Uncovering Sentiments using EDGAR Datasets

Team 9

Dawna Grace Raj

Jai Soni

Nikhil Kohli

Experiments:

Experiment 1:

Loading the data

sentiment	text
positive	thank good afternoon everyone welcome nvidias ...
positive	look past q1 expect channel inventory correcti...
negative	china game weakness slow economic environment ...
negative	dont know could tear apart tease apart harlan ...
positive	thank ill turn call back jenhsun close remark

Bag of Words Using Count Vectorizer

```
[ ]  from sklearn.feature_extraction.text import CountVectorizer
```

```
[ ]  vectorizer = CountVectorizer(min_df=0, lowercase=False)
```

```
[ ]  vectorizer.fit(new_df['text'])
```

```
CountVectorizer(analyzer='word', binary=False, decode_error='strict',
        dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
        lowercase=False, max_df=1.0, max_features=None, min_df=0,
        ngram_range=(1, 1), preprocessor=None, stop_words=None,
        strip_accents=None, token_pattern='(?u)\\b\\w\\w+\\b',
        tokenizer=None, vocabulary=None)
```

Converting it to an array of integers

```
[ ] a=vectorizer.transform(new_df['text']).toarray()
```

```
[ ] a
```

```
array([[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]], dtype=int64)
```

Creating Train and Test Data

```
[ ] from sklearn.model_selection import train_test_split
```

```
[ ] X_train, X_test, y_train, y_test = train_test_split(a, Y, test_size=0.2, random_state=1000)
```


Creating Train and Test Data

```
[ ] from sklearn.model_selection import train_test_split
```

```
[ ] X_train, X_test, y_train, y_test = train_test_split(a, Y, test_size=0.2, random_state=1000)
```

Creating a Logistic Regression Model:

```
[ ]  from sklearn.linear_model import LogisticRegression
```

```
[ ]  classifier = LogisticRegression()
     classifier.fit(X_train, y_train)
     score = classifier.score(X_test, y_test)
```

```
[ ]  print("Accuracy:", score)
```

```
Accuracy: 0.8098159509202454
```

Obtained an Accuracy of 81%

```
[ ]  y_pred = classifier.predict(X_test)
     y_pred
```

```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0,
       1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1], dtype=int64)
```

Creating a metrics to measure model performance

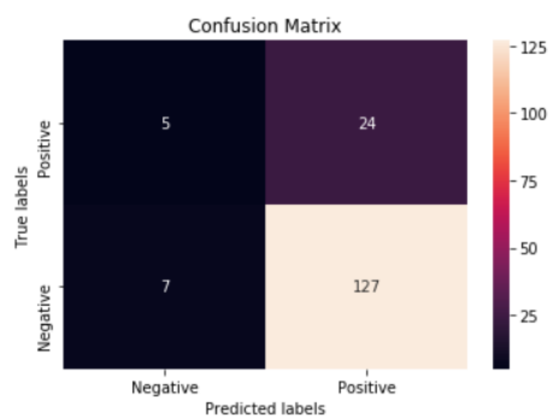
```
from sklearn.metrics import confusion_matrix

confusion_matrix(y_test, y_pred)
```

```
array([[  5,  24],
       [  7, 127]], dtype=int64)
```

```
from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred))
```

```
             precision    recall  f1-score   support

          0       0.42      0.17      0.24        29
          1       0.84      0.95      0.89       134

avg / total       0.77      0.81      0.78       163
```

```
[ ] import matplotlib.pyplot as plt
    import seaborn as sns

    ax= plt.subplot()
    sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, ax = ax, fmt='g'); #annot=True to annotate cells

    # labels, title and ticks
    ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
    ax.set_title('Confusion Matrix');
    ax.xaxis.set_ticklabels(['Negative', 'Positive']); ax.yaxis.set_ticklabels(['Positive', 'Negative']);
```



Fully connected Model using Keras

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

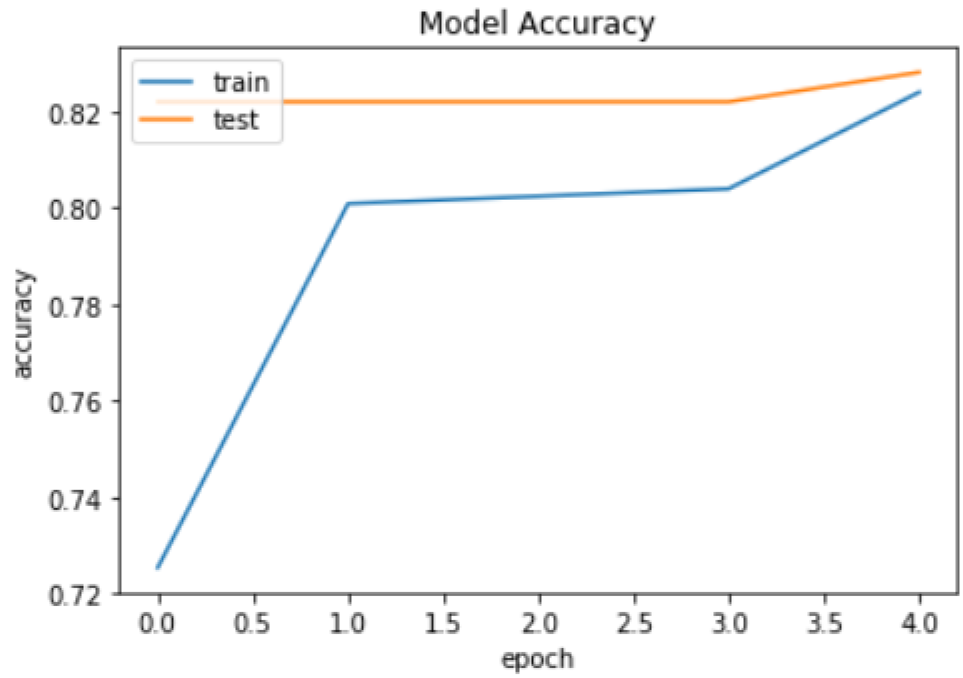
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_10 (Dense)             (None, 32)                128032    
_________________________________________________________________
dropout_7 (Dropout)          (None, 32)                0         
_________________________________________________________________
dense_11 (Dense)             (None, 32)                1056      
_________________________________________________________________
dropout_8 (Dropout)          (None, 32)                0         
_________________________________________________________________
dense_12 (Dense)             (None, 1)                 33        
=================================================================
Total params: 129,121
Trainable params: 129,121
Non-trainable params: 0
```

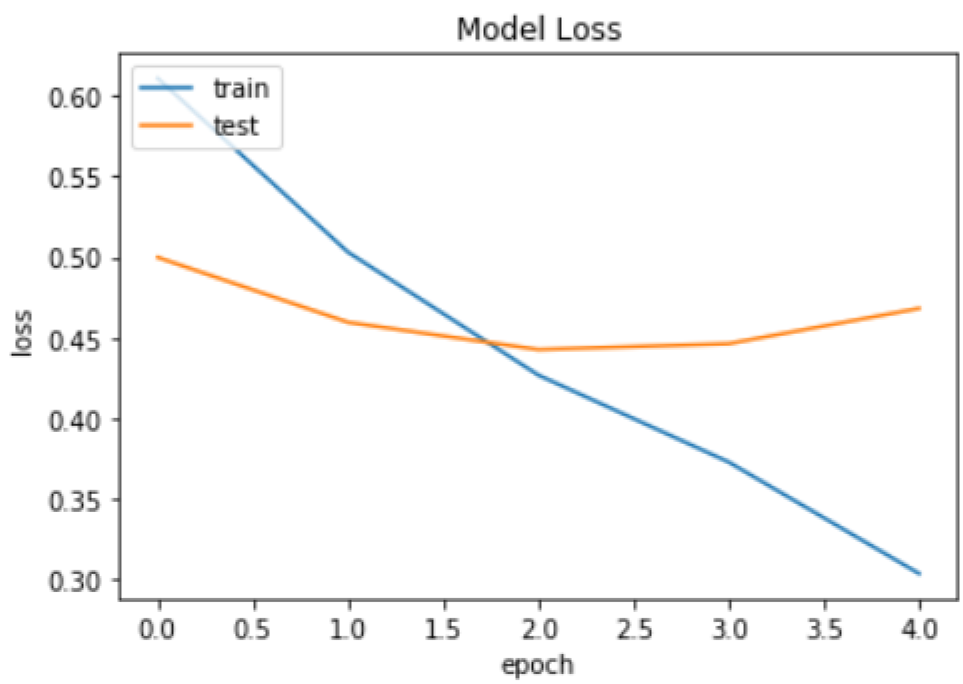
```
history_bow = model_bow.fit(X_train, y_train,epochs=5,verbose=False,validation_data=(X_test, y_test),batch_size=10)
```

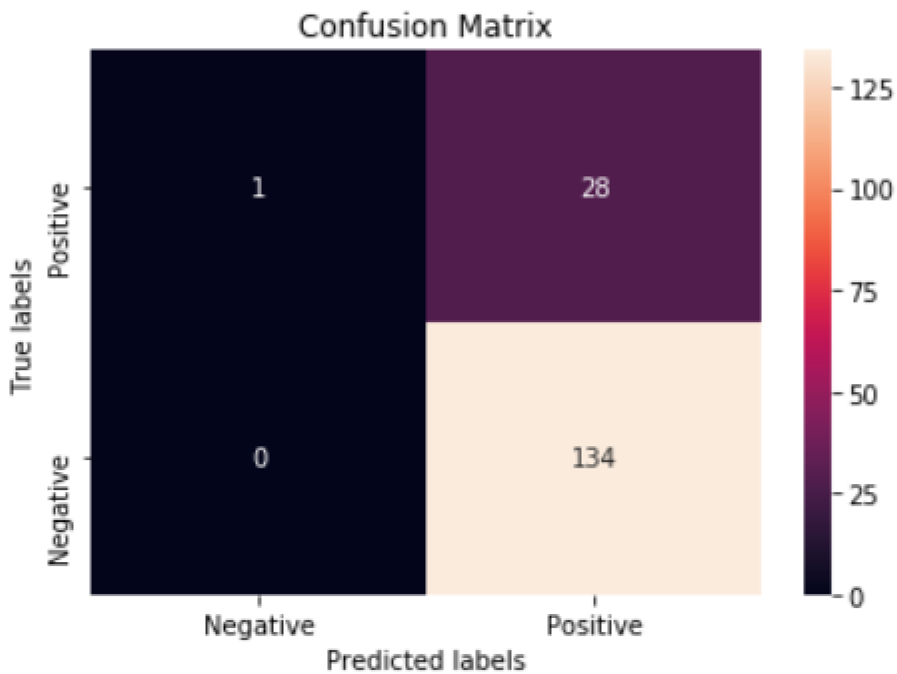
```
loss, accuracy = model_bow.evaluate(X_train, y_train)
print("Training Accuracy: {:.4f}".format(accuracy))
loss, accuracy = model_bow.evaluate(X_test, y_test)
print("Testing Accuracy:  {:.4f}".format(accuracy))
```

```
648/648 [==============================] - 0s 54us/step
Training Accuracy: 0.8673
163/163 [==============================] - 0s 61us/step
Testing Accuracy:  0.8282
```

```
Test Loss:  0.4681244474247189
Test Accuracy 0.8282208592613782
dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
```





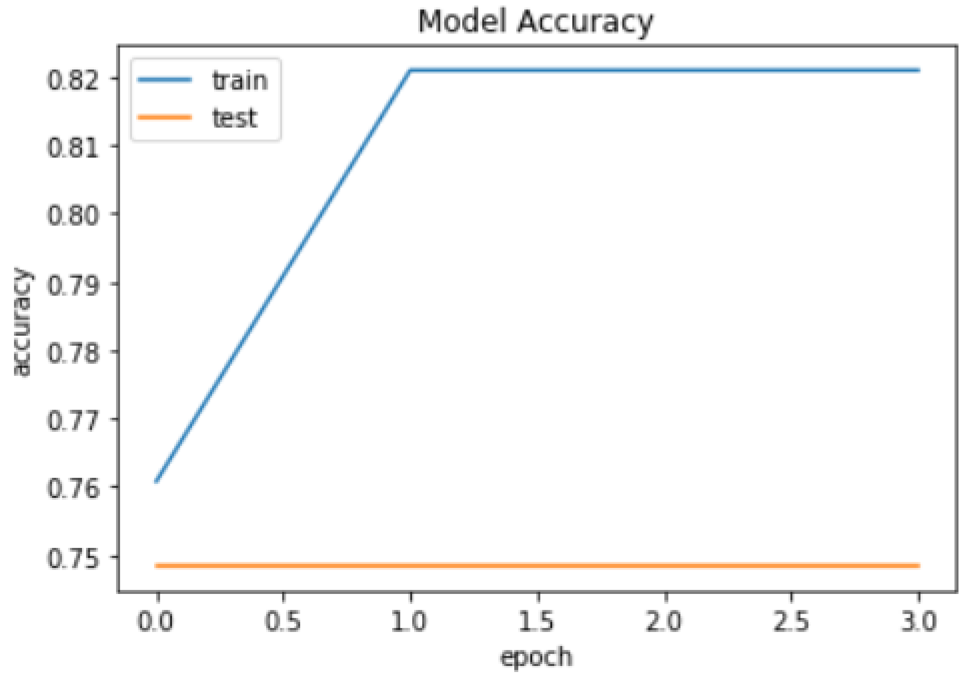


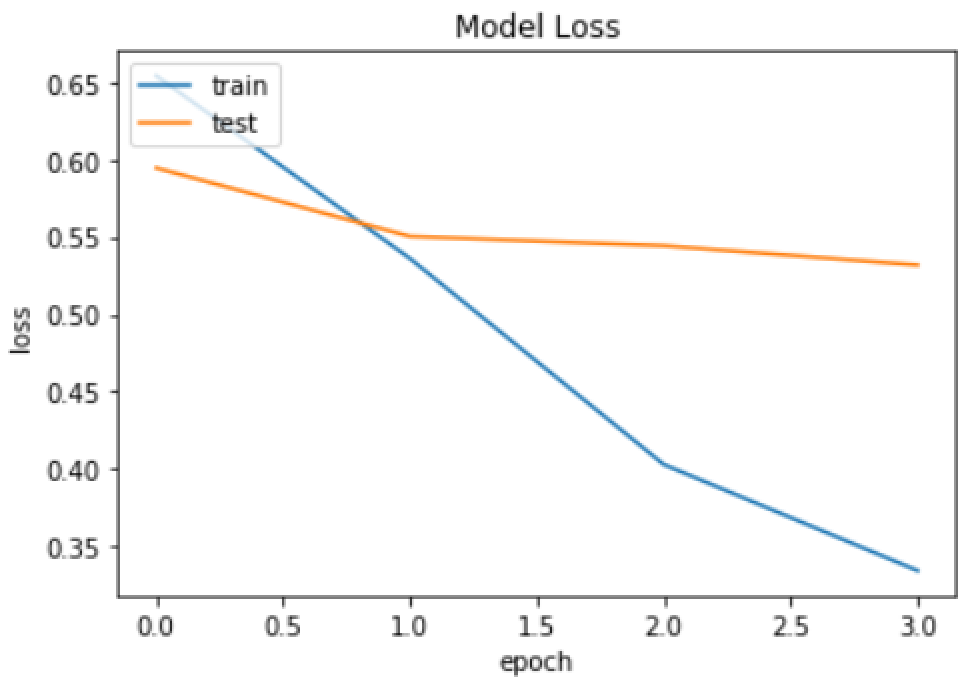
Creating a LSTM model using keras

```
[ ]  model_lstm = Sequential()
     model_lstm.add(Embedding(20000, 100, input_length=50))
     model_lstm.add(LSTM(100,activation='relu', dropout=0.5))
     model_lstm.add(Dense(1, activation='sigmoid'))
     model_lstm.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

```
Train on 648 samples, validate on 163 samples
Epoch 1/4
648/648 [==============================] - 5s 8ms/step - loss: 0.6542 - acc: 0.7608 - val_loss: 0.5946 - val_acc: 0.7485
Epoch 2/4
648/648 [==============================] - 2s 3ms/step - loss: 0.5361 - acc: 0.8210 - val_loss: 0.5504 - val_acc: 0.7485
Epoch 3/4
648/648 [==============================] - 2s 3ms/step - loss: 0.4030 - acc: 0.8210 - val_loss: 0.5443 - val_acc: 0.7485
Epoch 4/4
648/648 [==============================] - 2s 3ms/step - loss: 0.3345 - acc: 0.8210 - val_loss: 0.5319 - val_acc: 0.7485
```

```
Test Loss:  0.4681244474247189
Test Accuracy 0.8282208592613782
dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
```





Creating a Glove model

```
sequence_input = Input(shape=(50,), dtype='int32')
embedded_sequences = embedding_layer(sequence_input)
x = Conv1D(128, 5, activation='relu')(embedded_sequences)
x = MaxPooling1D(5)(x)
x = Conv1D(128, 5, activation='relu')(x)
x = MaxPooling1D(5)(x)
x = Conv1D(128, 5, activation='relu',data_format='channels_first')(x)
x = MaxPooling1D(35)(x)  # global max pooling
x = Flatten()(x)
x = Dense(128, activation='relu')(x)
preds = Dense(1, activation='softmax')(x)

model_glove = Model(sequence_input, preds)
model_glove.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['acc'])

# happy learning!
history_glove=model_glove.fit(x_train, y_train, validation_data=(x_val, y_val),epochs=10, batch_size=32)
```

```
Train on 649 samples, validate on 162 samples
Epoch 1/10
649/649 [==============================] - 4s 6ms/step - loss: 2.9477 - acc: 0.8151 - val_loss: 3.6412 - val_acc: 0.7716
Epoch 2/10
649/649 [==============================] - 1s 1ms/step - loss: 2.9477 - acc: 0.8151 - val_loss: 3.6412 - val_acc: 0.7716
Epoch 3/10
649/649 [==============================] - 1s 1ms/step - loss: 2.9477 - acc: 0.8151 - val_loss: 3.6412 - val_acc: 0.7716
Epoch 4/10
649/649 [==============================] - 1s 1ms/step - loss: 2.9477 - acc: 0.8151 - val_loss: 3.6412 - val_acc: 0.7716
Epoch 5/10
649/649 [==============================] - 1s 1ms/step - loss: 2.9477 - acc: 0.8151 - val_loss: 3.6412 - val_acc: 0.7716
Epoch 6/10
649/649 [==============================] - 1s 1ms/step - loss: 2.9477 - acc: 0.8151 - val_loss: 3.6412 - val_acc: 0.7716
Epoch 7/10
649/649 [==============================] - 1s 1ms/step - loss: 2.9477 - acc: 0.8151 - val_loss: 3.6412 - val_acc: 0.7716
Epoch 8/10
649/649 [==============================] - 1s 1ms/step - loss: 2.9477 - acc: 0.8151 - val_loss: 3.6412 - val_acc: 0.7716
Epoch 9/10
649/649 [==============================] - 1s 1ms/step - loss: 2.9477 - acc: 0.8151 - val_loss: 3.6412 - val_acc: 0.7716
Epoch 10/10
649/649 [==============================] - 1s 1ms/step - loss: 2.9477 - acc: 0.8151 - val_loss: 3.6412 - val_acc: 0.7716
```

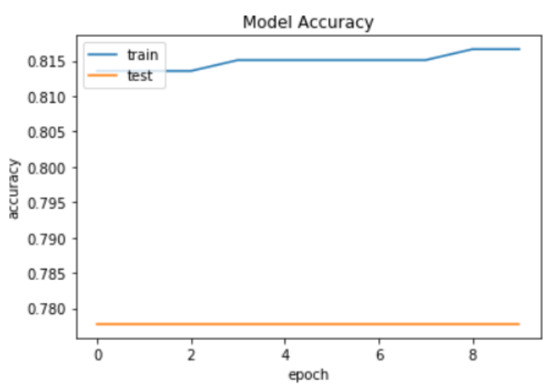
Getting a Test accuracy of 0.7716 with the Glove model

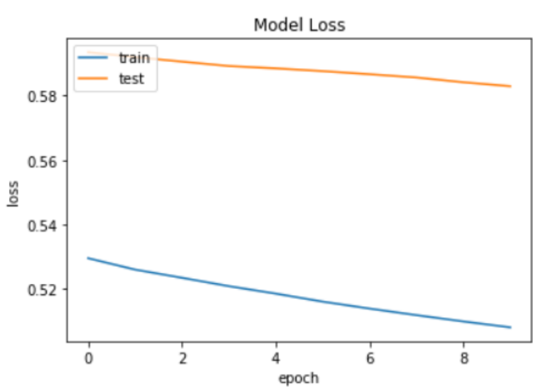
Trying out different models to get better accuracy

Model 2

```
from keras.optimizers import adam
sequence_input = Input(shape=(50,), dtype='int32')
embedded_sequences = embedding_layer_new(sequence_input)
x = Conv1D(32, 5, activation='relu')(embedded_sequences)
x = MaxPooling1D(5)(x)
#x = Conv1D(64, 5, activation='relu',data_format='channels_first')(x)
#x = MaxPooling1D(5)(x)  # global max pooling
x = Flatten()(x)
x = Dense(32, activation='relu')(x)
preds = Dense(1, activation='sigmoid')(x)

model_glove2 = Model(sequence_input, preds)
model_glove2.compile(loss='binary_crossentropy',
              optimizer=adam(lr=0.00001),
              metrics=['accuracy'])

# happy learning!
history_glove2=model_glove2.fit(x_train, y_train, validation_data=(x_val, y_val),epochs=10, batch_size=32)
```

```
Train on 649 samples, validate on 162 samples
Epoch 1/10
649/649 [==============================] - 3s 5ms/step - loss: 0.5295 - acc: 0.8136 - val_loss: 0.5935 - val_acc: 0.7778
Epoch 2/10
649/649 [==============================] - 0s 413us/step - loss: 0.5260 - acc: 0.8136 - val_loss: 0.5919 - val_acc: 0.7778
Epoch 3/10
649/649 [==============================] - 0s 421us/step - loss: 0.5235 - acc: 0.8136 - val_loss: 0.5905 - val_acc: 0.7778
Epoch 4/10
649/649 [==============================] - 0s 418us/step - loss: 0.5209 - acc: 0.8151 - val_loss: 0.5892 - val_acc: 0.7778
Epoch 5/10
649/649 [==============================] - 0s 438us/step - loss: 0.5185 - acc: 0.8151 - val_loss: 0.5884 - val_acc: 0.7778
Epoch 6/10
649/649 [==============================] - 0s 421us/step - loss: 0.5161 - acc: 0.8151 - val_loss: 0.5876 - val_acc: 0.7778
Epoch 7/10
649/649 [==============================] - 0s 433us/step - loss: 0.5139 - acc: 0.8151 - val_loss: 0.5866 - val_acc: 0.7778
Epoch 8/10
649/649 [==============================] - 0s 422us/step - loss: 0.5119 - acc: 0.8151 - val_loss: 0.5856 - val_acc: 0.7778
Epoch 9/10
649/649 [==============================] - 0s 420us/step - loss: 0.5099 - acc: 0.8166 - val_loss: 0.5841 - val_acc: 0.7778
Epoch 10/10
649/649 [==============================] - 0s 427us/step - loss: 0.5081 - acc: 0.8166 - val_loss: 0.5829 - val_acc: 0.7778
```



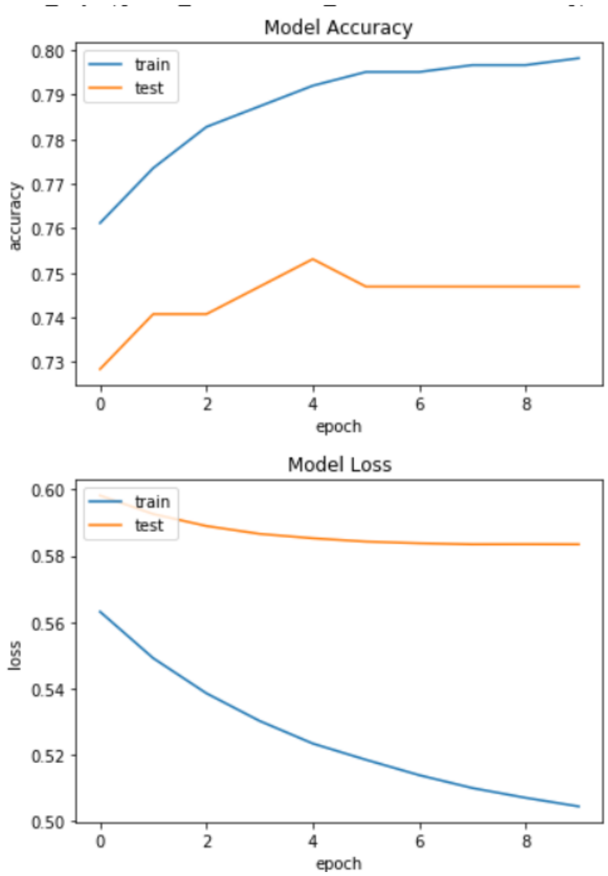


Model 3:

Model 3 Trainable True Plot

```
[ ]  import matplotlib.pyplot as plt
     #print(metrics.accuracy_score(Y_test, Y_predicted))

     #score = model_lstm.evaluate(X_test, y_test, verbose=3)
     #print('Test Loss: ', score[0])
     #print('Test Accuracy', score[1])


     # list all data in history
     print(history_glove2.history.keys())
     # summarize history for accuracy
     plt.plot(history_glove2.history['acc'])
     plt.plot(history_glove2.history['val_acc'])
     plt.title('Model Accuracy')
     plt.ylabel('accuracy')
     plt.xlabel('epoch')
     plt.legend(['train', 'test'], loc='upper left')
     plt.show()
     # summarize history for loss
     plt.plot(history_glove2.history['loss'])
     plt.plot(history_glove2.history['val_loss'])
     plt.title('Model Loss')
     plt.ylabel('loss')
     plt.xlabel('epoch')
     plt.legend(['train', 'test'], loc='upper left')
     plt.show()
```

```
Train on 649 samples, validate on 162 samples
Epoch 1/10
649/649 [==============================] - 4s 6ms/step - loss: 0.5631 - acc: 0.7612 - val_loss: 0.5982 - val_acc: 0.7284
Epoch 2/10
649/649 [==============================] - 0s 621us/step - loss: 0.5491 - acc: 0.7735 - val_loss: 0.5926 - val_acc: 0.7407
Epoch 3/10
649/649 [==============================] - 0s 644us/step - loss: 0.5385 - acc: 0.7827 - val_loss: 0.5890 - val_acc: 0.7407
Epoch 4/10
649/649 [==============================] - 0s 619us/step - loss: 0.5301 - acc: 0.7874 - val_loss: 0.5866 - val_acc: 0.7469
Epoch 5/10
649/649 [==============================] - 0s 633us/step - loss: 0.5233 - acc: 0.7920 - val_loss: 0.5853 - val_acc: 0.7531
Epoch 6/10
649/649 [==============================] - 0s 621us/step - loss: 0.5184 - acc: 0.7951 - val_loss: 0.5843 - val_acc: 0.7469
Epoch 7/10
649/649 [==============================] - 0s 612us/step - loss: 0.5138 - acc: 0.7951 - val_loss: 0.5838 - val_acc: 0.7469
Epoch 8/10
649/649 [==============================] - 0s 636us/step - loss: 0.5099 - acc: 0.7966 - val_loss: 0.5835 - val_acc: 0.7469
Epoch 9/10
649/649 [==============================] - 0s 615us/step - loss: 0.5069 - acc: 0.7966 - val_loss: 0.5835 - val_acc: 0.7469
Epoch 10/10
649/649 [==============================] - 0s 636us/step - loss: 0.5043 - acc: 0.7982 - val_loss: 0.5835 - val_acc: 0.7469
```



Experiment 2: Transfer learning

1. BOW

```
[ ] # Our vectorized labels
    y_train = np.asarray(train_labels).astype('float32')
    y_test = np.asarray(test_labels).astype('float32')
```

```
[ ] from sklearn.linear_model import LogisticRegression

    model = LogisticRegression()
    model.fit(x_train, y_train)
```

```
sr/local/lib/python3.6/dist-packages/sklearn/linear_model/logi
FutureWarning)
gisticRegression(C=1.0, class_weight=None, dual=False, fit_int
        intercept_scaling=1, max_iter=100, multi_class='warn',
        n_jobs=None, penalty='l2', random_state=None, solver='
        tol=0.0001, verbose=0, warm_start=False)
```

```
[ ] y_pred = model.predict(x_test)
    y_pred
```

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

```
[ ] from sklearn.metrics import classification_report, confusion_matr

    print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

         0.0       0.86      0.86      0.86     12500
         1.0       0.86      0.86      0.86     12500

   micro avg       0.86      0.86      0.86     25000
   macro avg       0.86      0.86      0.86     25000
weighted avg       0.86      0.86      0.86     25000
```

Used Grid Search and K-fold Cross Validation

```
GridSearchCV(cv=10, error_score='raise',
       estimator=SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False),
       fit_params=None, iid=True, n_jobs=-1,
       param_grid={'C': [1, 10, 100, 1000], 'gamma': [0.1, 0.01, 1, 0.001], 'kernel': ['linear', 'rbf']},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
```

[ ]
```
y_pred = CV.predict(X_test)
y_pred
```

```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
       1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0,
       1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 1], dtype=int64)
```

[ ]
```
print(metrics.accuracy_score(y_test, y_pred))
```

```
0.7914110429447853
```

[ ]
```
from sklearn.metrics import classification_report, confusion_matrix

print(classification_report(y_test, y_pred))
```

```
             precision    recall  f1-score   support

          0       0.42      0.26      0.32        31
          1       0.84      0.92      0.88       132

avg / total       0.76      0.79      0.77       163
```

[ ]
```
confusion_matrix(y_test, y_pred)
```

```
array([[  8,  23],
       [ 11, 121]], dtype=int64)
```

```
[ ]  tokenize = Tokenizer()
     tokenize.fit_on_texts(transcript['text'])
     seq = tokenize.texts_to_sequences(transcript['text'])
     pad = pad_sequences(seq , maxlen = 150)
     word_idx = tokenize1.word_index
     features = pad
     features
     #features.shape
```

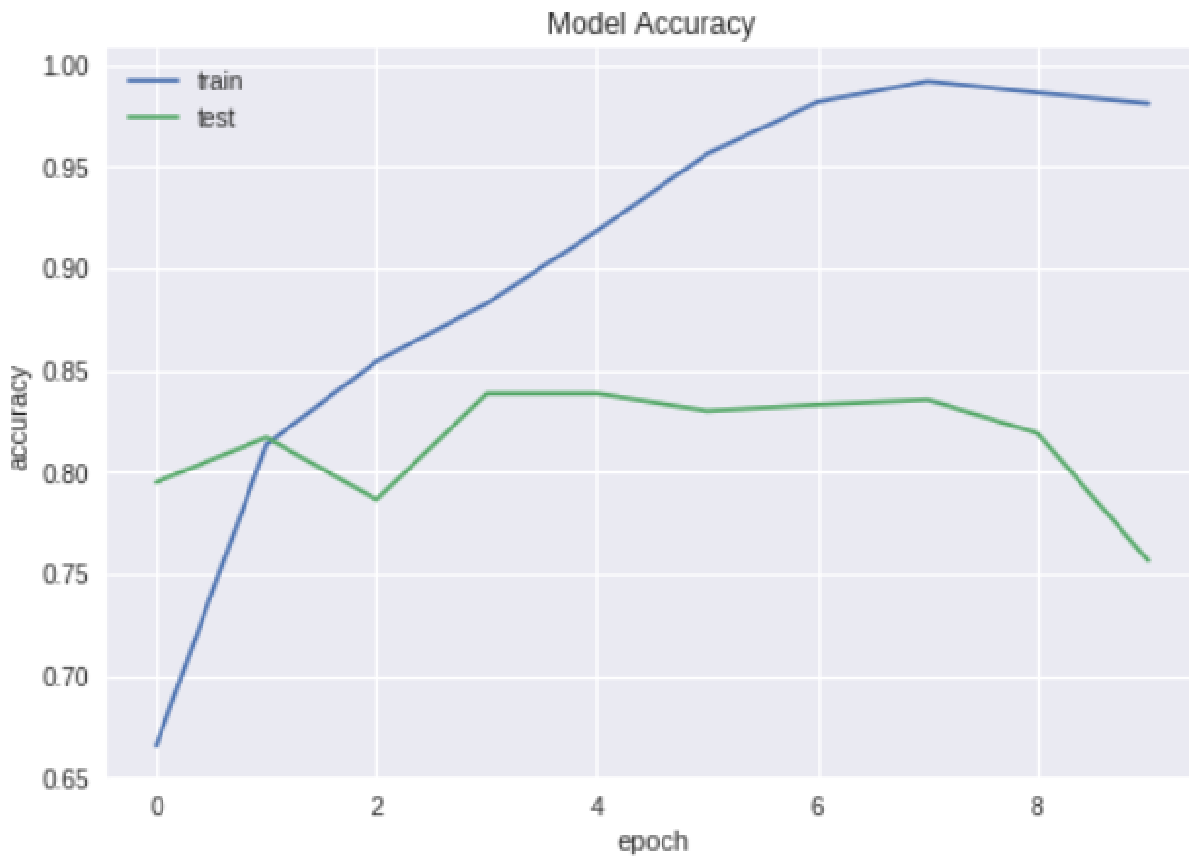
```
array([[   0,    0,    0, ..., 1137, 1477,  667],
       [   0,    0,    0, ...,  426,   51,   70],
       [   0,    0,    0, ...,  535, 1025,  276],
       ...,
       [   0,    0,    0, ...,  187, 1176,  709],
       [   0,    0,    0, ...,  137, 1133,  960],
       [   0,    0,    0, ...,  116, 1527,  474]])
```

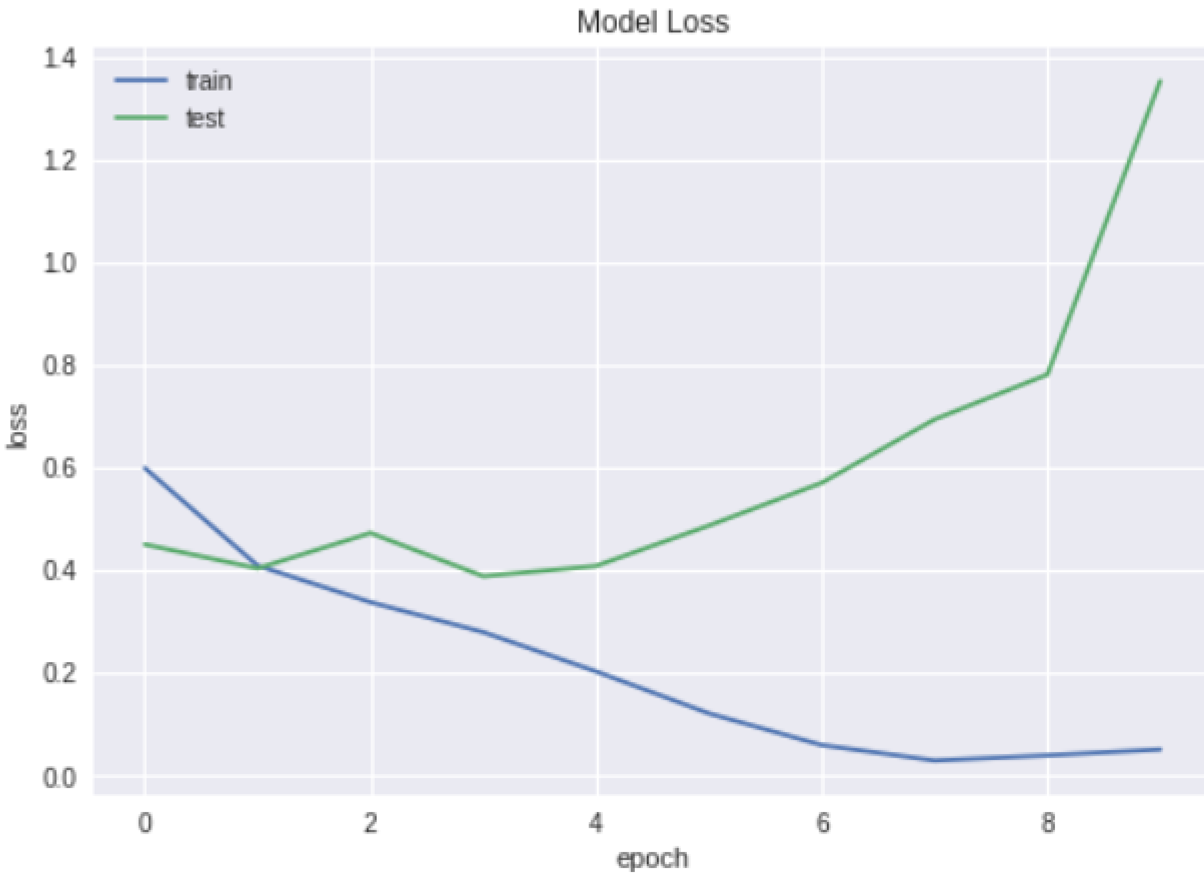
2. GLOVE

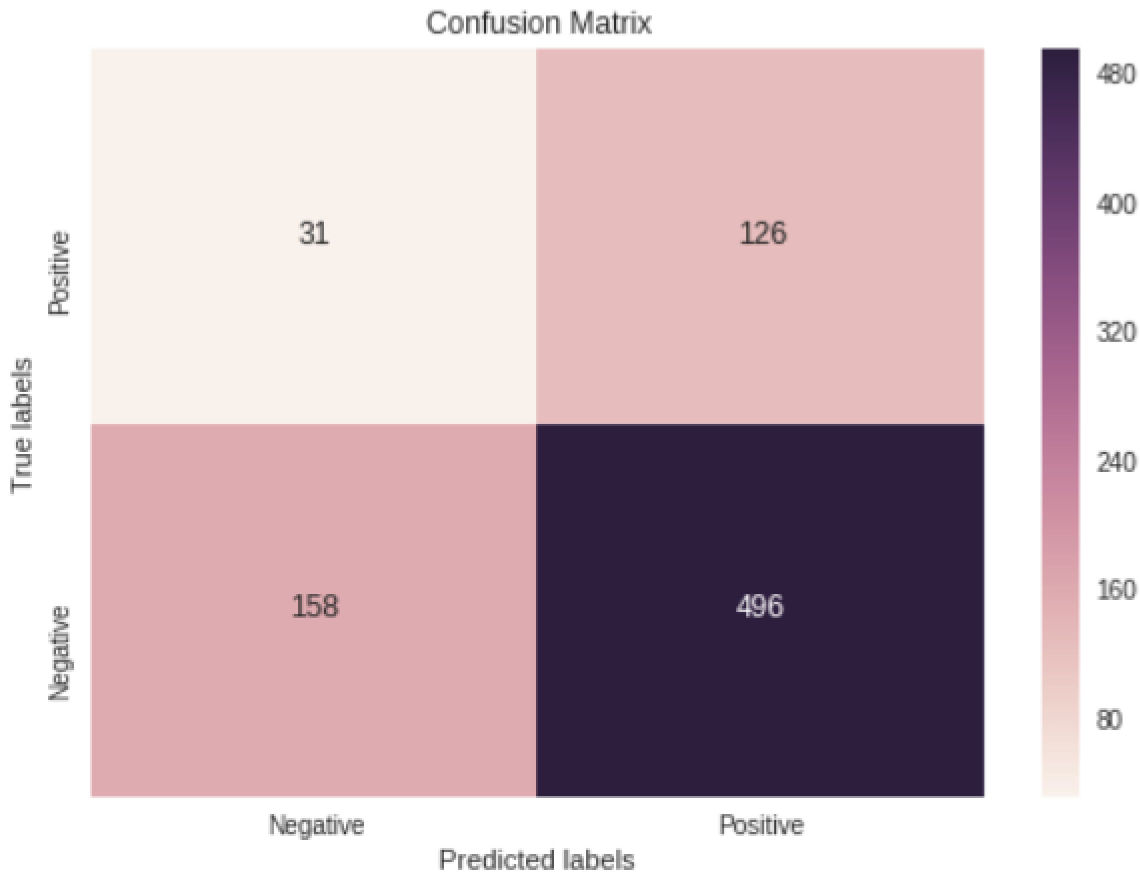
Used Pretrained model

```
from keras.models import Sequential
from keras.layers import Embedding, Flatten, Dense



from keras.layers import Embedding

embedding_layer = Embedding(max_words,
                            embedding_dim,
                            embeddings_initializer=Constant(embedding_matrix),
                            input_length=maxlen,
                            trainable=False)


sequence_input = Input(shape=(maxlen,), dtype='int32')
embedded_sequences = embedding_layer(sequence_input)
x = Conv1D(128, 5, activation='relu')(embedded_sequences)
x = MaxPooling1D(5)(x)
x = Conv1D(128, 5, activation='relu',data_format='channels_first')(x)
x = MaxPooling1D(35)(x)  # global max pooling
x = Flatten()(x)
x = Dense(128, activation='relu')(x)
preds = Dense(1, activation='sigmoid')(x)

model_c = Model(sequence_input, preds)
model_c.compile(loss='binary_crossentropy',
              optimizer=adam(lr=0.001),
              metrics=['acc'])

# happy learning!
history = model_c.fit(x_train, y_train, validation_data=(x_val, y_val),epochs=10, batch_size=32)
```

```
Train on 10000 samples, validate on 2000 samples
Epoch 1/10
10000/10000 [==============================] - 6s 590us/step - loss: 0.5987 - acc: 0.6656 - val_loss: 0.4501 - val_acc: 0.7950
Epoch 2/10
10000/10000 [==============================] - 3s 292us/step - loss: 0.4087 - acc: 0.8135 - val_loss: 0.4034 - val_acc: 0.8170
Epoch 3/10
10000/10000 [==============================] - 3s 295us/step - loss: 0.3374 - acc: 0.8544 - val_loss: 0.4723 - val_acc: 0.7865
Epoch 4/10
10000/10000 [==============================] - 3s 294us/step - loss: 0.2791 - acc: 0.8829 - val_loss: 0.3880 - val_acc: 0.8385
Epoch 5/10
10000/10000 [==============================] - 3s 293us/step - loss: 0.2026 - acc: 0.9185 - val_loss: 0.4081 - val_acc: 0.8385
Epoch 6/10
10000/10000 [==============================] - 3s 299us/step - loss: 0.1208 - acc: 0.9564 - val_loss: 0.4867 - val_acc: 0.8300
Epoch 7/10
10000/10000 [==============================] - 3s 299us/step - loss: 0.0592 - acc: 0.9819 - val_loss: 0.5700 - val_acc: 0.8330
Epoch 8/10
10000/10000 [==============================] - 3s 303us/step - loss: 0.0291 - acc: 0.9921 - val_loss: 0.6935 - val_acc: 0.8355
Epoch 9/10
10000/10000 [==============================] - 3s 320us/step - loss: 0.0391 - acc: 0.9864 - val_loss: 0.7809 - val_acc: 0.8190
Epoch 10/10
10000/10000 [==============================] - 3s 331us/step - loss: 0.0507 - acc: 0.9811 - val_loss: 1.3525 - val_acc: 0.7565
```




Model Loss
loss
epoch
train
test
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8

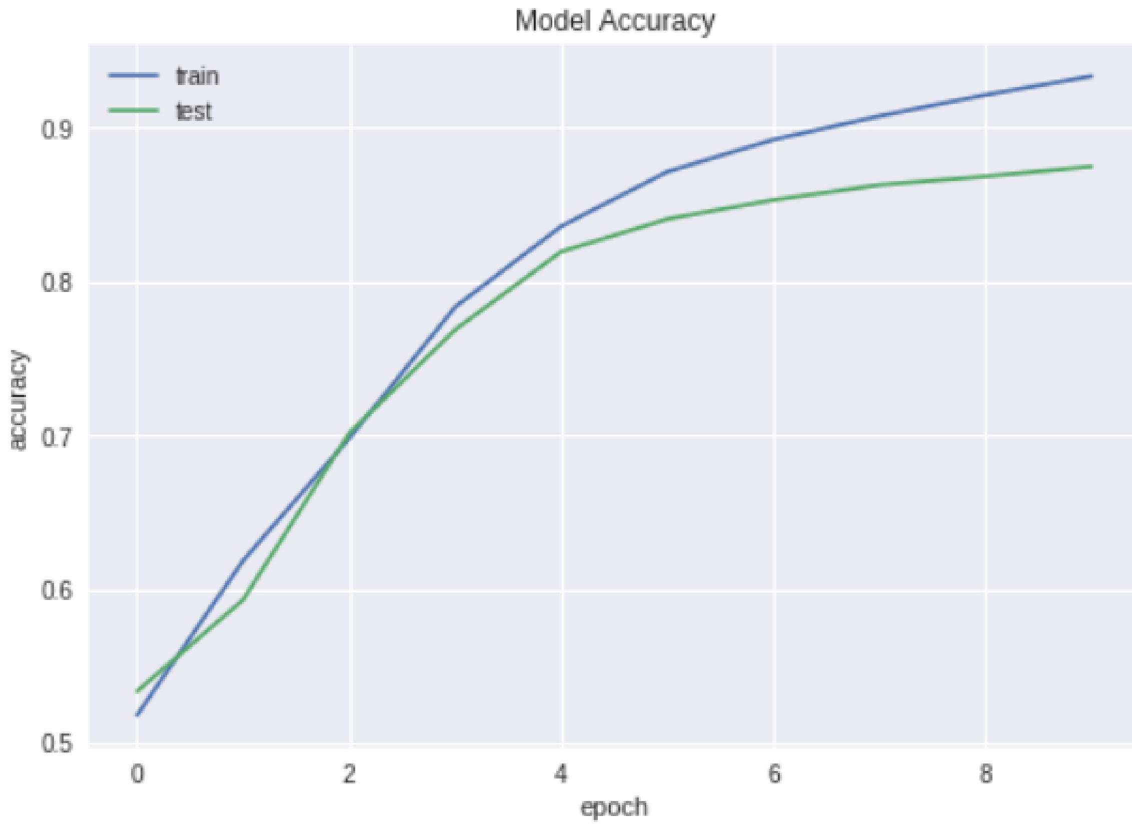


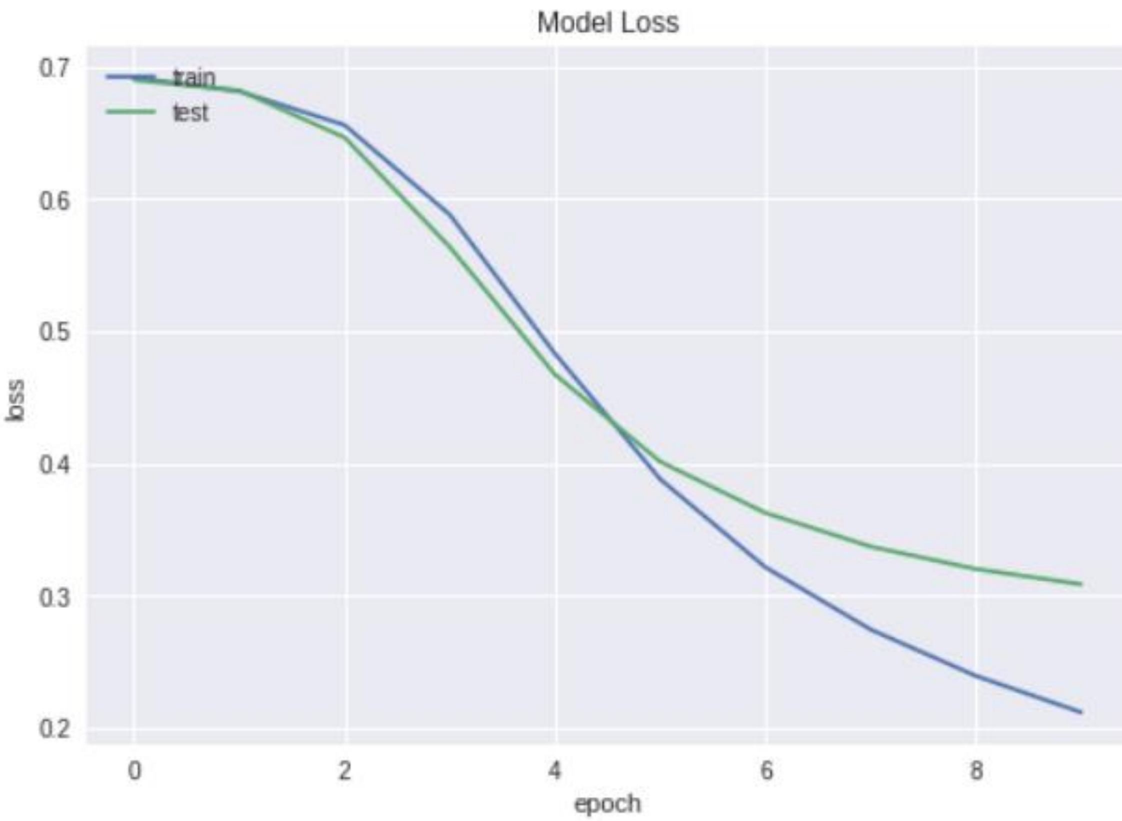
Word Embedding

```
from keras.models import Sequential
from keras.layers import Flatten, Dense, Dropout
from keras.optimizers import adam

model = Sequential()
model.add(Embedding(max_features, 64, input_length=maxlen))

model.add(Flatten())
model.add(Dense(32, activation='relu'))
model.add(Dropout(0.25))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer=adam(lr=0.0001), loss='binary_crossentropy', metrics=['accuracy'])
model.summary()

train_history = model.fit(x_train, y_train,
                    epochs=10,
                    batch_size=256,
                    validation_split=0.2)
```

```
_________________________________________________________________
flatten_3 (Flatten)          (None, 12800)             0
_________________________________________________________________
dense_5 (Dense)              (None, 32)                409632
_________________________________________________________________
dropout_3 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 33
=================================================================
Total params: 1,049,665
Trainable params: 1,049,665
Non-trainable params: 0
_________________________________________________________________
Train on 20000 samples, validate on 5000 samples
Epoch 1/10
20000/20000 [==============================] - 4s 214us/step - loss: 0.6920 - acc: 0.5179 - val_loss: 0.6902 - val_acc: 0.5334
Epoch 2/10
20000/20000 [==============================] - 4s 200us/step - loss: 0.6817 - acc: 0.6185 - val_loss: 0.6823 - val_acc: 0.5930
Epoch 3/10
20000/20000 [==============================] - 4s 203us/step - loss: 0.6561 - acc: 0.6981 - val_loss: 0.6467 - val_acc: 0.7014
Epoch 4/10
20000/20000 [==============================] - 4s 203us/step - loss: 0.5883 - acc: 0.7836 - val_loss: 0.5638 - val_acc: 0.7688
Epoch 5/10
20000/20000 [==============================] - 4s 200us/step - loss: 0.4831 - acc: 0.8361 - val_loss: 0.4673 - val_acc: 0.8196
Epoch 6/10
20000/20000 [==============================] - 4s 205us/step - loss: 0.3881 - acc: 0.8716 - val_loss: 0.4013 - val_acc: 0.8408
Epoch 7/10
20000/20000 [==============================] - 4s 198us/step - loss: 0.3214 - acc: 0.8925 - val_loss: 0.3626 - val_acc: 0.8532
Epoch 8/10
20000/20000 [==============================] - 4s 201us/step - loss: 0.2743 - acc: 0.9078 - val_loss: 0.3371 - val_acc: 0.8630
Epoch 9/10
20000/20000 [==============================] - 4s 206us/step - loss: 0.2392 - acc: 0.9216 - val_loss: 0.3201 - val_acc: 0.8686
Epoch 10/10
20000/20000 [==============================] - 4s 200us/step - loss: 0.2117 - acc: 0.9338 - val_loss: 0.3086 - val_acc: 0.8750
```

```
Test Loss:  0.30974822925567624
Test Accuracy 0.8712
dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
```





LSTM,

Stacked LSTM,

STACKED LSTM

```
[ ]  from keras.layers import LSTM

     model_lstm = Sequential()
     model_lstm.add(Embedding(max_features, 64))
     model_lstm.add(LSTM(64, dropout=0.2, recurrent_dropout=0.2, return_sequences=True))
     #model_lstm.add(Dropout(0.2))
     model_lstm.add(LSTM(32, dropout=0.2, recurrent_dropout=0.2))

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

     model_lstm.compile(optimizer=adam(lr=0.0001),
                  loss='binary_crossentropy',
                  metrics=['accuracy'])

     history = model_lstm.fit(x_train, y_train,
                       epochs=10,
                       batch_size=64,
                       validation_split=0.2)
```

```
Train on 20000 samples, validate on 5000 samples
Epoch 1/10
20000/20000 [==============================] - 157s 8ms/step - loss: 0.6425 - acc: 0.6336 - val_loss: 0.4882 - val_acc: 0.7878
Epoch 2/10
20000/20000 [==============================] - 155s 8ms/step - loss: 0.4454 - acc: 0.8076 - val_loss: 0.4223 - val_acc: 0.8152
Epoch 3/10
20000/20000 [==============================] - 154s 8ms/step - loss: 0.3710 - acc: 0.8507 - val_loss: 0.3842 - val_acc: 0.8356
Epoch 4/10
20000/20000 [==============================] - 157s 8ms/step - loss: 0.3313 - acc: 0.8678 - val_loss: 0.3746 - val_acc: 0.8382
Epoch 5/10
20000/20000 [==============================] - 156s 8ms/step - loss: 0.3085 - acc: 0.8803 - val_loss: 0.3744 - val_acc: 0.8400
Epoch 6/10
20000/20000 [==============================] - 156s 8ms/step - loss: 0.2877 - acc: 0.8913 - val_loss: 0.3756 - val_acc: 0.8412
Epoch 7/10
20000/20000 [==============================] - 155s 8ms/step - loss: 0.2745 - acc: 0.8944 - val_loss: 0.3813 - val_acc: 0.8388
Epoch 8/10
20000/20000 [==============================] - 156s 8ms/step - loss: 0.2582 - acc: 0.9045 - val_loss: 0.3961 - val_acc: 0.8294
Epoch 9/10
20000/20000 [==============================] - 156s 8ms/step - loss: 0.2488 - acc: 0.9096 - val_loss: 0.3994 - val_acc: 0.8414
Epoch 10/10
20000/20000 [==============================] - 156s 8ms/step - loss: 0.2334 - acc: 0.9148 - val_loss: 0.4042 - val_acc: 0.8398
```

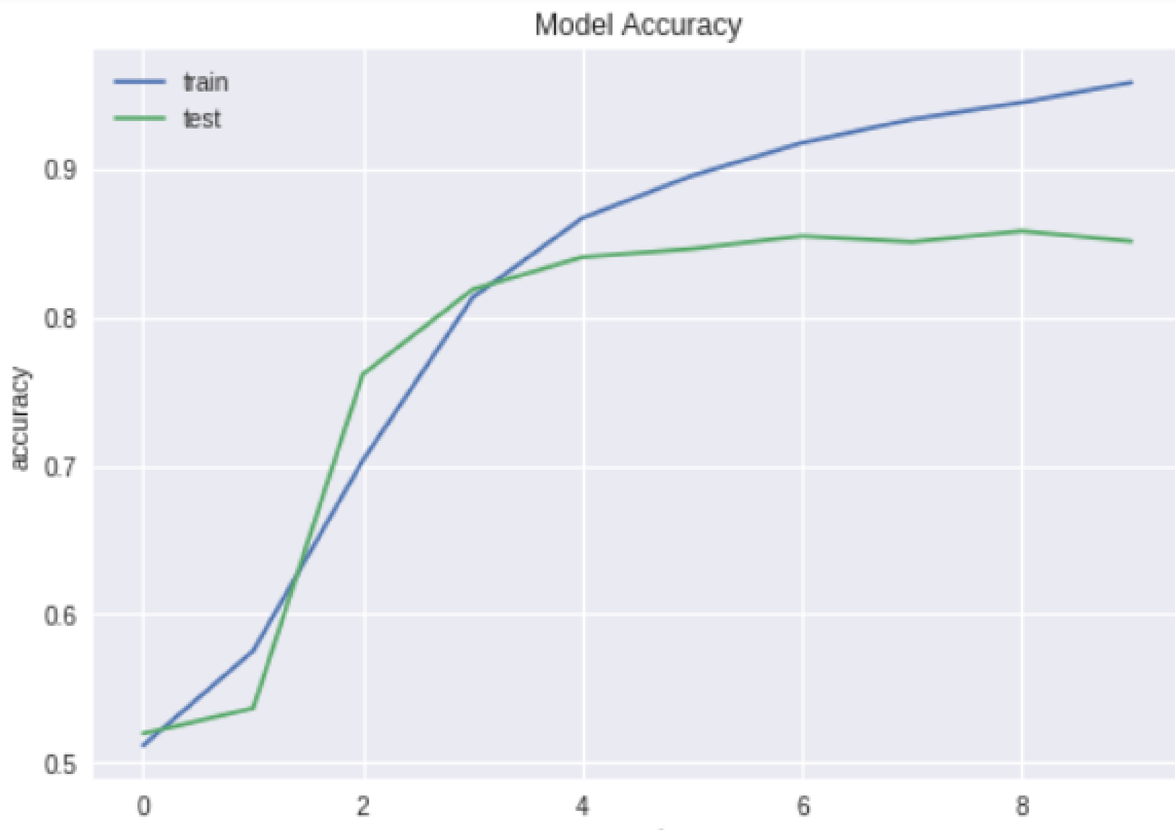
```
Test Loss:  0.41616990515708924
Test Accuracy 0.83632
```

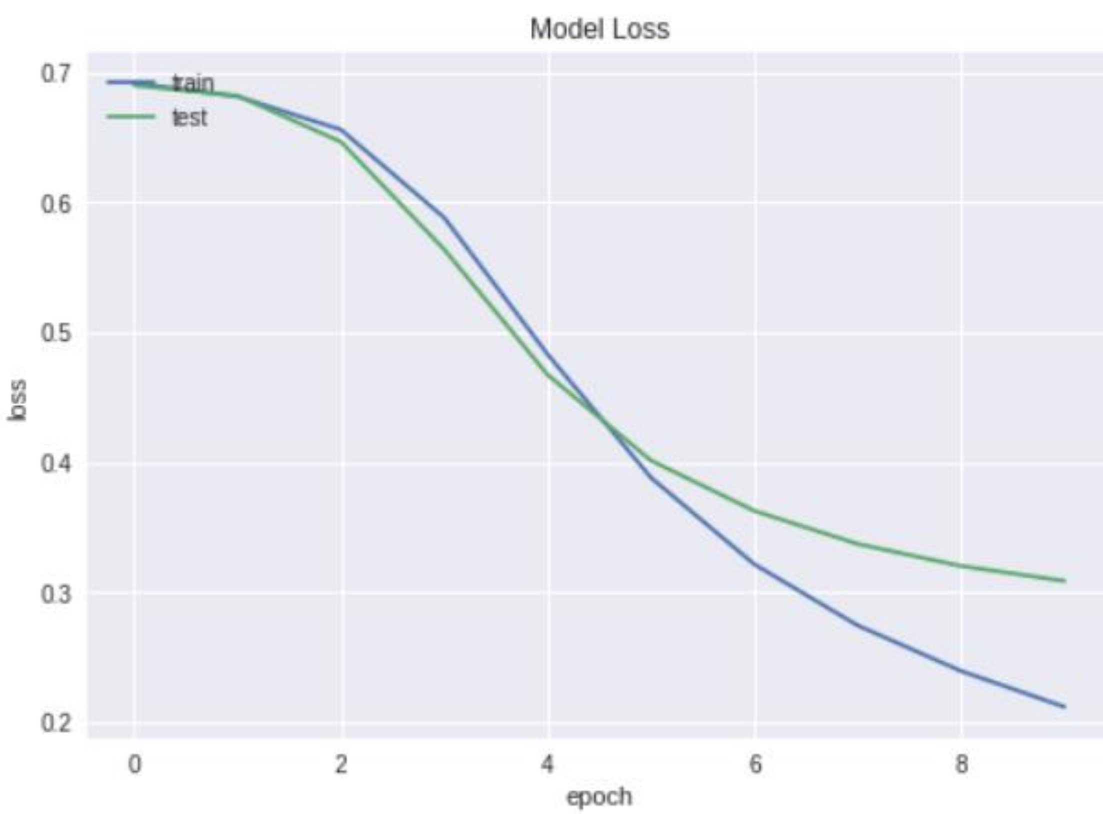
GRU(Gated Recurring Units),

RNN,

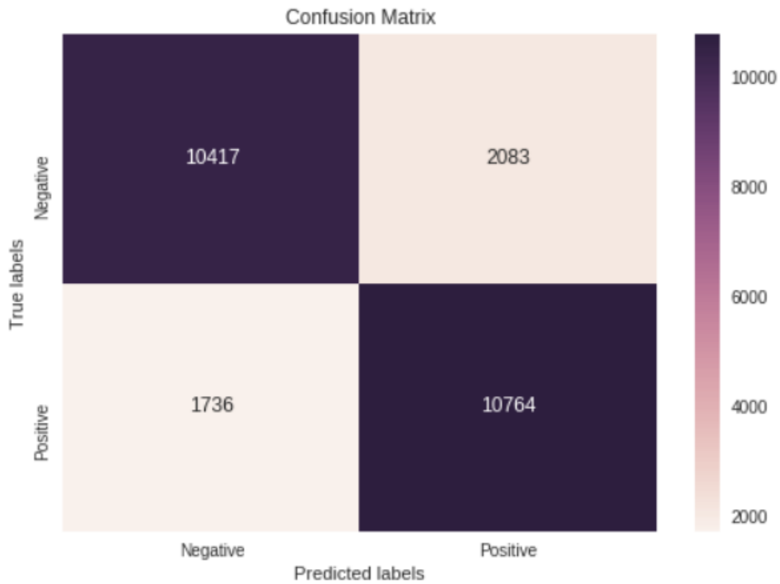
```
[ ]  #SImple RNN model
     from keras.layers import Dense, SimpleRNN

     model = Sequential()
     model.add(Embedding(max_features, 32))
     model.add(SimpleRNN(32))
     model.add(Dense(1, activation='sigmoid'))

     model.compile(optimizer=adam(0.0001), loss='binary_crossentropy', metrics=['accuracy'])
     history = model.fit(x_train, y_train,
                         epochs=10,
                         batch_size=64,
                         validation_split=0.2)
```

```
Train on 20000 samples, validate on 5000 samples
Epoch 1/10
20000/20000 [==============================] - 13s 632us/step - loss: 0.6946 - acc: 0.5116 - val_loss: 0.6919 - val_acc: 0.5196
Epoch 2/10
20000/20000 [==============================] - 12s 607us/step - loss: 0.6795 - acc: 0.5752 - val_loss: 0.6876 - val_acc: 0.5366
Epoch 3/10
20000/20000 [==============================] - 12s 607us/step - loss: 0.6026 - acc: 0.7038 - val_loss: 0.5313 - val_acc: 0.7618
Epoch 4/10
20000/20000 [==============================] - 12s 611us/step - loss: 0.4479 - acc: 0.8135 - val_loss: 0.4405 - val_acc: 0.8190
Epoch 5/10
20000/20000 [==============================] - 12s 618us/step - loss: 0.3451 - acc: 0.8672 - val_loss: 0.3889 - val_acc: 0.8408
Epoch 6/10
20000/20000 [==============================] - 12s 623us/step - loss: 0.2835 - acc: 0.8958 - val_loss: 0.3699 - val_acc: 0.8464
Epoch 7/10
20000/20000 [==============================] - 12s 621us/step - loss: 0.2360 - acc: 0.9178 - val_loss: 0.3565 - val_acc: 0.8550
Epoch 8/10
20000/20000 [==============================] - 12s 617us/step - loss: 0.2023 - acc: 0.9337 - val_loss: 0.3491 - val_acc: 0.8510
Epoch 9/10
20000/20000 [==============================] - 12s 611us/step - loss: 0.1745 - acc: 0.9451 - val_loss: 0.3481 - val_acc: 0.8584
Epoch 10/10
20000/20000 [==============================] - 12s 611us/step - loss: 0.1448 - acc: 0.9585 - val_loss: 0.3538 - val_acc: 0.8514
```


Model Accuracy
train
test
accuracy
0.5
0.6
0.7
0.8
0.9
0
2
4
6
8


Model Loss
train
test
loss
0.2
0.3
0.4
0.5
0.6
0.7
0
2
4
6
8
epoch

```
import matplotlib.pyplot as plt

ax= plt.subplot()
sns.heatmap(cm, annot=True, ax = ax, fmt='g'); #annot=True to annotate cells

# labels, title and ticks
ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
ax.xaxis.set_ticklabels(['Negative', 'Positive']); ax.yaxis.set_ticklabels(['Positive', 'Negative']);
```



Bi-directional RNN

We have created our own word embeddings on IMDB dataset and trained it

Used Transfer learning to predict sentiments for EDGAR datasets

Experiment 3: Using APIs

Using the Amazon, Google, Microsoft and Watson APIs, obtain the sentiment scores for your entire dataset.

Step 1:

Fetching the initial dataset with labelled transcripts

```
[ ] with open('./final_json_data.json') as f:
        df = pd.DataFrame(json.load(f))
```

```
[ ] df.head()
```

	sentiment	text
0	positive	Thank you. Good afternoon, everyone. And welco...
1	positive	As we look past Q1, we expect the channel inve...
2	neutral	And your last question comes from the line of ...
3	negative	On the China gaming weakness, is it the slower...
4	negative	I don't know that we could tear that apart, te...

Step 2a : Using Amazon API - aws comprehend

```
[ ] start = time.time()
    for text in text_list:
        my_json = json.loads(json.dumps(comprehend.detect_sentiment(Text=text, LanguageCode='en'), sort_keys=True))
        amazon_sentiments_label.append(my_json['Sentiment'].lower())
        amazon_score_mixed.append(my_json['SentimentScore']['Mixed'])
        amazon_score_negative.append(my_json['SentimentScore']['Negative'])
        amazon_score_positive.append(my_json['SentimentScore']['Positive'])
        amazon_score_neutral.append(my_json['SentimentScore']['Neutral'])
    end = time.time()
    print("Time to execute", end-start)
```

```
Time to execute 96.88824228052368
```

```
[ ] df.head()
```

	sentiment	text	amazon_sentiments_label	amazon_score_mixed	amazon_score_negative	amazon_score_neutral	amazon_score_positive
0	positive	Thank you. Good afternoon, everyone. And welco...	neutral	0.001549	0.000533	0.750581	0.247337
1	positive	As we look past Q1, we expect the channel inve...	neutral	0.009229	0.006072	0.880566	0.104133
2	neutral	And your last question comes from the line of ...	neutral	0.000831	0.008268	0.980631	0.010270
3	negative	On the China gaming weakness, is it the slower...	negative	0.011638	0.953385	0.034877	0.000099
4	negative	I don't know that we could tear that apart, te...	positive	0.014957	0.000221	0.007328	0.977495

Step 2b : Using Watson API – natural language understanding

```
[ ]  import os
     import json
     from watson_developer_cloud import NaturalLanguageUnderstandingV1
     from watson_developer_cloud.natural_language_understanding_v1 import Features, SentimentOptions
```

```
[ ]  api_key = os.environ.get('IAM_ACCESS_IBM')
     url = "https://gateway.watsonplatform.net/natural-language-understanding/api"
     natural_language_understanding = NaturalLanguageUnderstandingV1(
         version='2018-11-16',
         iam_apikey=api_key,
         url=url
     )
     def ibm_sentiments(data):
     #     data = ("I've seen things you people wouldn't believe. Attack ships on fire off the shoulder of Orion. I watched C-b
         response = natural_language_understanding.analyze(
             text=data,
             features=Features(sentiment=SentimentOptions()),
             language="en"
         ).get_result()
         response = json.loads(json.dumps(response))
         return response
```

```
[ ]  start = time.time()
     for text in text_list:
         my_json = ibm_sentiments(text)
         ibm_sentiments_label.append(my_json['sentiment']['document']['label'].lower())
         ibm_score.append(my_json['sentiment']['document']['score'])
     end = time.time()
     print("Time to execute", end-start)
```

```
Time to execute 626.9456360340118
```

ibm_sentiments_label	ibm_score
positive	0.816136
positive	0.558518
neutral	0.000000
negative	-0.598559
positive	0.790615

Step 2c : Google cloud language API

```
## This code runs on linux and not on Windows
# Imports the Google Cloud client library
from google.cloud import language
from google.cloud.language import enums
from google.cloud.language import types

# Instantiates a client
client = language.LanguageServiceClient()

def google_sentiments_api(text_in):
    # The text to analyze
    text = text_in
    document = types.Document(
        content=text,
        type=enums.Document.Type.PLAIN_TEXT)

    # Detects the sentiment of the text
    sentiment = client.analyze_sentiment(document=document).document_sentiment
    return sentiment

# print('Text: {}'.format(text))
# print('Sentiment: {}, {}'.format(sentiment.score, sentiment.magnitude))
```

The score of a document's sentiment indicates the overall emotion of a document. The magnitude of a document's sentiment indicates how much emotional content is present within the document, and this value is often proportional to the length of the document.

It is important to note that the Natural Language API indicates differences between positive and negative emotion in a document, but does not identify specific positive and negative emotions. For example, "angry" and "sad" are both considered negative emotions. However, when the Natural Language API analyzes text that is considered "angry", or text that is considered "sad", the response only indicates that the sentiment in the text is negative, not "sad" or "angry".

A document with a neutral score (around 0.0) may indicate a low-emotion document, or may indicate mixed emotions, with both high positive and negative values which cancel each out. Generally, you can use magnitude values to disambiguate these cases, as truly neutral documents will have a low magnitude value, while mixed documents will have higher magnitude values.

When comparing documents to each other (especially documents of different length), make sure to use the magnitude values to calibrate your scores, as they can help you gauge the relevant amount of emotional content.

The chart below shows some sample values and how to interpret them:

Sentiment Sample Values

Clearly Positive* "score": 0.8, "magnitude": 3.0

Clearly Negative* "score": -0.6, "magnitude": 4.0

Neutral "score": 0.1, "magnitude": 0.0

Mixed "score": 0.0, "magnitude": 4.0

* “Clearly positive” and “clearly negative” sentiment varies for different use cases and customers. You might find differing results for your specific scenario. We recommend that you define a threshold that works for you, and then adjust the threshold after testing and verifying the results. For example, you may define a threshold of any score over 0.25 as clearly positive, and then modify the score threshold to 0.15 after reviewing your data and results and finding that scores from 0.15-0.25 should be considered positive as well.

```
[ ]  def get_label(sentiment):
         label = ""
         if sentiment.score > 0.5 and sentiment.magnitude > 1.5:
             label = "positive"
         elif sentiment.score < -0.5 and sentiment.magnitude > 1.5:
             label = "negative"
         else:
             label = "neutral"
         return label
```

```
[ ]  google_sentiment_socre = []
     google_sentiment_magnitude = []
     google_sentiment_label = []
```

```
[ ]  start = time.time()
     # i = 0
     for text in text_list:
         sentiment = google_sentiments_api(text)
     #     print("Index" + str(i) + " Score: " + str(sentiment.score) + " Magn: " + str(sentiment.magnitude))
         google_sentiment_socre.append(sentiment.score)
         google_sentiment_magnitude.append(sentiment.magnitude)
         google_sentiment_label.append(get_label(sentiment))
     #     i+=1
     end = time.time()
     print("Time to execute", end-start)
```

```
Time to execute 238.47278022766113
```

google_sentiment_socre	google_sentiment_magnitude	google_sentiment_label
0.2	1.1	positive
0.2	1.4	positive
0.0	0.0	neutral
-0.3	1.2	negative
0.5	7.9	positive

Step 2d: Azure text analysis API

Preparing documents

```
document_list = []
for i,text in enumerate(text_list):
    document = {"id":str(i),
                "language":"en",
                "text":text[:5119]
    }
    document_list.append(document)
```

```
document_list[1]
```

```
{'id': '1',
 'language': 'en',
 'text': "As we look past Q1, we expect the channel invent
```

```
document_part_1 = {"documents" : document_list[:1000]}
```

```
document_part_2 = {"documents" : document_list[1000:]}
```

Getting sentiment score for part 1

```
[ ]  start = time.time()
     headers   = {"Ocp-Apim-Subscription-Key": subscription_key}
     response_1  = requests.post(sentiment_api_url, headers=headers, json=document_part_1)
     sentiments_part_1 = response_1.json()
     pprint(sentiments_part_1)
     end = time.time()
     print("Time to execute", end-start)
```

Getting sentiment score for part 2

```
[ ]  start = time.time()
     headers   = {"Ocp-Apim-Subscription-Key": subscription_key}
     response_2  = requests.post(sentiment_api_url, headers=headers, json=document_part_2)
     sentiments_part_2 = response_2.json()
     pprint(sentiments_part_2)
     end = time.time()
     print("Time to execute", end-start)
```

azure_api_score	azure_api_label
0.978328	positive
0.500000	neutral
0.500000	neutral
0.500000	neutral

Saving the file with sentiment data from all API's

```
df.to_json(path_or_buf ="final_label_json_data.json",orient='records')
```

Step 3 : Normalizing sentiment scores

- Removing Neutral sentiments

```
[ ]  #Removing the neutral sentiments for our model

     sentiment_df = sentiment_df[sentiment_df['sentiment'] != 'neutral']
     sentiment_df.head()
```

	azure_api_score	google_sentiment_socre	ibm_score	amazon_sentiment_score	sentiment
0	0.978328	0.2	0.816136	0.750581	positive
1	0.500000	0.2	0.558518	0.880566	positive
3	0.500000	-0.3	-0.598559	0.953385	negative
4	0.905933	0.5	0.790615	0.977495	negative
5	0.904133	0.0	0.988573	0.674756	positive

```
[ ]  sentiment_df.shape
```

```
(811, 5)
```

- Build a model to map the output Sentiement label with the sentiment scores from all 4 apis

```
#label encode the output variable
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
sentiment_df['sentiment'] = le.fit_transform(sentiment_df['sentiment'])
```

```
sentiment_df.head()
```

	azure_api_score	google_sentiment_socre	ibm_score	amazon_sentiment_score	sentiment
0	0.978328	0.2	0.816136	0.750581	1
1	0.500000	0.2	0.558518	0.880566	1
3	0.500000	-0.3	-0.598559	0.953385	0
4	0.905933	0.5	0.790615	0.977495	0
5	0.904133	0.0	0.988573	0.674756	1

- Scaling the scores

```
[ ]  from sklearn.preprocessing import MinMaxScaler
```

```
[ ]
     X = sentiment_df.iloc[:,:-1]
     Y = sentiment_df.iloc[:,-1]
     features = X.columns.values

     features
```

```
array(['azure_api_score', 'google_sentiment_socre', 'ibm_score',
       'amazon_sentiment_score'], dtype=object)
```

```
[ ]  #Scaling all variables to a range of 0 to 1
     sc = MinMaxScaler(feature_range= (0,1))
     X = pd.DataFrame(sc.fit_transform(X), columns= features)
     #features = X.columns
```

```
[ ]  features
```

```
array(['azure_api_score', 'google_sentiment_socre', 'ibm_score',
       'amazon_sentiment_score'], dtype=object)
```

- After normalization

```
X['Avg_Norm_Sentiment_Score'] = X[["azure_api_score","google_sentiment_socre","ibm_score", "amazon_sentiment_score"]].mean(axis=1)
X
```

	azure_api_score	google_sentiment_socre	ibm_score	amazon_sentiment_score	Avg_Norm_Sentiment_Score
0	0.980185	0.5625	0.903736	0.646323	0.773186
1	0.468419	0.5625	0.767878	0.831746	0.657636
2	0.468419	0.2500	0.157679	0.935622	0.452930
3	0.902730	0.7500	0.890277	0.970015	0.878256
4	0.900803	0.4375	0.994673	0.538159	0.717784
5	0.823834	0.5000	0.776448	0.108119	0.552100

- Labelling w.r.t Average sentiment score

```
[ ]  X.loc[ X['Avg_Norm_Sentiment_Score'] >= 0.5, 'API_Predicted_Sentiment'] = 1
     X.loc[ X['Avg_Norm_Sentiment_Score'] < 0.5, 'API_Predicted_Sentiment'] = 0
     X
```

	azure_api_score	google_sentiment_socre	ibm_score	amazon_sentiment_score	Avg_Norm_Sentiment_Score	API_Predicted_Sentiment
0	0.980185	0.5625	0.903736	0.646323	0.773186	1.0
1	0.468419	0.5625	0.767878	0.831746	0.657636	1.0
2	0.468419	0.2500	0.157679	0.935622	0.452930	0.0
3	0.902730	0.7500	0.890277	0.970015	0.878256	1.0
4	0.900803	0.4375	0.994673	0.538159	0.717784	1.0

-

Step 4:

Getting Metrics

```
[ ]  X.API_Predicted_Sentiment.value_counts()
```

```
1.0    710
0.0    101
Name: API_Predicted_Sentiment, dtype: int64
```

```
[ ]  Y.value_counts()
```

```
1    654
0    157
Name: sentiment, dtype: int64
```

```
[ ] from sklearn.metrics import confusion_matrix

    confusion_matrix(Y, X.API_Predicted_Sentiment)
```

```
array([[ 41, 116],
       [ 60, 594]], dtype=int64)
```

```
[ ] from sklearn.metrics import accuracy_score

    print(accuracy_score(Y, X.API_Predicted_Sentiment))
```

```
0.782983970406905
```

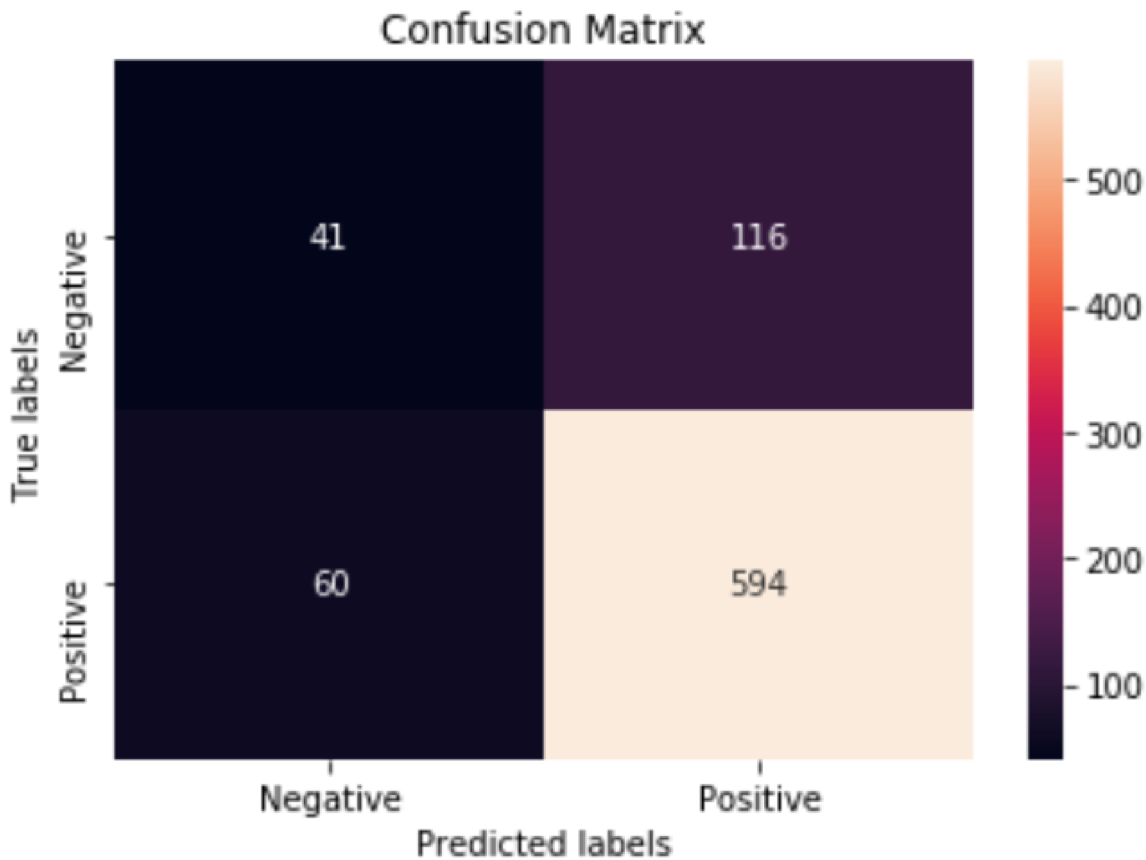
```
[ ] from sklearn.metrics import classification_report

    print(classification_report(Y, X.API_Predicted_Sentiment))
```

```
             precision    recall  f1-score   support

          0       0.41      0.26      0.32       157
          1       0.84      0.91      0.87       654

avg / total       0.75      0.78      0.76       811
```

PLotting the Confusion matrix in Heatmap

```
[ ] import matplotlib.pyplot as plt
    import seaborn as sns

    ax= plt.subplot()
    #cm = classification_report(y_test, y_pred_lstm)
    sns.heatmap(confusion_matrix(Y, X.API_Predicted_Sentiment), annot=True, ax = ax, fmt='g'); #annot=True to annotate cells

    # labels, title and ticks
    ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
    ax.set_title('Confusion Matrix');
    ax.xaxis.set_ticklabels(['Negative', 'Positive'])
    ax.yaxis.set_ticklabels(['Positive', 'Negative'])
```

```
[Text(0,0.5,'Negative'), Text(0,1.5,'Positive')]
```



Experiment 4: Ensemble learning using AutoML

In order to map the sentiment scores from all 4 API's to the sentiment's we labelled manually, we created a FC Neural network

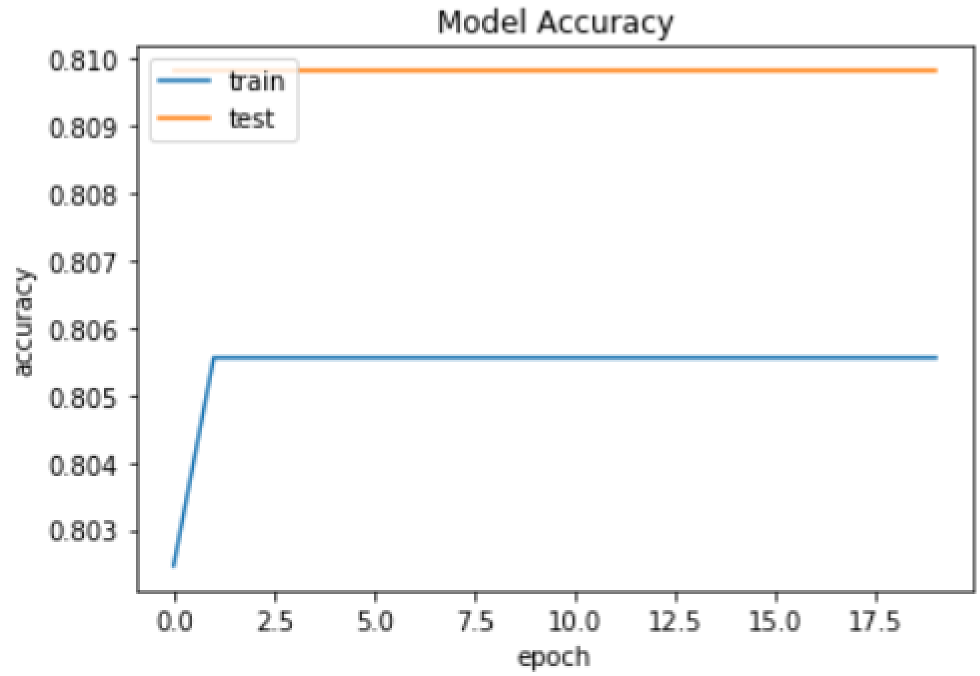
```
[ ] from keras.layers import Dense, Dropout
    from keras.models import Sequential

    model = Sequential()
```

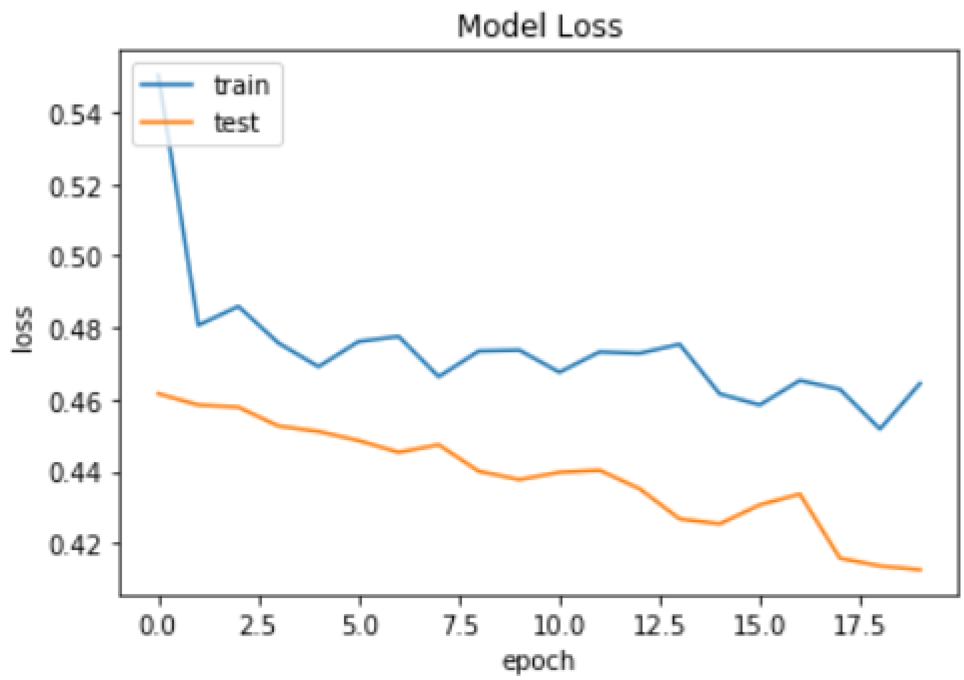
```
[ ] model.add(Dense(32, activation='relu', kernel_initializer='uniform', input_shape=(4,)))
    model.add(Dropout(0.15))
    model.add(Dense(16, activation='relu',kernel_initializer='uniform'))
    model.add(Dropout(0.2))
    model.add(Dense(1, activation='sigmoid', kernel_initializer='uniform'))

    model.summary()

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

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_25 (Dense)             (None, 32)                160
_________________________________________________________________
dropout_17 (Dropout)         (None, 32)                0
_________________________________________________________________
dense_26 (Dense)             (None, 16)                528
_________________________________________________________________
dropout_18 (Dropout)         (None, 16)                0
_________________________________________________________________
dense_27 (Dense)             (None, 1)                 17
=================================================================
Total params: 705
Trainable params: 705
Non-trainable params: 0
_________________________________________________________________
```

Followed by this we tried TPOT, AutoSKlearn , H20.ai in order to utilize autoML for mapping the same

```
Test Loss:  0.41255556056104553
Test Accuracy 0.8098159509202454
dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
```

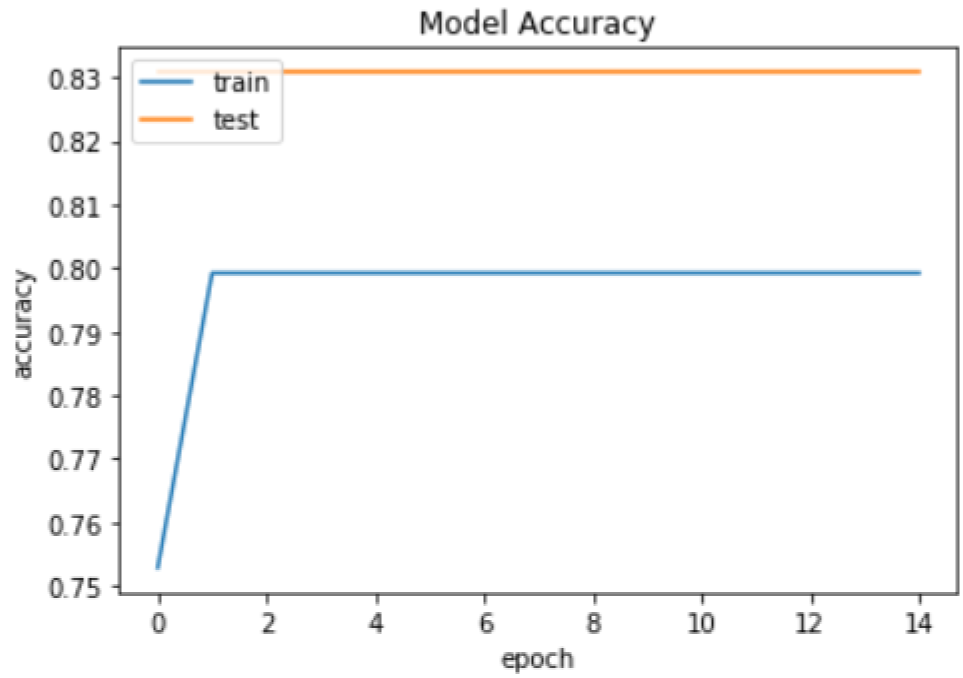


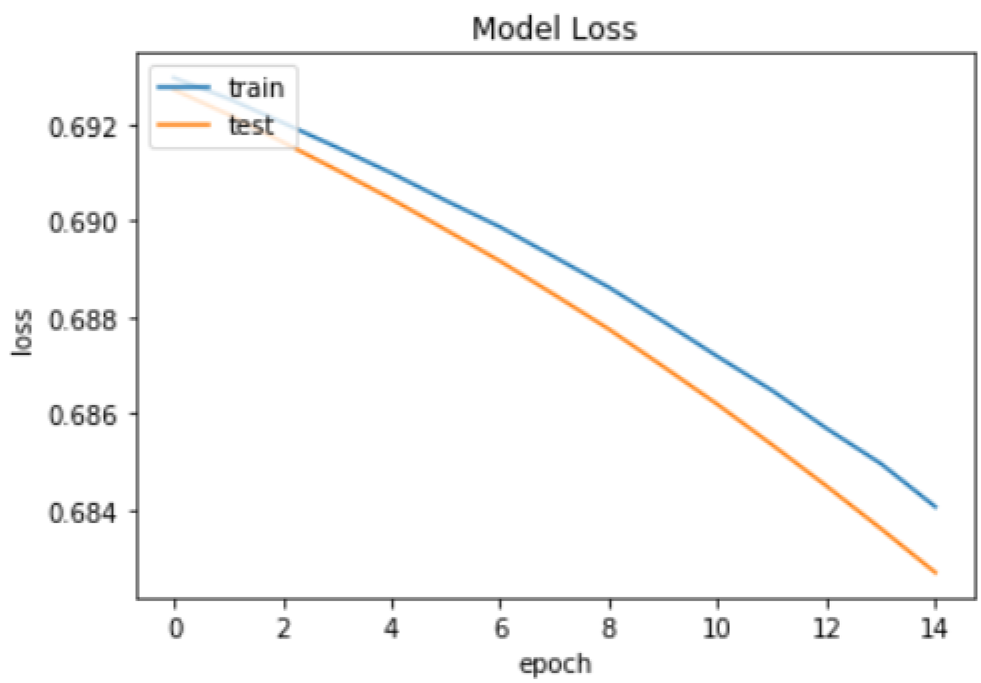


We did hyperparameter tuning for the model so that the loss converges better

```
_________________________________________________________________
dense_28 (Dense)             (None, 32)                160
_________________________________________________________________
dropout_19 (Dropout)         (None, 32)                0
_________________________________________________________________
dense_29 (Dense)             (None, 32)                1056
_________________________________________________________________
dropout_20 (Dropout)         (None, 32)                0
_________________________________________________________________
dense_30 (Dense)             (None, 1)                 33
_________________________________________________________________
dense_31 (Dense)             (None, 32)                64
_________________________________________________________________
dropout_21 (Dropout)         (None, 32)                0
_________________________________________________________________
dense_32 (Dense)             (None, 32)                1056
_________________________________________________________________
dropout_22 (Dropout)         (None, 32)                0
_________________________________________________________________
dense_33 (Dense)             (None, 1)                 33
_________________________________________________________________
dense_34 (Dense)             (None, 32)                64
_________________________________________________________________
dropout_23 (Dropout)         (None, 32)                0
_________________________________________________________________
dense_35 (Dense)             (None, 32)                1056
_________________________________________________________________
dropout_24 (Dropout)         (None, 32)                0
_________________________________________________________________
dense_36 (Dense)             (None, 1)                 33
=================================================================
C:\Users\nikhi\Anaconda3\lib\site-packages\keras\engine\training.py
  'Discrepancy between trainable weights and collected trainable'
Total params: 2,402
Trainable params: 2,402
Non-trainable params: 0
```

```
Test Loss:  0.6833624390005334
Test Accuracy 0.8098159509202454
dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
```





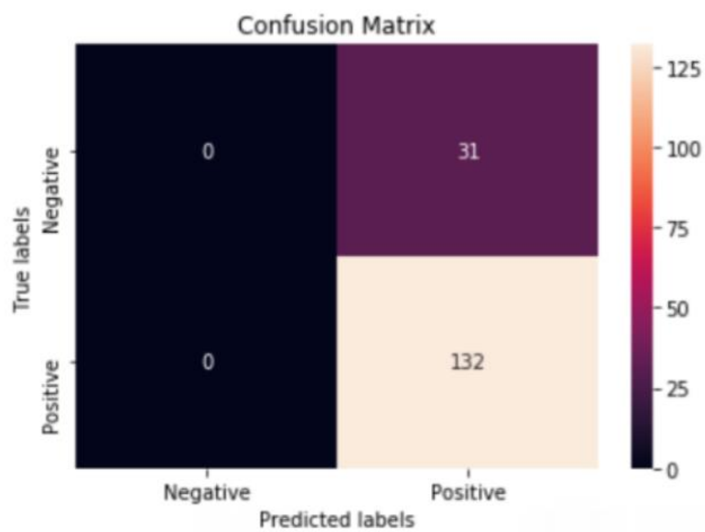
Since the model is trained on very few negative data, the model doesn't predict the negative sentiments at all and we find 0 True Negatives as a result

```
from sklearn.metrics import confusion_matrix, classification_report

confusion_matrix(Y_test, y_pred_class)
```

```
array([[  0,  31],
       [  0, 132]], dtype=int64)
```

```
ax= plt.subplot()
#cm = classification_report(y_test, y_pred_lstm)
sns.heatmap(confusion_matrix(Y_test, y_pred_class), annot=True, ax = ax, fmt='g');

# labels, title and ticks
ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
ax.xaxis.set_ticklabels(['Negative', 'Positive'])
ax.yaxis.set_ticklabels(['Negative', 'Positive'])
```

```
[Text(0,0.5,'Negative'), Text(0,1.5,'Positive')]
```



AUTO ML

Implementing TPOT to get a model with tuned hyperparameters

```
[ ] from tpot import TPOTClassifier
    #from tpot import TPOTRegressor

    tpot = TPOTClassifier(generations=5,verbosity=2)
```

```
[ ] tpot.fit(X_train,Y_train)
```

```
HBox(children=(IntProgress(value=0, description='Optimization Progress', max=600), HTML(value='')))
Generation 1 - Current best internal CV score: 0.8009970667380429
Generation 2 - Current best internal CV score: 0.8009970667380429
Generation 3 - Current best internal CV score: 0.8009970667380429
Generation 4 - Current best internal CV score: 0.8009970667380429
Generation 5 - Current best internal CV score: 0.8009970667380429

Best pipeline: ExtraTreesClassifier(input_matrix, bootstrap=True, criterion=entropy, max_features=0.8500000000000001,
TPOTClassifier(config_dict=None, crossover_rate=0.1, cv=5,
        disable_update_check=False, early_stop=None, generations=5,
        max_eval_time_mins=5, max_time_mins=None, memory=None,
        mutation_rate=0.9, n_jobs=1, offspring_size=None,
        periodic_checkpoint_folder=None, population_size=100,
        random_state=None, scoring=None, subsample=1.0, use_dask=False,
        verbosity=2, warm_start=False)
```

Output of TPOT:

```
[ ] tpot.fitted_pipeline_
```

```
Pipeline(memory=None,
     steps=[('extratreesclassifier', ExtraTreesClassifier(bootstrap=True, class_weight=None, criterion='entropy',
           max_depth=None, max_features=0.8500000000000001,
           max_leaf_nodes=None, min_impurity_decrease=0.0,
           min_impurity_split=None, min_samples_leaf=19,
           min_samples_split=11, min_weight_fraction_leaf=0.0,
           n_estimators=100, n_jobs=1, oob_score=False, random_state=None,
           verbose=0, warm_start=False))])
```

So the best pipeline as per TPOT is the below -

Best pipeline: KNeighborsClassifier(input_matrix, n_neighbors=90, p=1, weights=distance)

```
[ ] # !conda install -c anaconda py-xgboost
```

```
[ ] tpot.score(X_test,
               Y_test)
```

```
0.8226600985221675
```

Using H2O for creating a complete pipeline

```
[ ] import h2o
    from h2o.automl import H2OAutoML
    h2o.init()
```

```
Checking whether there is an H2O instance running at http://localhost:54321..... not found.
Attempting to start a local H2O server...
; Java HotSpot(TM) 64-Bit Server VM (build 25.191-b12, mixed mode)
  Starting server from C:\Users\nikhi\Anaconda3\lib\site-packages\h2o\backend\bin\h2o.jar
  Ice root: C:\Users\nikhi\AppData\Local\Temp\tmp3fvl92bt
  JVM stdout: C:\Users\nikhi\AppData\Local\Temp\tmp3fvl92bt\h2o_nikhi_started_from_python.out
  JVM stderr: C:\Users\nikhi\AppData\Local\Temp\tmp3fvl92bt\h2o_nikhi_started_from_python.err
  Server is running at http://127.0.0.1:54321
Connecting to H2O server at http://127.0.0.1:54321... successful.
```

H2O cluster uptime:	01 secs
H2O cluster timezone:	America/New_York
H2O data parsing timezone:	UTC
H2O cluster version:	3.22.1.2
H2O cluster version age:	2 months and 1 day
H2O cluster name:	H2O_from_python_nikhi_dvh1lf
H2O cluster total nodes:	1
H2O cluster free memory:	3.523 Gb
H2O cluster total cores:	8
H2O cluster allowed cores:	8
H2O cluster status:	accepting new members, healthy
H2O connection url:	http://127.0.0.1:54321
H2O connection proxy:	None
H2O internal security:	False
H2O API Extensions:	Algos, AutoML, Core V3, Core V4
Python version:	3.6.5 final

Description about data

```
[ ] df.describe()
```

```
Rows:811
Cols:6
```

	C1	azure_api_score	google_sentiment_socre	ibm_score	amazon_sentiment_score	sentiment
type	int	real	real	real	real	int
mins	0.0	0.0621877611	-0.6999999881	-0.897554	0.2974953949	0.0
mean	828.5326757090012	0.6427028278373615	0.21418002826979035	0.5957383181257702	0.7602460733112212	0.8064118372379778
maxs	1642.0	0.9968479276	0.8999999762	0.998674	0.9985154271	1.0
sigma	455.07252153575007	0.22324971364639776	0.24069582961015862	0.4168205697793954	0.17980453185630274	0.39535366015815204
zeros	1	0	187	18	0	157
missing	0	0	0	0	0	0
0	0.0	0.9783278108	0.200000003	0.816136	0.7505810857	1.0
1	1.0	0.5	0.200000003	0.558518	0.8805660009	1.0
2	3.0	0.5	-0.3000000119	-0.598559	0.9533853531	0.0
3	4.0	0.9059334397	0.5	0.790615	0.9774951339	0.0
4	5.0	0.9041327238	0.0	0.988573	0.6747556925	1.0
5	6.0	0.8321921229	0.1000000015	0.574769	0.3732891977	0.0
6	8.0	0.5	0.400000006	0.884016	0.7173274755	1.0
7	9.0	0.7486802936	0.6000000238	0.59894	0.6371069551	1.0
8	10.0	0.9522012472	0.400000006	0.875206	0.9451110363	1.0
9	12.0	0.5	0.200000003	0.569009	0.6385424137	0.0

Training 10 models and getting there metrics score as follows:

```
[ ] aml = H2OAutoML(max_models = 10, max_runtime_secs=120, seed = 1)
    aml.train(x = x, y = y, training_frame= df)
```

AutoML progress: |██| 100%

```
[ ] lb = aml.leaderboard
    lb.head()
    #lb.head(rows=lb.nrows)
```

model_id	auc	logloss	mean_per_class_error	rmse	mse
GLM_grid_1_AutoML_20190320_200417_model_1	0.742136	0.438973	0.4679	0.373468	0.139478
GBM_5_AutoML_20190320_200417	0.740748	0.427411	0.5	0.367227	0.134855
StackedEnsemble_BestOfFamily_AutoML_20190320_200417	0.737724	0.430725	0.5	0.36908	0.13622
StackedEnsemble_AllModels_AutoML_20190320_200417	0.736516	0.43116	0.5	0.369314	0.136393
GBM_2_AutoML_20190320_200417	0.723509	0.453431	0.5	0.380405	0.144708
GBM_grid_1_AutoML_20190320_200417_model_1	0.723383	0.488385	0.496815	0.393917	0.155171
GBM_3_AutoML_20190320_200417	0.72145	0.455546	0.487135	0.379813	0.144258
GBM_4_AutoML_20190320_200417	0.716015	0.462293	0.5	0.38265	0.146421
DeepLearning_1_AutoML_20190320_200417	0.714544	0.464378	0.496815	0.387359	0.150047
GBM_1_AutoML_20190320_200417	0.708935	0.47874	0.5	0.389983	0.152087

The best auc score we got is : 0.742

Getting predictions for Text, p0 is Probability to be negative and p1 is Probability to be Positive

```
[ ] df1
```

azure_api_score	google_sentiment_socre	ibm_score	amazon_sentiment_score
0.468419	0.6875	0.898289	0.611774
0.468419	0.625	0.876083	0.443459
0.705929	0.5625	0.751259	0.0525138
0.468419	0.4375	0.797158	0.690586
0.468419	0.5625	0.905007	0.828573
0.468419	0.5625	0.888162	0.944919
0.468419	0.625	0.937871	0.461272
0.468419	0.6875	0.789764	0.771942
0.947612	0.9375	0.950588	0.963662
0.747647	0.4375	0.809732	0.525685

```
[ ] y_pred_h2o = aml.predict(df1)
    y_pred_h2o
```

glm prediction progress: |██| 100%

predict	p0	p1
1	0.0591668	0.940833
1	0.0715695	0.928431
1	0.102093	0.897907
1	0.0987179	0.901282
1	0.0680783	0.931922
1	0.0665929	0.933407
1	0.0667654	0.933235
1	0.0626903	0.93731
1	0.0310117	0.968988
1	0.101915	0.898085

AutoSKlearn

Facing issue with autosklearn package

The issue is still open and the link to it is down below

```
[ ]  import autosklearn.classification
     import sklearn.model_selection
     import sklearn.datasets
     import sklearn.metrics
     from sklearn.model_selection import cross_val_score, train_test_split
```

```
[ ]  X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=42)
     automl = autosklearn.classification.AutoSklearnClassifier()
     automl.fit(X_train, Y_train)
     y_hat = automl.predict(X_test)
     print("Accuracy score", sklearn.metrics.accuracy_score(Y_test, y_hat))
```

```
ome/jai/miniconda3/lib/python3.7/site-packages/autosklearn/evaluation/train_evaluator.py:197: RuntimeWarning: Mean of empty slice
/_train_pred = np.nanmean(Y_train_pred_full, axis=0)
ARNING] [2019-03-21 08:38:02,092:EnsembleBuilder(1):9af0d6fbec1b15d1b06c1bc99a3f82d0] No models better than random - using Dummy Score!
ARNING] [2019-03-21 08:38:02,100:EnsembleBuilder(1):9af0d6fbec1b15d1b06c1bc99a3f82d0] No models better than random - using Dummy Score!
ome/jai/miniconda3/lib/python3.7/site-packages/autosklearn/evaluation/train_evaluator.py:197: RuntimeWarning: Mean of empty slice
/_train_pred = np.nanmean(Y_train_pred_full, axis=0)
ome/jai/miniconda3/lib/python3.7/site-packages/autosklearn/evaluation/train_evaluator.py:197: RuntimeWarning: Mean of empty slice
/_train_pred = np.nanmean(Y_train_pred_full, axis=0)
```

https://github.com/automl/auto-sklearn/issues/520