NLPDisasterTweets

November 8, 2024

1 Introduction to Deep Learning - Week 4

1.1 NLP Disaster Tweets Kaggle Mini Project

Github Link https://github.com/conditas/NLPDisasterTweet

This project uses Natural Language Processing and a Recurrent Neural Netwrok Model (RNN) for the Kaggle Competition "Natural Language Processing with Disaster Tweets." It involves being able to predict and classify whether twitter texts are about an actual disaster or not. This can be helpful for spreading real-time information during emergency situations.

1.1.1 Import Libraries

The following libraries will be used in my project.

[212]:
```
%matplotlib inline
import warnings
warnings.simplefilter("ignore", FutureWarning)

import pandas as pd
import numpy as np
import scipy as sp
import scipy.stats as stats
import seaborn as sns
import matplotlib.pyplot as plt


import gc
import os

import sklearn
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score,␣
 ↪accuracy_score,ConfusionMatrixDisplay,confusion_matrix, f1_score
from tensorflow.keras.preprocessing.text import Tokenizer

import tensorflow as tf
from tensorflow import keras
from keras import layers, optimizers
```

```
import keras_core as keras
import keras_nlp
from tensorflow.keras.models import Sequential
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.layers import Embedding, LSTM, Bidirectional,␣
↪Dense,GlobalMaxPool1D,BatchNormalization, Dropout
from tensorflow.keras.preprocessing.sequence import pad_sequences

os.environ['KERAS_BACKEND'] = 'tensorflow'
```

1.1.2 Load the Data

The data comes from the "Natural Language Processing with Disaster Tweets" Kaggle competition. It consists of a two files a train.csv and a test.csv. The train file contains 5 columns and 7613 rows. The test file will be used for the kaggle competition submission and has 4 columns with 3263 rows. The table below summarizes the variables and below the table I print out the data types for each column.

Variable	Description
id	unique identifier
keyword	keyword from tweet
location	where tweet was sent from
text	text of twitter tweet
target	classifier '1' for disaster and '0' for non-disaster

Data Citation

Addison Howard, devrishi, Phil Culliton, and Yufeng Guo. Natural Language Processing with Disaster Tweets. https://kaggle.com/competitions/nlp-getting-started/data, 2019. Kaggle.

[238]:
```
df_train = pd.read_csv("train.csv")
df_test = pd.read_csv("test.csv")

print('Training:\n')
print('Shape = {}'.format(df_train.shape))
print(df_train.dtypes)

print('\nTest:\n')
print('Shape = {}'.format(df_test.shape))
print(df_test.dtypes)

print("Example of Data:\n", df_train.head(5))
```

```
Training:

Shape = (7613, 5)
id           int64
keyword     object
```

```
location    object
text        object
target       int64
dtype: object

Test:

Shape = (3263, 4)
id           int64
keyword     object
location    object
text        object
dtype: object
Example of Data:
   id keyword location                                               text  \
0   1     NaN      NaN  Our Deeds are the Reason of this #earthquake M…
1   4     NaN      NaN             Forest fire near La Ronge Sask. Canada
2   5     NaN      NaN  All residents asked to 'shelter in place' are …
3   6     NaN      NaN  13,000 people receive #wildfires evacuation or…
4   7     NaN      NaN  Just got sent this photo from Ruby #Alaska as …

   target
0       1
1       1
2       1
3       1
4       1
```

1.1.3 Exploratory Data Analysis

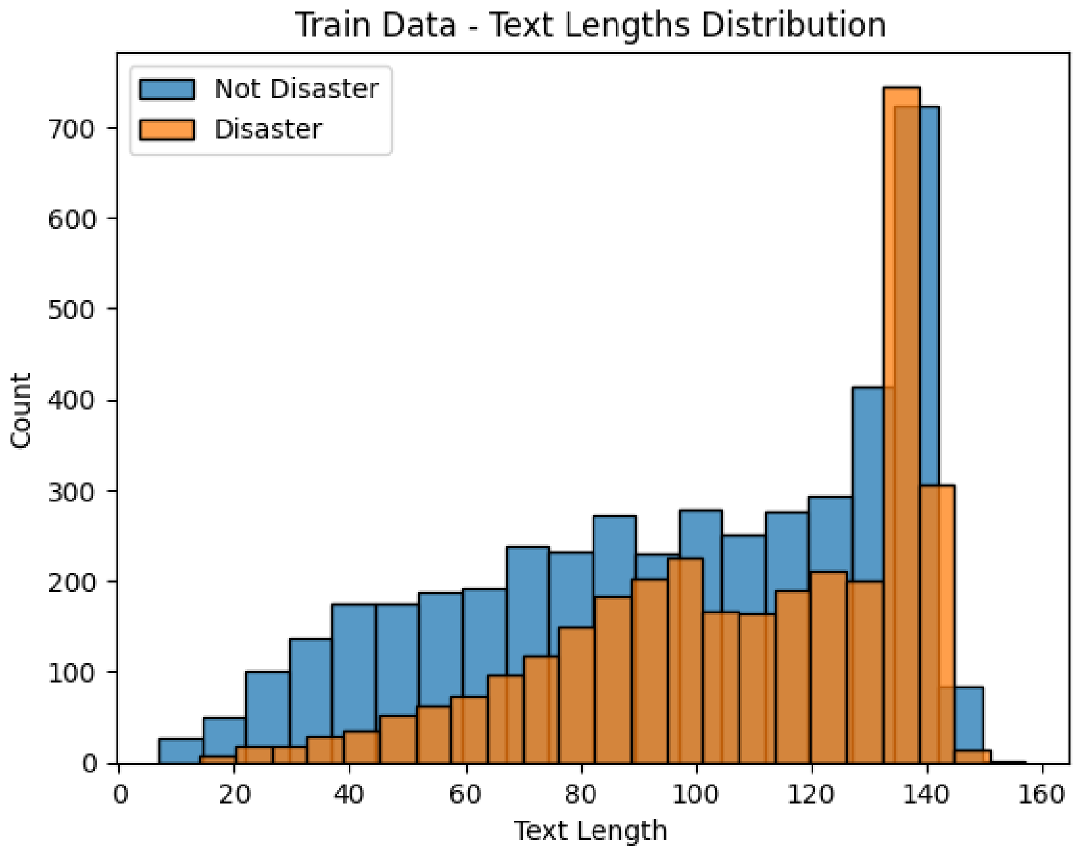
In my exploratory data analysis I compare the number of disaster tweets vs non-disaster tweets and calculate an imbalance ratio. This ratio was 1.35 which should be fine for modeling. I check for null values accross the columns. There were not any null or empty values in the text column. The distribution of locations shows most of the tweets came from the US. There were some descrepencies on the naming (USA vs United Staes) and the location column had a significant number of missing values (33%) so I will not be using this information. I compare text lengths of disaster vs non-disaster tweets. In general, the disaster tweets tend to be longer than non-disaster.

Finally I split the training data set 80/20 in order to train and validate models.

[239]:
```
text_lengths_1 = []
text_lengths_0 = []

####
#### This code plots text length distribution for disaster and non-disaster␣
↪tweets
####
for index,row in df_train.iterrows():
```

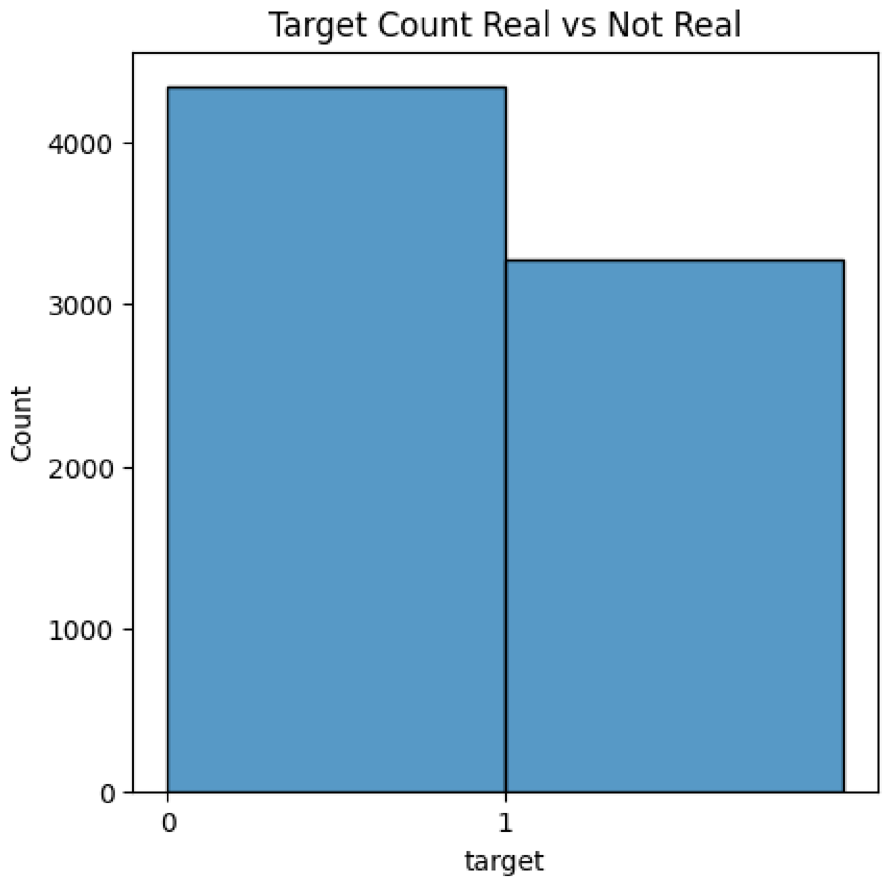
```
    if row['target']==1:
        text_lengths_1.append(len(row['text']))

    if row['target']==0:
        text_lengths_0.append(len(row['text']))

sns.histplot(text_lengths_0, label='Not Disaster')
sns.histplot(text_lengths_1, label='Disaster')
plt.title("Train Data - Text Lengths Distribution")
plt.xlabel('Text Length')
plt.legend(loc='upper left')
plt.show()

print("Examples of Disaster Tweets")
print(df_train[df_train['target']==1]['text'][0:10])

print("\nExamples of Non-Disaster Tweets")
print(df_train[df_train['target']==0]['text'][0:10])

####
## Clean data- looking for null values for text column
####
print("train dimensions: ", df_train.shape)
print("test dimensions: ", df_test.shape)

#Checking for null values and duplicates
print("\nTrain Dataset")
print("\nNull values in text?: ",df_train['text'].isnull().values.sum())
print("\nNull values in keyword?: ",df_train['keyword'].isnull().values.sum())
print("\nNull values in location?: ",df_train['location'].isnull().values.sum())
print("\n\nTest Dataset")
print("\nNull values in text?: ",df_test['text'].isnull().values.sum())
print("\nNull values in keyword?: ",df_test['keyword'].isnull().values.sum())
print("\nNull values in location?: ",df_test['location'].isnull().values.sum())


#####
# counts plot by target
####
print()
plt.figure(figsize = (5,5))
sns.histplot(df_train['target'],bins=range(3), ec='k')
plt.xticks([0,1])
plt.title("Target Count Real vs Not Real")
plt.show()

####
```

```
#calculate imbalance ratio
####
num_zeros = (y_train== 0).sum()
num_ones = (y_train == 1).sum()
imbalance = num_zeros/num_ones
print("Imbalance: ", imbalance)


#####
# counts plot by location
#####
print("\nWhere are tweets from?")
print(df_train['location'].value_counts()[:10])
plt.figure(figsize = (5,5))
sns.barplot(y=df_train['location'].value_counts()[:10].
 ↪index,x=df_train['location'].value_counts()[:10], color='maroon')
plt.title("Tweet Locations")
plt.show()


###
## Split into train file into train and validation datasets
###

target = df_train["target"]
x = df_train.drop(["id","location", "target"], axis=1)
X_train, X_test, y_train, y_test = train_test_split(df_train["text"], target,␣
 ↪test_size=0.2, random_state=1234)

print("Train Shape X/Y: ")
print(X_train.shape)
print(y_train.shape)
print("Validation Shape X/Y: ")
print(X_test.shape)
print(y_test.shape)
```

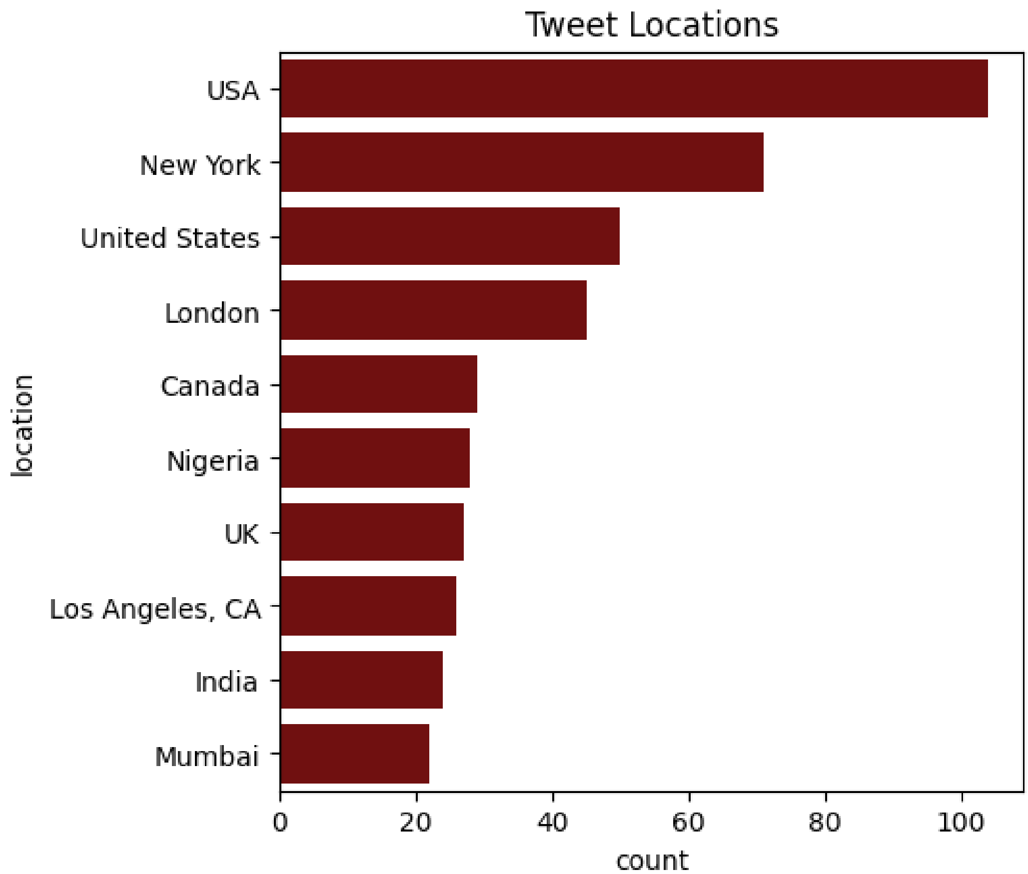



```
Examples of Disaster Tweets
0    Our Deeds are the Reason of this #earthquake M…
1                 Forest fire near La Ronge Sask. Canada
2    All residents asked to 'shelter in place' are …
3    13,000 people receive #wildfires evacuation or…
4    Just got sent this photo from Ruby #Alaska as …
5    #RockyFire Update => California Hwy. 20 closed…
6    #flood #disaster Heavy rain causes flash flood…
7    I'm on top of the hill and I can see a fire in…
8    There's an emergency evacuation happening now …
9    I'm afraid that the tornado is coming to our a…
Name: text, dtype: object

Examples of Non-Disaster Tweets
15                    What's up man?
16                     I love fruits
17                  Summer is lovely
18                 My car is so fast
19     What a goooooooaaaaaal!!!!!!
20              this is ridiculous…
```

```
21                London is cool ;)
22                      Love skiing
23            What a wonderful day!
24                         LOOOOOOL
Name: text, dtype: object
train dimensions:  (7613, 5)
test dimensions:  (3263, 4)

Train Dataset

Null values in text?:  0

Null values in keyword?:  61

Null values in location?:  2533


Test Dataset

Null values in text?:  0

Null values in keyword?:  26

Null values in location?:  1105
```




```
Imbalance:  1.3522595596755504

Where are tweets from?
location
USA                104
New York            71
United States       50
London              45
Canada              29
Nigeria             28
UK                  27
Los Angeles, CA     26
India               24
Mumbai              22
Name: count, dtype: int64
```




```
Train Shape X/Y:
(6090,)
(6090,)
Validation Shape X/Y:
(1523,)
(1523,)
```

1.1.4 Models

The first step in creating the model was to process the text. I used the Keras Tokenizer API which converts all characters to lowercase, it removes unnecessay characters such as symbols, puncuation, and numbers. It converts the text into a list of words in a string. I create a word embedding using GloVE (Global Vectors for Word Representation) for the model. This converts the sequence of words into a vectorized form. This vector format captures semantic relationships between words.

Since sequence in text is important, I use an Recurrent Neural Network model with LSTM (Long Short Term Memory). I use the embedding from GloVE in the input layer to transform the input sequences into a vector of size 100. I then use a bi-directional LSTM layer followed by a dense layer with an relu activation function. I use a drop out laye for regulation and then a final dense layer with a sigmoid activation function for the output label classification.

[287]:
```python
# Text Prepocessing
# This code performs tokenization of the tweet text
tokenizer = Tokenizer()  # # create the tokenizer
tokenizer.fit_on_texts(X_train)
tokenizer.fit_on_texts(X_test)


X_train_seq = tokenizer.texts_to_sequences(X_train)
X_test_seq = tokenizer.texts_to_sequences(X_test)
x_sub_test = tokenizer.texts_to_sequences(df_test["text"])
# Pad sequences to a fixed length
X_train_seq = pad_sequences(X_train_seq,truncating='post', padding='post')
X_test_seq = pad_sequences(X_test_seq,truncating='post', padding='post')
x_sub_test = pad_sequences(x_sub_test,truncating='post', padding='post')
X_train_seq.shape
```

[287]: (6090, 33)

[288]:
```python
### Embedding Function Using Glove 6B 100 text file

def embedding_for_vocab(filepath, word_index,
                        embedding_dim):
    vocab_size = len(word_index) + 1


    embedding_matrix_vocab = np.zeros((vocab_size,
                                       embedding_dim))

    with open(filepath, encoding="utf8") as f:
        for line in f:
            word, *vector = line.split()
            if word in word_index:
                idx = word_index[word]
                embedding_matrix_vocab[idx] = np.array(
                    vector, dtype=np.float32)[:embedding_dim]

    return embedding_matrix_vocab


embedding_matrix_vocab = embedding_for_vocab(
    'glove.6B.100d.txt', tokenizer.word_index,
  100)
```

[289]:
```python
#####
# This code builds the RNN model
#####
opt = optimizers.Adam(learning_rate=0.01, beta_1=0.9)
```

```
model = Sequential()
model.add(Embedding(input_dim=embedding_matrix_vocab.shape[0],
                    output_dim=embedding_matrix_vocab.shape[1],
                    weights=[embedding_matrix_vocab],
                    input_length=30,
                    trainable=False))
model.add(Bidirectional(LSTM(64,recurrent_dropout=0.2)))
model.add(Dropout(0.2))
model.add(Dense(64, activation = "relu"))
model.add(Dropout(0.2))
model.add(Dense(1, activation='sigmoid'))


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

```
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-
packages/keras/src/layers/core/embedding.py:90: UserWarning: Argument
`input_length` is deprecated. Just remove it.
  warnings.warn(
```

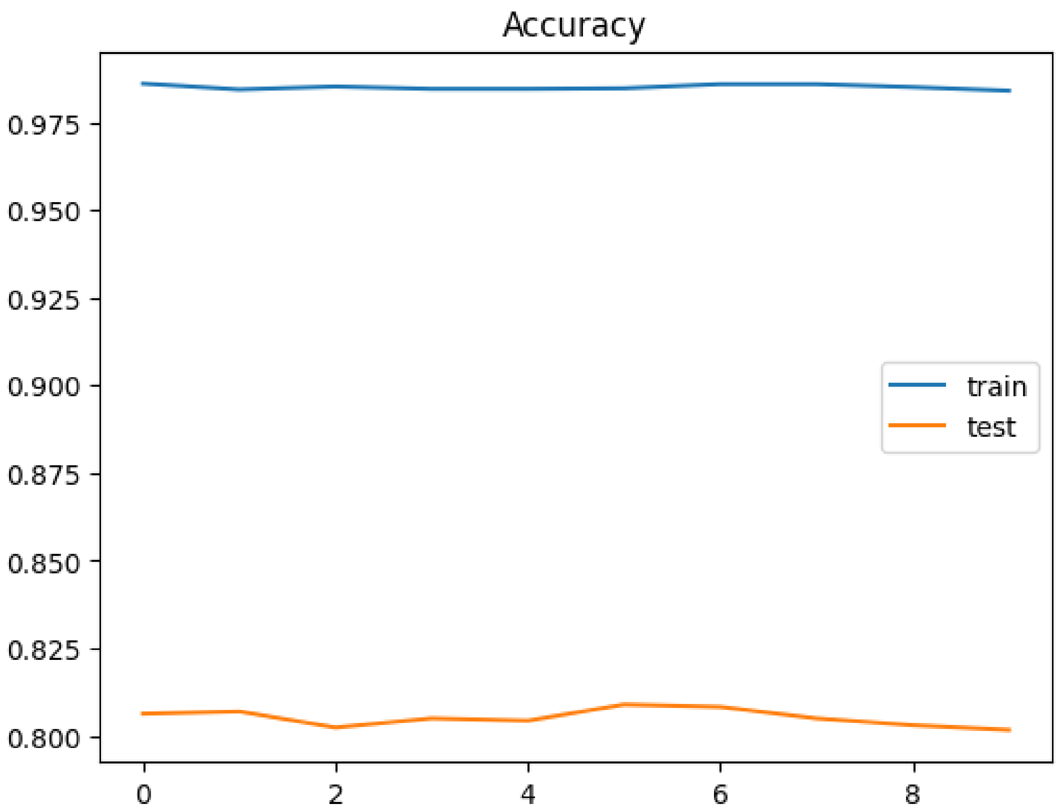
1.1.5 Results and Analysis

I run an initial model and plot the accuracy of the training data compared to the validation data. The accuracy of the training set is pretty decent- 0.98 at the 10th epoch. However the validation accuracy is only 0.80 which suggests there is overfitting in the model. The F1 score was 0.77. I perform some hyperparameter tuning to get this a little higher.

[323]:
```
####
# Train the model
###
lstm_model = model.fit(X_train_seq, y_train, epochs=10, batch_size=32,␣
 ↪validation_data = (X_test_seq, y_test))

plt.title('Accuracy')
plt.plot(lstm_model.history['accuracy'], label='train')
plt.plot(lstm_model.history['val_accuracy'], label='test')
plt.legend()
plt.show();

#predictions and f1 score
y_pred =np.round(model.predict(X_test_seq))

print("F1 Score: ",f1_score(y_test, y_pred))
```

```
Epoch 1/10
191/191              3s 15ms/step -
accuracy: 0.9845 - loss: 0.0296 - val_accuracy: 0.8063 - val_loss: 2.2264
Epoch 2/10
```

```
191/191              3s 15ms/step -
accuracy: 0.9838 - loss: 0.0302 - val_accuracy: 0.8070 - val_loss: 2.1891
Epoch 3/10
191/191              3s 15ms/step -
accuracy: 0.9849 - loss: 0.0302 - val_accuracy: 0.8024 - val_loss: 2.2214
Epoch 4/10
191/191              3s 15ms/step -
accuracy: 0.9817 - loss: 0.0310 - val_accuracy: 0.8050 - val_loss: 2.2162
Epoch 5/10
191/191              3s 15ms/step -
accuracy: 0.9856 - loss: 0.0390 - val_accuracy: 0.8043 - val_loss: 2.1365
Epoch 6/10
191/191              3s 15ms/step -
accuracy: 0.9825 - loss: 0.0315 - val_accuracy: 0.8089 - val_loss: 2.1296
Epoch 7/10
191/191              3s 15ms/step -
accuracy: 0.9837 - loss: 0.0323 - val_accuracy: 0.8083 - val_loss: 2.1701
Epoch 8/10
191/191              3s 15ms/step -
accuracy: 0.9846 - loss: 0.0291 - val_accuracy: 0.8050 - val_loss: 2.2688
Epoch 9/10
191/191              3s 15ms/step -
accuracy: 0.9825 - loss: 0.0339 - val_accuracy: 0.8030 - val_loss: 2.2499
Epoch 10/10
191/191              3s 15ms/step -
accuracy: 0.9839 - loss: 0.0314 - val_accuracy: 0.8017 - val_loss: 2.1582
```




```
48/48                 0s 4ms/step
F1 Score:  0.770516717325228
```

Hyperparameter Tuning

I tried different approaches to tuning the hyperparameters. I tried lowering the learning rate, changing the optimizer to RMS Prop, increasing batch size and epoch number. Finally I did I combination of increased epochs and decreased learning rate. The results are summarized in a table below.

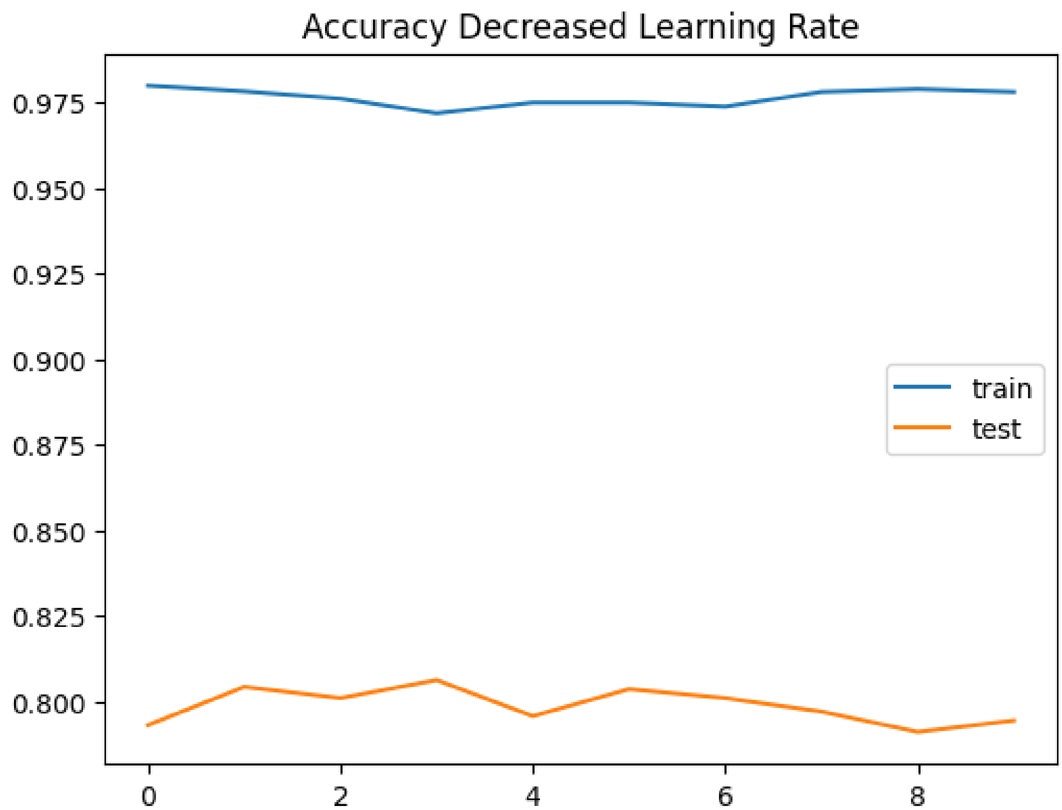
[324]:
```
####
# Hyperparameter Tuning -
###

#  Decrease Learning Rate by half
opt = optimizers.Adam(learning_rate=0.005, beta_1=0.9)
model.compile(optimizer=opt,loss='binary_crossentropy',metrics=['accuracy'])

lstm_model_2 = model.fit(X_train_seq, y_train, epochs=10, batch_size=32,␣
 ↪validation_data = (X_test_seq, y_test))
```

```
plt.title('Accuracy Decreased Learning Rate')
plt.plot(lstm_model_2.history['accuracy'], label='train')
plt.plot(lstm_model_2.history['val_accuracy'], label='test')
plt.legend()
plt.show();

#predictions and f1 score
y_pred =np.round(model.predict(X_test_seq))
print("F1 Score Decreased Learning Rate: ",f1_score(y_test, y_pred))



# Change Optimizer from ADAM to RMS Prop

#opt = optimizers.RMSPROP(learning_rate=0.01)
model.
 ↪compile(optimizer='rmsprop',loss='binary_crossentropy',metrics=['accuracy'])

lstm_model_3 = model.fit(X_train_seq, y_train, epochs=10, batch_size=32,␣
 ↪validation_data = (X_test_seq, y_test))

plt.title('Accuracy- RMS PROP ')
plt.plot(lstm_model_3.history['accuracy'], label='train')
plt.plot(lstm_model_3.history['val_accuracy'], label='test')
plt.legend()
plt.show();

#predictions and f1 score
y_pred =np.round(model.predict(X_test_seq))
print("F1 Score RMSProp Optimizer: ",f1_score(y_test, y_pred))


# Change Batch Size from 32 to 64

opt = optimizers.Adam(learning_rate=0.01, beta_1=0.9)
model.
 ↪compile(optimizer='rmsprop',loss='binary_crossentropy',metrics=['accuracy'])

lstm_model_4 = model.fit(X_train_seq, y_train, epochs=10, batch_size=64,␣
 ↪validation_data = (X_test_seq, y_test))

plt.title('Accuracy- Increased Batch Size')
plt.plot(lstm_model_4.history['accuracy'], label='train')
plt.plot(lstm_model_4.history['val_accuracy'], label='test')
plt.legend()
plt.show();
```

```
#predictions and f1 score
y_pred =np.round(model.predict(X_test_seq))
print("F1 Score Increased Batch Size: ",f1_score(y_test, y_pred))


# Increase Epoch
opt = optimizers.Adam(learning_rate=0.01, beta_1=0.9)
model.
 ↪compile(optimizer='rmsprop',loss='binary_crossentropy',metrics=['accuracy'])

lstm_model_5 = model.fit(X_train_seq, y_train, epochs=20, batch_size=32,␣
 ↪validation_data = (X_test_seq, y_test))

plt.title('Accuracy- Increased Epoch Size')
plt.plot(lstm_model_5.history['accuracy'], label='train')
plt.plot(lstm_model_5.history['val_accuracy'], label='test')
plt.legend()
plt.show();

#predictions and f1 score
y_pred =np.round(model.predict(X_test_seq))
print("F1 Score Increased Epoch Number: ",f1_score(y_test, y_pred))
```

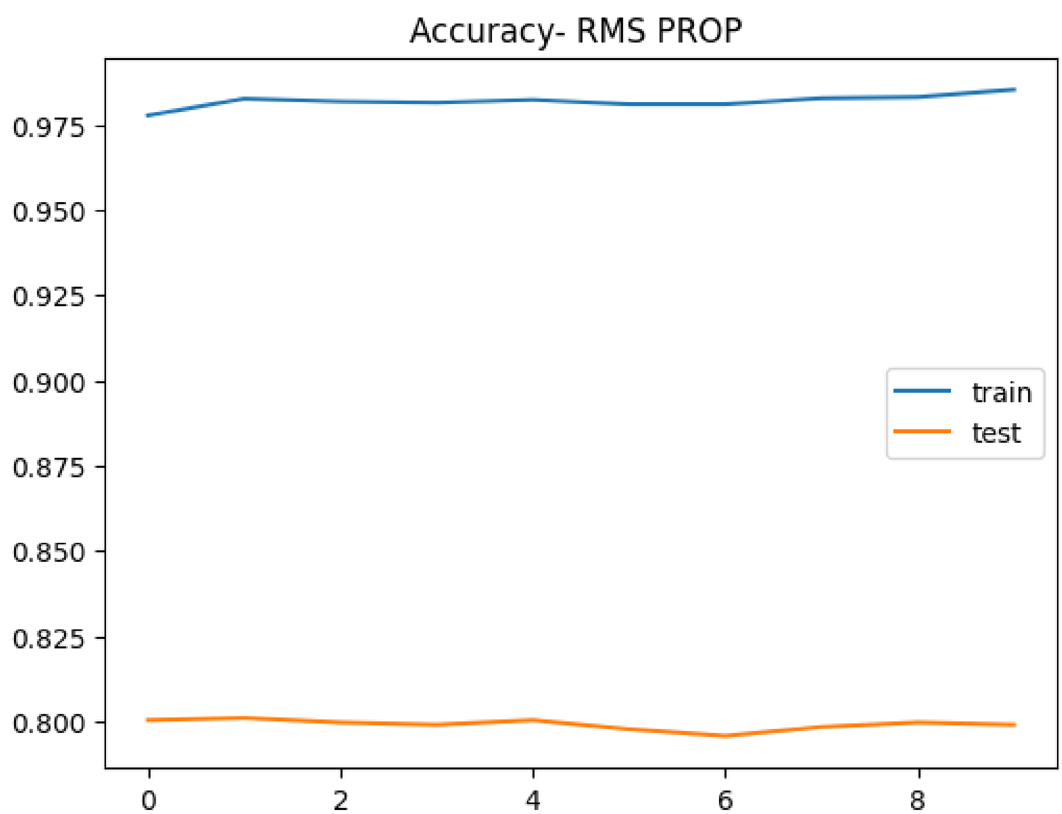
```
Epoch 1/10
191/191              4s 12ms/step -
accuracy: 0.9840 - loss: 0.0434 - val_accuracy: 0.7932 - val_loss: 1.8165
Epoch 2/10
191/191              3s 14ms/step -
accuracy: 0.9783 - loss: 0.0570 - val_accuracy: 0.8043 - val_loss: 1.8758
Epoch 3/10
191/191              3s 15ms/step -
accuracy: 0.9795 - loss: 0.0539 - val_accuracy: 0.8011 - val_loss: 1.7285
Epoch 4/10
191/191              3s 15ms/step -
accuracy: 0.9736 - loss: 0.0723 - val_accuracy: 0.8063 - val_loss: 1.4450
Epoch 5/10
191/191              3s 15ms/step -
accuracy: 0.9790 - loss: 0.0513 - val_accuracy: 0.7958 - val_loss: 1.3402
Epoch 6/10
191/191              3s 15ms/step -
accuracy: 0.9769 - loss: 0.0533 - val_accuracy: 0.8037 - val_loss: 1.3474
Epoch 7/10
191/191              3s 15ms/step -
accuracy: 0.9741 - loss: 0.0596 - val_accuracy: 0.8011 - val_loss: 1.3721
Epoch 8/10
191/191              3s 15ms/step -
accuracy: 0.9800 - loss: 0.0502 - val_accuracy: 0.7971 - val_loss: 1.1951
```

```
Epoch 9/10
191/191            3s 15ms/step -
accuracy: 0.9786 - loss: 0.0459 - val_accuracy: 0.7912 - val_loss: 1.5901
Epoch 10/10
191/191            3s 15ms/step -
accuracy: 0.9765 - loss: 0.0531 - val_accuracy: 0.7945 - val_loss: 1.5667
```



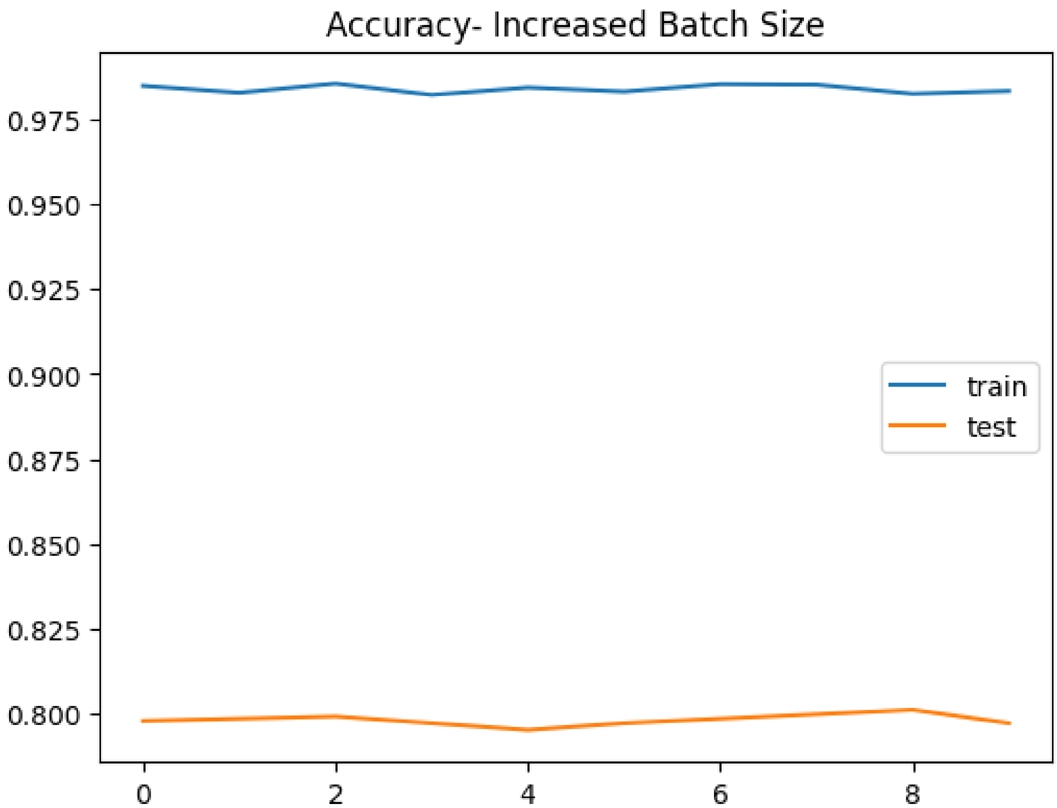

```
48/48            0s 5ms/step
F1 Score Decreased Learning Rate:  0.7658937920718025
Epoch 1/10
191/191            3s 12ms/step -
accuracy: 0.9774 - loss: 0.0494 - val_accuracy: 0.8004 - val_loss: 1.6714
Epoch 2/10
191/191            3s 14ms/step -
accuracy: 0.9848 - loss: 0.0350 - val_accuracy: 0.8011 - val_loss: 1.7988
Epoch 3/10
191/191            3s 15ms/step -
accuracy: 0.9829 - loss: 0.0389 - val_accuracy: 0.7997 - val_loss: 1.9420
Epoch 4/10
191/191            3s 15ms/step -
accuracy: 0.9831 - loss: 0.0393 - val_accuracy: 0.7991 - val_loss: 1.9633
Epoch 5/10
```

```
191/191             3s 15ms/step -
accuracy: 0.9840 - loss: 0.0378 - val_accuracy: 0.8004 - val_loss: 1.9944
Epoch 6/10
191/191             3s 15ms/step -
accuracy: 0.9827 - loss: 0.0383 - val_accuracy: 0.7978 - val_loss: 2.0073
Epoch 7/10
191/191             3s 15ms/step -
accuracy: 0.9824 - loss: 0.0375 - val_accuracy: 0.7958 - val_loss: 1.9941
Epoch 8/10
191/191             3s 14ms/step -
accuracy: 0.9833 - loss: 0.0363 - val_accuracy: 0.7984 - val_loss: 2.0600
Epoch 9/10
191/191             3s 15ms/step -
accuracy: 0.9822 - loss: 0.0383 - val_accuracy: 0.7997 - val_loss: 2.0870
Epoch 10/10
191/191             3s 15ms/step -
accuracy: 0.9834 - loss: 0.0355 - val_accuracy: 0.7991 - val_loss: 2.1883
```



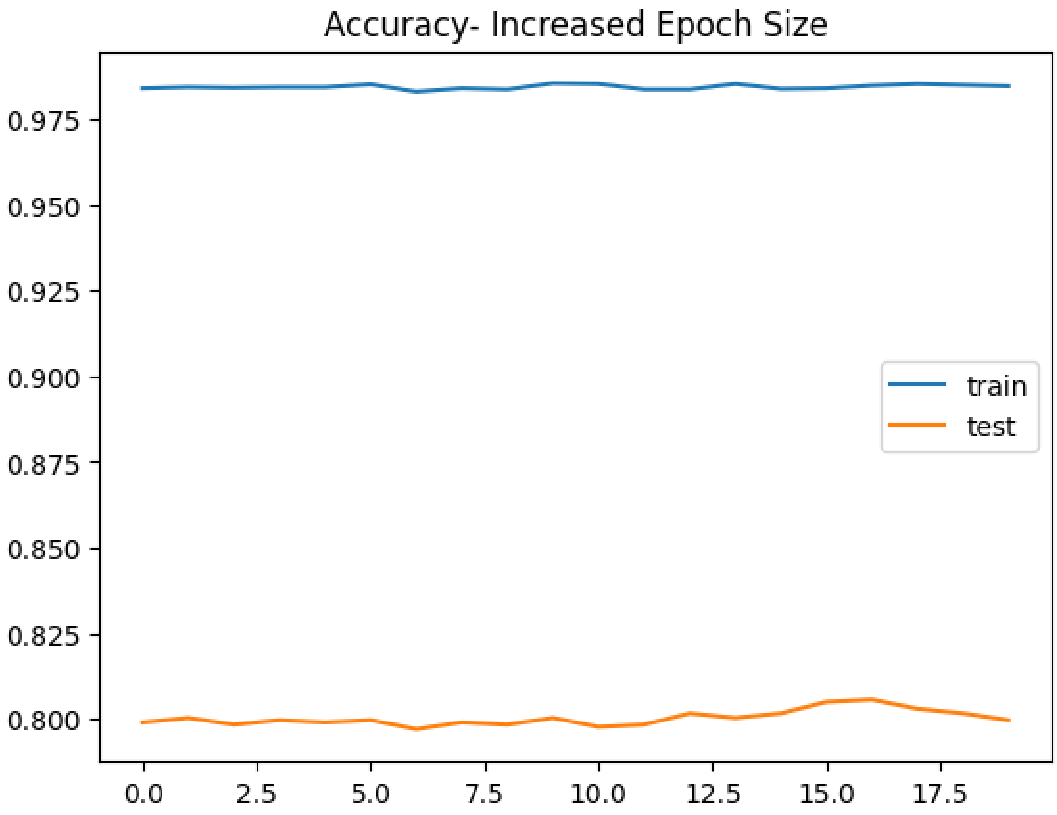

```
48/48             0s 5ms/step
F1 Score RMSProp Optimizer:  0.7649769585253456
Epoch 1/10
96/96             3s 16ms/step -
```

```
accuracy: 0.9856 - loss: 0.0307 - val_accuracy: 0.7978 - val_loss: 2.2161
Epoch 2/10
96/96            2s 24ms/step -
accuracy: 0.9834 - loss: 0.0395 - val_accuracy: 0.7984 - val_loss: 2.1742
Epoch 3/10
96/96            2s 24ms/step -
accuracy: 0.9871 - loss: 0.0316 - val_accuracy: 0.7991 - val_loss: 2.1967
Epoch 4/10
96/96            2s 24ms/step -
accuracy: 0.9822 - loss: 0.0352 - val_accuracy: 0.7971 - val_loss: 2.2207
Epoch 5/10
96/96            2s 24ms/step -
accuracy: 0.9852 - loss: 0.0344 - val_accuracy: 0.7951 - val_loss: 2.1893
Epoch 6/10
96/96            2s 25ms/step -
accuracy: 0.9843 - loss: 0.0319 - val_accuracy: 0.7971 - val_loss: 2.2188
Epoch 7/10
96/96            2s 25ms/step -
accuracy: 0.9865 - loss: 0.0368 - val_accuracy: 0.7984 - val_loss: 2.2432
Epoch 8/10
96/96            2s 25ms/step -
accuracy: 0.9861 - loss: 0.0307 - val_accuracy: 0.7997 - val_loss: 2.2656
Epoch 9/10
96/96            2s 25ms/step -
accuracy: 0.9833 - loss: 0.0356 - val_accuracy: 0.8011 - val_loss: 2.2200
Epoch 10/10
96/96            2s 25ms/step -
accuracy: 0.9821 - loss: 0.0375 - val_accuracy: 0.7971 - val_loss: 2.2704
```




```
48/48              0s 5ms/step
F1 Score Increased Batch Size:  0.7621247113163973
Epoch 1/20
191/191               5s 12ms/step -
accuracy: 0.9835 - loss: 0.0317 - val_accuracy: 0.7991 - val_loss: 2.3515
Epoch 2/20
191/191               3s 14ms/step -
accuracy: 0.9826 - loss: 0.0324 - val_accuracy: 0.8004 - val_loss: 2.4363
Epoch 3/20
191/191               3s 15ms/step -
accuracy: 0.9843 - loss: 0.0353 - val_accuracy: 0.7984 - val_loss: 2.4105
Epoch 4/20
191/191               3s 15ms/step -
accuracy: 0.9859 - loss: 0.0294 - val_accuracy: 0.7997 - val_loss: 2.5282
Epoch 5/20
191/191               3s 15ms/step -
accuracy: 0.9840 - loss: 0.0406 - val_accuracy: 0.7991 - val_loss: 2.4813
Epoch 6/20
191/191               3s 15ms/step -
accuracy: 0.9853 - loss: 0.0367 - val_accuracy: 0.7997 - val_loss: 2.4904
Epoch 7/20
```

```
191/191            3s 15ms/step -
accuracy: 0.9820 - loss: 0.0450 - val_accuracy: 0.7971 - val_loss: 2.5011
Epoch 8/20
191/191            3s 15ms/step -
accuracy: 0.9832 - loss: 0.0371 - val_accuracy: 0.7991 - val_loss: 2.5134
Epoch 9/20
191/191            3s 15ms/step -
accuracy: 0.9840 - loss: 0.0325 - val_accuracy: 0.7984 - val_loss: 2.6135
Epoch 10/20
191/191            3s 15ms/step -
accuracy: 0.9856 - loss: 0.0277 - val_accuracy: 0.8004 - val_loss: 2.7104
Epoch 11/20
191/191            3s 15ms/step -
accuracy: 0.9855 - loss: 0.0336 - val_accuracy: 0.7978 - val_loss: 2.8001
Epoch 12/20
191/191            3s 15ms/step -
accuracy: 0.9818 - loss: 0.0352 - val_accuracy: 0.7984 - val_loss: 2.7301
Epoch 13/20
191/191            3s 15ms/step -
accuracy: 0.9826 - loss: 0.0361 - val_accuracy: 0.8017 - val_loss: 2.6874
Epoch 14/20
191/191            3s 15ms/step -
accuracy: 0.9851 - loss: 0.0315 - val_accuracy: 0.8004 - val_loss: 2.6988
Epoch 15/20
191/191            3s 15ms/step -
accuracy: 0.9847 - loss: 0.0276 - val_accuracy: 0.8017 - val_loss: 2.7968
Epoch 16/20
191/191            3s 15ms/step -
accuracy: 0.9805 - loss: 0.0377 - val_accuracy: 0.8050 - val_loss: 2.7952
Epoch 17/20
191/191            3s 15ms/step -
accuracy: 0.9873 - loss: 0.0314 - val_accuracy: 0.8056 - val_loss: 2.7331
Epoch 18/20
191/191            3s 15ms/step -
accuracy: 0.9854 - loss: 0.0347 - val_accuracy: 0.8030 - val_loss: 2.8794
Epoch 19/20
191/191            3s 15ms/step -
accuracy: 0.9844 - loss: 0.0358 - val_accuracy: 0.8017 - val_loss: 2.9339
Epoch 20/20
191/191            3s 15ms/step -
accuracy: 0.9864 - loss: 0.0296 - val_accuracy: 0.7997 - val_loss: 3.0109
```




```
48/48              0s 5ms/step
F1 Score Increased Epoch Number:  0.765564950038432
```

[321]:

```
#Decrease Learning Increase Epoch
opt = optimizers.Adam(learning_rate=0.001, beta_1=0.9)
model.compile(optimizer=opt,loss='binary_crossentropy',metrics=['accuracy'])

lstm_model_6 = model.fit(X_train_seq, y_train, epochs=50, batch_size=32,␣
 ↪validation_data = (X_test_seq, y_test))

plt.title('Accuracy- Combined Increased Epoch Size/Decrease Learning Rate')
plt.plot(lstm_model_6.history['accuracy'], label='train')
plt.plot(lstm_model_6.history['val_accuracy'], label='test')
plt.legend()
plt.show();

#predictions and f1 score
y_pred =np.round(model.predict(X_test_seq))
print("F1 Score Combined Increased Epoch Size/Decrease Learning Rate:␣
 ↪",f1_score(y_test, y_pred))
```

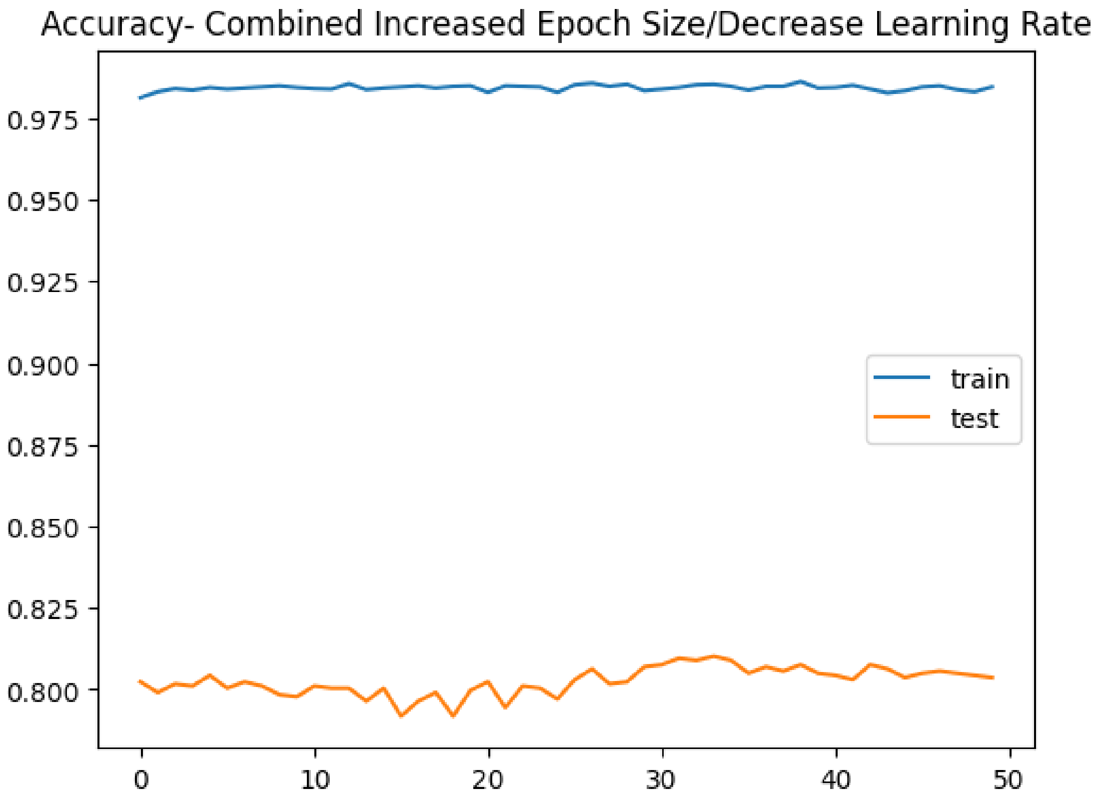
```
Epoch 1/50
```

```
191/191              4s 12ms/step -
accuracy: 0.9824 - loss: 0.0326 - val_accuracy: 0.8024 - val_loss: 2.3037
Epoch 2/50
191/191              3s 14ms/step -
accuracy: 0.9833 - loss: 0.0333 - val_accuracy: 0.7991 - val_loss: 2.1897
Epoch 3/50
191/191              3s 15ms/step -
accuracy: 0.9847 - loss: 0.0326 - val_accuracy: 0.8017 - val_loss: 2.1412
Epoch 4/50
191/191              3s 15ms/step -
accuracy: 0.9842 - loss: 0.0346 - val_accuracy: 0.8011 - val_loss: 2.1349
Epoch 5/50
191/191              3s 15ms/step -
accuracy: 0.9851 - loss: 0.0311 - val_accuracy: 0.8043 - val_loss: 2.1277
Epoch 6/50
191/191              3s 15ms/step -
accuracy: 0.9823 - loss: 0.0334 - val_accuracy: 0.8004 - val_loss: 2.1378
Epoch 7/50
191/191              3s 15ms/step -
accuracy: 0.9860 - loss: 0.0326 - val_accuracy: 0.8024 - val_loss: 2.1487
Epoch 8/50
191/191              3s 15ms/step -
accuracy: 0.9849 - loss: 0.0362 - val_accuracy: 0.8011 - val_loss: 2.1865
Epoch 9/50
191/191              3s 15ms/step -
accuracy: 0.9841 - loss: 0.0360 - val_accuracy: 0.7984 - val_loss: 2.1513
Epoch 10/50
191/191              3s 15ms/step -
accuracy: 0.9850 - loss: 0.0328 - val_accuracy: 0.7978 - val_loss: 2.2755
Epoch 11/50
191/191              3s 15ms/step -
accuracy: 0.9844 - loss: 0.0332 - val_accuracy: 0.8011 - val_loss: 2.2613
Epoch 12/50
191/191              3s 15ms/step -
accuracy: 0.9828 - loss: 0.0366 - val_accuracy: 0.8004 - val_loss: 2.2819
Epoch 13/50
191/191              3s 15ms/step -
accuracy: 0.9858 - loss: 0.0339 - val_accuracy: 0.8004 - val_loss: 2.2583
Epoch 14/50
191/191              3s 15ms/step -
accuracy: 0.9817 - loss: 0.0353 - val_accuracy: 0.7965 - val_loss: 2.1566
Epoch 15/50
191/191              3s 15ms/step -
accuracy: 0.9839 - loss: 0.0350 - val_accuracy: 0.8004 - val_loss: 2.2722
Epoch 16/50
191/191              3s 15ms/step -
accuracy: 0.9850 - loss: 0.0309 - val_accuracy: 0.7919 - val_loss: 2.3638
Epoch 17/50
```

```
191/191             3s 15ms/step -
accuracy: 0.9874 - loss: 0.0307 - val_accuracy: 0.7965 - val_loss: 2.2539
Epoch 18/50
191/191             3s 15ms/step -
accuracy: 0.9843 - loss: 0.0321 - val_accuracy: 0.7991 - val_loss: 2.1848
Epoch 19/50
191/191             3s 15ms/step -
accuracy: 0.9869 - loss: 0.0292 - val_accuracy: 0.7919 - val_loss: 2.0238
Epoch 20/50
191/191             3s 15ms/step -
accuracy: 0.9862 - loss: 0.0365 - val_accuracy: 0.7997 - val_loss: 2.1166
Epoch 21/50
191/191             3s 15ms/step -
accuracy: 0.9828 - loss: 0.0323 - val_accuracy: 0.8024 - val_loss: 2.0643
Epoch 22/50
191/191             3s 15ms/step -
accuracy: 0.9852 - loss: 0.0320 - val_accuracy: 0.7945 - val_loss: 2.1305
Epoch 23/50
191/191             3s 15ms/step -
accuracy: 0.9855 - loss: 0.0346 - val_accuracy: 0.8011 - val_loss: 2.2196
Epoch 24/50
191/191             3s 15ms/step -
accuracy: 0.9853 - loss: 0.0305 - val_accuracy: 0.8004 - val_loss: 2.2288
Epoch 25/50
191/191             3s 15ms/step -
accuracy: 0.9809 - loss: 0.0421 - val_accuracy: 0.7971 - val_loss: 2.1823
Epoch 26/50
191/191             3s 15ms/step -
accuracy: 0.9873 - loss: 0.0291 - val_accuracy: 0.8030 - val_loss: 2.2199
Epoch 27/50
191/191             3s 15ms/step -
accuracy: 0.9859 - loss: 0.0310 - val_accuracy: 0.8063 - val_loss: 2.2261
Epoch 28/50
191/191             3s 15ms/step -
accuracy: 0.9856 - loss: 0.0277 - val_accuracy: 0.8017 - val_loss: 2.2722
Epoch 29/50
191/191             3s 15ms/step -
accuracy: 0.9844 - loss: 0.0294 - val_accuracy: 0.8024 - val_loss: 2.3347
Epoch 30/50
191/191             3s 15ms/step -
accuracy: 0.9838 - loss: 0.0353 - val_accuracy: 0.8070 - val_loss: 2.1659
Epoch 31/50
191/191             3s 15ms/step -
accuracy: 0.9866 - loss: 0.0295 - val_accuracy: 0.8076 - val_loss: 2.2348
Epoch 32/50
191/191             3s 15ms/step -
accuracy: 0.9863 - loss: 0.0294 - val_accuracy: 0.8096 - val_loss: 2.1884
Epoch 33/50
```

```
191/191            3s 15ms/step -
accuracy: 0.9847 - loss: 0.0322 - val_accuracy: 0.8089 - val_loss: 2.2392
Epoch 34/50
191/191            3s 15ms/step -
accuracy: 0.9846 - loss: 0.0307 - val_accuracy: 0.8102 - val_loss: 2.2712
Epoch 35/50
191/191            3s 15ms/step -
accuracy: 0.9862 - loss: 0.0271 - val_accuracy: 0.8089 - val_loss: 2.1921
Epoch 36/50
191/191            3s 15ms/step -
accuracy: 0.9813 - loss: 0.0351 - val_accuracy: 0.8050 - val_loss: 2.1654
Epoch 37/50
191/191            3s 15ms/step -
accuracy: 0.9862 - loss: 0.0312 - val_accuracy: 0.8070 - val_loss: 2.1702
Epoch 38/50
191/191            3s 15ms/step -
accuracy: 0.9837 - loss: 0.0312 - val_accuracy: 0.8056 - val_loss: 2.1213
Epoch 39/50
191/191            3s 15ms/step -
accuracy: 0.9860 - loss: 0.0293 - val_accuracy: 0.8076 - val_loss: 2.2364
Epoch 40/50
191/191            3s 15ms/step -
accuracy: 0.9842 - loss: 0.0293 - val_accuracy: 0.8050 - val_loss: 2.1667
Epoch 41/50
191/191            3s 15ms/step -
accuracy: 0.9847 - loss: 0.0329 - val_accuracy: 0.8043 - val_loss: 2.2310
Epoch 42/50
191/191            3s 15ms/step -
accuracy: 0.9867 - loss: 0.0296 - val_accuracy: 0.8030 - val_loss: 2.3100
Epoch 43/50
191/191            3s 15ms/step -
accuracy: 0.9845 - loss: 0.0443 - val_accuracy: 0.8076 - val_loss: 2.1636
Epoch 44/50
191/191            3s 15ms/step -
accuracy: 0.9852 - loss: 0.0308 - val_accuracy: 0.8063 - val_loss: 2.1197
Epoch 45/50
191/191            3s 15ms/step -
accuracy: 0.9846 - loss: 0.0313 - val_accuracy: 0.8037 - val_loss: 2.1498
Epoch 46/50
191/191            3s 15ms/step -
accuracy: 0.9836 - loss: 0.0356 - val_accuracy: 0.8050 - val_loss: 2.1191
Epoch 47/50
191/191            3s 15ms/step -
accuracy: 0.9852 - loss: 0.0348 - val_accuracy: 0.8056 - val_loss: 2.1753
Epoch 48/50
191/191            3s 15ms/step -
accuracy: 0.9826 - loss: 0.0311 - val_accuracy: 0.8050 - val_loss: 2.2017
Epoch 49/50
```

```
191/191              3s 15ms/step -
accuracy: 0.9853 - loss: 0.0304 - val_accuracy: 0.8043 - val_loss: 2.1776
Epoch 50/50
191/191              3s 15ms/step -
accuracy: 0.9841 - loss: 0.0295 - val_accuracy: 0.8037 - val_loss: 2.1894
```




```
48/48              0s 5ms/step
F1 Score Combined Increased Epoch Size/Decrease Learning Rate:
0.7722772277227723
```

Hyperparameter Tuning Results

The table below shows the performance metrics for the different hyperparameter tuning strategies. These hyperparameter modifications did not appear to have a significant outcome on the accuracy or f1 score of the validation data. The accuracy of the training set is very good and the validation accuracy and f1 of 0.8 and 0.77 respectively are not too bad, but the differences suggest there is still inherent overfitting in the model.

Accuracy		
	Train	Val
Initial Model	0.9839	0.8017
Decrease Learning Rate	0.9765	0.7945

Accuracy		
Switch Optimizer	0.9834	0.7991
Increase Batch Size	0.9821	0.7971
Increase Epochs	0.9864	0.7979
Combined	0.9841	0.8037

F1 Score	
	Val
Initial Model	0.7705
Decrease Learning Rate	00.7659
Switch Optimizer	0.7650
Increase Batch Size	0.7610
Increase Epochs	0.7656
Combined	0.7723

1.1.6 Conclusion

This project provided a good exercise in NLP (Natural Language Processing) and also showed the potential in creating RNN/LSTM models. The model I created was able to classify whether a twitter tweet was about an actual disaster with approximately 80 percent accuracy. In this project I experimented with tuning hyperparameters to get an even better accuracy, however these results did not lead to too much improvement. For future modifications in attempting to achieve higher accuracy, making changes to the model's architecture such as adding additional layers may be needed. Making modifications to the text processing could also be an improvement to explore in the future.

1.1.7 References

AnmolsX_test_seq0, February 7). Disaster Tweets : Simple RNN Implementation. Kaggle.com; Kaggle. https://www.kaggle.com/code/anmolstha/disaster-tweets-simple-rnn-implementation

Andreshg. (2021, May 10). NLP GloVe, BERT, TF-IDF, LSTM… Explained. Kaggle.com; Kaggle. https://www.kaggle.com/code/andreshg/nlp-glove-bert-tf-idf-lstm-explained

Feldges, C. (2022, April 2). Text Classification with TF-IDF, LSTM, BERT: a quantitative comparison. Medium. https://medium.com/@claude.feldges/text-classification-with-tf-idf-lstm-bert-a-quantitative-comparison-b8409b556cb3

Kumar, V. (2020, February 23). Real or Not? NLP with Disaster Tweets (A Data science Capstone Project). Medium; Real or Not? NLP with Disaster Tweets. https://medium.com/real-or-not-nlp-with-disaster-tweets/real-or-not-nlp-with-disaster-tweets-a-data-science-capstone-project-fafa6c35c16f

[316]:
```
###
# Kaggle Submission Code - score 0.779
###
y_pred = model.predict(x_sub_test)
y_pred =np.round(y_pred).astype(int).reshape(3263)
```

```python
print(y_pred)

sub = pd.DataFrame(
    list(zip(df_test['id'], y_pred)),
    columns=["id", "target"],
)
print(sub)
sub.to_csv("submission.csv", index=False)
```

```
102/102              0s 4ms/step
[0 1 0 … 1 1 0]
         id  target
0         0       0
1         2       1
2         3       0
3         9       1
4        11       1
…       …     …
3258  10861       1
3259  10865       1
3260  10868       1
3261  10874       1
3262  10875       0

[3263 rows x 2 columns]
```

[ ]:

```

```