prescott-reddit-sentiment

June 6, 2021

[1336]:
```python
import gc
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras import Model, optimizers, Sequential
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.layers import (Input, Embedding, Bidirectional, LSTM,
                                     Dense, RepeatVector, Flatten,␣
 ↪BatchNormalization,
                                     LeakyReLU, Dropout)
from sklearn.metrics import confusion_matrix, roc_curve, auc
from sklearn.model_selection import train_test_split
from nltk.sentiment.vader import SentimentIntensityAnalyzer
import nltk
import seaborn as sns
nltk.download('vader_lexicon')
```

```
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     C:\Users\ben.prescott\AppData\Roaming\nltk_data…
[nltk_data]   Package vader_lexicon is already up-to-date!
```

[1336]: True

0.1 Importing Full Corpus

[4]:
```python
# Load JSON lines file for the full corpus
corpus = []
with open('redditjson.jl', encoding='utf8') as f:
    for line in f:
        corpus.append(json.loads(line))
```

[1255]:
```python
corpusDF.subreddit.unique()
```

[1255]:
```
array(['singapore', 'tifu', 'cringepics', 'motorcycles',
       'TwoXChromosomes', 'hockey', 'sex', 'Christianity', 'conspiracy',
```

```
'canada', 'pokemontrades', 'Guildwars2', 'askscience', 'IAmA',
'australia', 'relationships', 'Bitcoin', 'business',
'electronic_cigarette', 'MMA', 'DebateReligion', 'skyrim',
'movies', 'WTF', 'Android', 'OkCupid', 'Frugal', 'anime',
'todayilearned', 'Fitness', 'SquaredCircle', 'photography',
'hiphopheads', 'POLITIC', 'apple', 'science', 'AskMen', 'pokemon',
'offbeat', 'Games', 'Minecraft', 'guns', 'AskWomen', 'politics',
'technology', 'wow', 'Music', 'tf2', 'cringe', 'techsupport',
'news', 'cars', 'MensRights', 'malefashionadvice', 'buildapc',
'worldnews', 'gifs', 'soccer', 'asoiaf', 'explainlikeimfive',
'dayz', 'books', 'relationship_advice', 'aww', 'gonewild',
'fantasyfootball', 'unitedkingdom', 'AmItheAsshole',
'MovieDetails', 'nfl', 'AdviceAnimals', 'programming', 'Drugs',
'ShingekiNoKyojin', 'DotA2', 'Diablo', 'Random_Acts_Of_Amazon',
'Naruto', 'Marvel', 'starcraft', 'gaming', 'rupaulsdragrace',
'NoFap', 'travel', 'LifeProTips', 'teenagers', 'pics',
'leagueoflegends', 'atheism', 'trees', 'CFB', 'magicTCG',
'Economics', 'MakeupAddiction', 'nba', 'videos', 'baseball',
'funny', 'AskReddit', 'Libertarian'], dtype=object)
```

0.2 Select Only Games Subreddit

[1350]:
```
corpusDF = pd.DataFrame(corpus) # Assign corpus to dataframe for ease of review

reddits = ['Games'] # Subreddits I want to include in my analysis

reducedDF = corpusDF[corpusDF['subreddit'].isin(reddits)] # Creating a new␣
↪dataframe consisting of only Games subreddit
```

[1351]:
```
# Removing empty, deleted, removed, and automoderate utterances
reducedDF = reducedDF[~(reducedDF['text'] == '') &
                      ~(reducedDF['text'] == '[deleted]') &
                      ~(reducedDF['text'] == '[removed]') &
                      ~(reducedDF['speaker'] == 'AutoModerator')
                     ]
```

0.3 VADER

[1352]:
```
# Loading the VADER model
analyzer = SentimentIntensityAnalyzer()
```

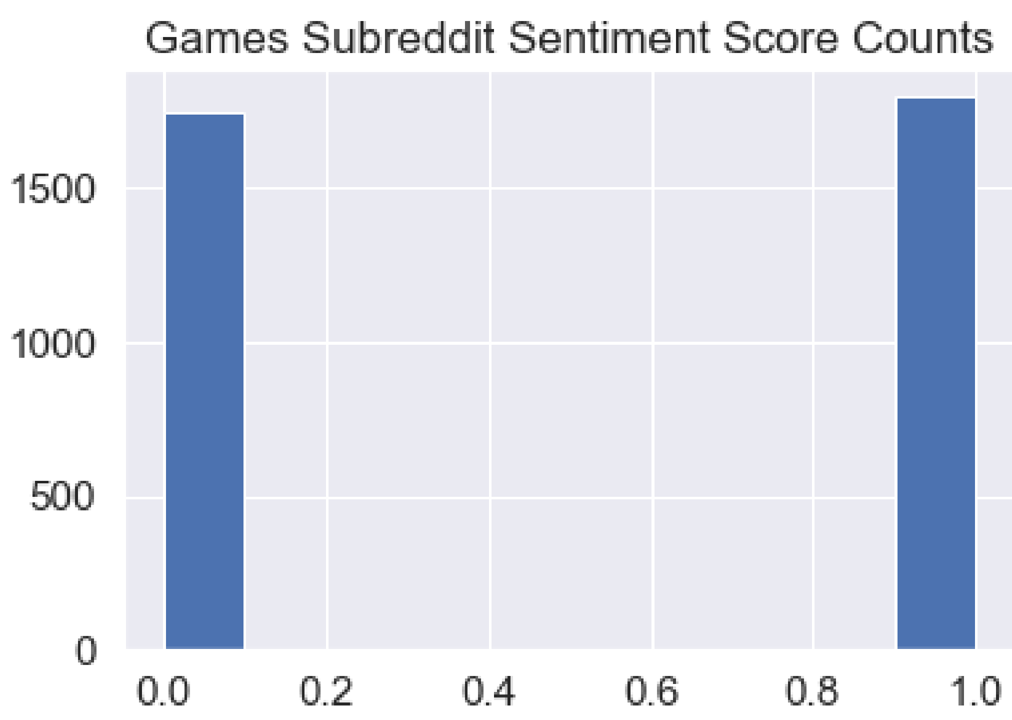
[1353]:
```
# Creating a copy of the Games dataframe and assigning a new column all values␣
↪of zero
reducedScored = reducedDF.copy()
reducedScored['sentiment'] = 0
comments = reducedScored.text.tolist()
```

[1354]:
```python
len(comments)
```

[1354]: 3542

[1355]:
```python
# Performing sentiment analysis on each utterance and updating the 'sentiment'␣
↪column score
reducedScored = reducedDF.copy()
reducedScored.reset_index(inplace=True)
reducedScored['sentiment'] = 0
comments = reducedScored.text.tolist()
count = 0
for comment in comments:
    score = analyzer.polarity_scores(comment)
    if score['compound'] > 0.05:
        reducedScored.at[count, 'sentiment'] = 1
    else:
        reducedScored.at[count, 'sentiment'] = 0
    count += 1
```

[1357]:
```python
# Visualizing the sentiment score counts
plt.hist(reducedScored.sentiment)
plt.title('Games Subreddit Sentiment Score Counts')
plt.show()
```



0.4 Tokenization and Padding

[1316]:
```
# Converting the utterances and their VADER sentiment labels to lists
sentences = reducedScored.text.tolist()
labels = np.array(reducedScored.sentiment.tolist())
```

[1317]:
```
# Reviewing the max sequence lengths to use for variables later
# Some look to be outliers - possibly long URLs. Majority seem to be under 55␣
↪words long.
seq_lengths = reducedScored.text.apply(lambda x: len(x.split(' ')))
seq_lengths.describe()
```

[1317]:
```
count    3542.000000
mean       50.200169
std       128.529918
min         1.000000
25%        16.000000
50%        29.000000
75%        55.000000
max      5203.000000
Name: text, dtype: float64
```

[1318]:
```
# Selecting some values for tokenization and word embeddings.
num_words = 300
max_len = 200
embed_dim = 10
```

[1319]:
```
# Tokenizing and zero-padding the sequences
tokenizer = Tokenizer(num_words = num_words,
                      split=' ')
tokenizer.fit_on_texts(sentences)
seqs = tokenizer.texts_to_sequences(sentences)
pad_seqs = pad_sequences(seqs, max_len)
```

[1320]:
```
len(pad_seqs[1])
```

[1320]:
```
200
```

0.5 Word Embeddings Using AutoEncoder

[1321]:
```
# Tripartite splitting of the datasets
X, y = pad_seqs, labels

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.10,␣
↪random_state=1)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.
↪25, random_state=1)
```

[1322]:
```
print('X_train:',X_train.shape,'\n','X_test:',X_test.shape,'\n','X_val:',X_val.
↪shape)
```

```
X_train: (2390, 200)
 X_test: (355, 200)
 X_val: (797, 200)
```

[1323]:
```
# define encoder/decoder
visible = Input(shape=(max_len,))
# encoder layer 1
encoder = Dense(max_len*2)(visible)
encoder = BatchNormalization()(encoder)
encoder = LeakyReLU()(encoder)
# encoder layer 2
encoder = Dense(max_len)(encoder)
encoder = BatchNormalization()(encoder)
encoder = LeakyReLU()(encoder)
# reduction
n_bottleneck = max_len
bottleneck = Dense(n_bottleneck)(encoder)

# decoder layer 1
decoder = Dense(max_len)(bottleneck)
decoder = BatchNormalization()(decoder)
decoder = LeakyReLU()(decoder)
# decoder layer 2
decoder = Dense(max_len*2)(decoder)
decoder = BatchNormalization()(decoder)
decoder = LeakyReLU()(decoder)
# output layer
output = Dense(max_len, activation='linear')(decoder)

gc.collect()

escallback = EarlyStopping(monitor='val_loss', patience=3)

model = Model(inputs=visible, outputs=output)
model.summary()
model.compile(optimizer='adam', loss='mse')
model.fit(X_train,
          X_train,
          epochs=100,
          validation_data=(X_val,X_val),
          callbacks=[escallback])

encoder = Model(inputs=visible, outputs=bottleneck)
encoder.save('encoder.h5')
```

```
Model: "model_47"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_29 (InputLayer)        [(None, 200)]             0
_________________________________________________________________
dense_564 (Dense)            (None, 400)               80400
_________________________________________________________________
batch_normalization_93 (Batc (None, 400)               1600
_________________________________________________________________
leaky_re_lu_114 (LeakyReLU)  (None, 400)               0
_________________________________________________________________
dense_565 (Dense)            (None, 200)               80200
_________________________________________________________________
batch_normalization_94 (Batc (None, 200)               800
_________________________________________________________________
leaky_re_lu_115 (LeakyReLU)  (None, 200)               0
_________________________________________________________________
dense_566 (Dense)            (None, 200)               40200
_________________________________________________________________
dense_567 (Dense)            (None, 200)               40200
_________________________________________________________________
batch_normalization_95 (Batc (None, 200)               800
_________________________________________________________________
leaky_re_lu_116 (LeakyReLU)  (None, 200)               0
_________________________________________________________________
dense_568 (Dense)            (None, 400)               80400
_________________________________________________________________
batch_normalization_96 (Batc (None, 400)               1600
_________________________________________________________________
leaky_re_lu_117 (LeakyReLU)  (None, 400)               0
_________________________________________________________________
dense_569 (Dense)            (None, 200)               80200
=================================================================
Total params: 406,400
Trainable params: 404,000
Non-trainable params: 2,400
_________________________________________________________________
Epoch 1/100
75/75 [==============================] - 2s 8ms/step - loss: 1294.4224 -
val_loss: 1100.6772
Epoch 2/100
75/75 [==============================] - 0s 6ms/step - loss: 1035.9602 -
val_loss: 926.6536
Epoch 3/100
75/75 [==============================] - 1s 7ms/step - loss: 951.2483 -
val_loss: 868.7836
Epoch 4/100
```

```
75/75 [==============================] - 0s 6ms/step - loss: 838.7691 -
val_loss: 804.7761
Epoch 5/100
75/75 [==============================] - 0s 6ms/step - loss: 792.4944 -
val_loss: 749.7614
Epoch 6/100
75/75 [==============================] - 0s 6ms/step - loss: 747.7604 -
val_loss: 700.8227
Epoch 7/100
75/75 [==============================] - 0s 6ms/step - loss: 661.4807 -
val_loss: 663.1226
Epoch 8/100
75/75 [==============================] - 0s 7ms/step - loss: 663.1271 -
val_loss: 606.9332
Epoch 9/100
75/75 [==============================] - 1s 7ms/step - loss: 603.3059 -
val_loss: 568.9296
Epoch 10/100
75/75 [==============================] - 0s 6ms/step - loss: 538.8212 -
val_loss: 533.3268
Epoch 11/100
75/75 [==============================] - 0s 6ms/step - loss: 519.6249 -
val_loss: 503.1592
Epoch 12/100
75/75 [==============================] - 0s 6ms/step - loss: 457.9767 -
val_loss: 477.5060
Epoch 13/100
75/75 [==============================] - 0s 6ms/step - loss: 451.2704 -
val_loss: 451.7427
Epoch 14/100
75/75 [==============================] - 0s 6ms/step - loss: 434.7517 -
val_loss: 421.1268
Epoch 15/100
75/75 [==============================] - 0s 6ms/step - loss: 413.4992 -
val_loss: 401.2543
Epoch 16/100
75/75 [==============================] - 1s 7ms/step - loss: 382.1207 -
val_loss: 387.2730
Epoch 17/100
75/75 [==============================] - 0s 6ms/step - loss: 374.2231 -
val_loss: 374.1621
Epoch 18/100
75/75 [==============================] - 0s 6ms/step - loss: 342.5318 -
val_loss: 366.4953
Epoch 19/100
75/75 [==============================] - 0s 6ms/step - loss: 350.8727 -
val_loss: 351.3207
Epoch 20/100
```

```
75/75 [==============================] - 0s 6ms/step - loss: 349.1638 -
val_loss: 351.4081
Epoch 21/100
75/75 [==============================] - 1s 7ms/step - loss: 346.2325 -
val_loss: 337.5947
Epoch 22/100
75/75 [==============================] - 1s 7ms/step - loss: 301.2316 -
val_loss: 332.6908
Epoch 23/100
75/75 [==============================] - 0s 6ms/step - loss: 306.0987 -
val_loss: 321.3126
Epoch 24/100
75/75 [==============================] - 0s 6ms/step - loss: 302.7002 -
val_loss: 318.5075
Epoch 25/100
75/75 [==============================] - 0s 7ms/step - loss: 289.7082 -
val_loss: 309.7708
Epoch 26/100
75/75 [==============================] - 0s 7ms/step - loss: 298.8308 -
val_loss: 312.4869
Epoch 27/100
75/75 [==============================] - 0s 7ms/step - loss: 267.2656 -
val_loss: 306.2052
Epoch 28/100
75/75 [==============================] - 0s 7ms/step - loss: 264.3420 -
val_loss: 299.5679
Epoch 29/100
75/75 [==============================] - 1s 7ms/step - loss: 261.7489 -
val_loss: 295.8536
Epoch 30/100
75/75 [==============================] - 1s 7ms/step - loss: 258.9780 -
val_loss: 291.6603
Epoch 31/100
75/75 [==============================] - 1s 7ms/step - loss: 233.5054 -
val_loss: 293.7874
Epoch 32/100
75/75 [==============================] - 0s 7ms/step - loss: 252.1711 -
val_loss: 290.7465
Epoch 33/100
75/75 [==============================] - 1s 7ms/step - loss: 248.1560 -
val_loss: 288.5827
Epoch 34/100
75/75 [==============================] - 1s 7ms/step - loss: 252.3236 -
val_loss: 285.0919
Epoch 35/100
75/75 [==============================] - 0s 7ms/step - loss: 237.7858 -
val_loss: 283.1170
Epoch 36/100
```

```
75/75 [==============================] - 1s 7ms/step - loss: 256.1667 -
val_loss: 279.2074
Epoch 37/100
75/75 [==============================] - 0s 6ms/step - loss: 221.2220 -
val_loss: 275.9750
Epoch 38/100
75/75 [==============================] - 1s 7ms/step - loss: 241.3310 -
val_loss: 274.8022
Epoch 39/100
75/75 [==============================] - 1s 7ms/step - loss: 220.3791 -
val_loss: 278.4431
Epoch 40/100
75/75 [==============================] - 1s 7ms/step - loss: 231.9716 -
val_loss: 271.4637
Epoch 41/100
75/75 [==============================] - 0s 7ms/step - loss: 216.0918 -
val_loss: 270.3765
Epoch 42/100
75/75 [==============================] - 1s 7ms/step - loss: 228.6897 -
val_loss: 269.4066
Epoch 43/100
75/75 [==============================] - 1s 7ms/step - loss: 229.3343 -
val_loss: 270.7491
Epoch 44/100
75/75 [==============================] - 0s 6ms/step - loss: 229.2695 -
val_loss: 268.6453
Epoch 45/100
75/75 [==============================] - 0s 7ms/step - loss: 229.1844 -
val_loss: 266.7638
Epoch 46/100
75/75 [==============================] - 0s 7ms/step - loss: 226.6756 -
val_loss: 265.0619
Epoch 47/100
75/75 [==============================] - 0s 7ms/step - loss: 224.8547 -
val_loss: 265.6024
Epoch 48/100
75/75 [==============================] - 1s 7ms/step - loss: 216.1346 -
val_loss: 264.6800
Epoch 49/100
75/75 [==============================] - 1s 7ms/step - loss: 201.6066 -
val_loss: 263.7982
Epoch 50/100
75/75 [==============================] - ETA: 0s - loss: 210.805 - 1s 7ms/step -
loss: 210.8446 - val_loss: 261.7806
Epoch 51/100
75/75 [==============================] - 1s 7ms/step - loss: 196.9283 -
val_loss: 260.5047
Epoch 52/100
```

```
75/75 [==============================] - 0s 6ms/step - loss: 212.4988 -
val_loss: 258.0635
Epoch 53/100
75/75 [==============================] - 0s 6ms/step - loss: 196.6521 -
val_loss: 259.1991
Epoch 54/100
75/75 [==============================] - 1s 7ms/step - loss: 195.9304 -
val_loss: 256.1610
Epoch 55/100
75/75 [==============================] - 0s 7ms/step - loss: 196.5669 -
val_loss: 257.0411
Epoch 56/100
75/75 [==============================] - 1s 8ms/step - loss: 201.9419 -
val_loss: 255.4944
Epoch 57/100
75/75 [==============================] - 0s 6ms/step - loss: 231.2992 -
val_loss: 253.1422
Epoch 58/100
75/75 [==============================] - 0s 7ms/step - loss: 190.3491 -
val_loss: 251.3075
Epoch 59/100
75/75 [==============================] - 1s 7ms/step - loss: 182.1631 -
val_loss: 252.4967
Epoch 60/100
75/75 [==============================] - 1s 7ms/step - loss: 201.8449 -
val_loss: 254.8852
Epoch 61/100
75/75 [==============================] - 1s 7ms/step - loss: 179.7725 -
val_loss: 247.3277
Epoch 62/100
75/75 [==============================] - 1s 7ms/step - loss: 192.0672 -
val_loss: 248.2090
Epoch 63/100
75/75 [==============================] - 0s 6ms/step - loss: 194.0441 -
val_loss: 249.8560
Epoch 64/100
75/75 [==============================] - 1s 7ms/step - loss: 187.5333 -
val_loss: 249.3679
```

[1324]:

```
# Predicting the word embeddings for the X value train/test/validation data
X_train_encode = encoder.predict(X_train)
X_test_encode = encoder.predict(X_test)
X_val_encode = encoder.predict(X_val)
```

0.6 Classification Network

[1346]:
```
# Model used to predict the sentiment scores using the learned word embeddings
model = Sequential()
model.add(Dense(128, input_dim=max_len, activation='relu'))
model.add(Dense(12, activation='relu'))

model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy', optimizer='adam',␣
 ↪metrics=['accuracy'])
escallback = EarlyStopping(monitor='val_loss', patience=5)

gc.collect()
model.fit(X_train_encode,
          y_train,
          validation_data = (X_val_encode,y_val),
          callbacks=[escallback],
          epochs=50)
```

```
Epoch 1/50
75/75 [==============================] - 1s 4ms/step - loss: 0.7406 - accuracy:
0.5183 - val_loss: 0.6920 - val_accuracy: 0.5471
Epoch 2/50
75/75 [==============================] - 0s 2ms/step - loss: 0.6621 - accuracy:
0.5811 - val_loss: 0.7089 - val_accuracy: 0.5684
Epoch 3/50
75/75 [==============================] - 0s 3ms/step - loss: 0.6250 - accuracy:
0.6424 - val_loss: 0.7306 - val_accuracy: 0.5521
Epoch 4/50
75/75 [==============================] - 0s 3ms/step - loss: 0.6011 - accuracy:
0.6726 - val_loss: 0.7424 - val_accuracy: 0.5646
Epoch 5/50
75/75 [==============================] - 0s 3ms/step - loss: 0.5609 - accuracy:
0.6947 - val_loss: 0.7634 - val_accuracy: 0.5634
Epoch 6/50
75/75 [==============================] - 0s 3ms/step - loss: 0.5384 - accuracy:
0.7181 - val_loss: 0.7946 - val_accuracy: 0.5445
```

[1346]:
```
<tensorflow.python.keras.callbacks.History at 0x1d1f456ae08>
```

0.7 Sentiment Predictions & Performance

[1347]:
```
# Predicting sentiment scores for the X_Test word embeddings
pred = (model.predict(X_test_encode) > 0.5).astype("int32")
predDF = pd.DataFrame({'actual':y_test.tolist(),'predicted':[item for sublist␣
 ↪in pred for item in sublist]})
```

```
predDF
```

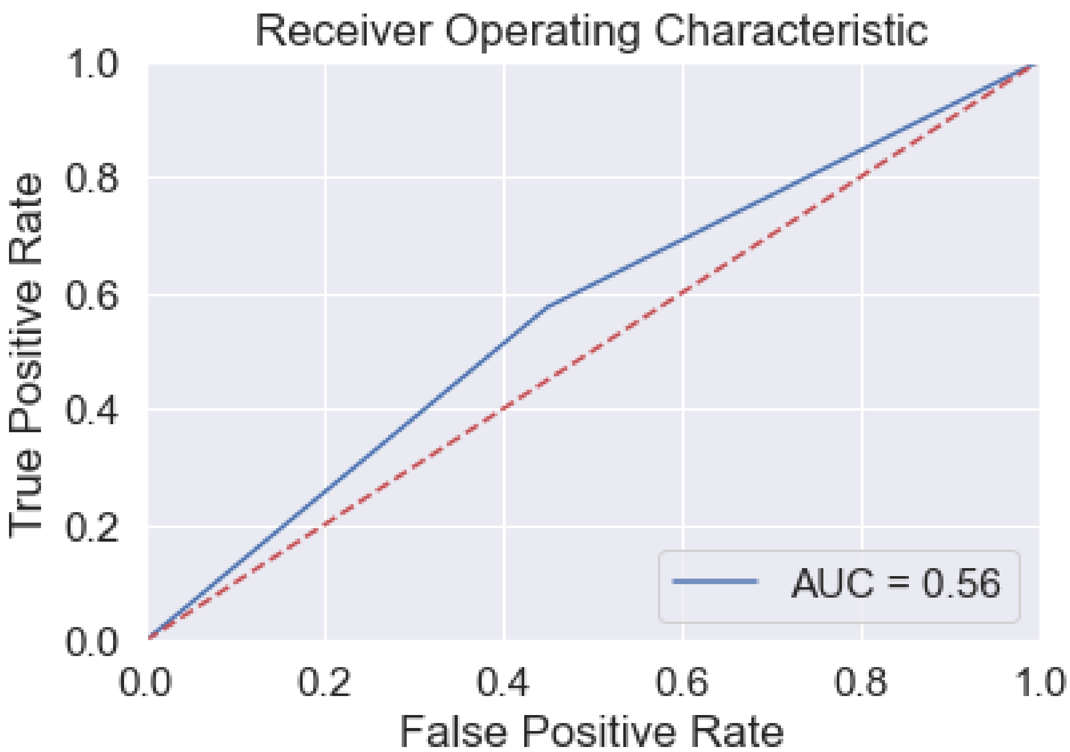
[1347]:

```
     actual  predicted
0         1          1
1         1          1
2         0          1
3         0          0
4         1          1
..      …        …
350       0          1
351       1          1
352       0          0
353       1          1
354       0          1

[355 rows x 2 columns]
```

0.7.1 ROC Curve

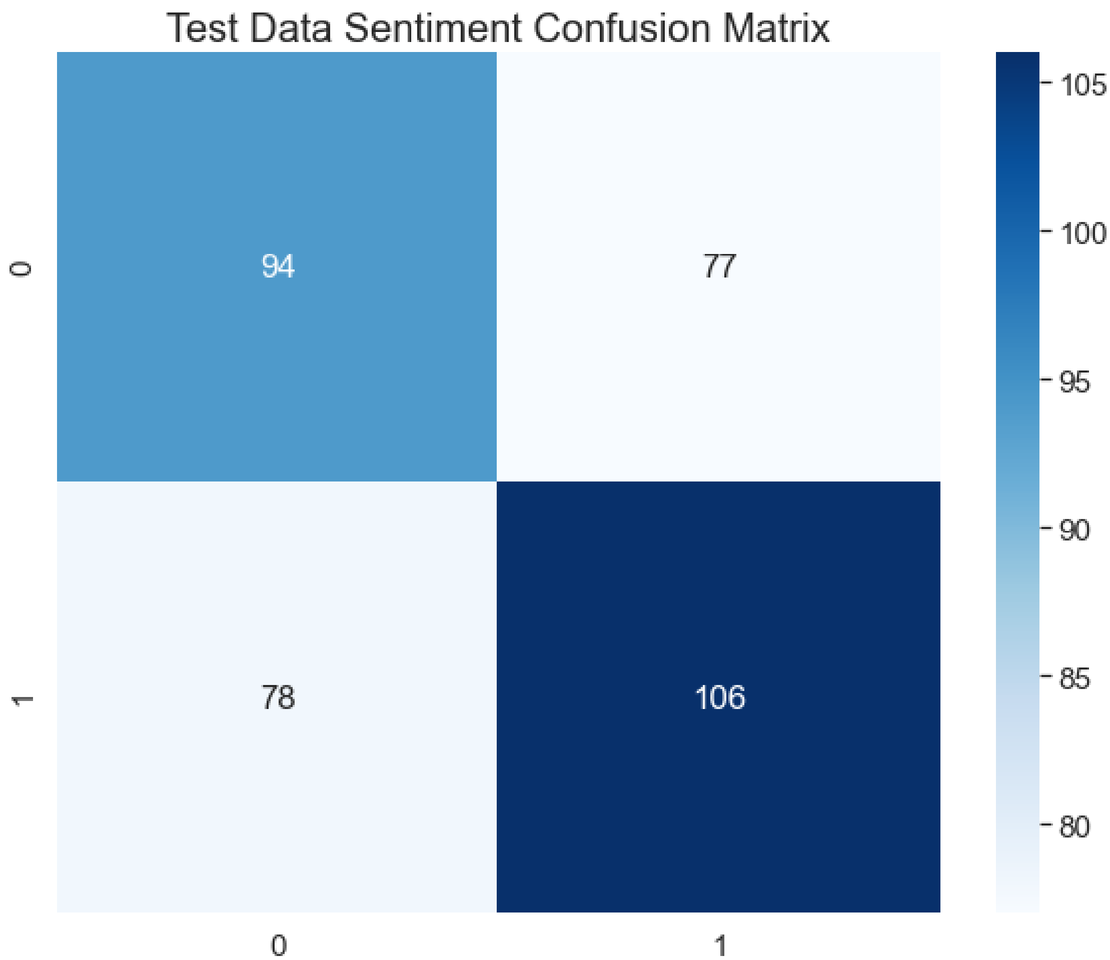
[1348]:

```
# Reviewing ROC AUC score

fpr, tpr, threshold = roc_curve(y_test, pred)
roc_auc = auc(fpr,tpr)

plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



0.7.2 Confusion Matrix

[1349]:
```
# Reviewing the confusion matrix of correctly and incorrectly classified data

cm = confusion_matrix(predDF.actual,predDF.predicted)
plt.figure(figsize=(10,8))
sns.set(font_scale=1.4)
sns.heatmap(cm, annot=True,cmap='Blues',fmt='d')
plt.title('Test Data Sentiment Confusion Matrix', fontsize=20)
plt.show()
```


Test Data Sentiment Confusion Matrix
94
77
78
106
0
1
0
1
105
100
95
90
85
80