Description of Data

The dataset that is being used are short texts that are identified as either jokes or as regular statements. The value of true means that the short text is a joke and if the value is false then the short text is just a regular statement. The total number of short texts are 200,000, and the number of true and false values are an even split of 100,000 each. Although it is great there is a lot of data, I unfortunately had to split it even further into a dataframe containing only 25000 values. Colab kept crashing and would not properly start with the values of 200,000. So the new data is split into 12,500 even of true and false. The models should be able to classify whether the text is a joke or not.

```
import urllib
import os
import pandas as pd
import pathlib
import zipfile
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras import layers, models
from sklearn.preprocessing import LabelEncoder
import pickle
import numpy as np
```

```
#getting file and dowloading
drive_url = 'https://utdallas.box.com/shared/static/e4p5l9eias6rqegn0d72mxpoklw8bnl9.zip'
file_name = 'Jokes.zip'

urllib.request.urlretrieve(drive_url, file_name)

os.listdir()
```

```
['.config',
 'glove',
 'glove.6B.zip',
 'dataset.csv',
 'Jokes.zip',
 'glove.6B.zip.1',
 'sample_data']
```

```
zip_ref = zipfile.ZipFile("Jokes.zip", "r")
zip_ref.extractall()
zip_ref.close()
```

```
Jokes_file = '/content/dataset.csv'

data = pd.read_csv(Jokes_file, encoding= 'unicode_escape')
df = pd.DataFrame(data)
df
```

	text	humor
0	Joe biden rules out 2020 bid: 'guys, i'm not r...	False
1	Watch: darvish gave hitter whiplash with slow ...	False
2	What do you call a turtle without its shell? d...	True
3	5 reasons the 2016 election feels so personal	False
4	Pasco police shot mexican migrant from behind,...	False
...	...	...
199995	Conor maynard seamlessly fits old-school r&b h...	False
199996	How to you make holy water? you boil the hell ...	True
199997	How many optometrists does it take to screw in...	True
199998	Mcdonald's will officially kick off all-day br...	False
199999	An irish man walks on the street and ignores a...	True

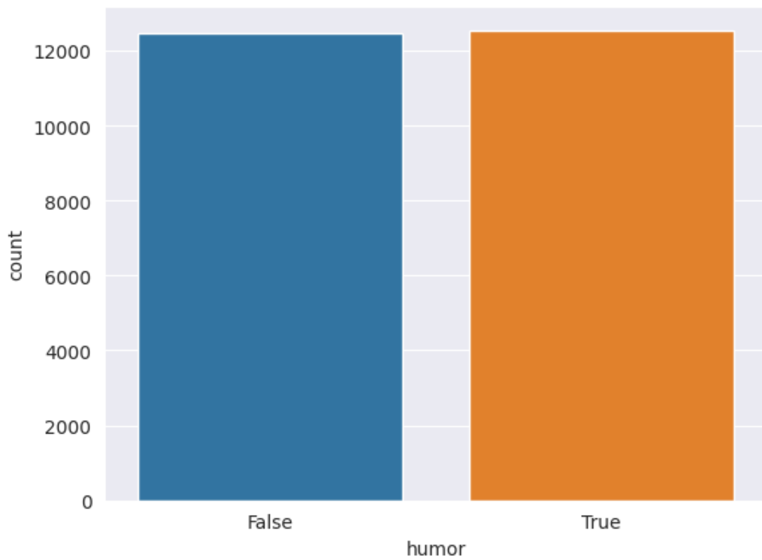
200000 rows × 2 columns

```
# splitting the data because it was too large
df_1 = df.iloc[:25000,:]
df_2 = df.iloc[25000:,:]
```

```
import seaborn as sns

sns.set_style('darkgrid')
sns.countplot(x ='humor', data = df_1)
```

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



Sequential Model

```
np.random.seed(1234)

#creating test and training data
i = np.random.rand(len(df_1)) < 0.75
train = df_1[i]
test = df_1[~i]
print("train data size: ", train.shape)
print("test data size: ", test.shape)
```

```
train data size:  (18710, 2)
test data size:  (6290, 2)
```

```
num_labels = 2
vocab_size = 25000
batch_size = 32

# 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.humor)
y_train = encoder.transform(train.humor)
y_test = encoder.transform(test.humor)

# 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:", y_test[:5])
print("test first five text:", x_test[:5])
```

```
train shapes: (18710, 25000) (18710,)
test shapes: (6290, 25000) (6290,)
test first five labels: [0 0 1 0 1]
test first five text: [[0.         1.39876921 0.         ... 0.         0.         0.         ]
```

```
 [0.         0.         0.         ... 0.         0.         0.        ]
 [0.         0.         0.         ... 0.         0.         0.        ]
 [0.         0.         0.         ... 0.         0.         0.        ]
 [0.         0.         1.46715988 ... 0.         0.         0.        ]]
```

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

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

history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=30,
                    verbose=1,
                    validation_split=0.1)
```

```
Epoch 1/30
527/527 [==============================] - 9s 15ms/step - loss: 0.4627 - accuracy: 0.8429 - val_loss: 0.2708 - val_accuracy:
Epoch 2/30
527/527 [==============================] - 10s 18ms/step - loss: 0.1434 - accuracy: 0.9735 - val_loss: 0.3330 - val_accuracy
Epoch 3/30
527/527 [==============================] - 9s 17ms/step - loss: 0.1058 - accuracy: 0.9853 - val_loss: 0.3729 - val_accuracy:
Epoch 4/30
527/527 [==============================] - 9s 16ms/step - loss: 0.0751 - accuracy: 0.9926 - val_loss: 0.4322 - val_accuracy:
Epoch 5/30
527/527 [==============================] - 12s 23ms/step - loss: 0.0635 - accuracy: 0.9954 - val_loss: 0.5038 - val_accuracy
Epoch 6/30
527/527 [==============================] - 10s 19ms/step - loss: 0.0595 - accuracy: 0.9960 - val_loss: 0.5037 - val_accuracy
Epoch 7/30
527/527 [==============================] - 9s 17ms/step - loss: 0.0571 - accuracy: 0.9964 - val_loss: 0.5458 - val_accuracy:
Epoch 8/30
527/527 [==============================] - 10s 20ms/step - loss: 0.0541 - accuracy: 0.9964 - val_loss: 0.5783 - val_accuracy
Epoch 9/30
527/527 [==============================] - 8s 15ms/step - loss: 0.0541 - accuracy: 0.9966 - val_loss: 0.5811 - val_accuracy:
Epoch 10/30
527/527 [==============================] - 10s 18ms/step - loss: 0.0509 - accuracy: 0.9966 - val_loss: 0.5787 - val_accuracy
Epoch 11/30
527/527 [==============================] - 8s 16ms/step - loss: 0.0714 - accuracy: 0.9911 - val_loss: 0.5749 - val_accuracy:
Epoch 12/30
527/527 [==============================] - 9s 18ms/step - loss: 0.0474 - accuracy: 0.9969 - val_loss: 0.6394 - val_accuracy:
Epoch 13/30
527/527 [==============================] - 9s 18ms/step - loss: 0.0459 - accuracy: 0.9970 - val_loss: 0.6254 - val_accuracy:
Epoch 14/30
527/527 [==============================] - 8s 16ms/step - loss: 0.0457 - accuracy: 0.9970 - val_loss: 0.6730 - val_accuracy:
Epoch 15/30
527/527 [==============================] - 10s 19ms/step - loss: 0.0457 - accuracy: 0.9970 - val_loss: 0.6534 - val_accuracy
Epoch 16/30
527/527 [==============================] - 8s 15ms/step - loss: 0.0457 - accuracy: 0.9970 - val_loss: 0.6418 - val_accuracy:
Epoch 17/30
527/527 [==============================] - 10s 18ms/step - loss: 0.0457 - accuracy: 0.9970 - val_loss: 0.6835 - val_accuracy
Epoch 18/30
527/527 [==============================] - 9s 16ms/step - loss: 0.0457 - accuracy: 0.9970 - val_loss: 0.6316 - val_accuracy:
Epoch 19/30
527/527 [==============================] - 9s 18ms/step - loss: 0.0457 - accuracy: 0.9970 - val_loss: 0.6837 - val_accuracy:
Epoch 20/30
527/527 [==============================] - 10s 19ms/step - loss: 0.0457 - accuracy: 0.9970 - val_loss: 0.6480 - val_accuracy
Epoch 21/30
527/527 [==============================] - 8s 15ms/step - loss: 0.0458 - accuracy: 0.9970 - val_loss: 0.6924 - val_accuracy:
Epoch 22/30
527/527 [==============================] - 9s 18ms/step - loss: 0.0460 - accuracy: 0.9970 - val_loss: 0.6879 - val_accuracy:
Epoch 23/30
527/527 [==============================] - 8s 15ms/step - loss: 0.0460 - accuracy: 0.9970 - val_loss: 0.7894 - val_accuracy:
Epoch 24/30
527/527 [==============================] - 10s 19ms/step - loss: 0.0458 - accuracy: 0.9970 - val_loss: 0.7599 - val_accuracy
Epoch 25/30
527/527 [==============================] - 9s 16ms/step - loss: 0.0457 - accuracy: 0.9970 - val_loss: 0.7530 - val_accuracy:
Epoch 26/30
527/527 [==============================] - 9s 17ms/step - loss: 0.0457 - accuracy: 0.9970 - val_loss: 0.7886 - val_accuracy:
Epoch 27/30
527/527 [==============================] - 10s 18ms/step - loss: 0.0456 - accuracy: 0.9970 - val_loss: 0.7910 - val_accuracy
Epoch 28/30
527/527 [==============================] - 8s 15ms/step - loss: 0.0456 - accuracy: 0.9970 - val_loss: 0.8165 - val_accuracy:
Epoch 29/30
527/527 [==============================] - 10s 18ms/step - loss: 0.0456 - accuracy: 0.9970 - val_loss: 0.7758 - val_accuracy
```

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

```
197/197 [==============================] - 1s 5ms/step - loss: 0.7687 - accuracy: 0.9108
Accuracy:  0.9108108282089233
```

```
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

pred = model.predict(x_test)
pred_labels = [1 if p>0.5 else 0 for p in pred]
print(classification_report(y_test, pred_labels))
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))
```

```
197/197 [==============================] - 1s 6ms/step
              precision    recall  f1-score   support

           0       0.92      0.90      0.91      3115
           1       0.90      0.92      0.91      3175

    accuracy                           0.91      6290
   macro avg       0.91      0.91      0.91      6290
weighted avg       0.91      0.91      0.91      6290

accuracy score:  0.9108108108108108
precision score:  0.9048946716232962
recall score:  0.92
f1 score:  0.9123848196158051
```

Sequential Model Report

90% relu sigmoid adam 81% selu relu sgd 91% relu relu adam

The sequential model worked fairly well when the data was broken up into a smaller dataset. Before when the data was large in the 200,000 values, it had a difficult time trying to load everything and kept on crashing. I tried out different types of optimizers as well as activation combinations and recorded their scores. A few of the notable ones were the original relu sigmoid activations with the adam optimizer which achieved an accuracy of 90%. I then transitioned it into a selu relu activation with and sgd optimizer and the accuracy and overall model dropped to 81%. Finally I did a relu relu activation with an adam optimizer and had the highest accuracy of 91%. It was also interesting to play around with the batch size as 32 got the highest accuracy as well. The model did fairly well when guessing whether a text was humor or not and it had higher precision for not humor and higher recall for humor. One thing that I want to try in the future is to add more layers with different activations and see if that may change any of the accuracies or if it may bring it down.

RNN

```
from tensorflow.keras import preprocessing

max_features = 10000
maxlen = 500
batch_size = 32

#preprocess data to fit for model
train_data = preprocessing.sequence.pad_sequences(x_train, maxlen=maxlen)
test_data = preprocessing.sequence.pad_sequences(x_test, maxlen=maxlen)


#adding layers for the model
model_RNN = models.Sequential()
model_RNN.add(layers.Embedding(max_features, 32))
model_RNN.add(layers.SimpleRNN(32))
model_RNN.add(layers.Dense(1, activation='sigmoid')) #sigmoid #relu

model_RNN.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_2 (Embedding)     (None, None, 32)          320000

 simple_rnn_1 (SimpleRNN)    (None, 32)                2080
```

```
 dense_7 (Dense)             (None, 1)                 33

=================================================================
Total params: 322,113
Trainable params: 322,113
Non-trainable params: 0
_________________________________________________________________
```

```
#model is compiled and running
model_RNN.compile(optimizer='adam', #rmsprop
              loss='binary_crossentropy',
              metrics=['accuracy'])

history = model_RNN.fit(train_data,
                    y_train,
                    epochs=10,
                    batch_size=128, #128
                    validation_split=0.1)
```

```
Epoch 1/10
132/132 [==============================] - 35s 235ms/step - loss: 0.6933 - accuracy: 0.5000 - val_loss: 0.6937 - val_accurac
Epoch 2/10
132/132 [==============================] - 27s 202ms/step - loss: 0.6933 - accuracy: 0.4995 - val_loss: 0.6935 - val_accurac
Epoch 3/10
132/132 [==============================] - 27s 202ms/step - loss: 0.6932 - accuracy: 0.4951 - val_loss: 0.6941 - val_accurac
Epoch 4/10
132/132 [==============================] - 29s 217ms/step - loss: 0.6932 - accuracy: 0.5006 - val_loss: 0.6928 - val_accurac
Epoch 5/10
132/132 [==============================] - 36s 272ms/step - loss: 0.6934 - accuracy: 0.4940 - val_loss: 0.6928 - val_accurac
Epoch 6/10
132/132 [==============================] - 28s 214ms/step - loss: 0.6933 - accuracy: 0.4997 - val_loss: 0.6930 - val_accurac
Epoch 7/10
132/132 [==============================] - 34s 259ms/step - loss: 0.6933 - accuracy: 0.4972 - val_loss: 0.6940 - val_accurac
Epoch 8/10
132/132 [==============================] - 28s 214ms/step - loss: 0.6933 - accuracy: 0.5013 - val_loss: 0.6937 - val_accurac
Epoch 9/10
132/132 [==============================] - 29s 223ms/step - loss: 0.6932 - accuracy: 0.4997 - val_loss: 0.6939 - val_accurac
Epoch 10/10
132/132 [==============================] - 41s 309ms/step - loss: 0.6932 - accuracy: 0.5016 - val_loss: 0.6934 - val_accurac
```

```
from sklearn.metrics import classification_report

pred_RNN = model_RNN.predict(test_data)
pred_labels_RNN = [1.0 if p>= 0.5 else 0.0 for p in pred_RNN]
print(classification_report(y_test, pred_labels_RNN))
```

```
197/197 [==============================] - 12s 61ms/step
              precision    recall  f1-score   support

           0       0.50      1.00      0.66      3115
           1       0.00      0.00      0.00      3175

    accuracy                           0.50      6290
   macro avg       0.25      0.50      0.33      6290
weighted avg       0.25      0.50      0.33      6290

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

RNN Report

When doing the RNN although I tried out different activation functions, optimizers as well as batch sizes the accuracy for the model still came out to be 50% making it not only a weak classifier but no better than flipping a coin to see what the value may be. Even when the activation was relu, sigmoid, or selu it kept on giving the accuracy of 50%. In comparison to the sequential model RNN does poorly and does not really have a strong case to be a better model. I would like to see if adding more layers to the RNN might change the model or accuracy or if different types of data may affect the RNN. Perhaps if the data was more complex it may perform better.

LSTM

```
#adding layers for the lstm
model_LSTM = models.Sequential()
model_LSTM.add(layers.Embedding(max_features, 32))
model_LSTM.add(layers.LSTM(32))
model_LSTM.add(layers.Dense(1, activation='relu')) # sigmoid

model_LSTM.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding_4 (Embedding)     (None, None, 32)          320000    
                                                                 
 lstm_1 (LSTM)               (None, 32)                8320      
                                                                 
 dense_9 (Dense)             (None, 1)                 33        
                                                                 
=================================================================
Total params: 328,353
Trainable params: 328,353
Non-trainable params: 0
_________________________________________________________________
```

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

history = model_LSTM.fit(train_data,
                    y_train,
                    epochs=10,
                    batch_size=128,
                    validation_split=0.1)
```

```
Epoch 1/10
132/132 [==============================] - 58s 410ms/step - loss: 7.6891 - accuracy: 0.5015 - val_loss: 8.0464 - val_accurac
Epoch 2/10
132/132 [==============================] - 57s 429ms/step - loss: 7.6891 - accuracy: 0.5015 - val_loss: 8.0464 - val_accurac
Epoch 3/10
132/132 [==============================] - 74s 564ms/step - loss: 7.6891 - accuracy: 0.5015 - val_loss: 8.0464 - val_accurac
Epoch 4/10
132/132 [==============================] - 58s 441ms/step - loss: 7.6891 - accuracy: 0.5015 - val_loss: 8.0464 - val_accurac
Epoch 5/10
132/132 [==============================] - 65s 494ms/step - loss: 7.6891 - accuracy: 0.5015 - val_loss: 8.0464 - val_accurac
Epoch 6/10
132/132 [==============================] - 61s 462ms/step - loss: 7.6891 - accuracy: 0.5015 - val_loss: 8.0464 - val_accurac
Epoch 7/10
132/132 [==============================] - 55s 420ms/step - loss: 7.6891 - accuracy: 0.5015 - val_loss: 8.0464 - val_accurac
Epoch 8/10
132/132 [==============================] - 56s 423ms/step - loss: 7.6891 - accuracy: 0.5015 - val_loss: 8.0464 - val_accurac
Epoch 9/10
132/132 [==============================] - 55s 416ms/step - loss: 7.6891 - accuracy: 0.5015 - val_loss: 8.0464 - val_accurac
Epoch 10/10
132/132 [==============================] - 60s 452ms/step - loss: 7.6891 - accuracy: 0.5015 - val_loss: 8.0464 - val_accurac
```

```
pred_LSTM = model_LSTM.predict(test_data)
pred_labels_LSTM = [1.0 if i>= 0.5 else 0.0 for i in pred_LSTM]
print(classification_report(y_test, pred_labels_LSTM))
```

```
197/197 [==============================] - 12s 58ms/step
              precision    recall  f1-score   support

           0       0.50      1.00      0.66      3115
           1       0.00      0.00      0.00      3175

    accuracy                           0.50      6290
   macro avg       0.25      0.50      0.33      6290
weighted avg       0.25      0.50      0.33      6290

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

LSTM Report

Just like the RNN model, the LSTM model does not beat the sequential model. It has an accuracy of 50% as well even if the activation functions and optimizers are changed. It can be labeled as a weak classifier, but it still is no better than flipping a coin to see what the value should be. Like the RNN model I am curious to see if adding more layers or changing the type of data and values it is may alter the results and get us a better accuracy and description.

GRU

```
#adding layeers for the GRU
model_GRU = models.Sequential()
model_GRU.add(layers.Embedding(max_features, 32))
model_GRU.add(layers.GRU(32))
model_GRU.add(layers.Dense(1, activation='relu')) #sigmord


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

history = model_GRU.fit(train_data,
                    y_train,
                    epochs=10,
                    batch_size=128,
                    validation_split=0.1)
```

```
Epoch 1/10
132/132 [==============================] - 57s 379ms/step - loss: 0.7379 - accuracy: 0.4999 - val_loss: 0.6927 - val_accurac
Epoch 2/10
132/132 [==============================] - 59s 449ms/step - loss: 0.6934 - accuracy: 0.4983 - val_loss: 0.6930 - val_accurac
Epoch 3/10
132/132 [==============================] - 52s 391ms/step - loss: 0.6935 - accuracy: 0.5057 - val_loss: 0.6980 - val_accurac
Epoch 4/10
132/132 [==============================] - 51s 390ms/step - loss: 0.6941 - accuracy: 0.4934 - val_loss: 0.6945 - val_accurac
Epoch 5/10
132/132 [==============================] - 52s 391ms/step - loss: 0.6940 - accuracy: 0.4942 - val_loss: 0.6936 - val_accurac
Epoch 6/10
132/132 [==============================] - 50s 378ms/step - loss: 0.6938 - accuracy: 0.4988 - val_loss: 0.6931 - val_accurac
Epoch 7/10
132/132 [==============================] - 52s 391ms/step - loss: 0.6951 - accuracy: 0.4941 - val_loss: 0.6922 - val_accurac
Epoch 8/10
132/132 [==============================] - 56s 422ms/step - loss: 0.6941 - accuracy: 0.4977 - val_loss: 0.6958 - val_accurac
Epoch 9/10
132/132 [==============================] - 52s 396ms/step - loss: 0.6936 - accuracy: 0.5052 - val_loss: 0.7018 - val_accurac
Epoch 10/10
132/132 [==============================] - 51s 384ms/step - loss: 0.6945 - accuracy: 0.4952 - val_loss: 0.6931 - val_accurac
```

```
pred_GRU = model_GRU.predict(test_data)
pred_labels_GRU = [1.0 if p>= 0.5 else 0.0 for p in pred_GRU]
print(classification_report(y_test, pred_labels_GRU))
```

```
197/197 [==============================] - 11s 56ms/step
              precision    recall  f1-score   support

           0       0.00      0.00      0.00      3115
           1       0.50      1.00      0.67      3175

    accuracy                           0.50      6290
   macro avg       0.25      0.50      0.34      6290
weighted avg       0.25      0.50      0.34      6290

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

GRU Report

With the gru it is very similar to the RNN and LSTM with the classifier being a weak classifier with a low accuracy rate. I think one thing moving forward I may need to alter is to make sure that there should be more layers perhaps allowing for the model to do a better job in classifying. It is interesting however because in a previous project using images, the RNN, LSTM, and GRU did a fairly strong job so it may be that the data is not complex enough as the models may do a better job with more complex data.

CNN

```
#adding layers for CNN
model_CNN = models.Sequential()
model_CNN.add(layers.Embedding(max_features, 128, input_length=maxlen))
model_CNN.add(layers.Conv1D(32, 7, activation='relu'))
model_CNN.add(layers.MaxPooling1D(5))
model_CNN.add(layers.Conv1D(32, 7, activation='sigmoid'))
model_CNN.add(layers.GlobalMaxPooling1D())
model_CNN.add(layers.Dense(1))

model_CNN.summary()
```

```
Model: "sequential_9"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding_9 (Embedding)     (None, 500, 128)          1280000   
                                                                 
 conv1d_8 (Conv1D)           (None, 494, 32)           28704     
                                                                 
 max_pooling1d_5 (MaxPooling  (None, 98, 32)           0         
 1D)                                                             
                                                                 
 conv1d_9 (Conv1D)           (None, 92, 32)            7200      
                                                                 
 global_max_pooling1d_3 (Glo  (None, 32)               0         
 balMaxPooling1D)                                                
                                                                 
 dense_15 (Dense)            (None, 1)                 33        
                                                                 
=================================================================
Total params: 1,315,937
Trainable params: 1,315,937
Non-trainable params: 0
_________________________________________________________________
```

```
model_CNN.compile(optimizer=tf.keras.optimizers.RMSprop(lr=1e-4),  # set learning rate
              loss='binary_crossentropy',
              metrics=['accuracy'])

history = model_CNN.fit(train_data,
                    y_train,
                    epochs=10,
                    batch_size=128,
                    validation_split=0.1)
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.op
Epoch 1/10
132/132 [==============================] - 80s 603ms/step - loss: 0.6938 - accuracy: 0.5067 - val_loss: 0.7012 - val_accurac
Epoch 2/10
132/132 [==============================] - 80s 604ms/step - loss: 0.6939 - accuracy: 0.5007 - val_loss: 0.6922 - val_accurac
Epoch 3/10
132/132 [==============================] - 78s 592ms/step - loss: 0.6940 - accuracy: 0.5003 - val_loss: 0.6943 - val_accurac
Epoch 4/10
132/132 [==============================] - 79s 600ms/step - loss: 0.6935 - accuracy: 0.5046 - val_loss: 0.6994 - val_accurac
Epoch 5/10
132/132 [==============================] - 79s 599ms/step - loss: 0.6941 - accuracy: 0.4965 - val_loss: 0.6930 - val_accurac
Epoch 6/10
132/132 [==============================] - 79s 596ms/step - loss: 0.6939 - accuracy: 0.5030 - val_loss: 0.6927 - val_accurac
Epoch 7/10
132/132 [==============================] - 79s 597ms/step - loss: 0.6940 - accuracy: 0.5064 - val_loss: 0.6991 - val_accurac
Epoch 8/10
132/132 [==============================] - 79s 600ms/step - loss: 0.6941 - accuracy: 0.5015 - val_loss: 0.6928 - val_accurac
Epoch 9/10
132/132 [==============================] - 78s 590ms/step - loss: 0.6939 - accuracy: 0.5025 - val_loss: 0.6922 - val_accurac
Epoch 10/10
132/132 [==============================] - 79s 602ms/step - loss: 0.6939 - accuracy: 0.5057 - val_loss: 0.6922 - val_accurac
```

```
pred_CNN = model_CNN.predict(test_data)
pred_labels_CNN = [1.0 if p>= 0.5 else 0.0 for p in pred_CNN]
print(classification_report(y_test, pred_labels_CNN))
```

```
197/197 [==============================] - 7s 34ms/step
              precision    recall  f1-score   support

           0       0.00      0.00      0.00      3115
```

```
           1       0.50      1.00      0.67      3175

    accuracy                           0.50      6290
   macro avg       0.25      0.50      0.34      6290
weighted avg       0.25      0.50      0.34      6290

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

CNN Report

Similar to the other deep learning models the CNN once again had a score that was of 50% just like the other models. Even with addes layers that differ or did not differ with relu and sigmoid the accuracy did not go up. The number of epochs may have to go up possibly allowing for the model to learn better, or transfer learning could be used to allow for a better accuracy.

Embedding

```
maxlen = 20

#preprocessing data
train_data = preprocessing.sequence.pad_sequences(x_train, maxlen=maxlen)
test_data = preprocessing.sequence.pad_sequences(x_test, maxlen=maxlen)



model_EMB = models.Sequential()
model_EMB.add(layers.Embedding(max_features, 8, input_length=maxlen))
model_EMB.add(layers.Flatten())
model_EMB.add(layers.Dense(16, activation='relu'))
model_EMB.add(layers.Dense(1, activation='sigmoid'))

model_EMB.summary()
```

```
Model: "sequential_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_10 (Embedding)    (None, 20, 8)             80000

 flatten (Flatten)           (None, 160)               0

 dense_16 (Dense)            (None, 16)                2576

 dense_17 (Dense)            (None, 1)                 17

=================================================================
Total params: 82,593
Trainable params: 82,593
Non-trainable params: 0
_________________________________________________________________
```

```
model_EMB.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
history = model_EMB.fit(train_data, y_train, epochs=10, batch_size=32, validation_split=0.1)
```

```
Epoch 1/10
527/527 [==============================] - 2s 3ms/step - loss: 0.6933 - acc: 0.4997 - val_loss: 0.6934 - val_acc: 0.4784
Epoch 2/10
527/527 [==============================] - 1s 2ms/step - loss: 0.6932 - acc: 0.4982 - val_loss: 0.6932 - val_acc: 0.4784
Epoch 3/10
527/527 [==============================] - 1s 3ms/step - loss: 0.6932 - acc: 0.4991 - val_loss: 0.6932 - val_acc: 0.4784
Epoch 4/10
527/527 [==============================] - 1s 3ms/step - loss: 0.6932 - acc: 0.4966 - val_loss: 0.6934 - val_acc: 0.4784
Epoch 5/10
527/527 [==============================] - 2s 3ms/step - loss: 0.6932 - acc: 0.5000 - val_loss: 0.6932 - val_acc: 0.4784
Epoch 6/10
527/527 [==============================] - 2s 4ms/step - loss: 0.6932 - acc: 0.4996 - val_loss: 0.6932 - val_acc: 0.4784
Epoch 7/10
527/527 [==============================] - 2s 4ms/step - loss: 0.6932 - acc: 0.4956 - val_loss: 0.6932 - val_acc: 0.4784
Epoch 8/10
527/527 [==============================] - 1s 3ms/step - loss: 0.6932 - acc: 0.5010 - val_loss: 0.6936 - val_acc: 0.4784
Epoch 9/10
```

```
527/527 [==============================] - 1s 3ms/step - loss: 0.6932 - acc: 0.5006 - val_loss: 0.6935 - val_acc: 0.4784
Epoch 10/10
527/527 [==============================] - 1s 2ms/step - loss: 0.6932 - acc: 0.4976 - val_loss: 0.6936 - val_acc: 0.4784
```

```
pred_EMB = model_EMB.predict(test_data)
pred_labels_EMB = [1.0 if p>= 0.5 else 0.0 for p in pred_EMB]
print(classification_report(y_test, pred_labels_EMB))
```

```
197/197 [==============================] - 0s 2ms/step
              precision    recall  f1-score   support

           0       0.50      1.00      0.66      3115
           1       0.00      0.00      0.00      3175

    accuracy                           0.50      6290
   macro avg       0.25      0.50      0.33      6290
weighted avg       0.25      0.50      0.33      6290

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

Embedding Layer

```
from tensorflow.keras.layers.experimental.preprocessing import TextVectorization

vectorizer = TextVectorization(max_tokens=20000, output_sequence_length=200)
text_ds = tf.data.Dataset.from_tensor_slices(train.text).batch(128)
vectorizer.adapt(text_ds)
```

```
voc = vectorizer.get_vocabulary()
word_index = dict(zip(voc, range(len(voc))))

test_words = ["the", "cat", "sat", "on", "the", "mat"]
[word_index[w] for w in test_words]
```

```
[2, 437, 4548, 14, 2, 9354]
```

```
EMBEDDING_DIM = 128
MAX_SEQUENCE_LENGTH = 200

embedding_layer = layers.Embedding(len(word_index) + 1,
                            EMBEDDING_DIM,
                            input_length=MAX_SEQUENCE_LENGTH)
```

```
from tensorflow import keras

#adding new layers for the model
int_sequences_input = keras.Input(shape=(None,), dtype="int64")
embedded_sequences = embedding_layer(int_sequences_input)
x = layers.Conv1D(128, 5, activation="relu")(embedded_sequences)
x = layers.MaxPooling1D(5)(x)
x = layers.Conv1D(128, 5, activation="sigmoid")(x) #relu
x = layers.MaxPooling1D(5)(x)
x = layers.Conv1D(128, 5, activation="relu")(x)
x = layers.GlobalMaxPooling1D()(x)
x = layers.Dense(128, activation="sigmoid")(x) #relu
x = layers.Dropout(0.5)(x)
preds = layers.Dense(2 , activation="softmax")(x)
model_EMB2 = keras.Model(int_sequences_input, preds)
model_EMB2.summary()
```

```
Model: "model_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_5 (InputLayer)        [(None, None)]            0

 embedding_11 (Embedding)    (None, None, 128)         2560128

 conv1d_10 (Conv1D)          (None, None, 128)         82048
```

```
 max_pooling1d_6 (MaxPooling  (None, None, 128)        0
 1D)

 conv1d_11 (Conv1D)          (None, None, 128)         82048

 max_pooling1d_7 (MaxPooling  (None, None, 128)        0
 1D)

 conv1d_12 (Conv1D)          (None, None, 128)         82048

 global_max_pooling1d_4 (Glo  (None, 128)              0
 balMaxPooling1D)

 dense_18 (Dense)            (None, 128)               16512

 dropout_2 (Dropout)         (None, 128)               0

 dense_19 (Dense)            (None, 2)                 258

=================================================================
Total params: 2,823,042
Trainable params: 2,823,042
Non-trainable params: 0
_________________________________________________________________
```

```
x_train = vectorizer(np.array([[s] for s in train.text])).numpy()
#x_val = vectorizer(np.array([[s] for s in val_samples])).numpy()

y_train = np.array(train.humor)
#y_val = np.array(val_labels)

model_EMB2.compile(
    loss="sparse_categorical_crossentropy", optimizer="adam", metrics=["acc"] #rmsprop
)
model_EMB2.fit(x_train, y_train, batch_size=128, epochs=20, validation_split = 0.1)#validation_data=(x_val, y_val))
```

```
Epoch 1/20
132/132 [==============================] - 97s 675ms/step - loss: 0.0024 - acc: 0.9993 - val_loss: 0.6006 - val_acc: 0.9209
Epoch 2/20
132/132 [==============================] - 85s 641ms/step - loss: 0.0059 - acc: 0.9985 - val_loss: 0.5703 - val_acc: 0.9150
Epoch 3/20
132/132 [==============================] - 71s 540ms/step - loss: 0.0025 - acc: 0.9991 - val_loss: 0.5981 - val_acc: 0.9134
Epoch 4/20
132/132 [==============================] - 78s 593ms/step - loss: 0.0011 - acc: 0.9998 - val_loss: 0.5591 - val_acc: 0.9161
Epoch 5/20
132/132 [==============================] - 75s 570ms/step - loss: 5.6568e-04 - acc: 0.9999 - val_loss: 0.5933 - val_acc: 0.9
Epoch 6/20
132/132 [==============================] - 65s 494ms/step - loss: 5.8539e-04 - acc: 0.9999 - val_loss: 0.6035 - val_acc: 0.9
Epoch 7/20
132/132 [==============================] - 75s 572ms/step - loss: 7.3731e-04 - acc: 0.9999 - val_loss: 0.6107 - val_acc: 0.9
Epoch 8/20
132/132 [==============================] - 81s 614ms/step - loss: 8.0288e-04 - acc: 0.9999 - val_loss: 0.6225 - val_acc: 0.9
Epoch 9/20
132/132 [==============================] - 66s 498ms/step - loss: 6.7367e-04 - acc: 0.9999 - val_loss: 0.6333 - val_acc: 0.9
Epoch 10/20
132/132 [==============================] - 64s 483ms/step - loss: 6.9573e-04 - acc: 0.9998 - val_loss: 0.5933 - val_acc: 0.9
Epoch 11/20
132/132 [==============================] - 63s 478ms/step - loss: 0.0055 - acc: 0.9989 - val_loss: 0.4716 - val_acc: 0.9113
Epoch 12/20
132/132 [==============================] - 63s 482ms/step - loss: 0.0060 - acc: 0.9981 - val_loss: 0.5121 - val_acc: 0.9091
Epoch 13/20
132/132 [==============================] - 62s 473ms/step - loss: 0.0024 - acc: 0.9994 - val_loss: 0.5459 - val_acc: 0.9091
Epoch 14/20
132/132 [==============================] - 64s 484ms/step - loss: 6.8507e-04 - acc: 0.9999 - val_loss: 0.6114 - val_acc: 0.9
Epoch 15/20
132/132 [==============================] - 63s 476ms/step - loss: 8.6429e-05 - acc: 1.0000 - val_loss: 0.6946 - val_acc: 0.9
Epoch 16/20
132/132 [==============================] - 64s 482ms/step - loss: 3.4724e-05 - acc: 1.0000 - val_loss: 0.7127 - val_acc: 0.9
Epoch 17/20
132/132 [==============================] - 63s 474ms/step - loss: 2.5603e-05 - acc: 1.0000 - val_loss: 0.7309 - val_acc: 0.9
Epoch 18/20
132/132 [==============================] - 63s 478ms/step - loss: 2.1004e-05 - acc: 1.0000 - val_loss: 0.7525 - val_acc: 0.9
Epoch 19/20
132/132 [==============================] - 63s 475ms/step - loss: 1.6997e-05 - acc: 1.0000 - val_loss: 0.7661 - val_acc: 0.9
Epoch 20/20
132/132 [==============================] - 64s 482ms/step - loss: 1.4141e-05 - acc: 1.0000 - val_loss: 0.7774 - val_acc: 0.9
<keras.callbacks.History at 0x7f49cb464d00>
```

```
class_names = []
class_names.append("True")
```

```
class_names.append("False")

#allow model to take in text and output class
string_input = keras.Input(shape=(1,), dtype="string")
x = vectorizer(string_input)
preds = model_EMB2(x)
end_to_end_model = keras.Model(string_input, preds)

probabilities = end_to_end_model.predict(
    [["this message is about funny text and humor"]]
)

class_names[np.argmax(probabilities[0])]
```

```
1/1 [==============================] - 0s 187ms/step
'True'
```

```
from sklearn.metrics import classification_report

test_x = vectorizer(np.array([[s] for s in test.text])).numpy()

preds_EMB2 = model_EMB2.predict(test_x)
pred_labels_EMB2 = [np.argmax(p) for p in preds_EMB2]

print(classification_report(test.humor, pred_labels_EMB2))
```

```
197/197 [==============================] - 8s 41ms/step
              precision    recall  f1-score   support

       False       0.92      0.92      0.92      3115
        True       0.92      0.92      0.92      3175

    accuracy                           0.92      6290
   macro avg       0.92      0.92      0.92      6290
weighted avg       0.92      0.92      0.92      6290
```

GloVe

```
vectorizer = TextVectorization(max_tokens=20000, output_sequence_length=200)
text_ds = tf.data.Dataset.from_tensor_slices(train.text).batch(128)
vectorizer.adapt(text_ds)

voc = vectorizer.get_vocabulary()
word_index = dict(zip(voc, range(len(voc))))
```

```
drive_url = 'https://utdallas.box.com/shared/static/ylqadgjzkrcwho4yzgn5huok58adhmov.zip'
file_name = 'glove.zip'

urllib.request.urlretrieve(drive_url, file_name)

os.listdir()
```

```
['.config',
 'glove',
 'glove.6B.zip',
 'glove.6B.100d.txt',
 'dataset.csv',
 'glove.6B.50d.txt',
 'glove.6B.200d.txt',
 'glove.zip',
 'Jokes.zip',
 'glove.6B.zip.1',
 'sample_data']
```

```
zip_ref = zipfile.ZipFile("/content/glove.zip", "r")
zip_ref.extractall()
zip_ref.close()
```

```
import os
```

```
#path_to_glove_file = os.path.join(
```

```
#    os.path.expanduser("~"), ".glove.6B.100d.txt"#".keras/datasets/glove.6B/glove.6B.100d.txt"
#)

#path for glove file
path_to_glove_file = "/content/glove.6B.200d.txt"
embeddings_index = {}
with open(path_to_glove_file) as f:
    for line in f:
        word, coefs = line.split(maxsplit=1)
        coefs = np.fromstring(coefs, "f", sep=" ")
        embeddings_index[word] = coefs

print("Found %s word vectors." % len(embeddings_index))
```

```
Found 400000 word vectors.
```

```
num_tokens = len(voc) + 2
embedding_dim = 200#100
hits = 0
misses = 0

# Prepare embedding matrix
embedding_matrix = np.zeros((num_tokens, embedding_dim))
for word, i in word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        # Words not found in embedding index will be all-zeros.
        # This includes the representation for "padding" and "OOV"
        embedding_matrix[i] = embedding_vector
        hits += 1
    else:
        misses += 1
print("Converted %d words (%d misses)" % (hits, misses))
```

```
Converted 17445 words (2555 misses)
```

```
from tensorflow.keras.layers import Embedding

embedding_layer = Embedding(
    num_tokens,
    embedding_dim,
    embeddings_initializer=keras.initializers.Constant(embedding_matrix),
    trainable=False,
)
```

```
from tensorflow.keras import layers

#layers to the model are being added
int_sequences_input = keras.Input(shape=(None,), dtype="int64")
embedded_sequences = embedding_layer(int_sequences_input)
x = layers.Conv1D(128, 5, activation="relu")(embedded_sequences)
x = layers.MaxPooling1D(5)(x)
x = layers.Conv1D(128, 5, activation="relu")(x) #relu
x = layers.MaxPooling1D(5)(x)
x = layers.Conv1D(128, 5, activation="relu")(x)
x = layers.GlobalMaxPooling1D()(x)
x = layers.Dense(128, activation="relu")(x) #relu
x = layers.Dropout(0.5)(x)
preds = layers.Dense(len(class_names), activation="softmax")(x)
model_glove = keras.Model(int_sequences_input, preds)
model_glove.summary()
```

```
Model: "model_9"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_10 (InputLayer)       [(None, None)]            0

 embedding_13 (Embedding)    (None, None, 200)         4000400

 conv1d_19 (Conv1D)          (None, None, 128)         128128

 max_pooling1d_12 (MaxPoolin  (None, None, 128)        0
 g1D)

 conv1d_20 (Conv1D)          (None, None, 128)         82048
```

```
 max_pooling1d_13 (MaxPoolin  (None, None, 128)        0
 g1D)

 conv1d_21 (Conv1D)          (None, None, 128)         82048

 global_max_pooling1d_7 (Glo  (None, 128)              0
 balMaxPooling1D)

 dense_24 (Dense)            (None, 128)               16512

 dropout_5 (Dropout)         (None, 128)               0

 dense_25 (Dense)            (None, 2)                 258

=================================================================
Total params: 4,309,394
Trainable params: 308,994
Non-trainable params: 4,000,400
_________________________________________________________________
```

```
x_train = vectorizer(np.array([[s] for s in train.text])).numpy()
#x_val = vectorizer(np.array([[s] for s in val_samples])).numpy()

y_train = np.array(train.humor)
#y_val = np.array(val_labels)


model_glove.compile(
    loss="sparse_categorical_crossentropy", optimizer="rmsprop", metrics=["acc"] #rmsprop
)
model_glove.fit(x_train, y_train, batch_size=128, epochs=20, validation_split = 0.1)
```

```
Epoch 1/20
132/132 [==============================] - 86s 608ms/step - loss: 0.0088 - acc: 0.9967 - val_loss: 0.8120 - val_acc: 0.8979
Epoch 2/20
132/132 [==============================] - 104s 792ms/step - loss: 0.0088 - acc: 0.9973 - val_loss: 0.6913 - val_acc: 0.9081
Epoch 3/20
132/132 [==============================] - 88s 664ms/step - loss: 0.0064 - acc: 0.9976 - val_loss: 0.8172 - val_acc: 0.9011
Epoch 4/20
132/132 [==============================] - 64s 483ms/step - loss: 0.0104 - acc: 0.9966 - val_loss: 0.8322 - val_acc: 0.9006
Epoch 5/20
132/132 [==============================] - 60s 454ms/step - loss: 0.0058 - acc: 0.9983 - val_loss: 0.8434 - val_acc: 0.9102
Epoch 6/20
132/132 [==============================] - 59s 445ms/step - loss: 0.0059 - acc: 0.9981 - val_loss: 0.8334 - val_acc: 0.9102
Epoch 7/20
132/132 [==============================] - 59s 447ms/step - loss: 0.0080 - acc: 0.9985 - val_loss: 0.7998 - val_acc: 0.9043
Epoch 8/20
132/132 [==============================] - 58s 441ms/step - loss: 0.0049 - acc: 0.9984 - val_loss: 0.8319 - val_acc: 0.9081
Epoch 9/20
132/132 [==============================] - 60s 452ms/step - loss: 0.0058 - acc: 0.9983 - val_loss: 1.2450 - val_acc: 0.8840
Epoch 10/20
132/132 [==============================] - 58s 442ms/step - loss: 0.0049 - acc: 0.9989 - val_loss: 0.8252 - val_acc: 0.9075
Epoch 11/20
132/132 [==============================] - 59s 449ms/step - loss: 0.0040 - acc: 0.9988 - val_loss: 0.9338 - val_acc: 0.9059
Epoch 12/20
132/132 [==============================] - 58s 437ms/step - loss: 0.0029 - acc: 0.9989 - val_loss: 0.9911 - val_acc: 0.9113
Epoch 13/20
132/132 [==============================] - 57s 432ms/step - loss: 0.0070 - acc: 0.9982 - val_loss: 0.7636 - val_acc: 0.9129
Epoch 14/20
132/132 [==============================] - 59s 447ms/step - loss: 0.0048 - acc: 0.9981 - val_loss: 0.8128 - val_acc: 0.9134
Epoch 15/20
132/132 [==============================] - 62s 470ms/step - loss: 0.0028 - acc: 0.9992 - val_loss: 0.9447 - val_acc: 0.9139
Epoch 16/20
132/132 [==============================] - 58s 435ms/step - loss: 0.0054 - acc: 0.9989 - val_loss: 0.8605 - val_acc: 0.9150
Epoch 17/20
132/132 [==============================] - 59s 446ms/step - loss: 0.0025 - acc: 0.9992 - val_loss: 0.9944 - val_acc: 0.9129
Epoch 18/20
132/132 [==============================] - 58s 440ms/step - loss: 0.0040 - acc: 0.9992 - val_loss: 1.0061 - val_acc: 0.9097
Epoch 19/20
132/132 [==============================] - 59s 445ms/step - loss: 0.0062 - acc: 0.9983 - val_loss: 0.8907 - val_acc: 0.9102
Epoch 20/20
132/132 [==============================] - 59s 445ms/step - loss: 0.0054 - acc: 0.9985 - val_loss: 1.0116 - val_acc: 0.8963
<keras.callbacks.History at 0x7f49d0fbc4f0>
```

```
string_input = keras.Input(shape=(1,), dtype="string")
x = vectorizer(string_input)
preds = model_glove(x)
end_to_end_model = keras.Model(string_input, preds)

probabilities = end_to_end_model.predict(
    [["this message is about texts and humor"]]
```

```
)

class_names[np.argmax(probabilities[0])]
```

```
1/1 [==============================] - 0s 269ms/step
'False'
```

```
test_x = vectorizer(np.array([[s] for s in test.text])).numpy()

preds_glove = model_glove.predict(test_x)
pred_labels_glove = [np.argmax(p) for p in preds_glove]
```

```
197/197 [==============================] - 8s 42ms/step
```

```
print(classification_report(test.humor, pred_labels_glove))
```

```
              precision    recall  f1-score   support

       False       0.88      0.95      0.91      3115
        True       0.95      0.87      0.91      3175

    accuracy                           0.91      6290
   macro avg       0.91      0.91      0.91      6290
weighted avg       0.91      0.91      0.91      6290
```

Embedding Report

Overall the embedding models did much better than the RNN, CNN, LSTM, and GRU models. Aside from the first embedding model which also scored fairly low with 50%, the two other models: layered embedding and GloVe, scored above 90%. I went ahead and experimented by changing some of the activation functions and most of the time the values and accuracies were fairly similar if not the same. The highest score between the two of them though was 93% accuracy from using all relu activation with an rmsprop optimizer. The GloVe model was fairly interesting to run however when I first ran the model with 100d text file I had an accuracy of 91%, when I ran the model with the 200d text file it had a similar result which was interesting. I thought that it may do better but it did fairly similar. Overall the embedding method was much more successful for classifying instead of the other methods.

Overall Report

Overall the sequential model and embedding did the best in terms of accurately classifying the text into either jokes or not jokes. The sequential model was able to score above a 90% in terms of accuracy and this was similar to two of the embedding models. The model that did the best however was the Embedding model with multiple embedding layers in the model. This may have been the reason for the high accuracy as with more layers it allowed for the model to properly train and understand what is going on and what the text is being classified as. The interesting thing though was the scores with the RNN, LSTM, GRU, and CNN. The scores with these four classifiers were very poor with all four of them scoring around 50% accuracy. This is no different than taking a coin and flipping it to see if it lands on heads or tails. I think there were multiple reasons as to why this may have been the case though. For instance there were not many layers that were added to each of the models and perhaps with more layers added it could possibly score much higher. If there were perhaps more epochs the model may have had a better time training over the data and could have done better as well. I tried to adjust the activation models as well as the optimizers, but with any combination that I did the score stayed the same regardless of the optimizers and activation functions. With the embedding it was interesting to see how the different functions worked in regards to the layered embedding and the GloVe model. Before I ran the models I thought that the GloVe may perform better only due to the fact the pretrained model had many words already stored on it allowing for a better read and training model. However, to my surprise the layered embedding did a much better job. Regardless though, both models were still very strong and had a very high accuracy rate along with high recalls precision and f1 scores. Overall the two models that perforemed the best were the sequential and layered Embedding.

0s completed at 5:27 PM