoch 10/10
163/163 [==============================] - 2s 14ms/step - loss: -909.5621 - accuracy: 0.4245 - val_loss: -1334.7695 - val_acc
```

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

```
46/46 [==============================] - 0s 5ms/step - loss: -922.3441 - accuracy: 0.4012
Accuracy:  0.4011949598789215
```

```
print(score)
```

```
[-922.3440551757812, 0.4011949598789215]
```

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

```
142/142 [==============================] - 0s 2ms/step
```

```
pred[:10]
```

```
array([[1.],
       [1.],
       [1.],
       [0.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.]], dtype=float32)
```

```
pred_labels[:10]
```

```
[1, 1, 1, 0, 1, 1, 1, 1, 1, 1]
```

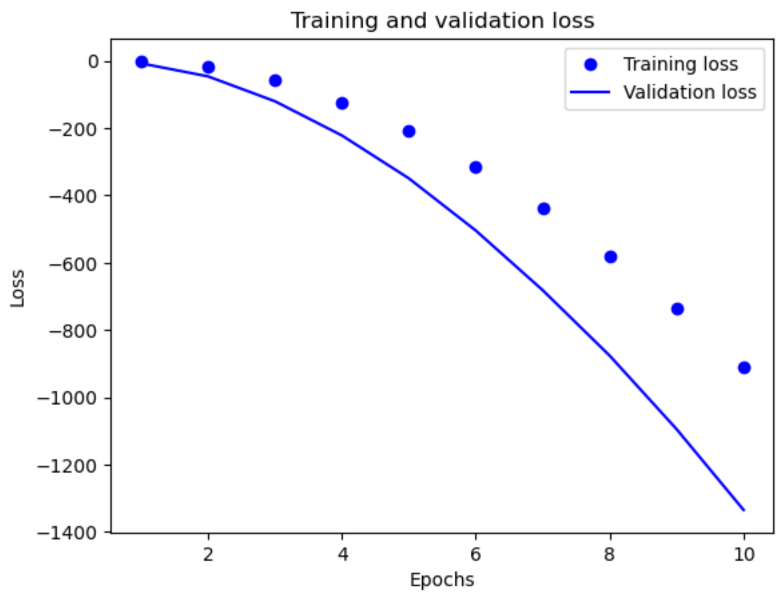
```
#print our accuracy, precision, recall, and f1 scores
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
print('accuracy score: ', accuracy_score(y_test, pred_labels))
print('precision score: ', precision_score(y_test, pred_labels, average='micro'))
print('recall score: ', recall_score(y_test, pred_labels, average='micro'))
print('f1 score: ', f1_score(y_test, pred_labels, average='micro'))
```

```
accuracy score:  0.4011949546359814
precision score:  0.4011949546359814
recall score:  0.4011949546359814
f1 score:  0.4011949546359815
```

```
# plot the training and validation loss
import matplotlib.pyplot as plt

loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(loss)+1)

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



Write up your analysis of the performance of various approaches

My sequential model was not very accurate, only getting up to 40 percent accuracy with my data set. I tried using different amounts of epochs, layers, and different batch sizes. I could probably get the accuracy higher with more experience with using Keras, but for now that's the accuracy I have.