CSYE7374 Cognitive Computing Assignment 3 - NLP

Professor: Sri Krishnamurthy

Team Members: Shiqi Dai(Sean), Gaurang Davda

Case Instruction

> “
>
> *Sentiment analysis is becoming more and more popular with availability of alternative datasets such as Twitter etc. Companies like Amazon, Yelp, Uber etc. routinely analyze user reviews to understand user sentiments. With chatbots and conversational agents becoming more and more popular, sentiment analysis is gaining mainstream acceptance. Building a sentiment analysis engine from scratch is a challenge. Traditionally, bow models, word embeddings using glove/word2vec have been popular and the Movie review database (http://ai.stanford.edu/~amaas/data/sentiment/) is used to illustrate these concepts. In the last few years, Deep Neural networks and RNNs have been tried for sentiment analysis. Many cloud vendors are offering sentiment analysis APIs[2,3,4,5]. Sentiment analysis is particularly challenging in domains such as finance. Financial Sentiment analysis is a challenging problem considering it is domain specific and there is lack of good labeled data to train. Traditionally, analysts conduct in-depth fundamental analysis and issue guidance and reports which are used by financial professionals for investment analysis. In the last few years, with the growth of social networks and alternative means of getting information about market events, using Alternative data to evaluate sentiments is becoming popular. Twitter, EDGAR and news articles are typical data sources for information and for evaluating sentiments.*
>
> *In this case, we will review some methods to evaluate sentiment analysis techniques.The goal of this project is to build a sentiment analysis engine that can be used to understand the sentiments of earnings call transcripts.*
>
> ”

Data

Data Source: Facebook Q3 2018 earnings call transcripts

Web scraping from SeekingAlpha:
https://seekingalpha.com/article/4216207-facebook-fb-q3-2018-results-earnings-call-transcript

Manual Labeling data

How we define positive/negative/neutral

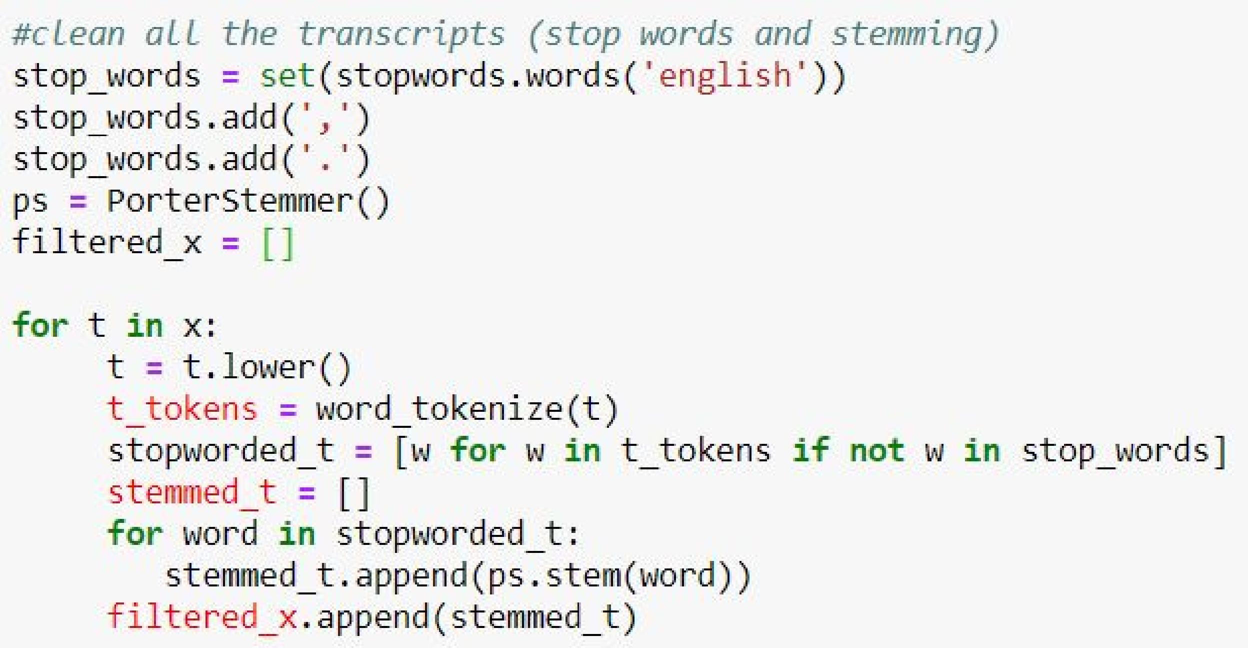
- **Positive**: if one paragraph talks something good to/about the company
- **Negative**: if one paragraph talks something bad to/about the company
- **Neutral**: neither any of above.

Experiments

EX1 - BOW

Preprocessing

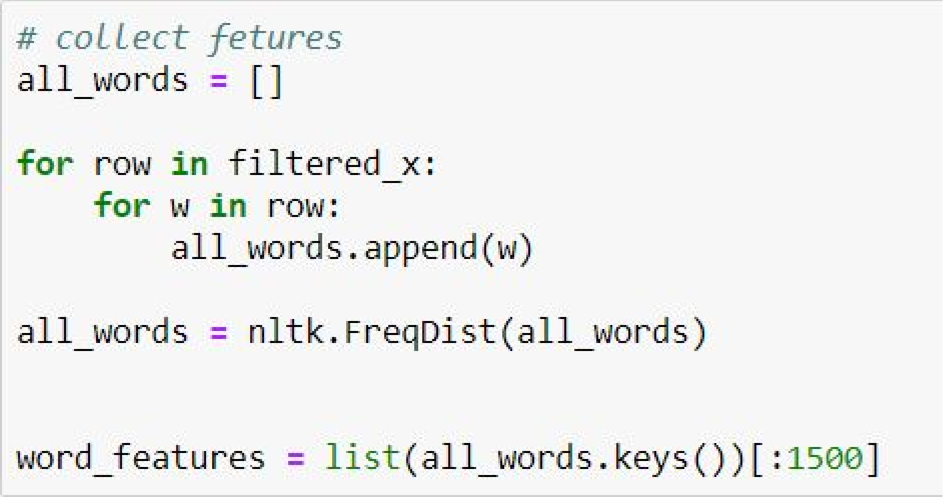
In this part, we used NLTK package, and mainly applied English stop words, Potter Stammer, and a tokenizer.



```
#clean all the transcripts (stop words and stemming)
stop_words = set(stopwords.words('english'))
stop_words.add(',')
stop_words.add('.')
ps = PorterStemmer()
filtered_x = []

for t in x:
     t = t.lower()
     t_tokens = word_tokenize(t)
     stopworded_t = [w for w in t_tokens if not w in stop_words]
     stemmed_t = []
     for word in stopworded_t:
        stemmed_t.append(ps.stem(word))
     filtered_x.append(stemmed_t)
```

Feature Collection

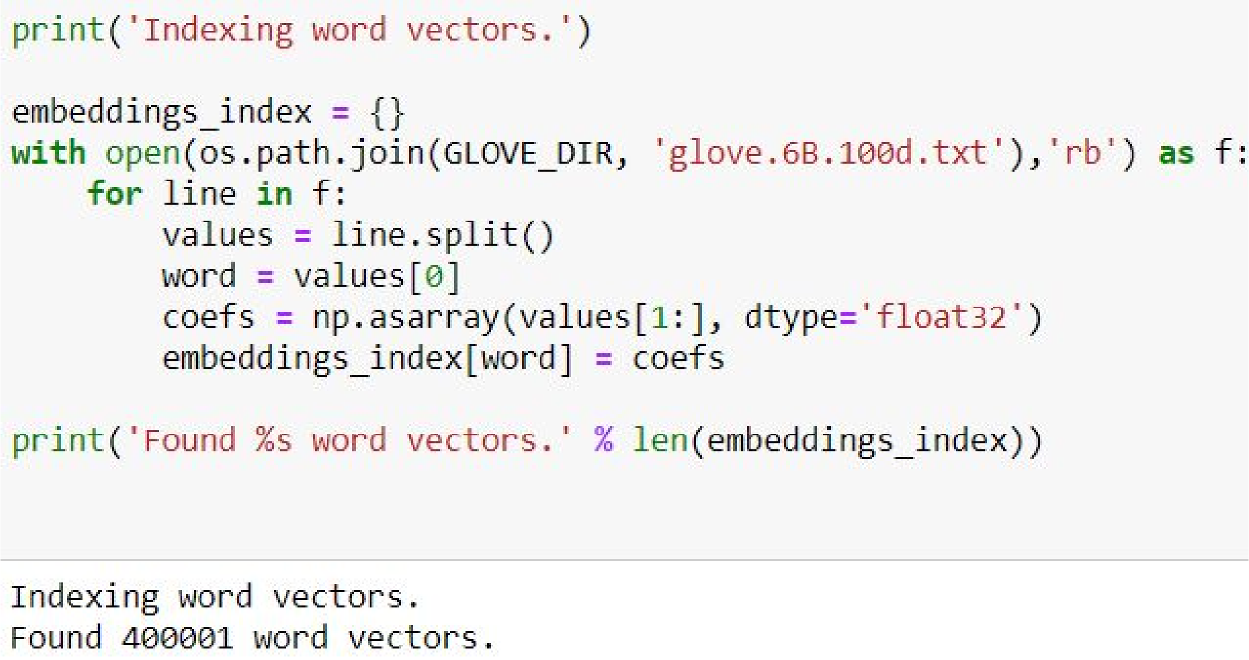
We did not use all the words to build a word bag. Instead, we decided to collect word features from our data set. In practice, we first collected all the words in a list, then build a dictionary which contains the vocabulary(getting rid of those repeated words). By using nltk.FreqDist API, we can easily obtain such a dictionary ordered by word frequency. Last, extract a certain amount of words from the very top(high frequency).



```
# collect fetures
all_words = []

for row in filtered_x:
    for w in row:
        all_words.append(w)

all_words = nltk.FreqDist(all_words)


word_features = list(all_words.keys())[:1500]
```

Word Embedding

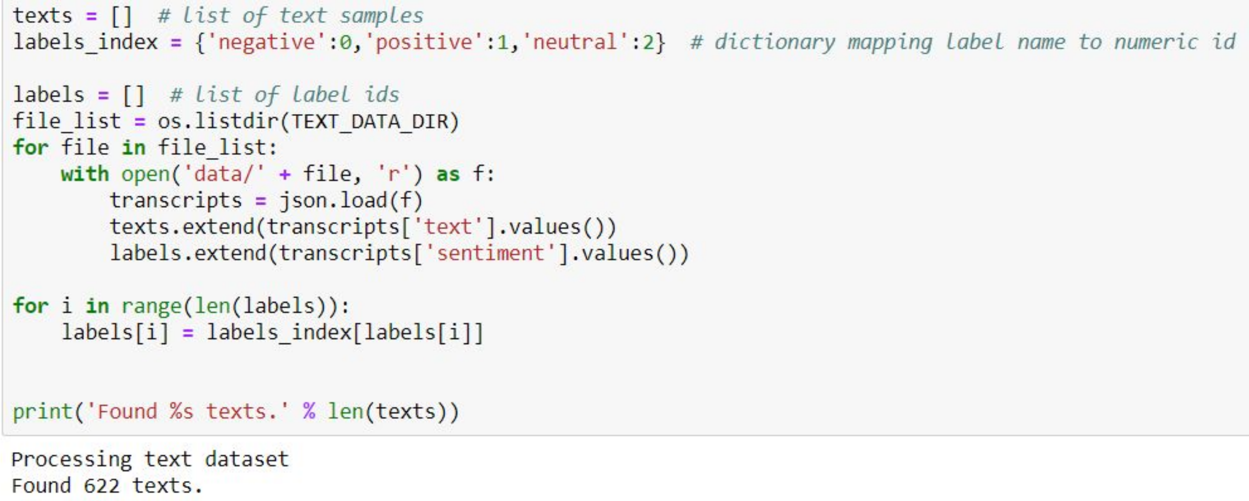
First, load the downloaded glove word embeddings and build index mapping words in the embeddings set to their embedding vector.



```
print('Indexing word vectors.')

embeddings_index = {}
with open(os.path.join(GLOVE_DIR, 'glove.6B.100d.txt'),'rb') as f:
    for line in f:
        values = line.split()
        word = values[0]
        coefs = np.asarray(values[1:], dtype='float32')
        embeddings_index[word] = coefs

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

```
Indexing word vectors.
Found 400001 word vectors.
```

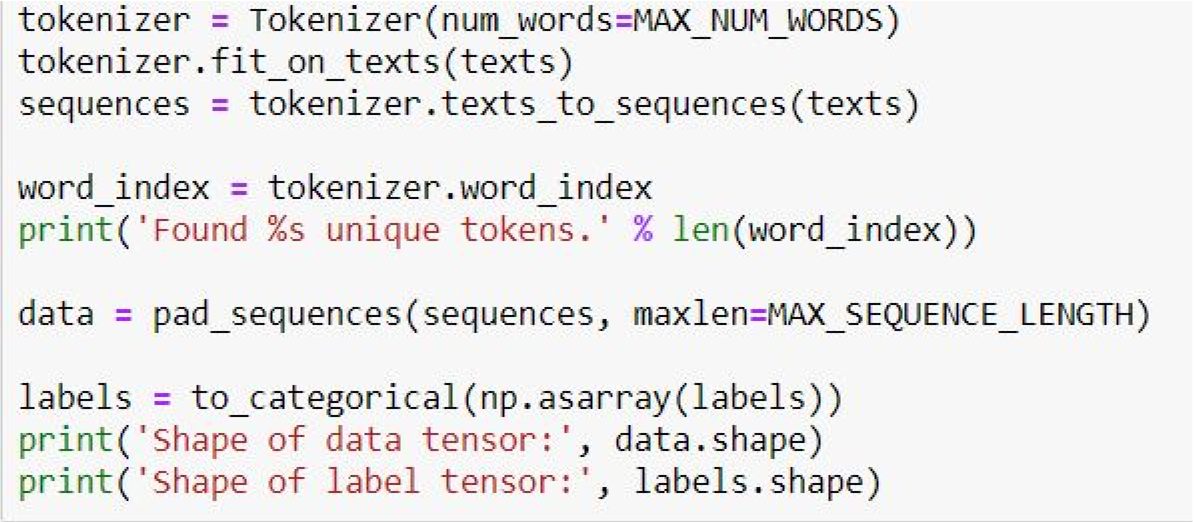
Second, prepare text samples and their labels. Will read all data files(json) into the program and store them in ‘text’ and ‘sentiment’ lists.



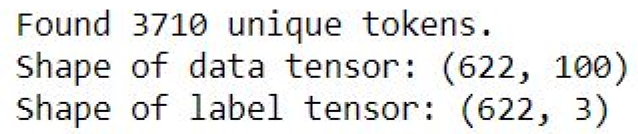
```
texts = []  # list of text samples
labels_index = {'negative':0,'positive':1,'neutral':2}  # dictionary mapping label name to numeric id

labels = []  # list of label ids
file_list = os.listdir(TEXT_DATA_DIR)
for file in file_list:
    with open('data/' + file, 'r') as f:
        transcripts = json.load(f)
        texts.extend(transcripts['text'].values())
        labels.extend(transcripts['sentiment'].values())

for i in range(len(labels)):
    labels[i] = labels_index[labels[i]]


print('Found %s texts.' % len(texts))
```

```
Processing text dataset
Found 622 texts.
```

Finally, vectorize the text samples into a 2D integer tensor

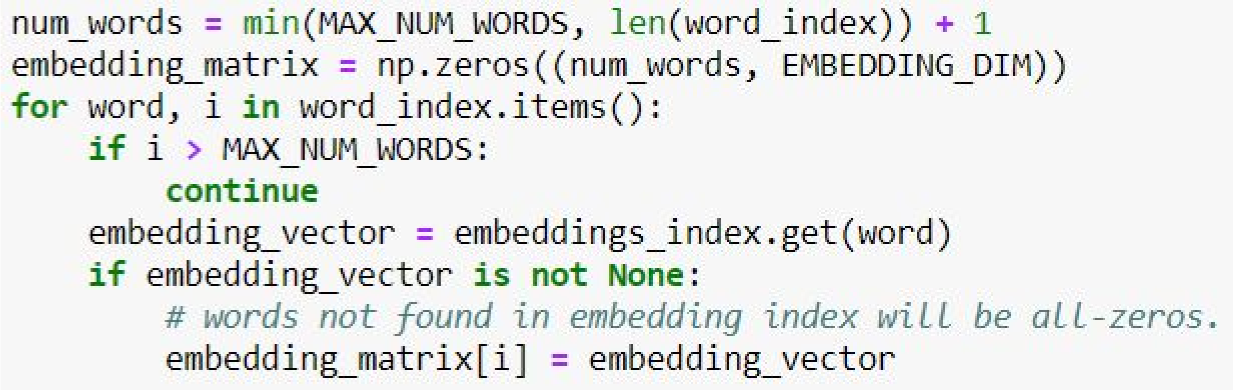


```
tokenizer = Tokenizer(num_words=MAX_NUM_WORDS)
tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)

word_index = tokenizer.word_index
print('Found %s unique tokens.' % len(word_index))

data = pad_sequences(sequences, maxlen=MAX_SEQUENCE_LENGTH)

labels = to_categorical(np.asarray(labels))
print('Shape of data tensor:', data.shape)
print('Shape of label tensor:', labels.shape)
```



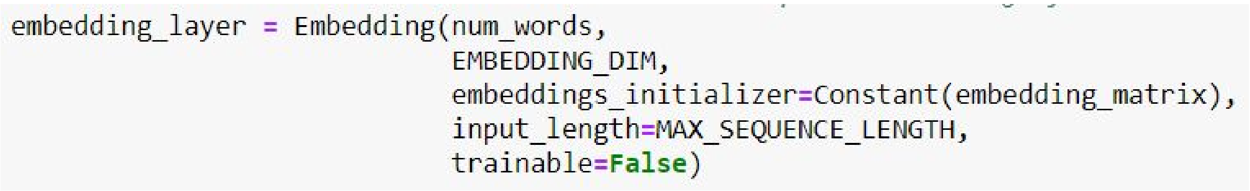
```
Found 3710 unique tokens.
Shape of data tensor: (622, 100)
Shape of label tensor: (622, 3)
```

Use the embedding index to create an embedding matrix for later creating the embedding layer.



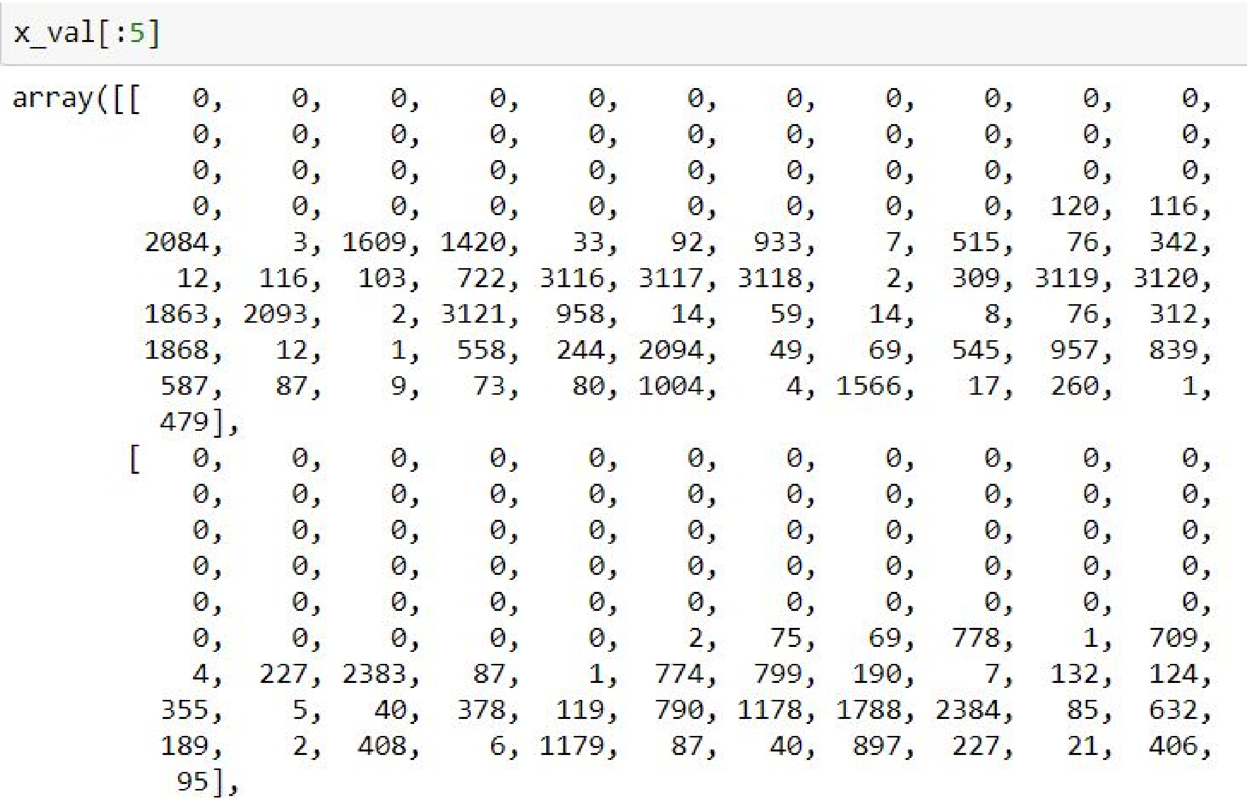
```
num_words = min(MAX_NUM_WORDS, len(word_index)) + 1
embedding_matrix = np.zeros((num_words, EMBEDDING_DIM))
for word, i in word_index.items():
    if i > MAX_NUM_WORDS:
        continue
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        # words not found in embedding index will be all-zeros.
        embedding_matrix[i] = embedding_vector
```

Create a keras embedding layer. Note that we set trainable = False so as to keep the embeddings fixed



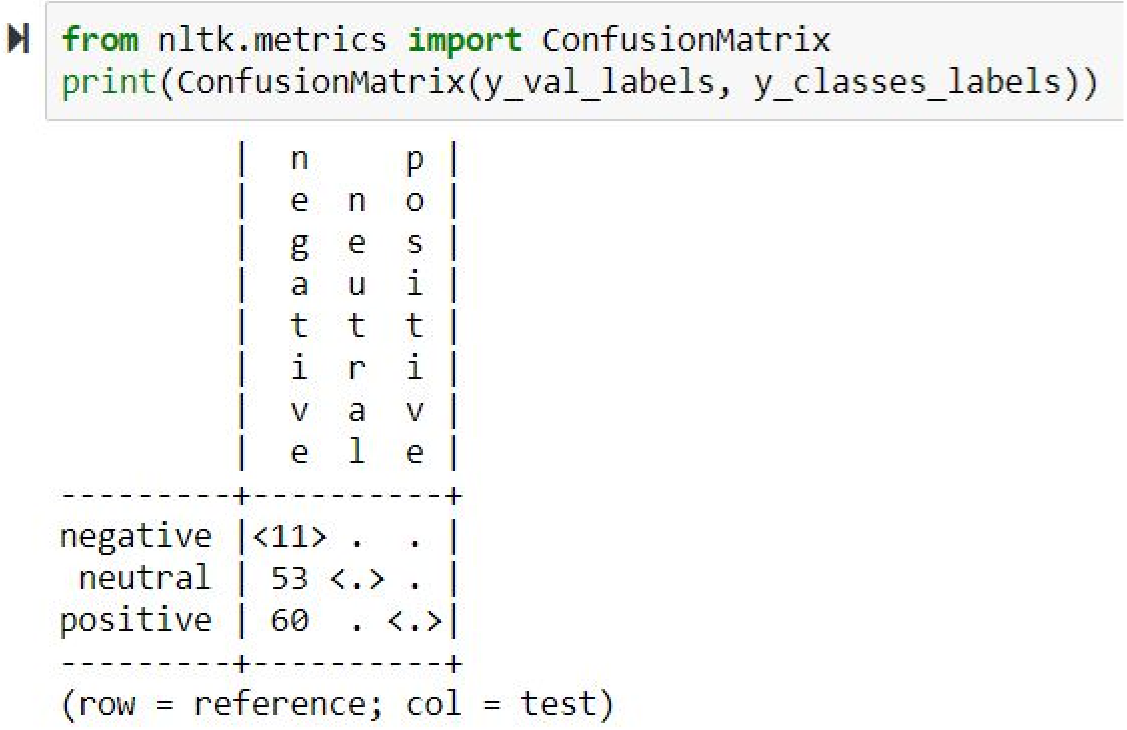
```
embedding_layer = Embedding(num_words,
                            EMBEDDING_DIM,
                            embeddings_initializer=Constant(embedding_matrix),
                            input_length=MAX_SEQUENCE_LENGTH,
                            trainable=False)
```

Check if the validation set is okay.



```
x_val[:5]
```

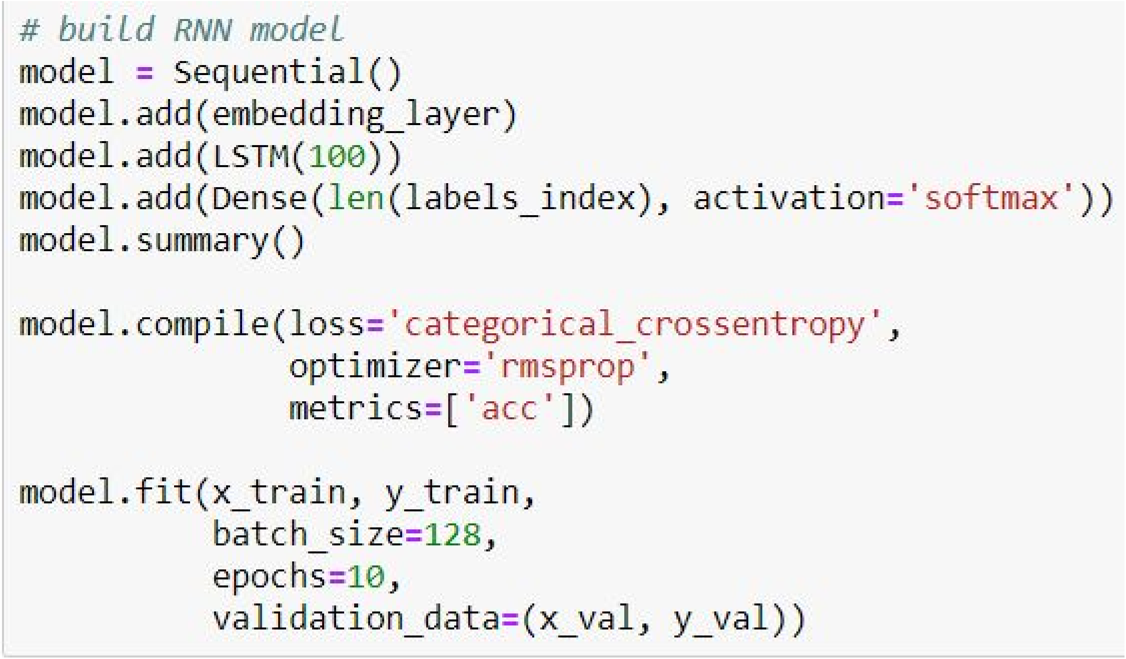
```
array([[   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,  120,  116,
        2084,    3, 1609, 1420,   33,   92,  933,    7,  515,   76,  342,
          12,  116,  103,  722, 3116, 3117, 3118,    2,  309, 3119, 3120,
        1863, 2093,    2, 3121,  958,   14,   59,   14,    8,   76,  312,
        1868,   12,    1,  558,  244, 2094,   49,   69,  545,  957,  839,
         587,   87,    9,   73,   80, 1004,    4, 1566,   17,  260,    1,
         479],
       [   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,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    2,   75,   69,  778,    1,  709,
           4,  227, 2383,   87,    1,  774,  799,  190,    7,  132,  124,
         355,    5,   40,  378,  119,  790, 1178, 1788, 2384,   85,  632,
         189,    2,  408,    6, 1179,   87,   40,  897,  227,   21,  406,
          95],
```



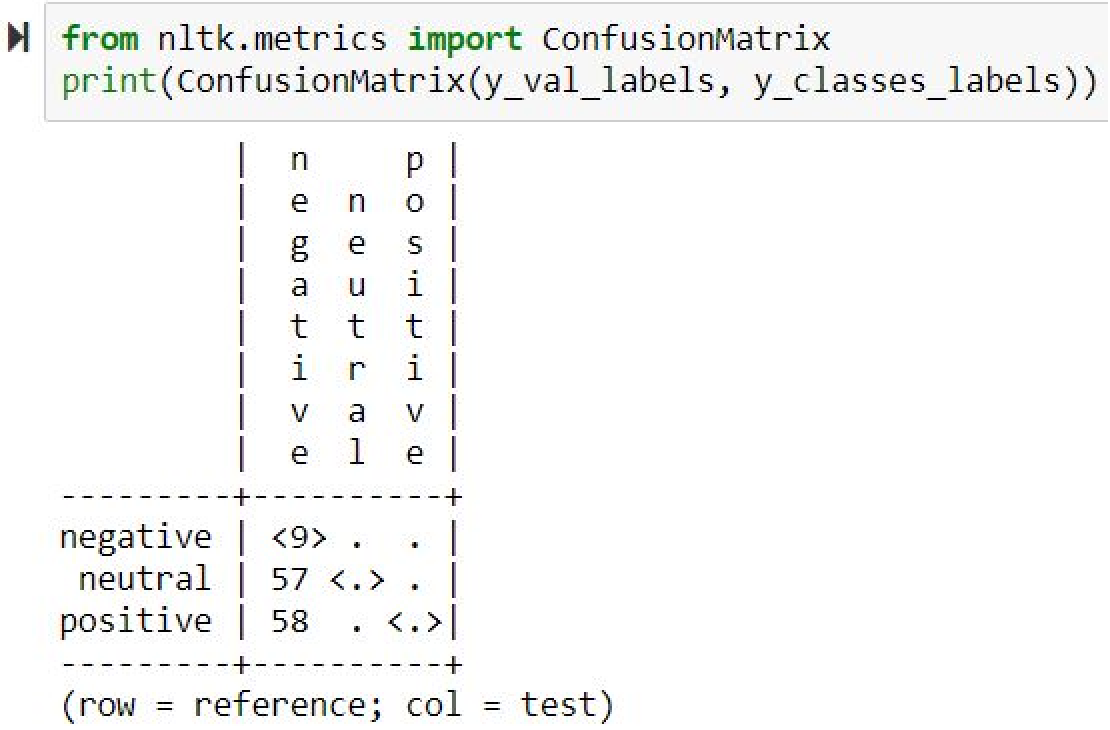
```
from nltk.metrics import ConfusionMatrix
print(ConfusionMatrix(y_val_labels, y_classes_labels))
```

```
         |   n       p |
         |   e   n   o |
         |   g   e   s |
         |   a   u   i |
         |   t   t   t |
         |   i   r   i |
         |   v   a   v |
         |   e   l   e |
---------+-------------+
negative |<11>   .   . |
 neutral |  53  <.>  . |
positive |  60   .  <.>|
---------+-------------+
(row = reference; col = test)
```

RNN



```
# build RNN model
model = Sequential()
model.add(embedding_layer)
model.add(LSTM(100))
model.add(Dense(len(labels_index), activation='softmax'))
model.summary()

model.compile(loss='categorical_crossentropy',
              optimizer='rmsprop',
              metrics=['acc'])

model.fit(x_train, y_train,
          batch_size=128,
          epochs=10,
          validation_data=(x_val, y_val))
```

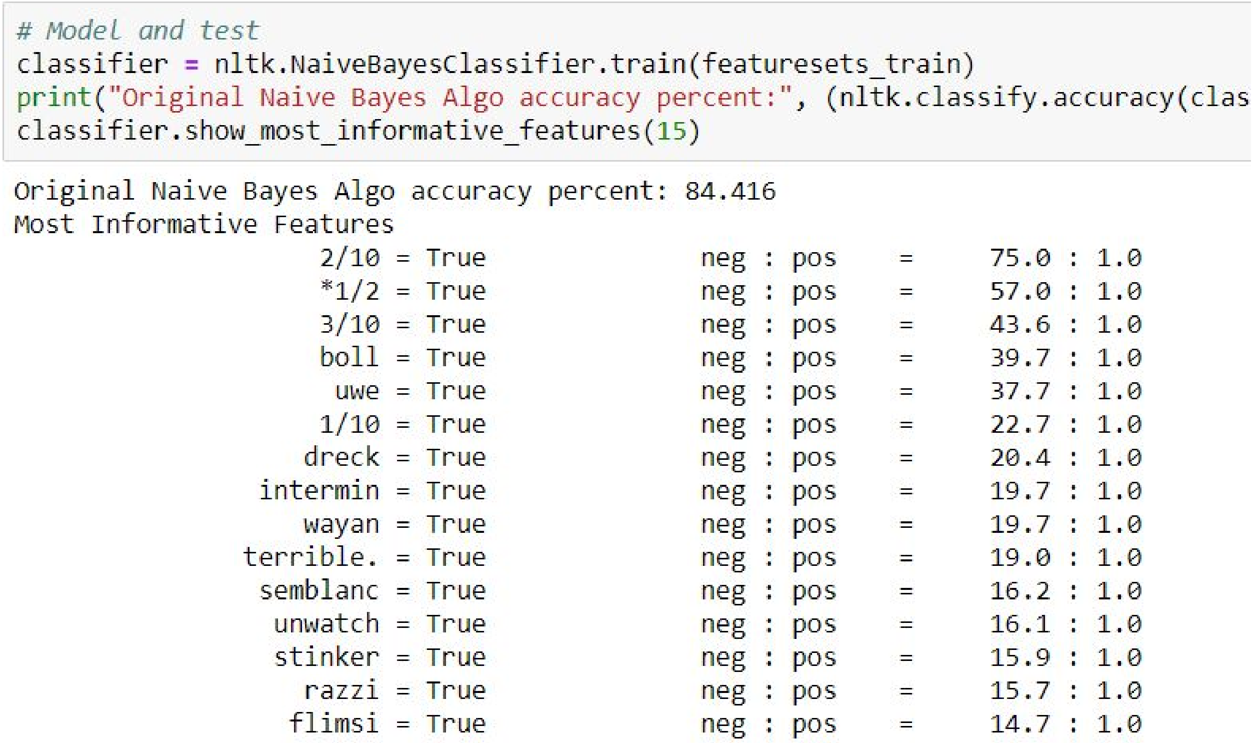


```
from nltk.metrics import ConfusionMatrix
print(ConfusionMatrix(y_val_labels, y_classes_labels))
```

```
         |  n     p |
         |  e  n  o |
         |  g  e  s |
         |  a  u  i |
         |  t  t  t |
         |  i  r  i |
         |  v  a  v |
         |  e  l  e |
---------+----------+
negative | <9> .  . |
 neutral | 57 <.> . |
positive | 58  . <.>|
---------+----------+
(row = reference; col = test)
```

EX2

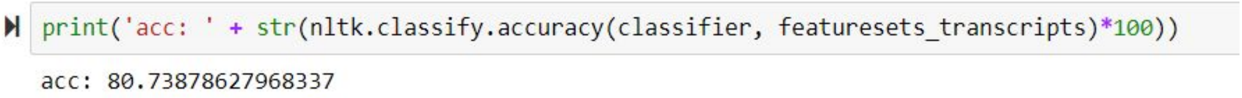
BOW



```
# Model and test
classifier = nltk.NaiveBayesClassifier.train(featuresets_train)
print("Original Naive Bayes Algo accuracy percent:", (nltk.classify.accuracy(clas
classifier.show_most_informative_features(15)
```

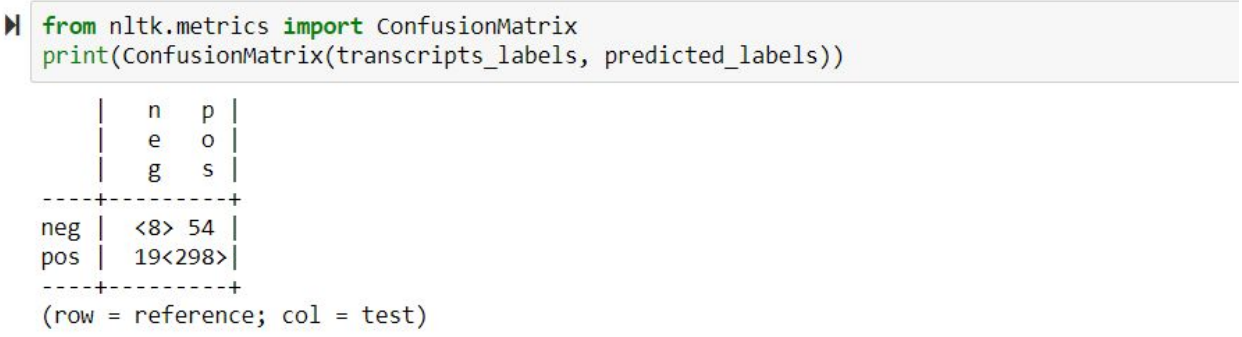
```
Original Naive Bayes Algo accuracy percent: 84.416
Most Informative Features
                    2/10 = True              neg : pos    =     75.0 : 1.0
                   *1/2 = True              neg : pos    =     57.0 : 1.0
                    3/10 = True              neg : pos    =     43.6 : 1.0
                    boll = True              neg : pos    =     39.7 : 1.0
                     uwe = True              neg : pos    =     37.7 : 1.0
                    1/10 = True              neg : pos    =     22.7 : 1.0
                   dreck = True              neg : pos    =     20.4 : 1.0
                intermin = True              neg : pos    =     19.7 : 1.0
                   wayan = True              neg : pos    =     19.7 : 1.0
               terrible. = True              neg : pos    =     19.0 : 1.0
                semblanc = True              neg : pos    =     16.2 : 1.0
                 unwatch = True              neg : pos    =     16.1 : 1.0
                 stinker = True              neg : pos    =     15.9 : 1.0
                   razzi = True              neg : pos    =     15.7 : 1.0
                  flimsi = True              neg : pos    =     14.7 : 1.0
```

This is the naive Bayes model on IMDB dataset. We used BOW to represent the data. The accuracy is around 84%, not bad.



```
print('acc: ' + str(nltk.classify.accuracy(classifier, featuresets_transcripts)*100))
```

```
acc: 80.73878627968337
```



```
from nltk.metrics import ConfusionMatrix
print(ConfusionMatrix(transcripts_labels, predicted_labels))
```

```
    |   n   p |
    |   e   o |
    |   g   s |
----+---------+
neg |  <8> 54 |
pos |  19<298>|
----+---------+
(row = reference; col = test)
```

Then we keep training the Bayes model with our transcripts data. And the acc is show as 80% as well. Great!

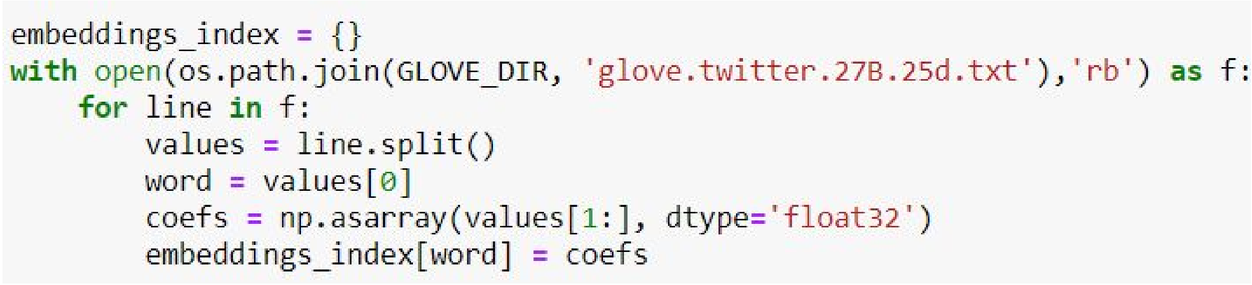
Word Embedding

We actually tried 3 different glove embeddings. They are Common Crawl (42B tokens, 300 dimensions), Wiki (6B tokens, 100 dimensions), and Twitter (27B tokens, 100 dimensions)

EX2 Word Embedding(twitter glove) on imdb.ipynb

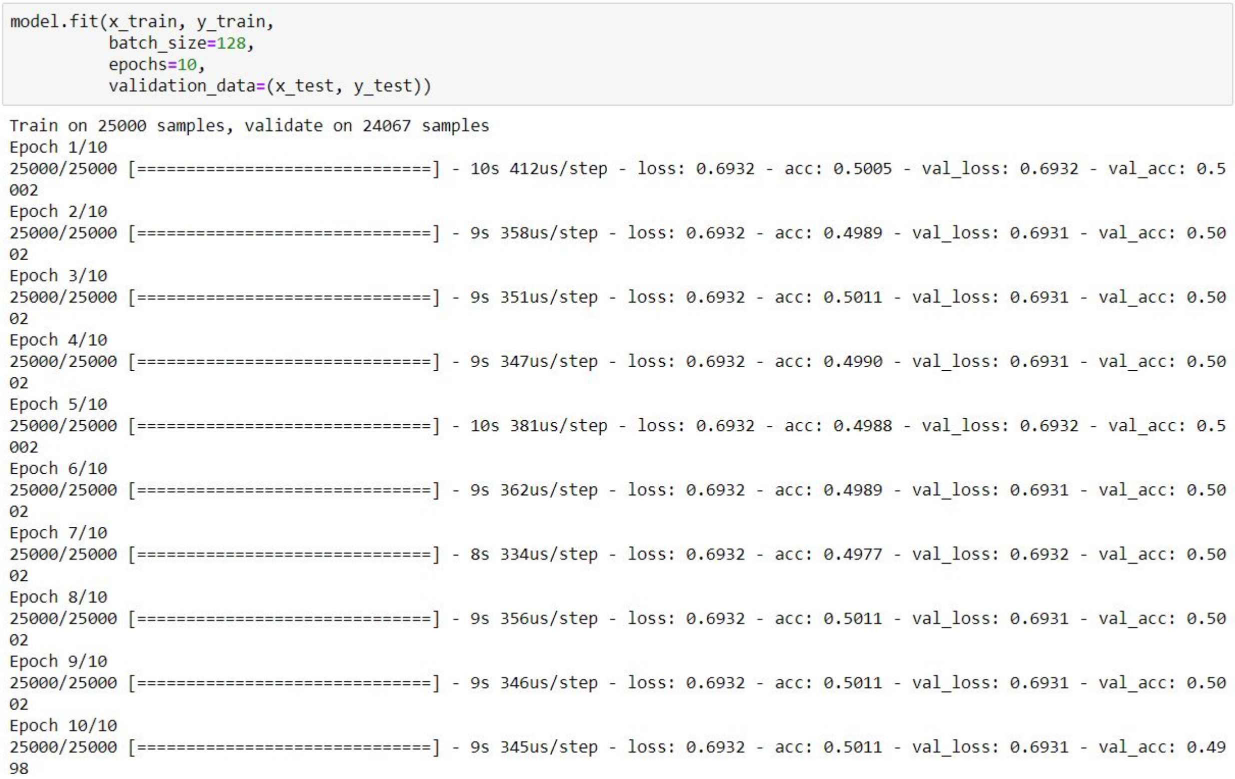
EX2 Word Embedding(wiki glove) on imdb.ipynb

First, we need to load the embeddings creating an embeddings_index.



```
embeddings_index = {}
with open(os.path.join(GLOVE_DIR, 'glove.twitter.27B.25d.txt'),'rb') as f:
    for line in f:
        values = line.split()
        word = values[0]
        coefs = np.asarray(values[1:], dtype='float32')
        embeddings_index[word] = coefs
```

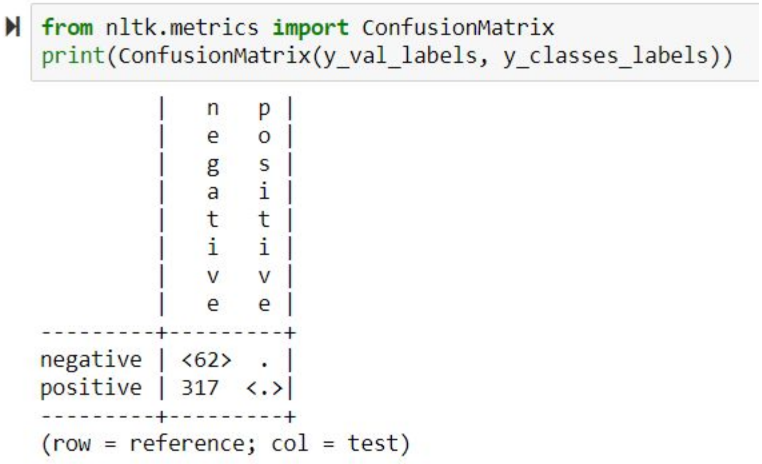
For this part, we used a simple neural network suggested by tensorflow to perform the classification task. Thus we used imdb data to train the simple nn.



```
model.fit(x_train, y_train,
          batch_size=128,
          epochs=10,
          validation_data=(x_test, y_test))
```

```
Train on 25000 samples, validate on 24067 samples
Epoch 1/10
25000/25000 [==============================] - 10s 412us/step - loss: 0.6932 - acc: 0.5005 - val_loss: 0.6932 - val_acc: 0.5
002
Epoch 2/10
25000/25000 [==============================] - 9s 358us/step - loss: 0.6932 - acc: 0.4989 - val_loss: 0.6931 - val_acc: 0.50
02
Epoch 3/10
25000/25000 [==============================] - 9s 351us/step - loss: 0.6932 - acc: 0.5011 - val_loss: 0.6931 - val_acc: 0.50
02
Epoch 4/10
25000/25000 [==============================] - 9s 347us/step - loss: 0.6932 - acc: 0.4990 - val_loss: 0.6931 - val_acc: 0.50
02
Epoch 5/10
25000/25000 [==============================] - 10s 381us/step - loss: 0.6932 - acc: 0.4988 - val_loss: 0.6932 - val_acc: 0.5
002
Epoch 6/10
25000/25000 [==============================] - 9s 362us/step - loss: 0.6932 - acc: 0.4989 - val_loss: 0.6931 - val_acc: 0.50
02
Epoch 7/10
25000/25000 [==============================] - 8s 334us/step - loss: 0.6932 - acc: 0.4977 - val_loss: 0.6932 - val_acc: 0.50
02
Epoch 8/10
25000/25000 [==============================] - 9s 356us/step - loss: 0.6932 - acc: 0.5011 - val_loss: 0.6931 - val_acc: 0.50
02
Epoch 9/10
25000/25000 [==============================] - 9s 346us/step - loss: 0.6932 - acc: 0.5011 - val_loss: 0.6931 - val_acc: 0.50
02
Epoch 10/10
25000/25000 [==============================] - 9s 345us/step - loss: 0.6932 - acc: 0.5011 - val_loss: 0.6931 - val_acc: 0.49
98
```

Later, we used the same pre-processing method on our transcripts data and put the data into the pre-trained model to make predictions.

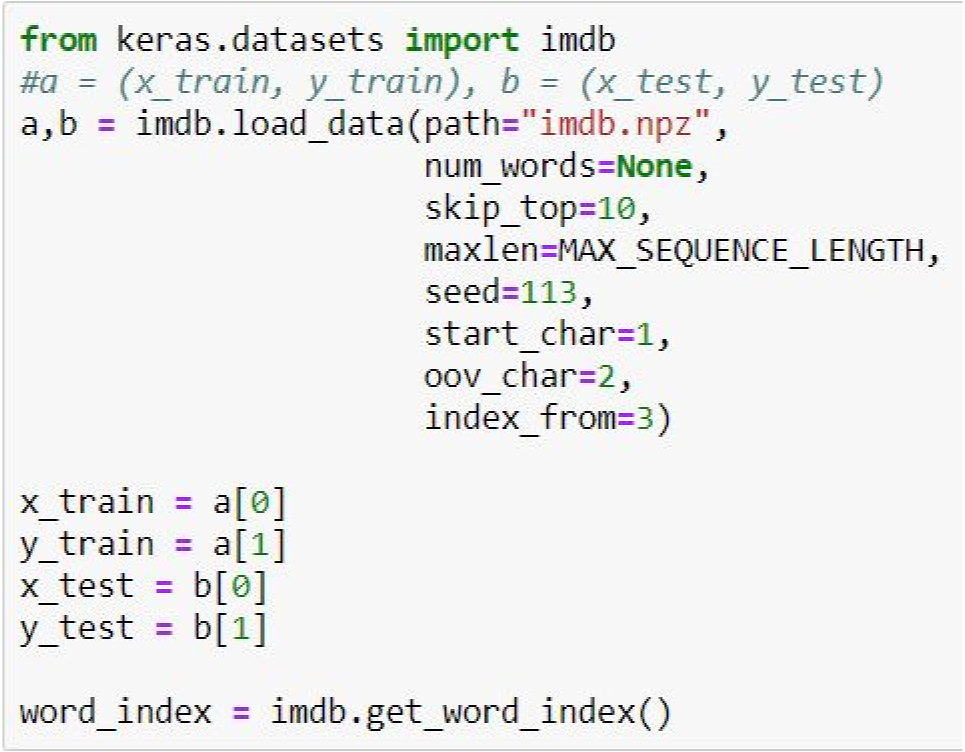


```
from nltk.metrics import ConfusionMatrix
print(ConfusionMatrix(y_val_labels, y_classes_labels))
```

```
         |   n   p |
         |   e   o |
         |   g   s |
         |   a   i |
         |   t   t |
         |   i   i |
         |   v   v |
         |   e   e |
---------+---------+
negative | <62>  . |
positive | 317  <.>|
---------+---------+
(row = reference; col = test)
```

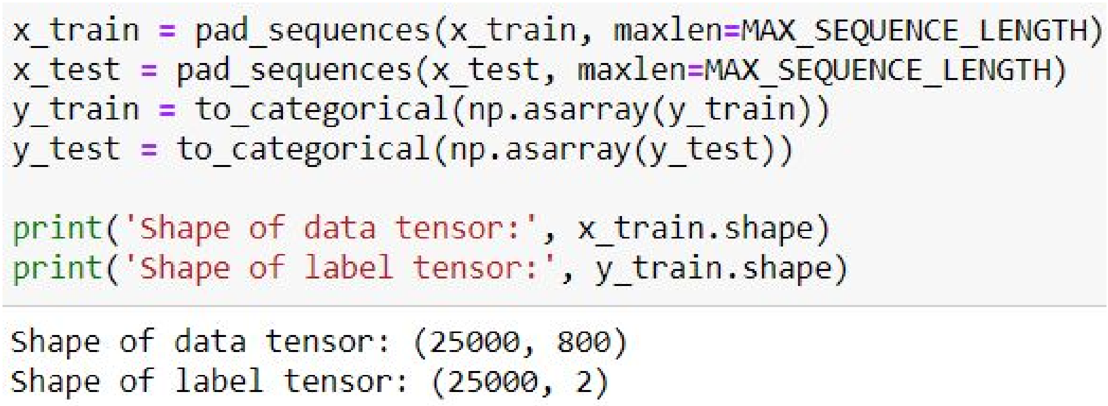
RNN(LSTM) on IMDB dataset without word embedding, then continuous train the mode with transcripts data.

Simply load the data from keras.datasets. The data is in sequences, thus for convenience, we will call imdb.get_word_index().



```
from keras.datasets import imdb
#a = (x_train, y_train), b = (x_test, y_test)
a,b = imdb.load_data(path="imdb.npz",
                     num_words=None,
                     skip_top=10,
                     maxlen=MAX_SEQUENCE_LENGTH,
                     seed=113,
                     start_char=1,
                     oov_char=2,
                     index_from=3)

x_train = a[0]
y_train = a[1]
x_test = b[0]
y_test = b[1]

word_index = imdb.get_word_index()
```

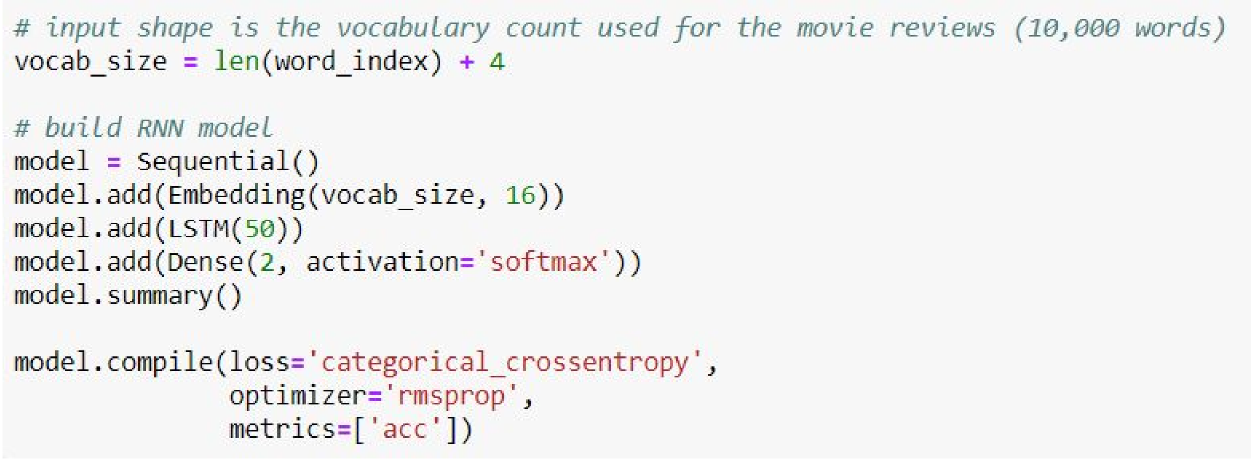
Padding and to_categorical



```
x_train = pad_sequences(x_train, maxlen=MAX_SEQUENCE_LENGTH)
x_test = pad_sequences(x_test, maxlen=MAX_SEQUENCE_LENGTH)
y_train = to_categorical(np.asarray(y_train))
y_test = to_categorical(np.asarray(y_test))

print('Shape of data tensor:', x_train.shape)
print('Shape of label tensor:', y_train.shape)
```

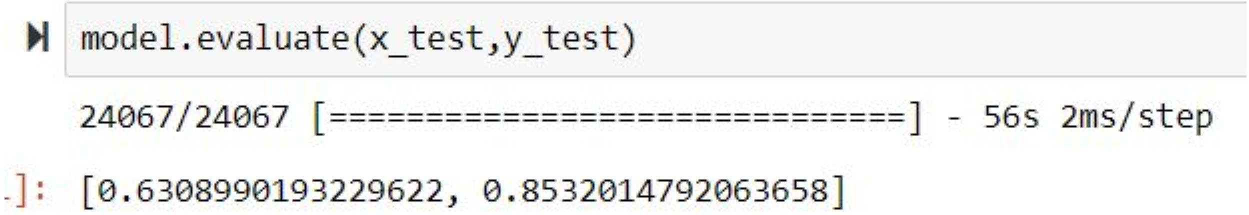
```
Shape of data tensor: (25000, 800)
Shape of label tensor: (25000, 2)
```

Build a simple LSTM model:



```
# input shape is the vocabulary count used for the movie reviews (10,000 words)
vocab_size = len(word_index) + 4

# build RNN model
model = Sequential()
model.add(Embedding(vocab_size, 16))
model.add(LSTM(50))
model.add(Dense(2, activation='softmax'))
model.summary()

model.compile(loss='categorical_crossentropy',
              optimizer='rmsprop',
              metrics=['acc'])
```

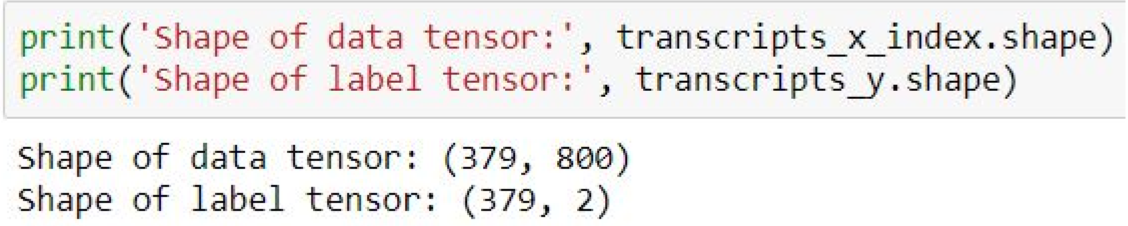
Train the model on imdb dataset, here is the evaluation:



```
model.evaluate(x_test,y_test)

24067/24067 [==============================] - 56s 2ms/step

.]: [0.6308990193229622, 0.8532014792063658]
```

The acc is 85%, not bad.

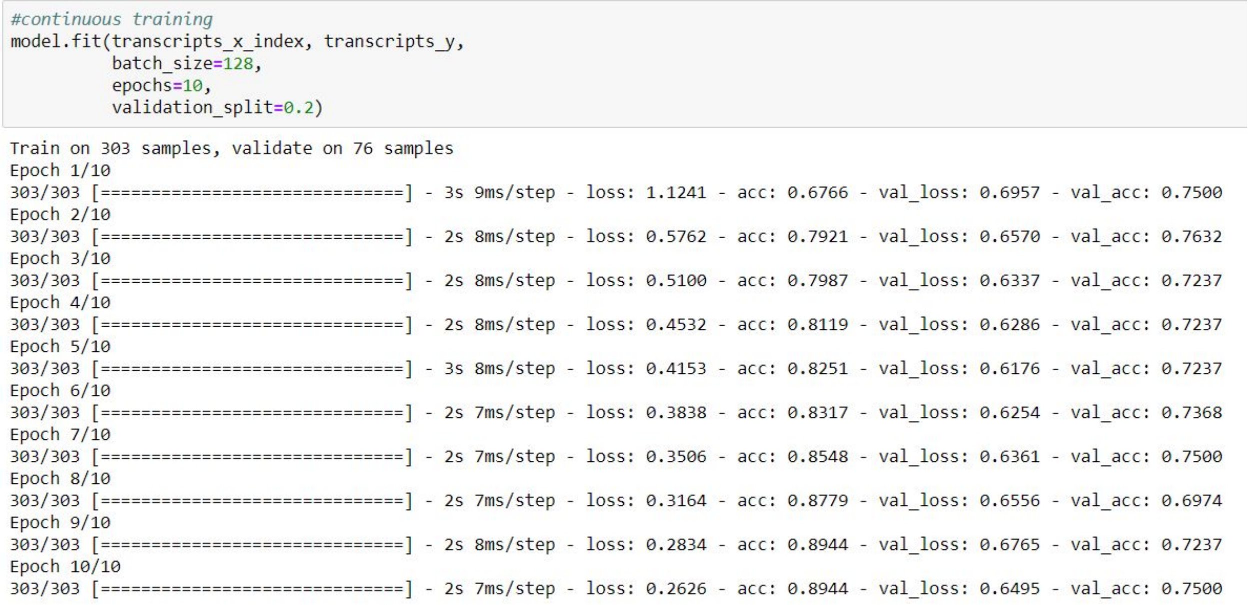
Then load our data. Because of the lack of ‘neutral’ class in imdb dataset. We decided to drop all those neutral texts in our dataset to make the prediction possible.



```
print('Shape of data tensor:', transcripts_x_index.shape)
print('Shape of label tensor:', transcripts_y.shape)
```

```
Shape of data tensor: (379, 800)
Shape of label tensor: (379, 2)
```

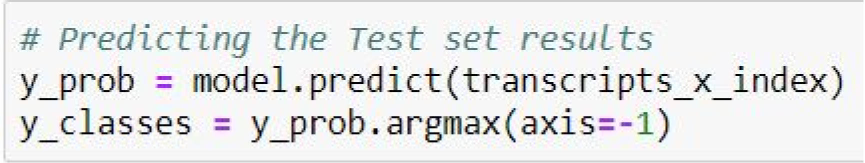
Continuous training the model using our data.



```
#continuous training
model.fit(transcripts_x_index, transcripts_y,
          batch_size=128,
          epochs=10,
          validation_split=0.2)
```

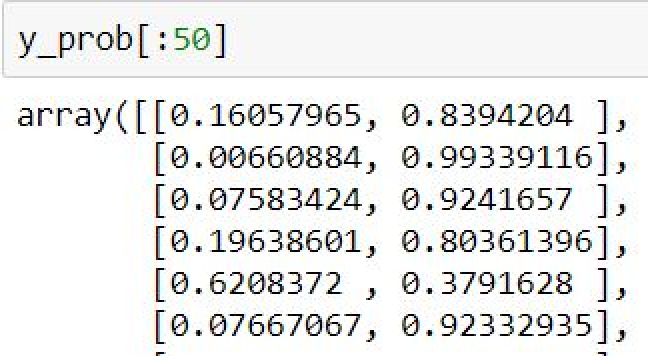
```
Train on 303 samples, validate on 76 samples
Epoch 1/10
303/303 [==============================] - 3s 9ms/step - loss: 1.1241 - acc: 0.6766 - val_loss: 0.6957 - val_acc: 0.7500
Epoch 2/10
303/303 [==============================] - 2s 8ms/step - loss: 0.5762 - acc: 0.7921 - val_loss: 0.6570 - val_acc: 0.7632
Epoch 3/10
303/303 [==============================] - 2s 8ms/step - loss: 0.5100 - acc: 0.7987 - val_loss: 0.6337 - val_acc: 0.7237
Epoch 4/10
303/303 [==============================] - 2s 8ms/step - loss: 0.4532 - acc: 0.8119 - val_loss: 0.6286 - val_acc: 0.7237
Epoch 5/10
303/303 [==============================] - 3s 8ms/step - loss: 0.4153 - acc: 0.8251 - val_loss: 0.6176 - val_acc: 0.7237
Epoch 6/10
303/303 [==============================] - 2s 7ms/step - loss: 0.3838 - acc: 0.8317 - val_loss: 0.6254 - val_acc: 0.7368
Epoch 7/10
303/303 [==============================] - 2s 7ms/step - loss: 0.3506 - acc: 0.8548 - val_loss: 0.6361 - val_acc: 0.7500
Epoch 8/10
303/303 [==============================] - 2s 7ms/step - loss: 0.3164 - acc: 0.8779 - val_loss: 0.6556 - val_acc: 0.6974
Epoch 9/10
303/303 [==============================] - 2s 8ms/step - loss: 0.2834 - acc: 0.8944 - val_loss: 0.6765 - val_acc: 0.7237
Epoch 10/10
303/303 [==============================] - 2s 7ms/step - loss: 0.2626 - acc: 0.8944 - val_loss: 0.6495 - val_acc: 0.7500
```

After that, predict our data. The second line code is to convert probabilities to class indices.



```
# Predicting the Test set results
y_prob = model.predict(transcripts_x_index)
y_classes = y_prob.argmax(axis=-1)
```

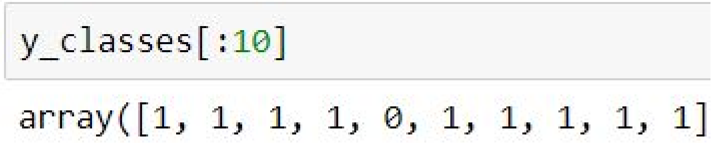
Before converting:



```
y_prob[:50]
```

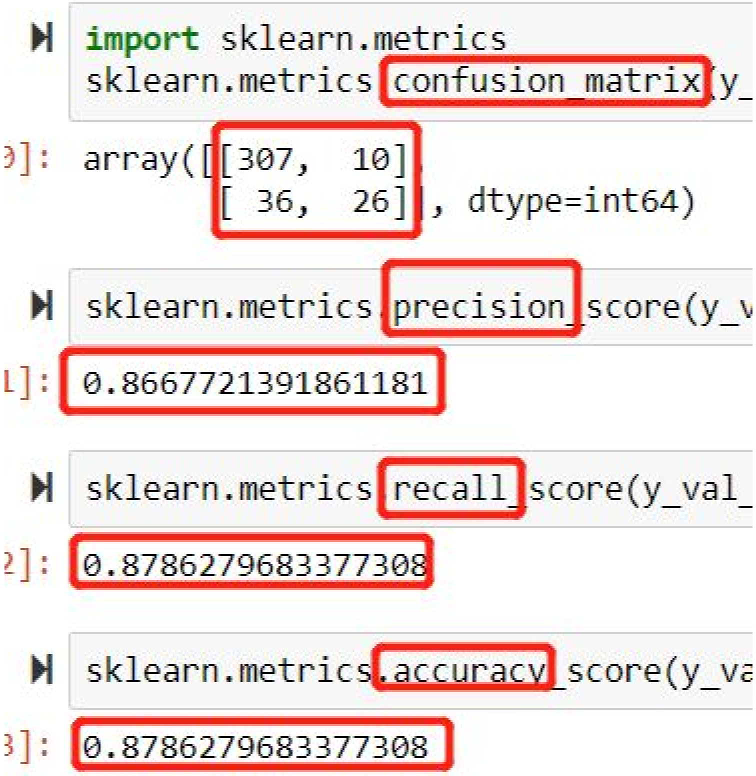
```
array([[0.16057965, 0.8394204 ],
       [0.00660884, 0.99339116],
       [0.07583424, 0.9241657 ],
       [0.19638601, 0.80361396],
       [0.6208372 , 0.3791628 ],
       [0.07667067, 0.92332935],
```

After converting:



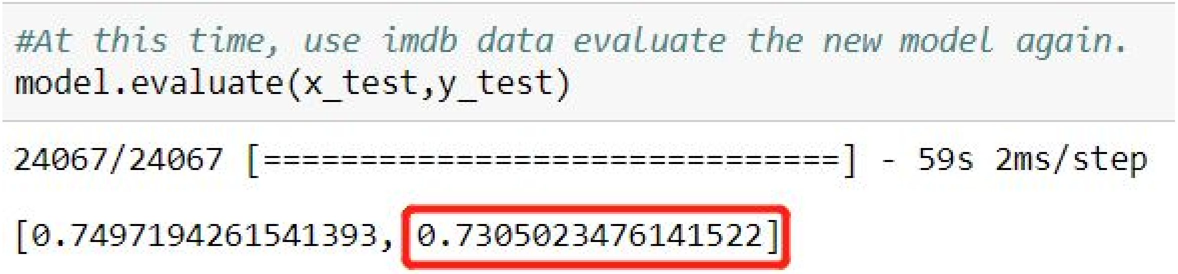
```
y_classes[:10]

array([1, 1, 1, 1, 0, 1, 1, 1, 1, 1])
```

Now evaluate the transfer model.

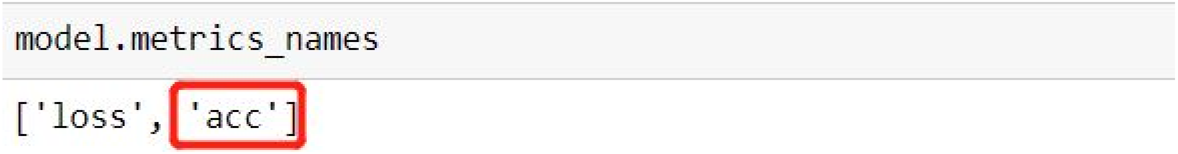


```
import sklearn.metrics
sklearn.metrics.confusion_matrix(y_

0]: array([[307,  10],
           [ 36,  26]], dtype=int64)

sklearn.metrics.precision_score(y_v

1]: 0.8667721391861181

sklearn.metrics.recall_score(y_val_

2]: 0.8786279683377308

sklearn.metrics.accuracy_score(y_va

3]: 0.8786279683377308
```

At this time, use imdb data evaluate the new model again.



```
#At this time, use imdb data evaluate the new model again.
model.evaluate(x_test,y_test)

24067/24067 [==============================] - 59s 2ms/step

[0.7497194261541393, 0.7305023476141522]
```



```
model.metrics_names

['loss', 'acc']
```

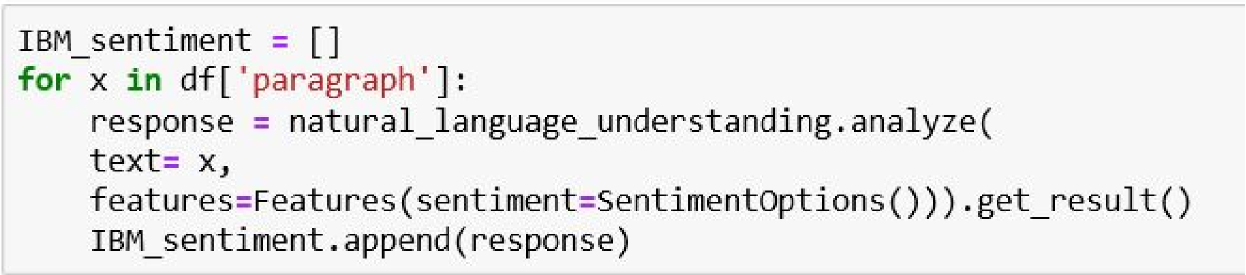
The transfer model successfully learned from our transcripts data, meanwhile, it keeps the accuracy on imdb data.

EX3

Obtain the sentiments using the Cloud APIs and calculate the confusion matrices for them.

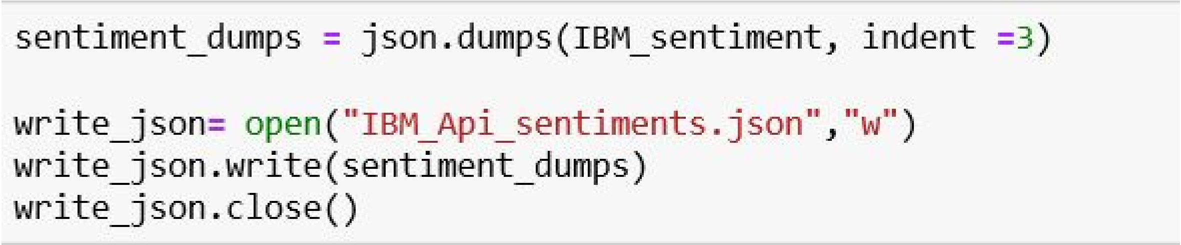
IBM - Natural Language Understanding

Watson Developer Cloud - 2.4.4



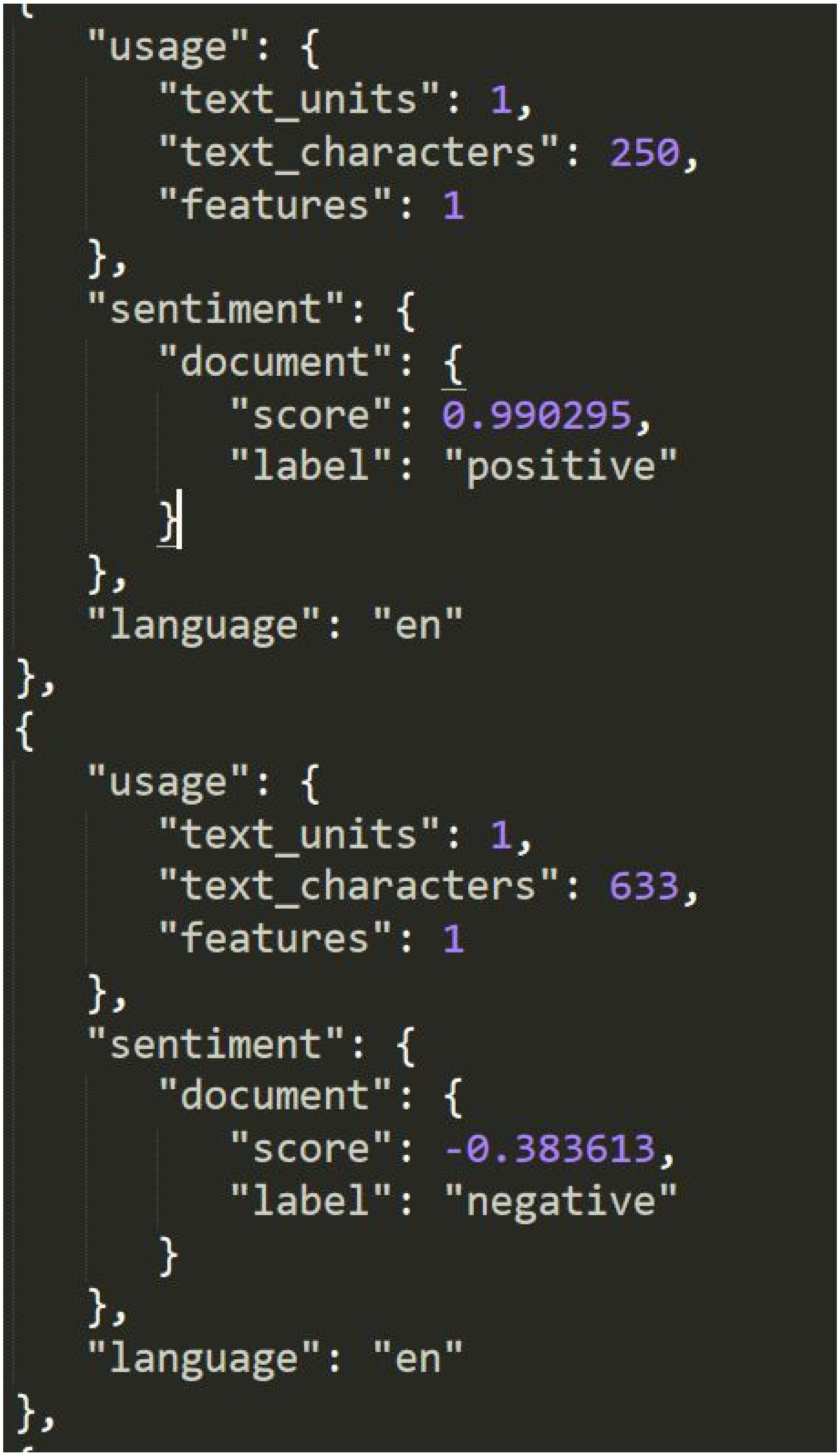
```
IBM_sentiment = []
for x in df['paragraph']:
    response = natural_language_understanding.analyze(
    text= x,
    features=Features(sentiment=SentimentOptions())).get_result()
    IBM_sentiment.append(response)
```

The IBM sentiments are dumped in a json file as follows.



```
sentiment_dumps = json.dumps(IBM_sentiment, indent =3)

write_json= open("IBM_Api_sentiments.json","w")
write_json.write(sentiment_dumps)
write_json.close()
```

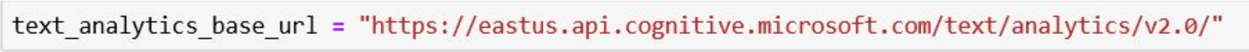
The sentiment scores given by the api fall in range -1 to +1 and it also gives the final sentiment label as shown below.



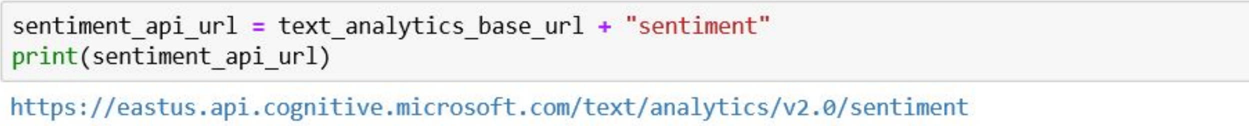
```
{
    "usage": {
        "text_units": 1,
        "text_characters": 250,
        "features": 1
    },
    "sentiment": {
        "document": {
            "score": 0.990295,
            "label": "positive"
        }
    },
    "language": "en"
},
{
    "usage": {
        "text_units": 1,
        "text_characters": 633,
        "features": 1
    },
    "sentiment": {
        "document": {
            "score": -0.383613,
            "label": "negative"
        }
    },
    "language": "en"
},
{
```

Microsoft - Text Analytics

The Microsoft text analytics cloud api is url based as shown below



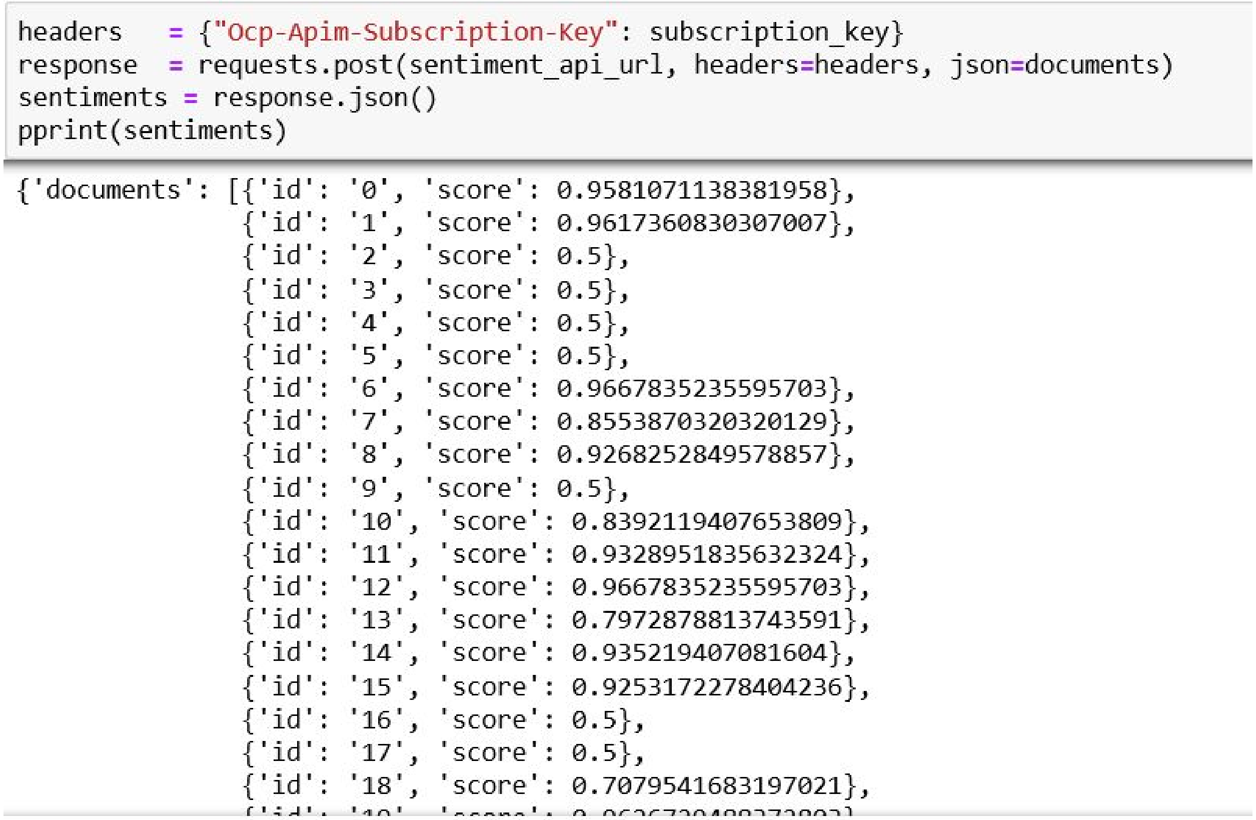
```
text_analytics_base_url = "https://eastus.api.cognitive.microsoft.com/text/analytics/v2.0/"
```



```
sentiment_api_url = text_analytics_base_url + "sentiment"
print(sentiment_api_url)
```

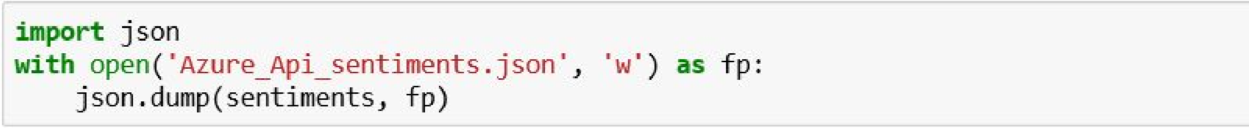
```
https://eastus.api.cognitive.microsoft.com/text/analytics/v2.0/sentiment
```

All the paragraphs are taken, processed into a json with documents as header and posted as a request to the url.



```
headers   = {"Ocp-Apim-Subscription-Key": subscription_key}
response  = requests.post(sentiment_api_url, headers=headers, json=documents)
sentiments = response.json()
pprint(sentiments)
```

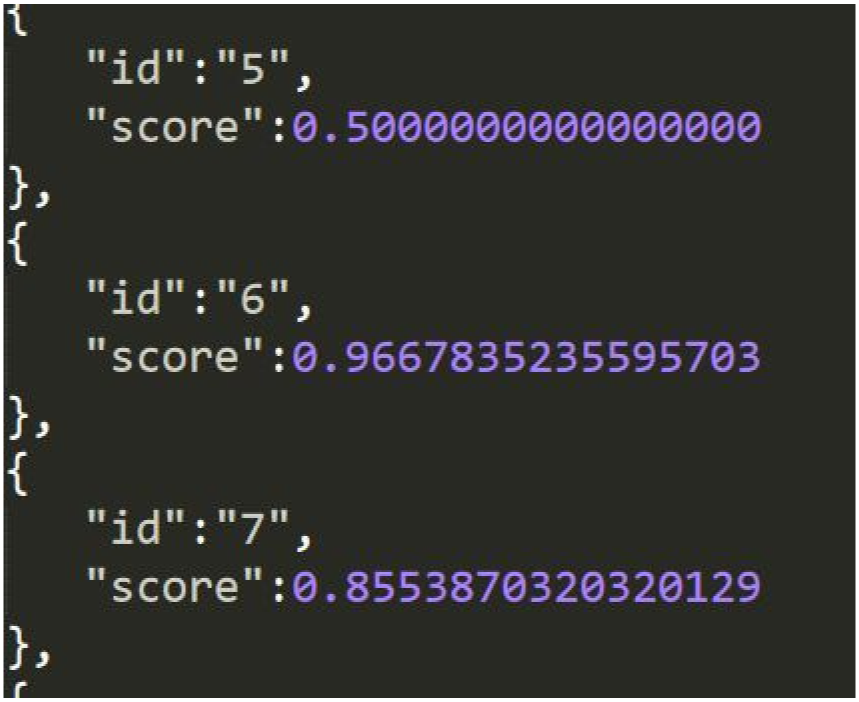
```
{'documents': [{'id': '0', 'score': 0.9581071138381958},
               {'id': '1', 'score': 0.9617360830307007},
               {'id': '2', 'score': 0.5},
               {'id': '3', 'score': 0.5},
               {'id': '4', 'score': 0.5},
               {'id': '5', 'score': 0.5},
               {'id': '6', 'score': 0.9667835235595703},
               {'id': '7', 'score': 0.8553870320320129},
               {'id': '8', 'score': 0.9268252849578857},
               {'id': '9', 'score': 0.5},
               {'id': '10', 'score': 0.8392119407653809},
               {'id': '11', 'score': 0.9328951835632324},
               {'id': '12', 'score': 0.9667835235595703},
               {'id': '13', 'score': 0.7972878813743591},
               {'id': '14', 'score': 0.935219407081604},
               {'id': '15', 'score': 0.9253172278404236},
               {'id': '16', 'score': 0.5},
               {'id': '17', 'score': 0.5},
               {'id': '18', 'score': 0.7079541683197021},
               {'id': '19', 'score': 0.9626720488372892}
```



```
import json
with open('Azure_Api_sentiments.json', 'w') as fp:
    json.dump(sentiments, fp)
```

The response sentiments are dumped in a json file.

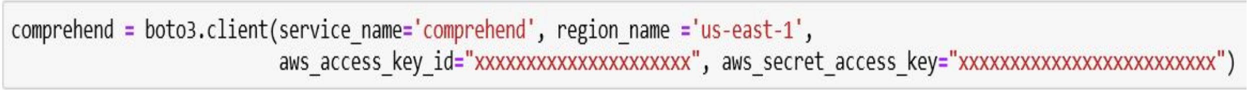
The sentiment scores given by the api are from 0 to 1 with 0.5 as neutral. Also, there are no sentiment labels given in response.



```
{
    "id":"5",
    "score":0.5000000000000000
},
{
    "id":"6",
    "score":0.9667835235595703
},
{
    "id":"7",
    "score":0.8553870320320129
},
{
```

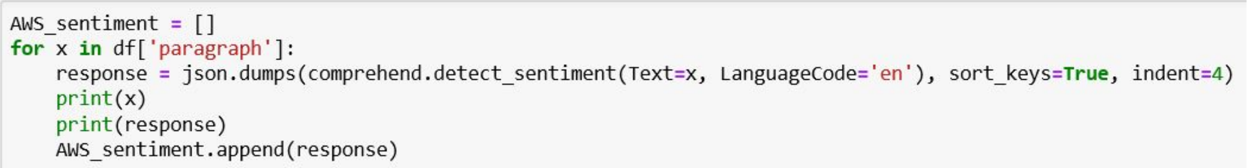
Amazon - Comprehend

It is a service which can be used using boto3 client.

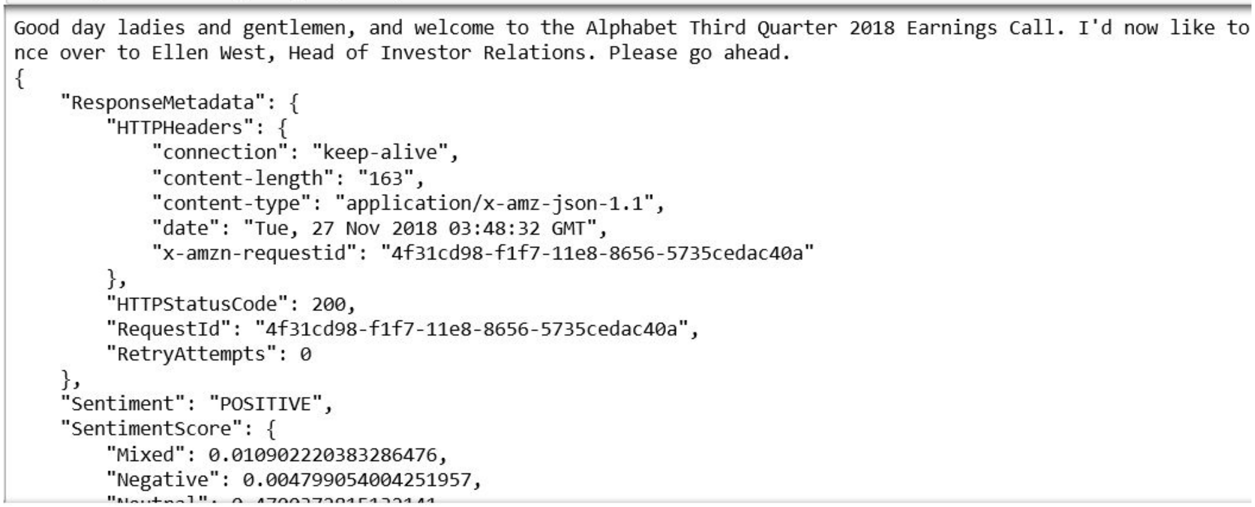


```
comprehend = boto3.client(service_name='comprehend', region_name ='us-east-1',
                          aws_access_key_id="xxxxxxxxxxxxxxxxxxxx", aws_secret_access_key="xxxxxxxxxxxxxxxxxxxxxxxxx")
```

Sentiments are detected using the following code and dumped to json file.

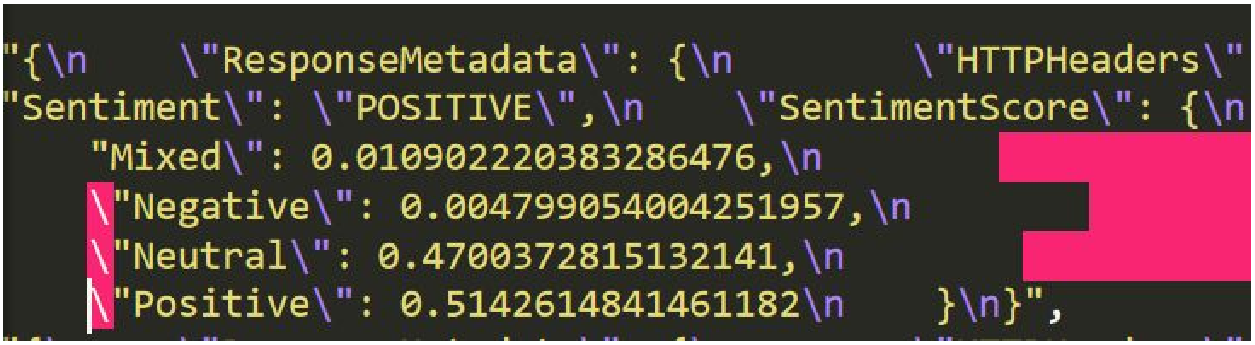


```
AWS_sentiment = []
for x in df['paragraph']:
    response = json.dumps(comprehend.detect_sentiment(Text=x, LanguageCode='en'), sort_keys=True, indent=4)
    print(x)
    print(response)
    AWS_sentiment.append(response)
```



```
Good day ladies and gentlemen, and welcome to the Alphabet Third Quarter 2018 Earnings Call. I'd now like to
nce over to Ellen West, Head of Investor Relations. Please go ahead.
{
    "ResponseMetadata": {
        "HTTPHeaders": {
            "connection": "keep-alive",
            "content-length": "163",
            "content-type": "application/x-amz-json-1.1",
            "date": "Tue, 27 Nov 2018 03:48:32 GMT",
            "x-amzn-requestid": "4f31cd98-f1f7-11e8-8656-5735cedac40a"
        },
        "HTTPStatusCode": 200,
        "RequestId": "4f31cd98-f1f7-11e8-8656-5735cedac40a",
        "RetryAttempts": 0
    },
    "Sentiment": "POSITIVE",
    "SentimentScore": {
        "Mixed": 0.010902220383286476,
        "Negative": 0.004799054004251957,
        "Neutral": 0.4790272815132141
```

The api gives 4 scores for sentiments Mixed, Neutral, Negative and Positive. All the scores add up to 1 like softmax probabilities.

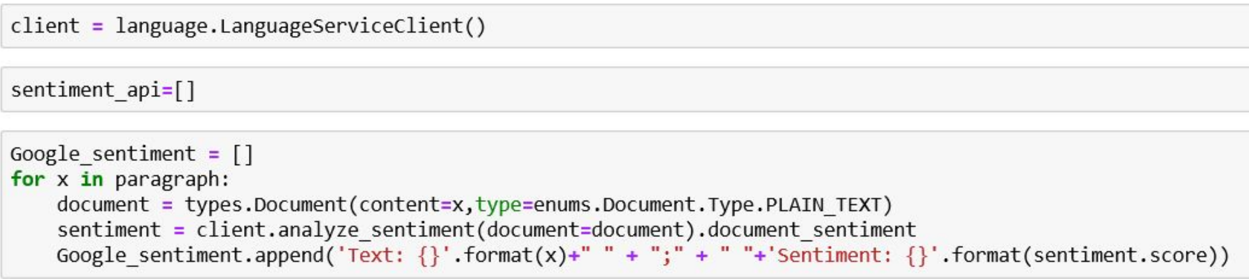


```
"{\n    \"ResponseMetadata\": {\n        \"HTTPHeaders\"
"Sentiment\": \"POSITIVE\",\n    \"SentimentScore\": {\n
    "Mixed\": 0.010902220383286476,\n
    \"Negative\": 0.004799054004251957,\n
    \"Neutral\": 0.4700372815132141,\n
    \"Positive\": 0.5142614841461182\n    }\n}",
```

It also tells which is the sentiment it detected which is nothing but the max value of the 4 sentiments. I didn't find any scenario where 2 sentiment scores are max and equal as i wanted to see what would be the overall sentiment for that scenario.

Google Cloud Language Service

Dealing with Google cloud api was a bit tricky. It was not working on laptops and was only working on google colab because of some installation constraints.

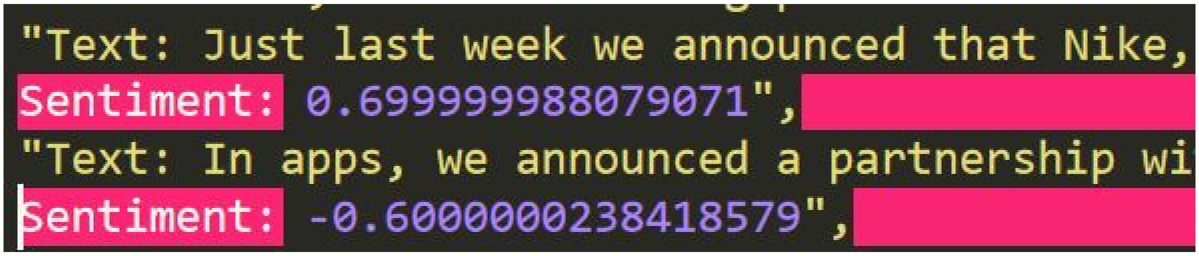


```
client = language.LanguageServiceClient()
```

```
sentiment_api=[]
```

```
Google_sentiment = []
for x in paragraph:
    document = types.Document(content=x,type=enums.Document.Type.PLAIN_TEXT)
    sentiment = client.analyze_sentiment(document=document).document_sentiment
    Google_sentiment.append('Text: {}'.format(x)+" " + ";" + " "+'Sentiment: {}'.format(sentiment.score))
```

Another tricky part was that passing all the paragraphs to the api was throwing errors that it is not possible. So, each paragraph was passed as a Text and its sentiment was recorded and dumped in json file.

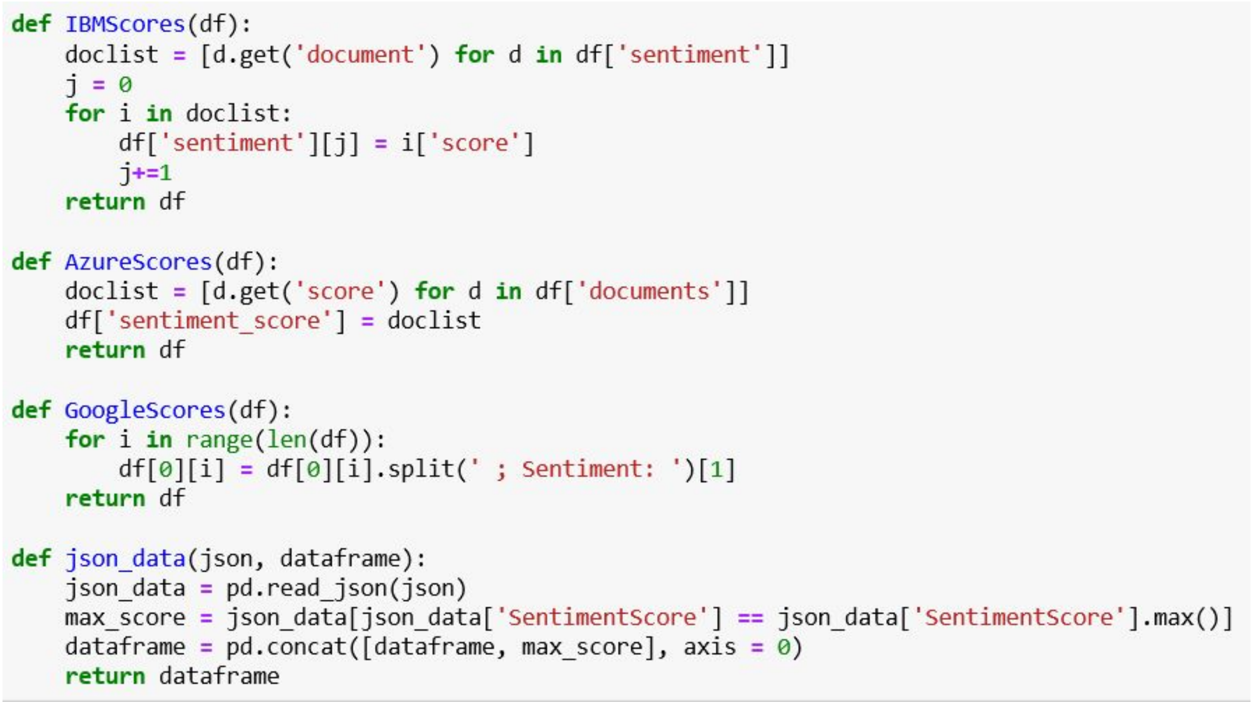


```
"Text: Just last week we announced that Nike,
Sentiment: 0.699999988079071",
"Text: In apps, we announced a partnership wi
Sentiment: -0.6000000238418579",
```

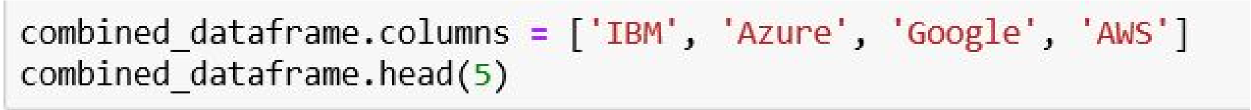
The api gave scores from -1 to +1 with 0 as neutral. The sentiment labels were not returned in the response.

Normalization

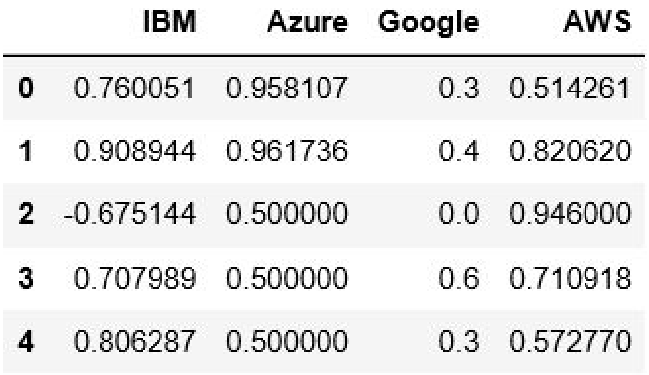
All the json files are now parsed and loaded in 1 dataframe as follows.



```
def IBMScores(df):
    doclist = [d.get('document') for d in df['sentiment']]
    j = 0
    for i in doclist:
        df['sentiment'][j] = i['score']
        j+=1
    return df

def AzureScores(df):
    doclist = [d.get('score') for d in df['documents']]
    df['sentiment_score'] = doclist
    return df

def GoogleScores(df):
    for i in range(len(df)):
        df[0][i] = df[0][i].split(' ; Sentiment: ')[1]
    return df

def json_data(json, dataframe):
    json_data = pd.read_json(json)
    max_score = json_data[json_data['SentimentScore'] == json_data['SentimentScore'].max()]
    dataframe = pd.concat([dataframe, max_score], axis = 0)
    return dataframe
```

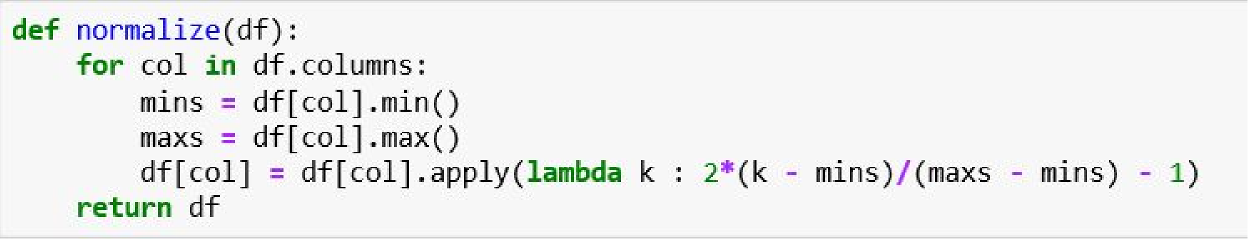


```
combined_dataframe.columns = ['IBM', 'Azure', 'Google', 'AWS']
combined_dataframe.head(5)
```

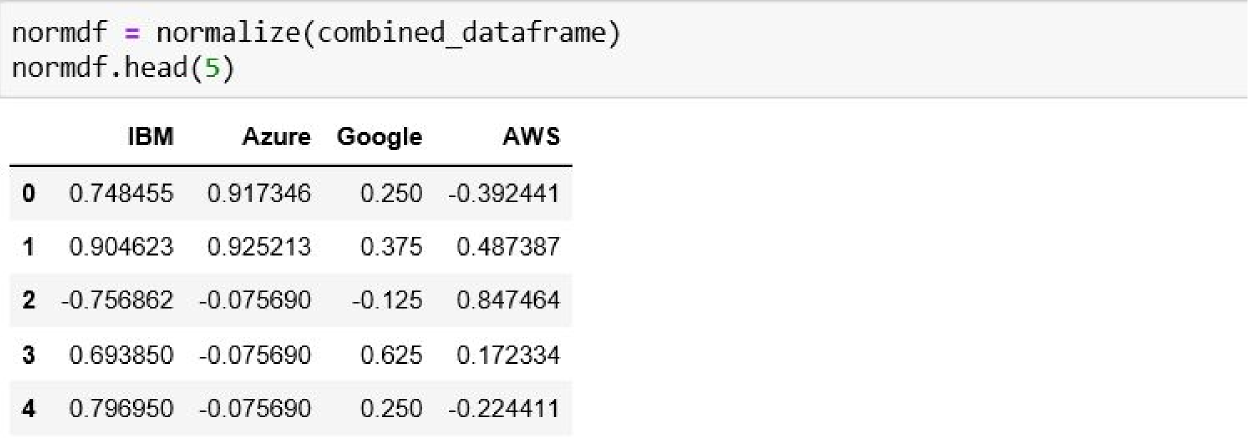


	IBM	Azure	Google	AWS
0	0.760051	0.958107	0.3	0.514261
1	0.908944	0.961736	0.4	0.820620
2	-0.675144	0.500000	0.0	0.946000
3	0.707989	0.500000	0.6	0.710918
4	0.806287	0.500000	0.3	0.572770

Basic formula for normalization is used as follows.



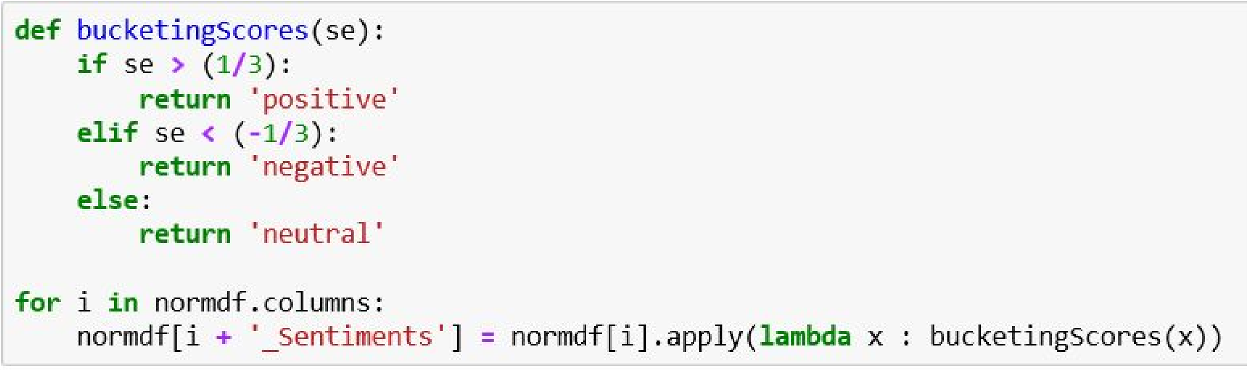
```python
def normalize(df):
    for col in df.columns:
        mins = df[col].min()
        maxs = df[col].max()
        df[col] = df[col].apply(lambda k : 2*(k - mins)/(maxs - mins) - 1)
    return df
```



```python
normdf = normalize(combined_dataframe)
normdf.head(5)
```

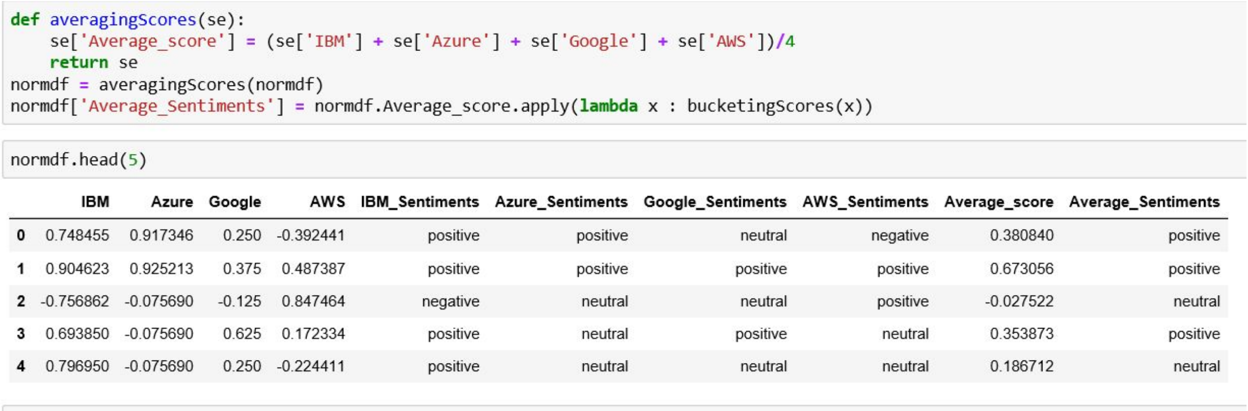
	IBM	Azure	Google	AWS
0	0.748455	0.917346	0.250	-0.392441
1	0.904623	0.925213	0.375	0.487387
2	-0.756862	-0.075690	-0.125	0.847464
3	0.693850	-0.075690	0.625	0.172334
4	0.796950	-0.075690	0.250	-0.224411

Then we defined the buckets as the normalization range is -1 to +1.



```python
def bucketingScores(se):
    if se > (1/3):
        return 'positive'
    elif se < (-1/3):
        return 'negative'
    else:
        return 'neutral'

for i in normdf.columns:
    normdf[i + '_Sentiments'] = normdf[i].apply(lambda x : bucketingScores(x))
```

Averaging out the scores and then adding the proper labels.

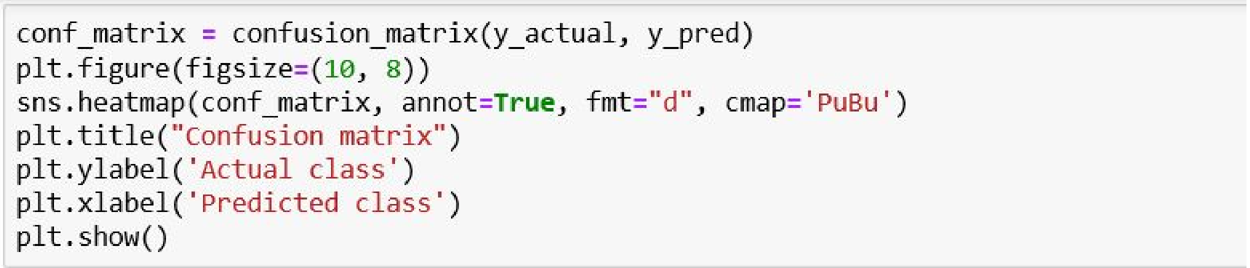


```python
def averagingScores(se):
    se['Average_score'] = (se['IBM'] + se['Azure'] + se['Google'] + se['AWS'])/4
    return se
normdf = averagingScores(normdf)
normdf['Average_Sentiments'] = normdf.Average_score.apply(lambda x : bucketingScores(x))
```

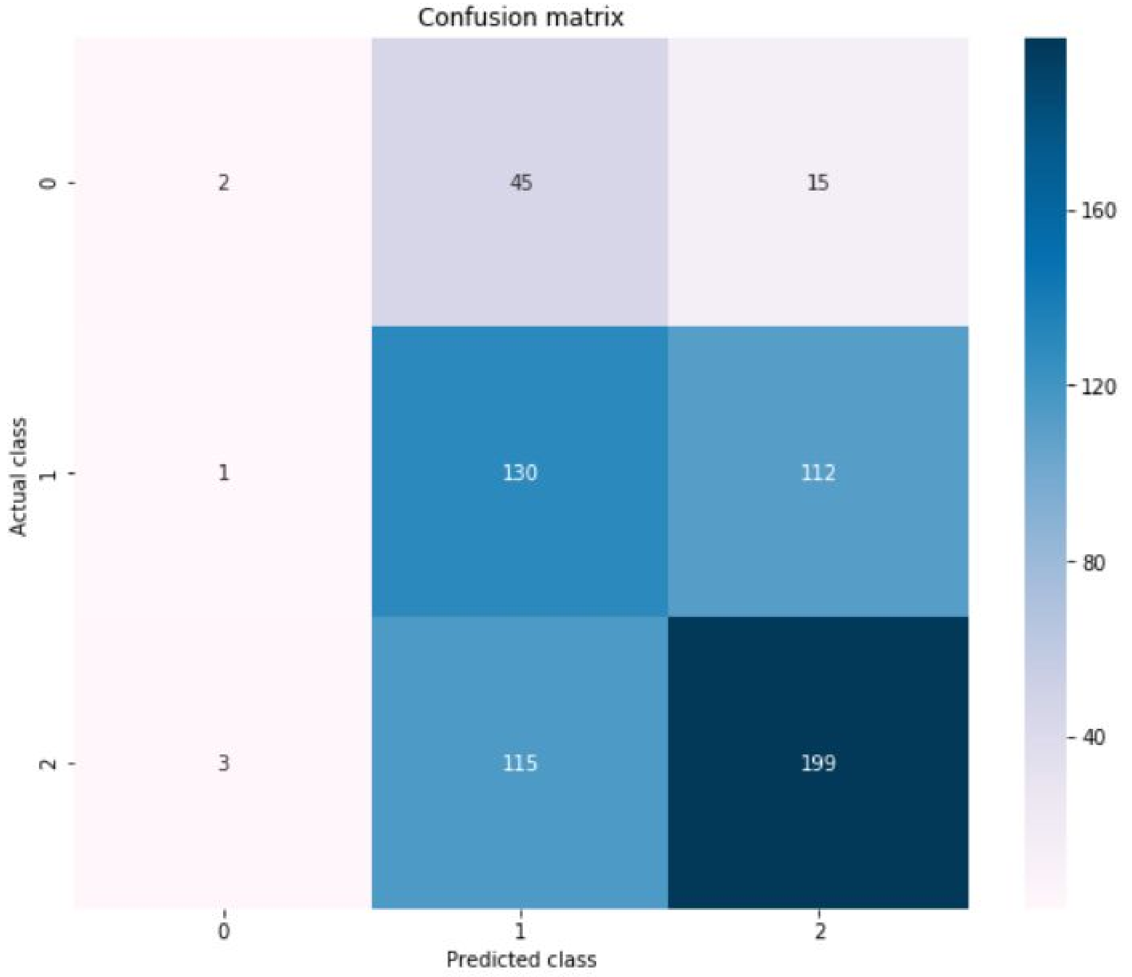
```python
normdf.head(5)
```

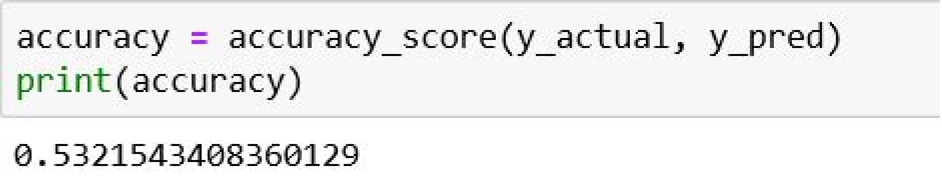
	IBM	Azure	Google	AWS	IBM_Sentiments	Azure_Sentiments	Google_Sentiments	AWS_Sentiments	Average_score	Average_Sentiments
0	0.748455	0.917346	0.250	-0.392441	positive	positive	neutral	negative	0.380840	positive
1	0.904623	0.925213	0.375	0.487387	positive	positive	positive	positive	0.673056	positive
2	-0.756862	-0.075690	-0.125	0.847464	negative	neutral	neutral	positive	-0.027522	neutral
3	0.693850	-0.075690	0.625	0.172334	positive	neutral	positive	neutral	0.353873	positive
4	0.796950	-0.075690	0.250	-0.224411	positive	neutral	neutral	neutral	0.186712	neutral

Calculated the confusion matrix of the average sentiments with manually labelled sentiments.



```
conf_matrix = confusion_matrix(y_actual, y_pred)
plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap='PuBu')
plt.title("Confusion matrix")
plt.ylabel('Actual class')
plt.xlabel('Predicted class')
plt.show()
```





```
accuracy = accuracy_score(y_actual, y_pred)
print(accuracy)
```

```
0.5321543408360129
```

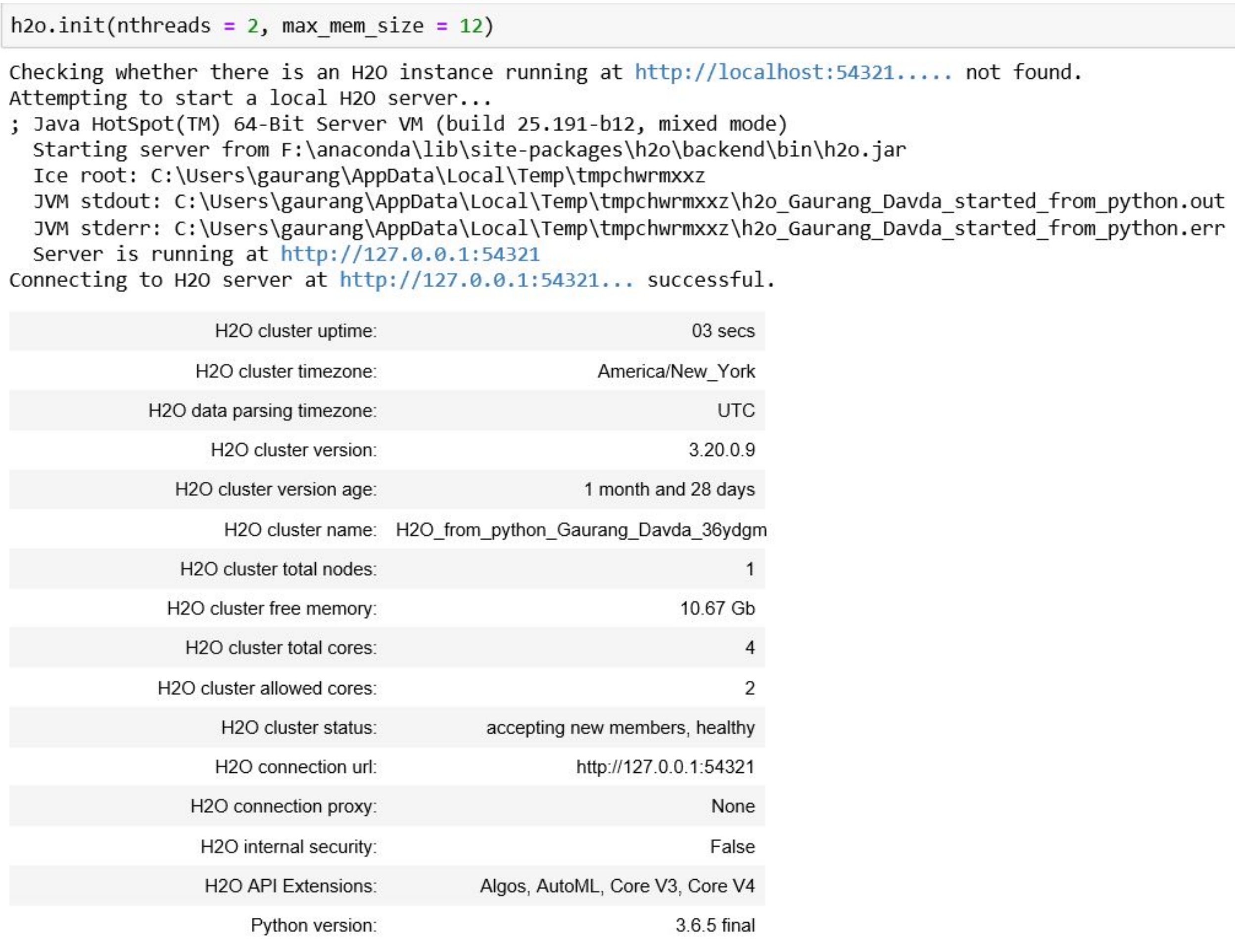
The averaging out experiment gave accuracy score of 0.53

The accuracy seems to be bad as the labelled data from the 4 APIs are compared with the manually labelled data which is highly likely to be wrong as the financial data was a bit tricky to differentiate between neutral and positives in some cases. Also, the way a CEO or CTO describes a negative point in a positive way is tricky and confuses many people whether that paragraph is positive negative or neutral sentiment.

EX4

Just averaging the 4 scores is not a good idea over here as there were clear differences in the sentiment scores of different cloud apis. Trying to build a model using AutoML can be beneficial in this scenario.

H2O Deep Learning Estimator



```
h2o.init(nthreads = 2, max_mem_size = 12)
```

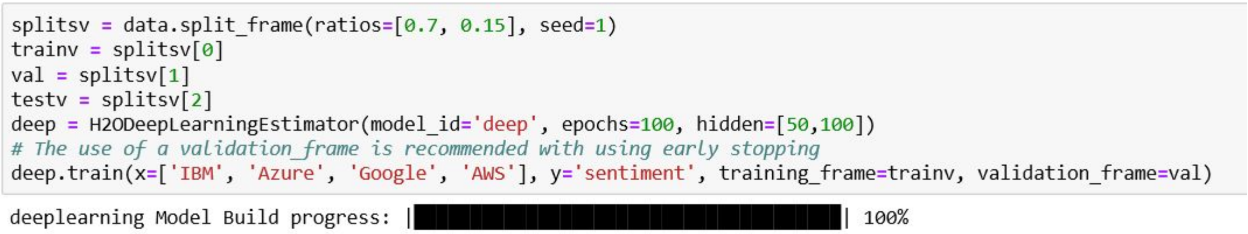
```
Checking whether there is an H2O instance running at http://localhost:54321..... not found.
Attempting to start a local H2O server...
; Java HotSpot(TM) 64-Bit Server VM (build 25.191-b12, mixed mode)
  Starting server from F:\anaconda\lib\site-packages\h2o\backend\bin\h2o.jar
  Ice root: C:\Users\gaurang\AppData\Local\Temp\tmpchwrmxxz
  JVM stdout: C:\Users\gaurang\AppData\Local\Temp\tmpchwrmxxz\h2o_Gaurang_Davda_started_from_python.out
  JVM stderr: C:\Users\gaurang\AppData\Local\Temp\tmpchwrmxxz\h2o_Gaurang_Davda_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:	03 secs
H2O cluster timezone:	America/New_York
H2O data parsing timezone:	UTC
H2O cluster version:	3.20.0.9
H2O cluster version age:	1 month and 28 days
H2O cluster name:	H2O_from_python_Gaurang_Davda_36ydgm
H2O cluster total nodes:	1
H2O cluster free memory:	10.67 Gb
H2O cluster total cores:	4
H2O cluster allowed cores:	2
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

H2O models use H2OFrame instead of Pandas DataFrame for processing.

```
data = h2o.import_file('data_for_automl.csv')
```

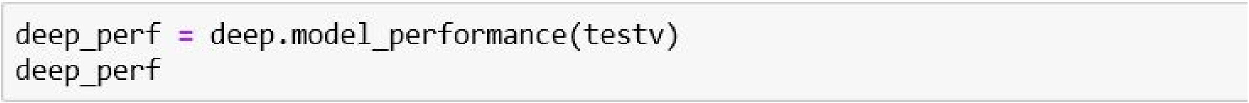
```
Parse progress: |████████████████████████████████████████████████████████| 100%
```



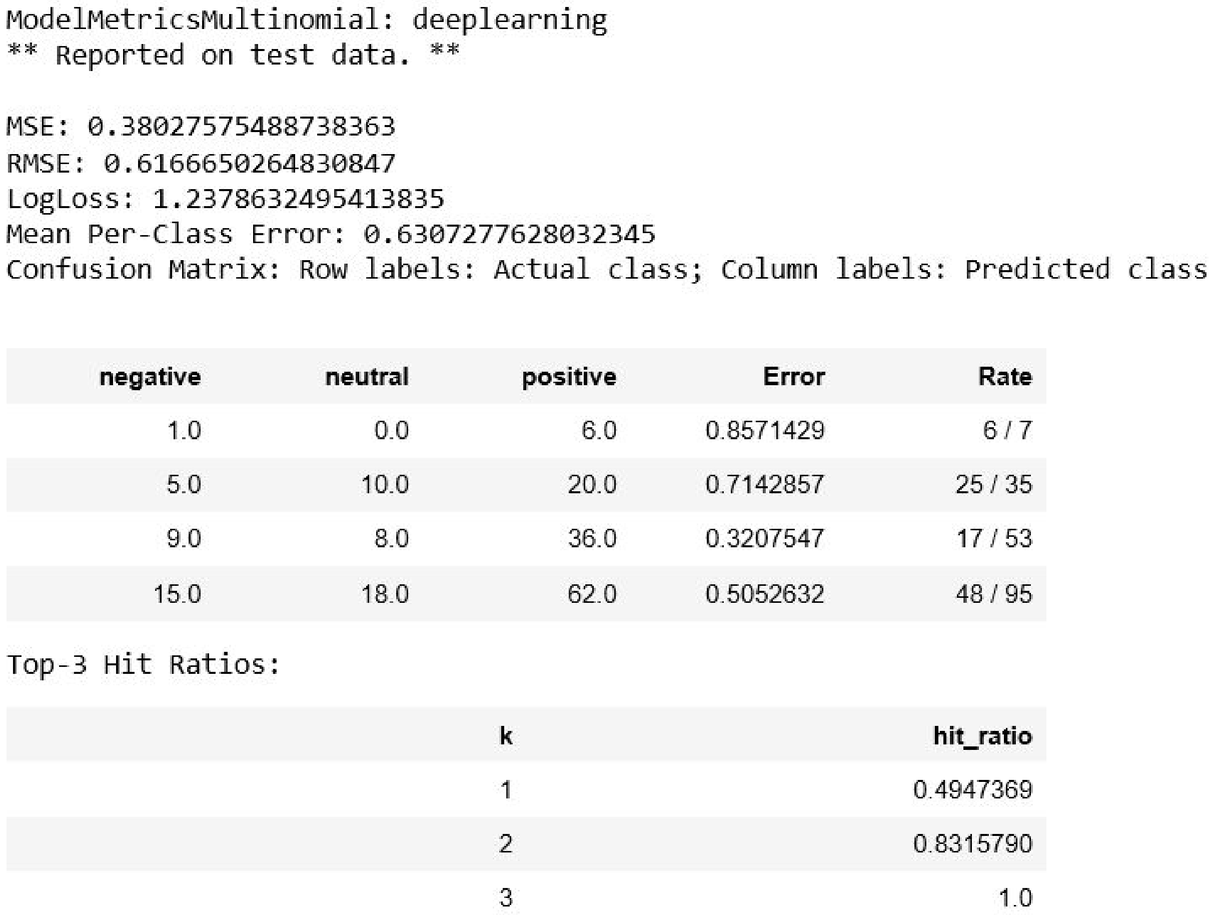
```
splitsv = data.split_frame(ratios=[0.7, 0.15], seed=1)
trainv = splitsv[0]
val = splitsv[1]
testv = splitsv[2]
deep = H2ODeepLearningEstimator(model_id='deep', epochs=100, hidden=[50,100])
# The use of a validation_frame is recommended with using early stopping
deep.train(x=['IBM', 'Azure', 'Google', 'AWS'], y='sentiment', training_frame=trainv, validation_frame=val)
```

```
deeplearning Model Build progress: |████████████████████████████████████████| 100%
```

We have to pass the whole h2oframe and just specify the features and target column.
H2O understands the nature of target column and decides if it is binary/multi class classification or regression.



```
deep_perf = deep.model_performance(testv)
deep_perf
```

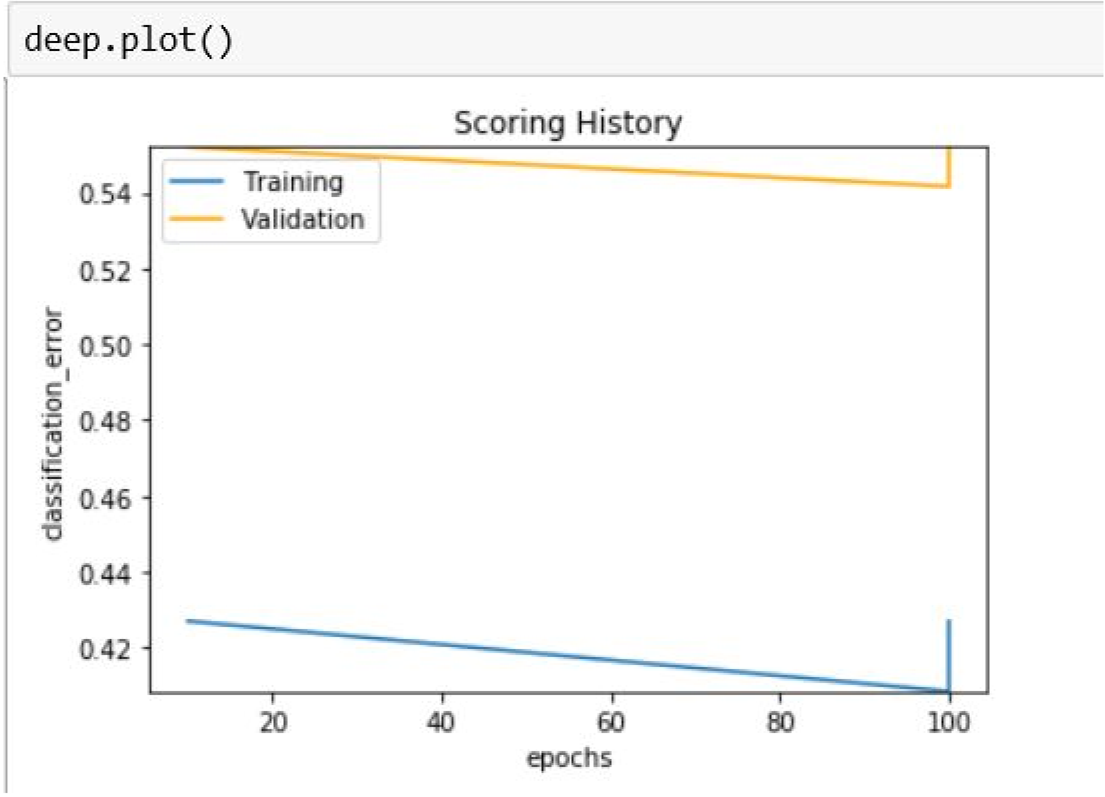


```
ModelMetricsMultinomial: deeplearning
** Reported on test data. **

MSE: 0.38027575488738363
RMSE: 0.6166650264830847
LogLoss: 1.2378632495413835
Mean Per-Class Error: 0.6307277628032345
Confusion Matrix: Row labels: Actual class; Column labels: Predicted class
```

negative	neutral	positive	Error	Rate
1.0	0.0	6.0	0.8571429	6 / 7
5.0	10.0	20.0	0.7142857	25 / 35
9.0	8.0	36.0	0.3207547	17 / 53
15.0	18.0	62.0	0.5052632	48 / 95

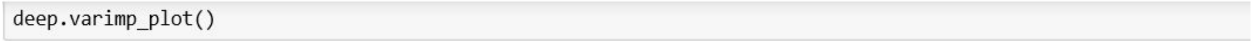
```
Top-3 Hit Ratios:
```

k	hit_ratio
1	0.4947369
2	0.8315790
3	1.0

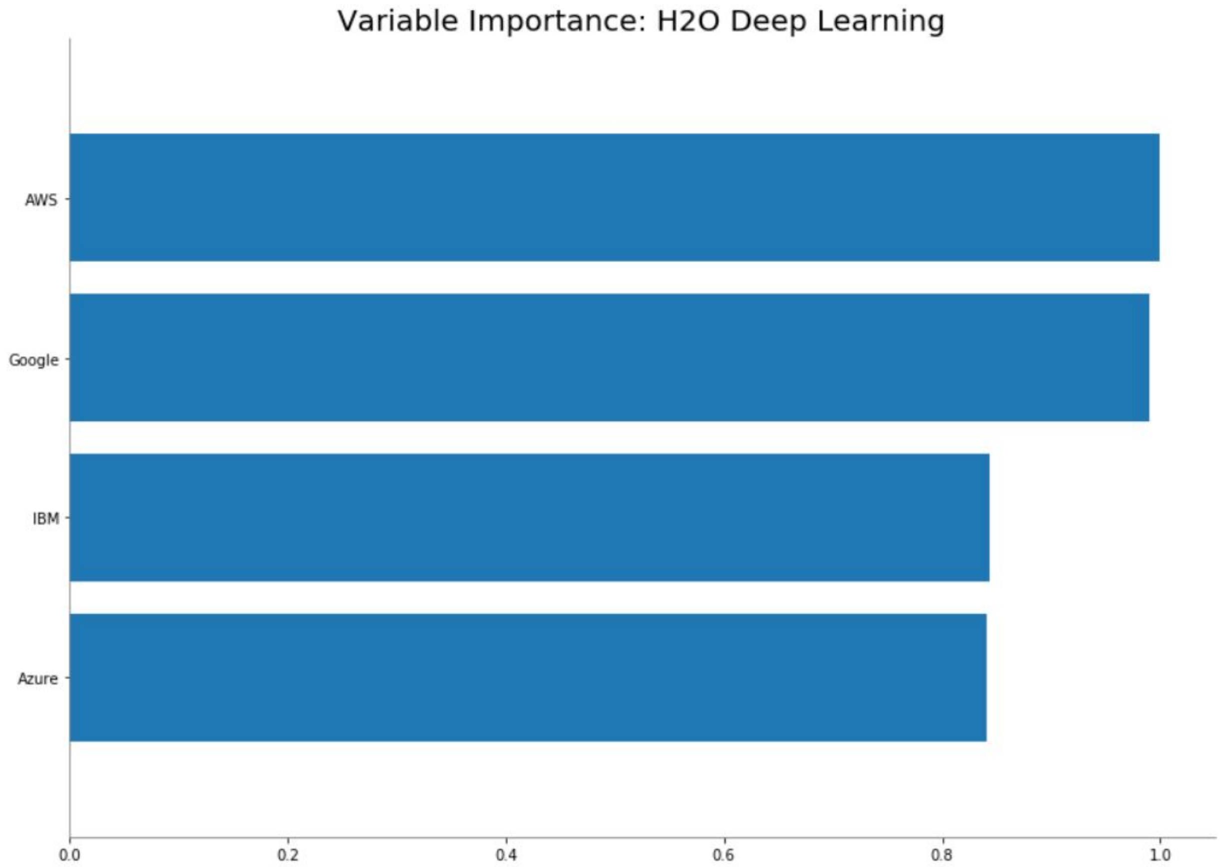


```
deep.plot()
```

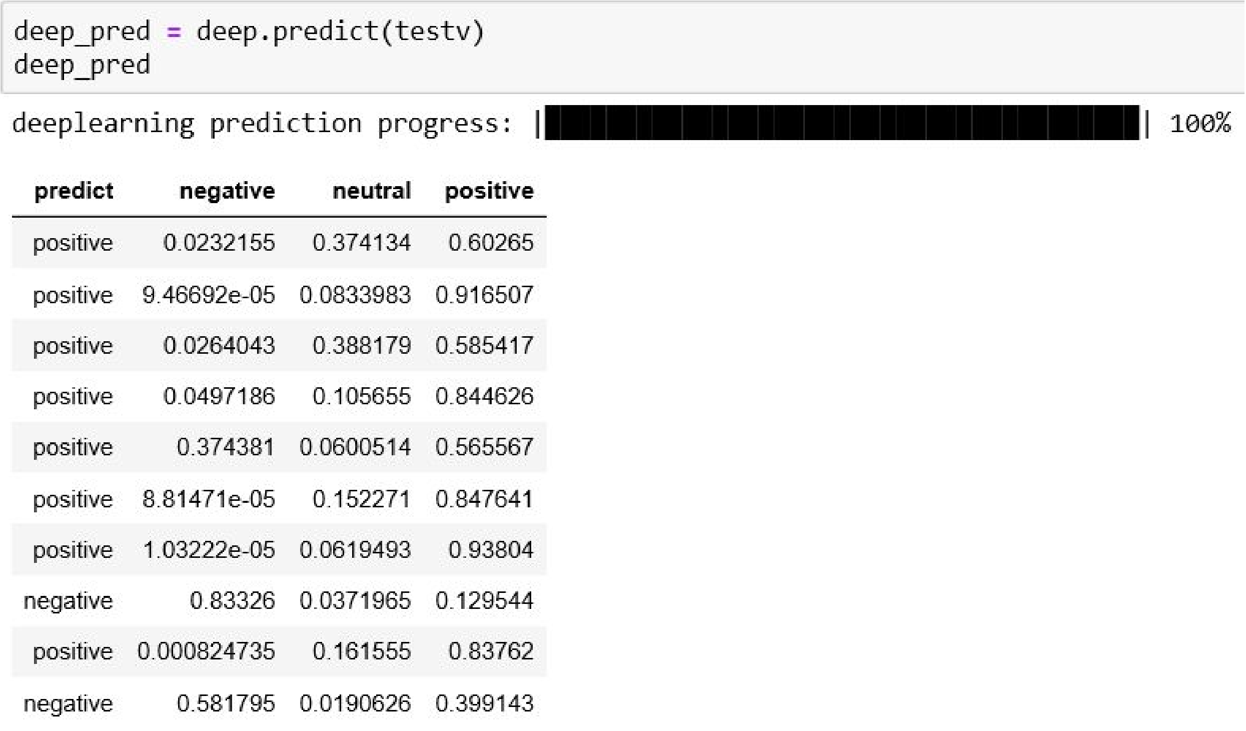
Scoring History
Training
Validation
classification_error
0.54
0.52
0.50
0.48
0.46
0.44
0.42
20
40
60
80
100
epochs



```
deep.varimp_plot()
```



As you can see that the model is underfitting and all the features have significant importance.

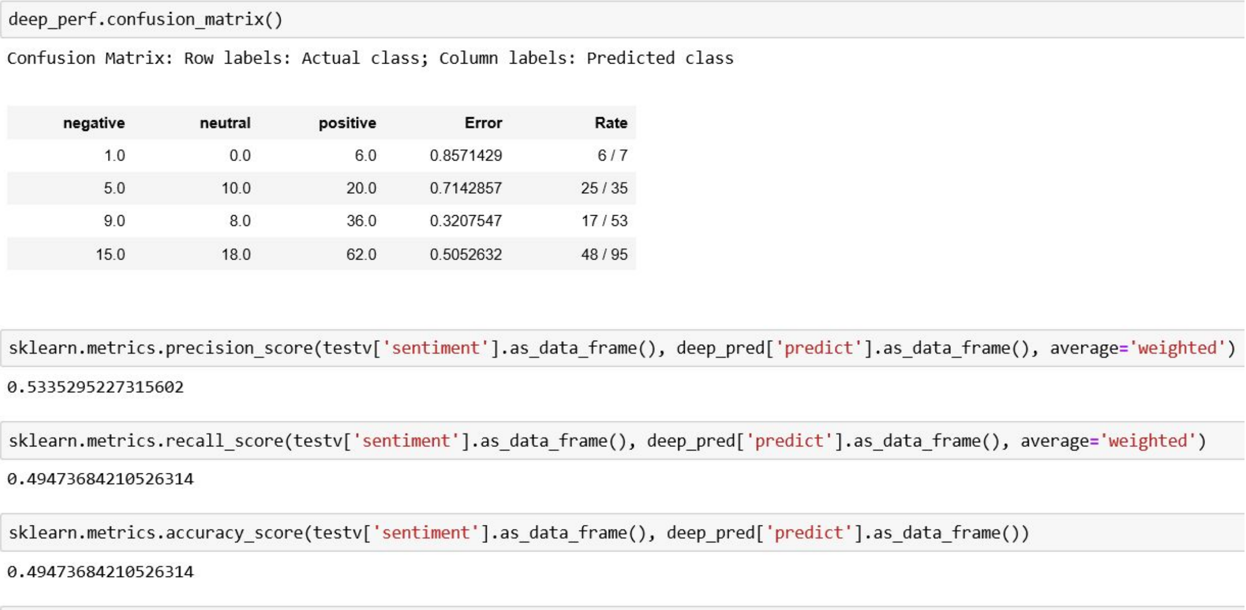


```
deep_pred = deep.predict(testv)
deep_pred
```

```
deeplearning prediction progress: |████████████████████████████████████████████████████████████████| 100%
```

predict	negative	neutral	positive
positive	0.0232155	0.374134	0.60265
positive	9.46692e-05	0.0833983	0.916507
positive	0.0264043	0.388179	0.585417
positive	0.0497186	0.105655	0.844626
positive	0.374381	0.0600514	0.565567
positive	8.81471e-05	0.152271	0.847641
positive	1.03222e-05	0.0619493	0.93804
negative	0.83326	0.0371965	0.129544
positive	0.000824735	0.161555	0.83762
negative	0.581795	0.0190626	0.399143

The above figure is the predicted sentiments which are used to determine accuracy.



```
deep_perf.confusion_matrix()
```

```
Confusion Matrix: Row labels: Actual class; Column labels: Predicted class
```

negative	neutral	positive	Error	Rate
1.0	0.0	6.0	0.8571429	6 / 7
5.0	10.0	20.0	0.7142857	25 / 35
9.0	8.0	36.0	0.3207547	17 / 53
15.0	18.0	62.0	0.5052632	48 / 95

```
sklearn.metrics.precision_score(testv['sentiment'].as_data_frame(), deep_pred['predict'].as_data_frame(), average='weighted')
```

```
0.5335295227315602
```

```
sklearn.metrics.recall_score(testv['sentiment'].as_data_frame(), deep_pred['predict'].as_data_frame(), average='weighted')
```

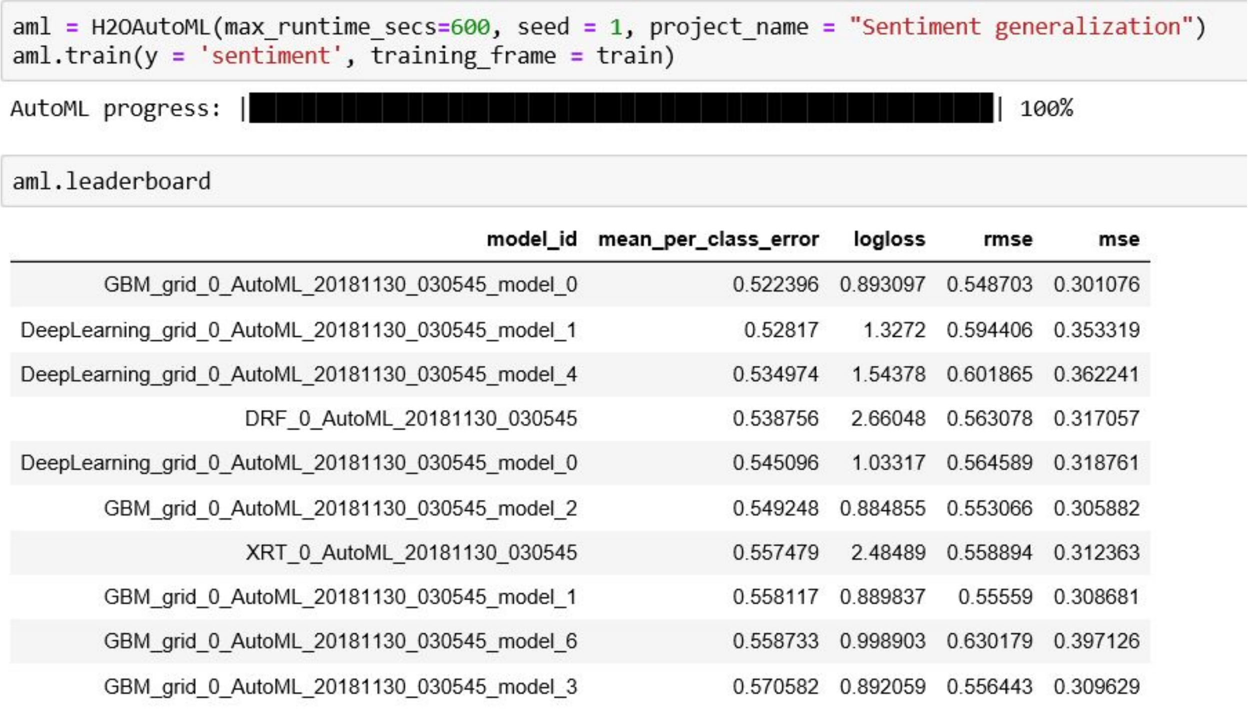
```
0.49473684210526314
```

```
sklearn.metrics.accuracy_score(testv['sentiment'].as_data_frame(), deep_pred['predict'].as_data_frame())
```

```
0.49473684210526314
```

H2O AutoML

Lets not stick to just MLP and use all possible models using AutoML.



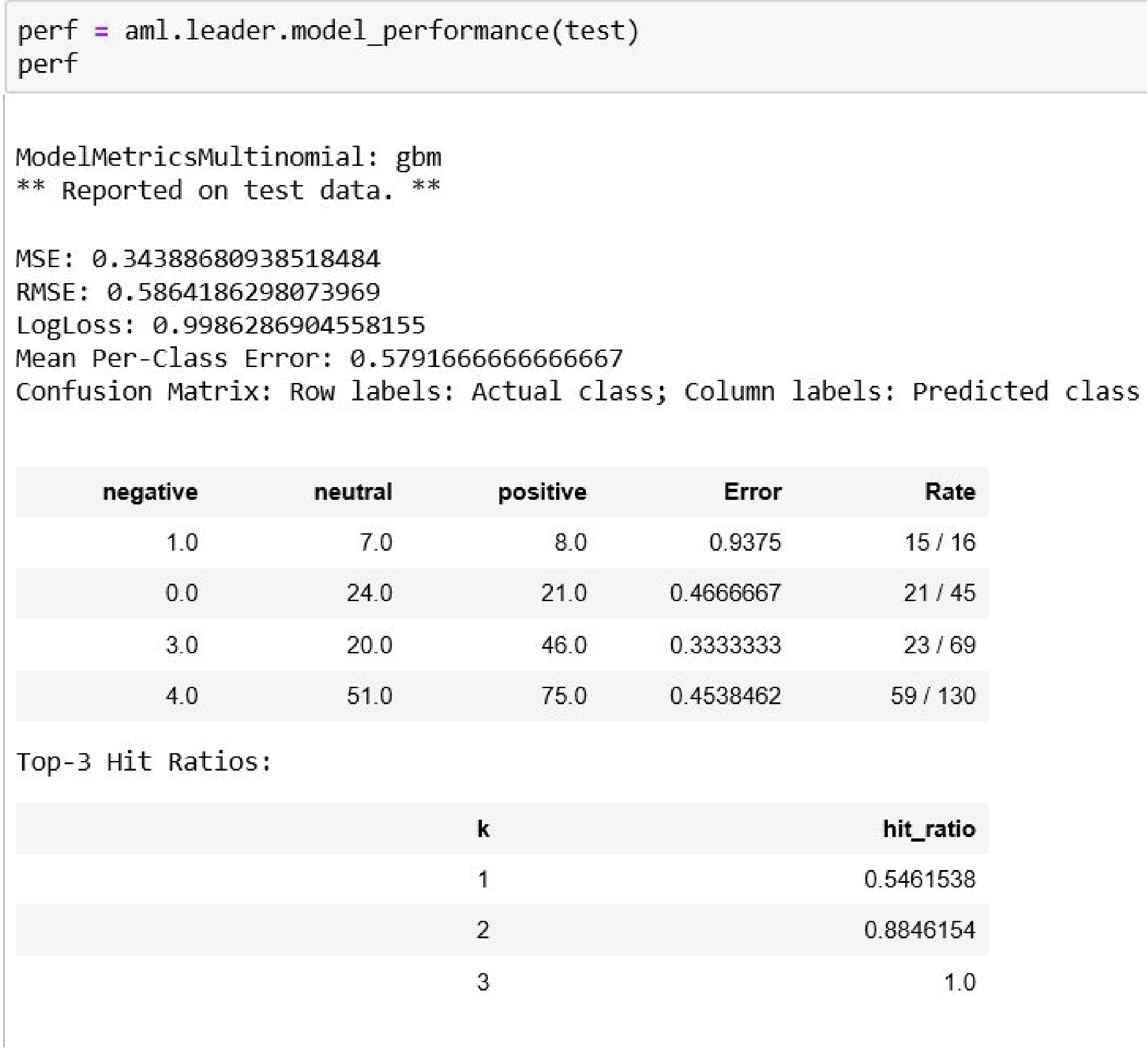
```
aml = H2OAutoML(max_runtime_secs=600, seed = 1, project_name = "Sentiment generalization")
aml.train(y = 'sentiment', training_frame = train)
```

```
AutoML progress: |███████████████████████████████████████████████████████████████| 100%
```

```
aml.leaderboard
```

model_id	mean_per_class_error	logloss	rmse	mse
GBM_grid_0_AutoML_20181130_030545_model_0	0.522396	0.893097	0.548703	0.301076
DeepLearning_grid_0_AutoML_20181130_030545_model_1	0.52817	1.3272	0.594406	0.353319
DeepLearning_grid_0_AutoML_20181130_030545_model_4	0.534974	1.54378	0.601865	0.362241
DRF_0_AutoML_20181130_030545	0.538756	2.66048	0.563078	0.317057
DeepLearning_grid_0_AutoML_20181130_030545_model_0	0.545096	1.03317	0.564589	0.318761
GBM_grid_0_AutoML_20181130_030545_model_2	0.549248	0.884855	0.553066	0.305882
XRT_0_AutoML_20181130_030545	0.557479	2.48489	0.558894	0.312363
GBM_grid_0_AutoML_20181130_030545_model_1	0.558117	0.889837	0.55559	0.308681
GBM_grid_0_AutoML_20181130_030545_model_6	0.558733	0.998903	0.630179	0.397126
GBM_grid_0_AutoML_20181130_030545_model_3	0.570582	0.892059	0.556443	0.309629

GBM model is at the top of the leaderboard. Lets evaluate the model performance.



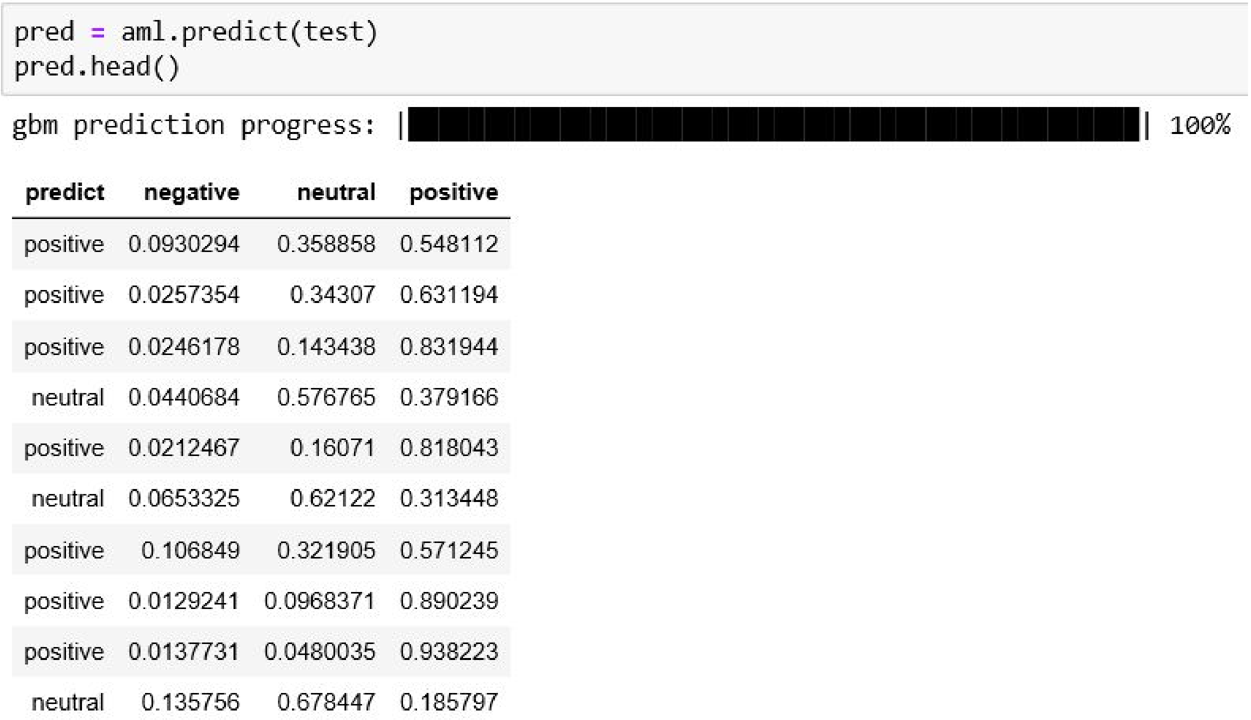
```
perf = aml.leader.model_performance(test)
perf
```

```
ModelMetricsMultinomial: gbm
** Reported on test data. **

MSE: 0.34388680938518484
RMSE: 0.5864186298073969
LogLoss: 0.9986286904558155
Mean Per-Class Error: 0.5791666666666667
Confusion Matrix: Row labels: Actual class; Column labels: Predicted class
```

negative	neutral	positive	Error	Rate
1.0	7.0	8.0	0.9375	15 / 16
0.0	24.0	21.0	0.4666667	21 / 45
3.0	20.0	46.0	0.3333333	23 / 69
4.0	51.0	75.0	0.4538462	59 / 130

```
Top-3 Hit Ratios:
```

k	hit_ratio
1	0.5461538
2	0.8846154
3	1.0

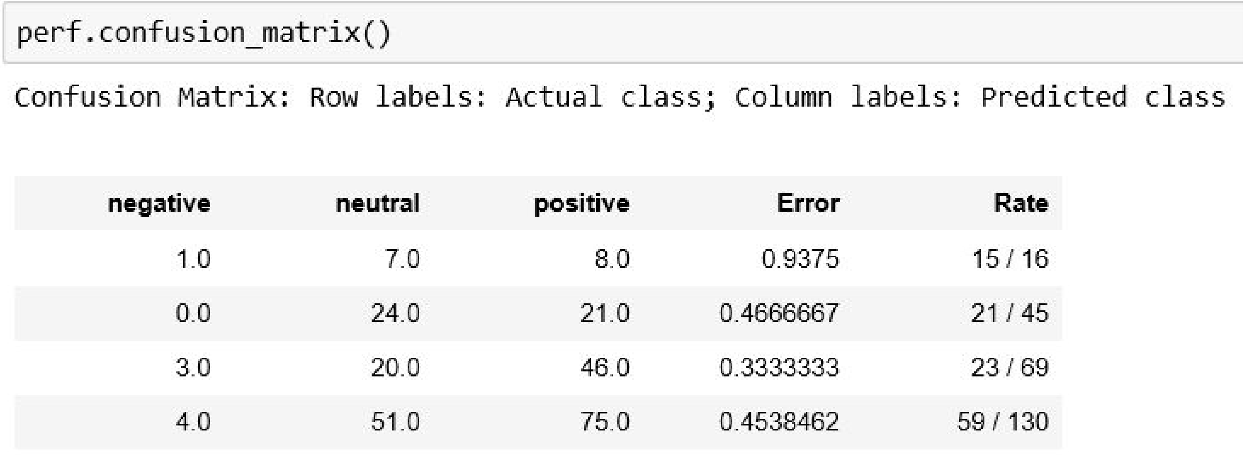


```
pred = aml.predict(test)
pred.head()
```

```
gbm prediction progress: |████████████████████████████████████████████████████| 100%
```

predict	negative	neutral	positive
positive	0.0930294	0.358858	0.548112
positive	0.0257354	0.34307	0.631194
positive	0.0246178	0.143438	0.831944
neutral	0.0440684	0.576765	0.379166
positive	0.0212467	0.16071	0.818043
neutral	0.0653325	0.62122	0.313448
positive	0.106849	0.321905	0.571245
positive	0.0129241	0.0968371	0.890239
positive	0.0137731	0.0480035	0.938223
neutral	0.135756	0.678447	0.185797

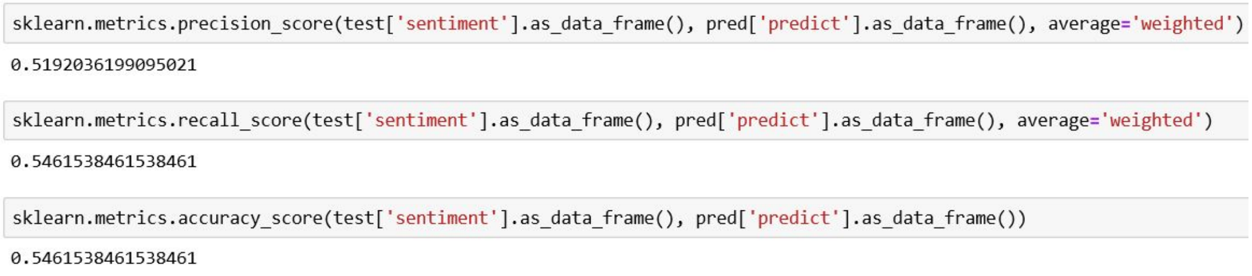
Above is the predicted data which is used to calculate the accuracy.



```
perf.confusion_matrix()
```

```
Confusion Matrix: Row labels: Actual class; Column labels: Predicted class
```

negative	neutral	positive	Error	Rate
1.0	7.0	8.0	0.9375	15 / 16
0.0	24.0	21.0	0.4666667	21 / 45
3.0	20.0	46.0	0.3333333	23 / 69
4.0	51.0	75.0	0.4538462	59 / 130



```
sklearn.metrics.precision_score(test['sentiment'].as_data_frame(), pred['predict'].as_data_frame(), average='weighted')
```

```
0.5192036199095021
```

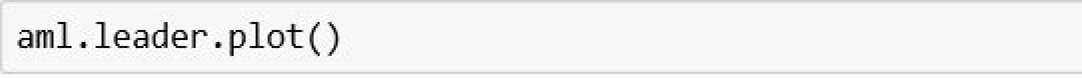
```
sklearn.metrics.recall_score(test['sentiment'].as_data_frame(), pred['predict'].as_data_frame(), average='weighted')
```

```
0.5461538461538461
```

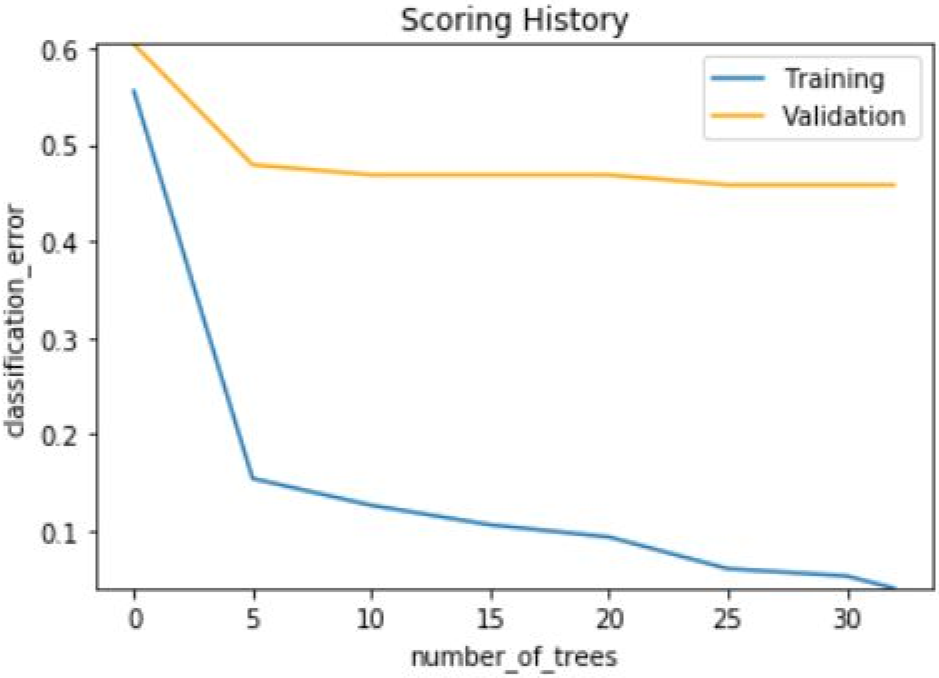
```
sklearn.metrics.accuracy_score(test['sentiment'].as_data_frame(), pred['predict'].as_data_frame())
```

```
0.5461538461538461
```

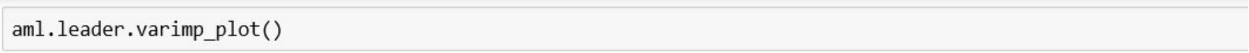
The accuracy can be seen as 54% only which is very less. The model plots are as follows.



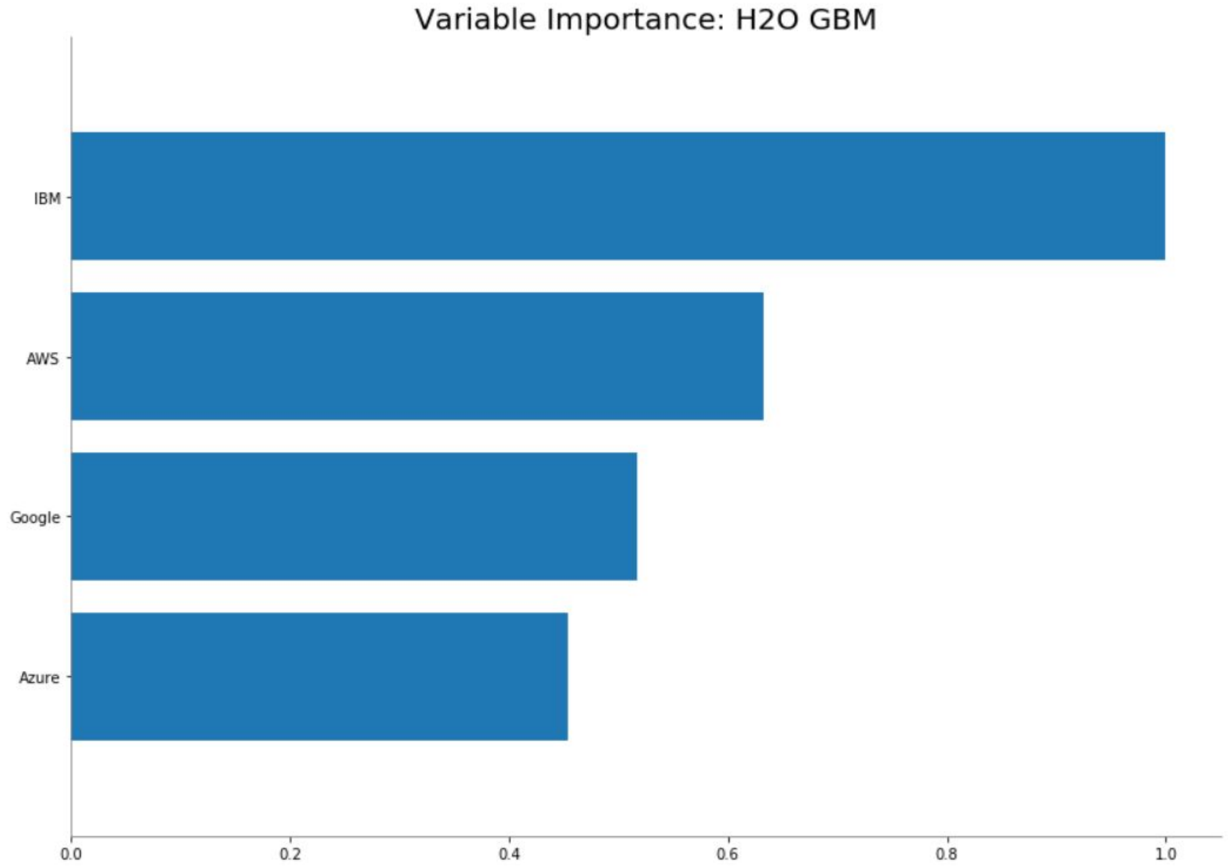
```
aml.leader.plot()
```



The plot shows that even this model is underfitting.

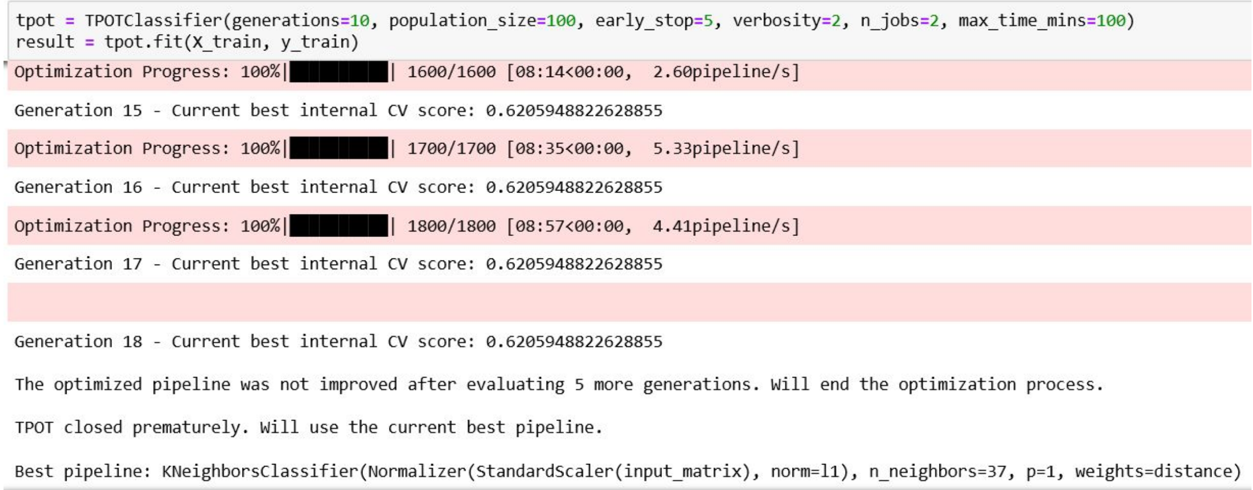


```
aml.leader.varimp_plot()
```



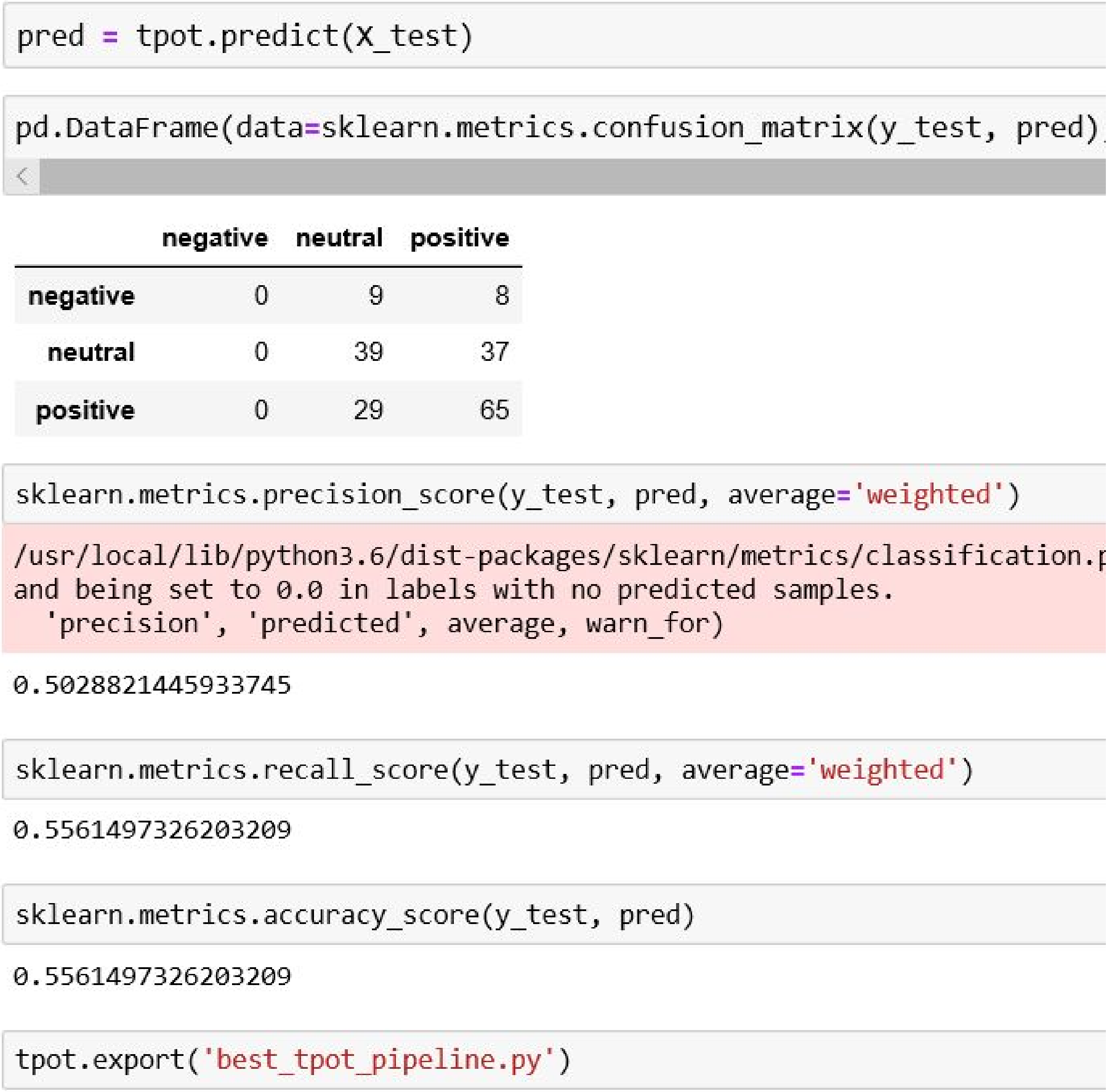
TPOT

Its a genetic algorithm based package which does AutoML.



```
tpot = TPOTClassifier(generations=10, population_size=100, early_stop=5, verbosity=2, n_jobs=2, max_time_mins=100)
result = tpot.fit(X_train, y_train)
Optimization Progress: 100%|██████████| 1600/1600 [08:14<00:00,  2.60pipeline/s]

Generation 15 - Current best internal CV score: 0.6205948822628855

Optimization Progress: 100%|██████████| 1700/1700 [08:35<00:00,  5.33pipeline/s]

Generation 16 - Current best internal CV score: 0.6205948822628855

Optimization Progress: 100%|██████████| 1800/1800 [08:57<00:00,  4.41pipeline/s]

Generation 17 - Current best internal CV score: 0.6205948822628855


Generation 18 - Current best internal CV score: 0.6205948822628855

The optimized pipeline was not improved after evaluating 5 more generations. Will end the optimization process.

TPOT closed prematurely. Will use the current best pipeline.

Best pipeline: KNeighborsClassifier(Normalizer(StandardScaler(input_matrix), norm=l1), n_neighbors=37, p=1, weights=distance)
```

The confusion matrix and accuracy scores are below. The best generation of TPOT pipeline had internal Cross Validation score of 0.62 which is good.



```
pred = tpot.predict(X_test)
```

```
pd.DataFrame(data=sklearn.metrics.confusion_matrix(y_test, pred)
```

	negative	neutral	positive
negative	0	9	8
neutral	0	39	37
positive	0	29	65

```
sklearn.metrics.precision_score(y_test, pred, average='weighted')
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/metrics/classification.
and being set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
```

```
0.5028821445933745
```

```
sklearn.metrics.recall_score(y_test, pred, average='weighted')
```

```
0.5561497326203209
```

```
sklearn.metrics.accuracy_score(y_test, pred)
```

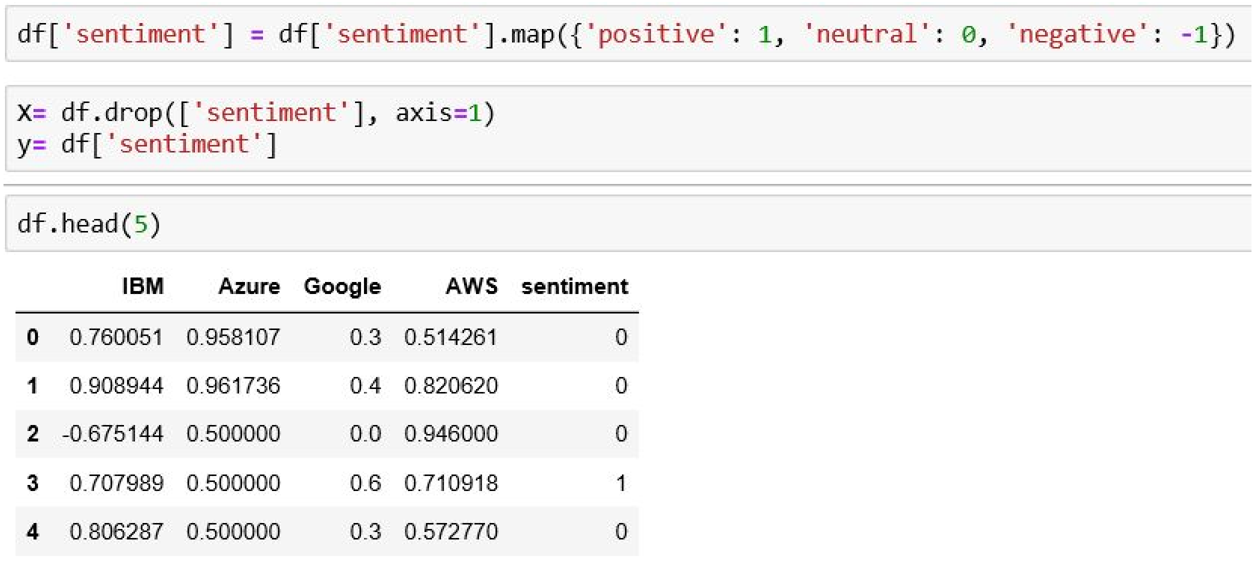
```
0.5561497326203209
```

```
tpot.export('best_tpot_pipeline.py')
```

So the accuracy is 55.6% which is better than H2O.

Auto-Sklearn

There were lot of problems while installing Auto-Sklearn. It is compatible with only Linux OS. Running it on Google Colab was also not possible due to installation errors. Even using deep learning docker images on GCP Ubuntu instance was creating problems because Auto-Sklearn requires specific versions of python and ubuntu packages. Somehow we were able to create a GCP instance with vanilla ubuntu and installed it with lot of modifications to the standard installation instructions on its website.



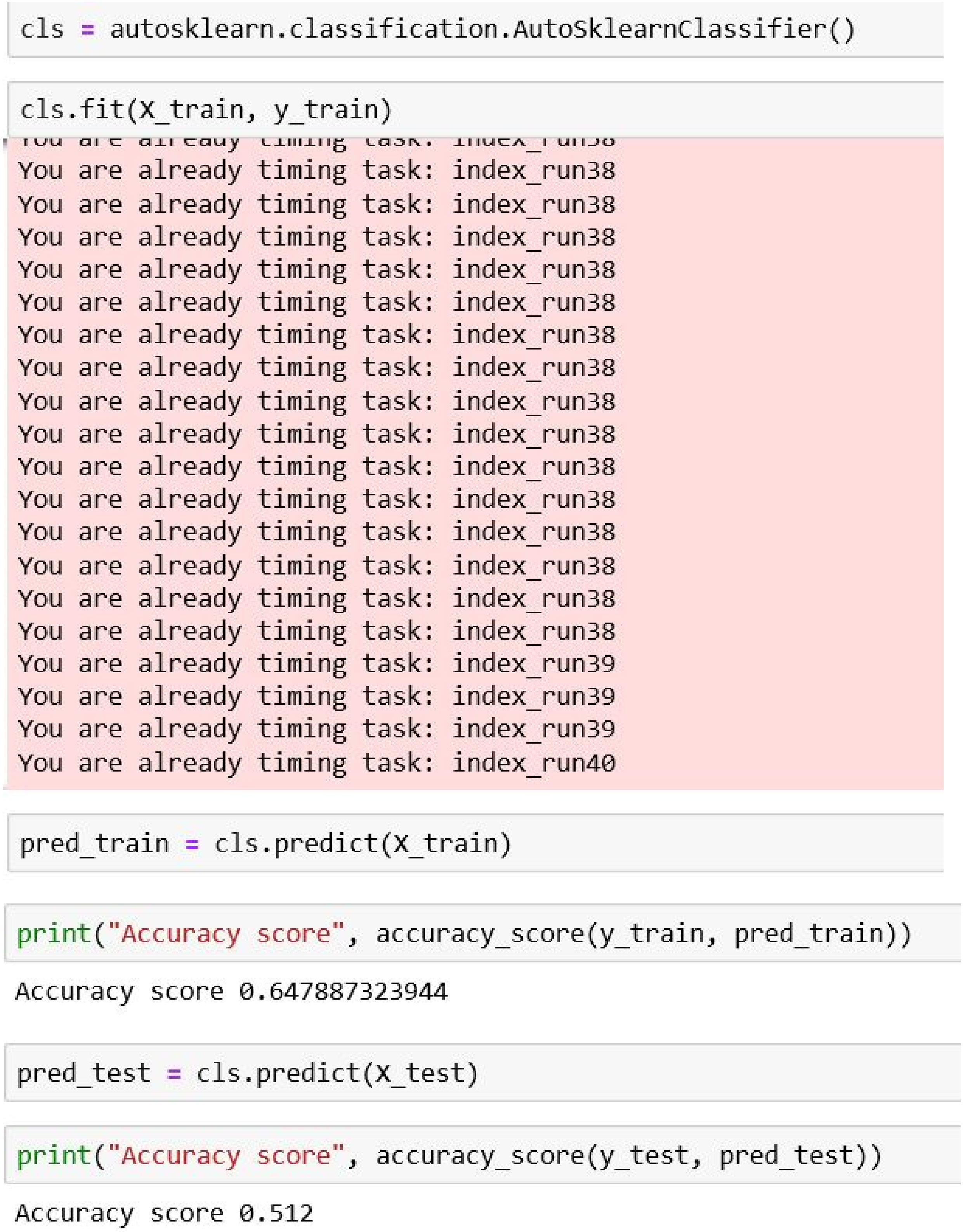
```
df['sentiment'] = df['sentiment'].map({'positive': 1, 'neutral': 0, 'negative': -1})
```

```
X= df.drop(['sentiment'], axis=1)
y= df['sentiment']
```

```
df.head(5)
```

	IBM	Azure	Google	AWS	sentiment
0	0.760051	0.958107	0.3	0.514261	0
1	0.908944	0.961736	0.4	0.820620	0
2	-0.675144	0.500000	0.0	0.946000	0
3	0.707989	0.500000	0.6	0.710918	1
4	0.806287	0.500000	0.3	0.572770	0

As you can see that the target variable is mapped to integer values. When we tried using the labels as categories, the model gave error of converting str to float. So we thought that the best way to keep it categorical would be by keeping the values as -1 for negative, 0 for neutral and 1 for positive.



```
cls = autosklearn.classification.AutoSklearnClassifier()
```

```
cls.fit(X_train, y_train)
```

```
You are already timing task: index_run38
You are already timing task: index_run38
You are already timing task: index_run38
You are already timing task: index_run38
You are already timing task: index_run38
You are already timing task: index_run38
You are already timing task: index_run38
You are already timing task: index_run38
You are already timing task: index_run38
You are already timing task: index_run38
You are already timing task: index_run38
You are already timing task: index_run38
You are already timing task: index_run38
You are already timing task: index_run38
You are already timing task: index_run38
You are already timing task: index_run38
You are already timing task: index_run39
You are already timing task: index_run39
You are already timing task: index_run39
You are already timing task: index_run40
```

```
pred_train = cls.predict(X_train)
```

```
print("Accuracy score", accuracy_score(y_train, pred_train))
```

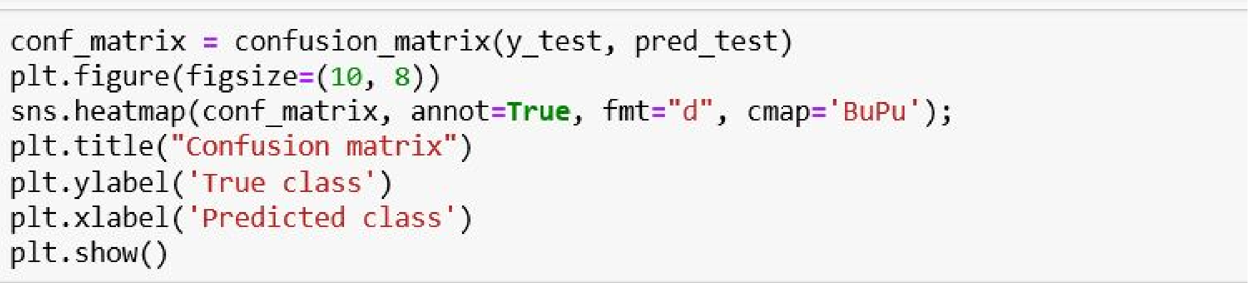
```
Accuracy score 0.647887323944
```

```
pred_test = cls.predict(X_test)
```

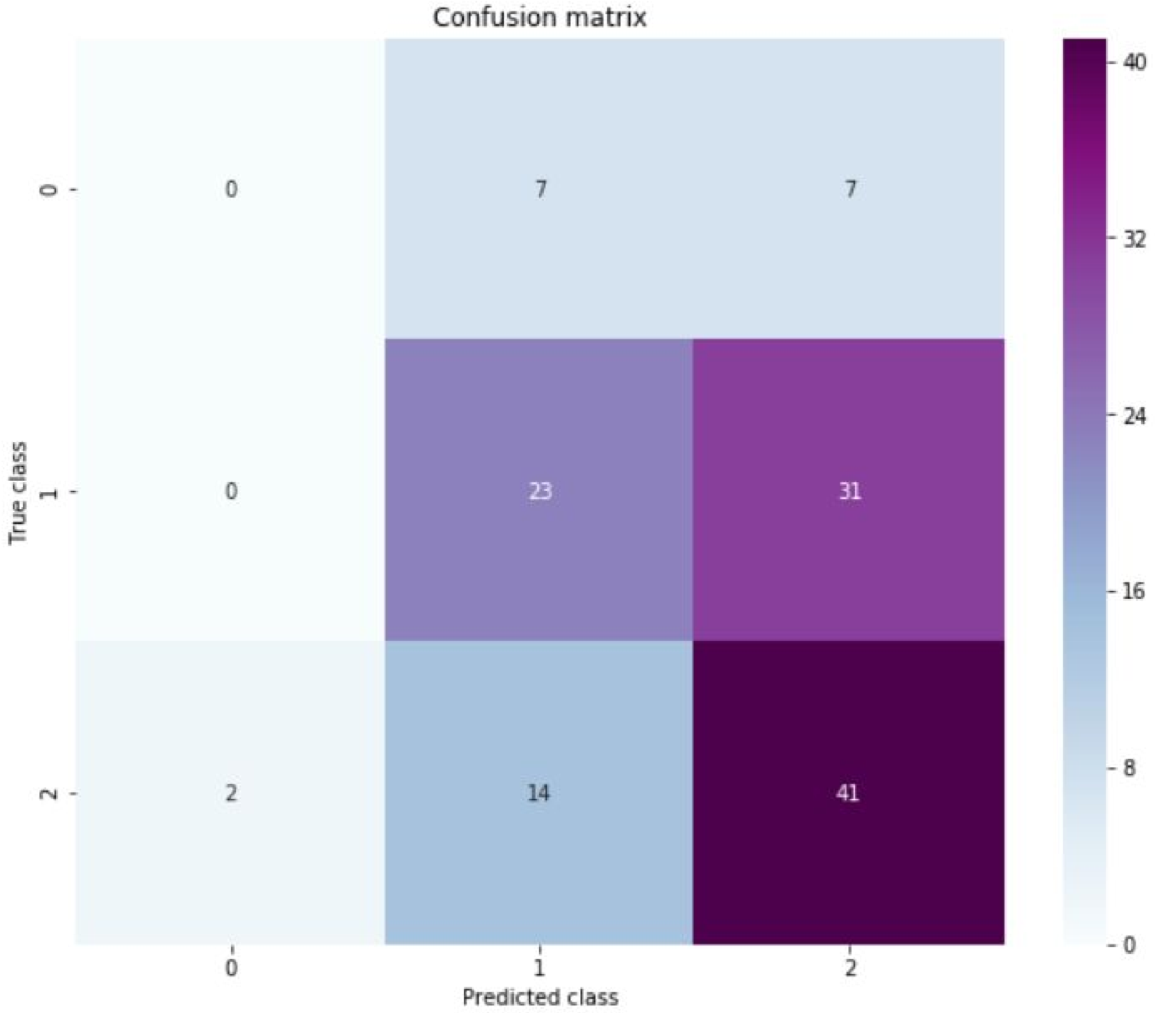
```
print("Accuracy score", accuracy_score(y_test, pred_test))
```

```
Accuracy score 0.512
```

So the training accuracy is 0.64 and test accuracy is 0.512



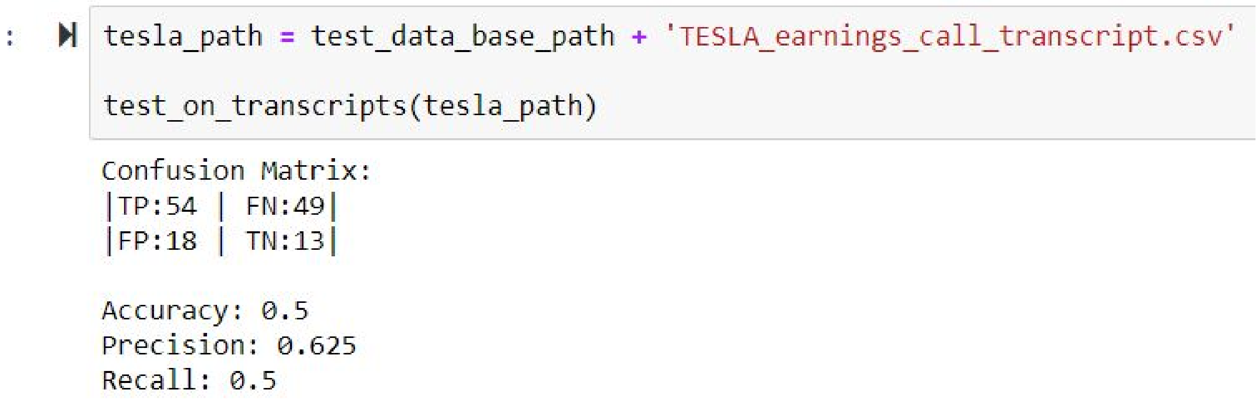
```
conf_matrix = confusion_matrix(y_test, pred_test)
plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap='BuPu');
plt.title("Confusion matrix")
plt.ylabel('True class')
plt.xlabel('Predicted class')
plt.show()
```



The above figure is the heat map of confusion matrix

Final Model Testing

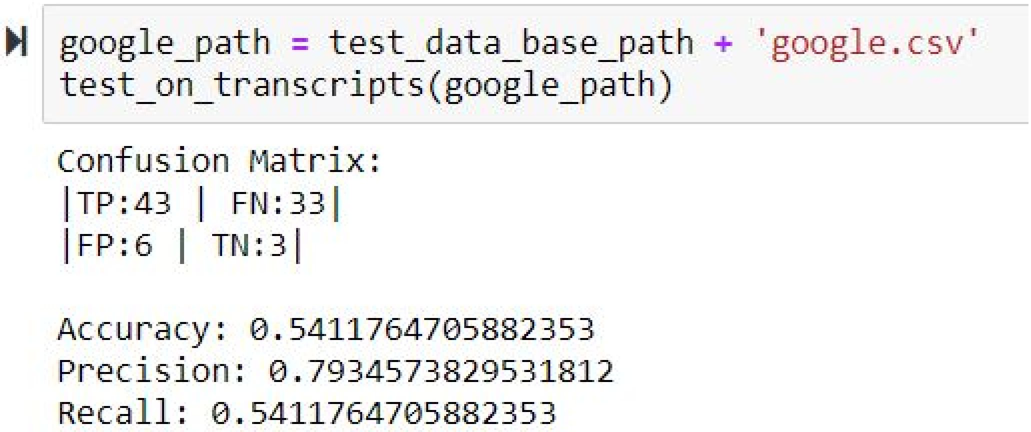
Tesla



```
tesla_path = test_data_base_path + 'TESLA_earnings_call_transcript.csv'

test_on_transcripts(tesla_path)
```

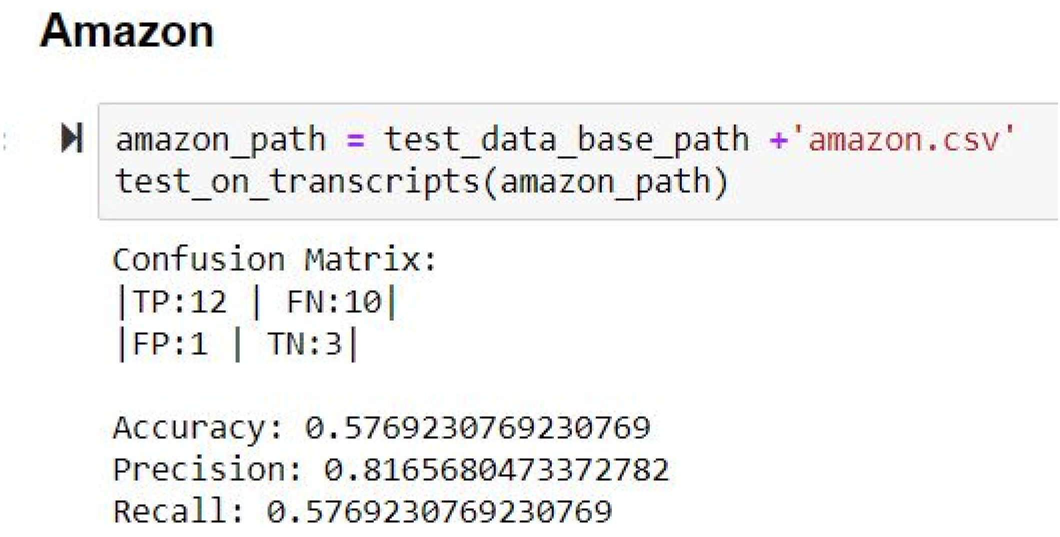
```
Confusion Matrix:
|TP:54 | FN:49|
|FP:18 | TN:13|

Accuracy: 0.5
Precision: 0.625
Recall: 0.5
```

Google



```
google_path = test_data_base_path + 'google.csv'
test_on_transcripts(google_path)
```

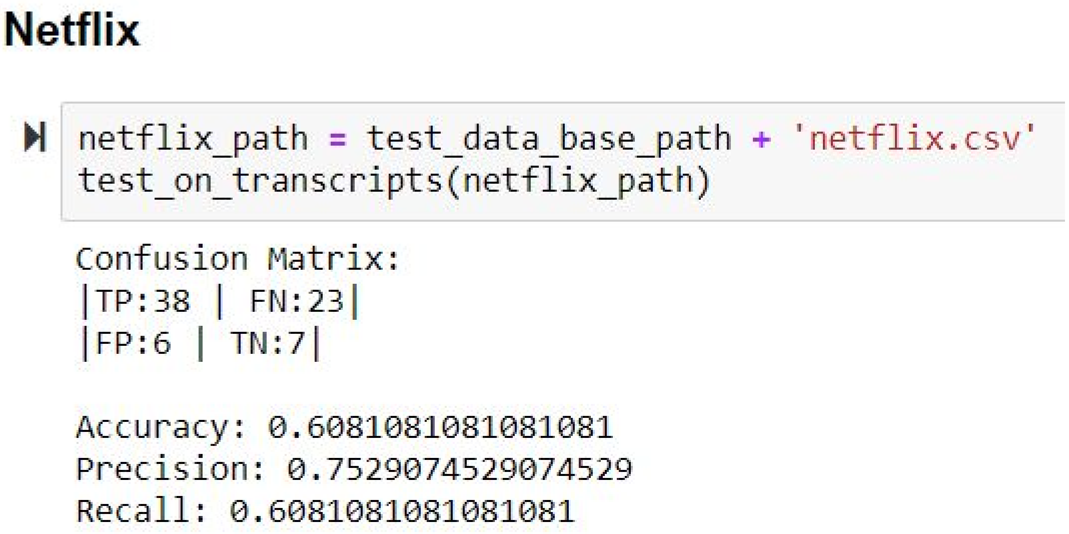
```
Confusion Matrix:
|TP:43 | FN:33|
|FP:6 | TN:3|

Accuracy: 0.5411764705882353
Precision: 0.7934573829531812
Recall: 0.5411764705882353
```



Amazon

```
amazon_path = test_data_base_path +'amazon.csv'
test_on_transcripts(amazon_path)
```

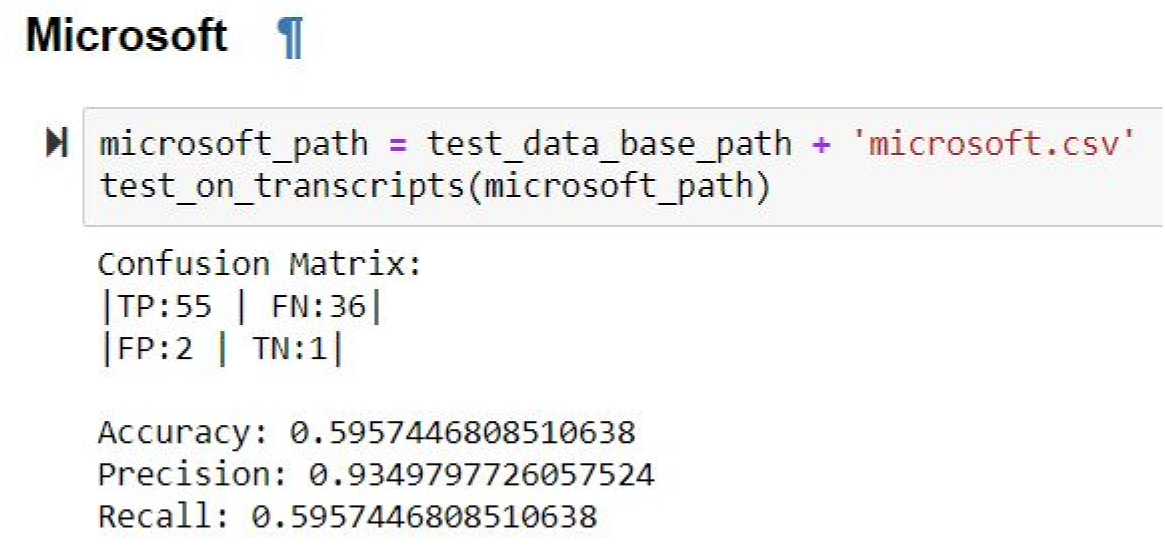
```
Confusion Matrix:
|TP:12 | FN:10|
|FP:1 | TN:3|

Accuracy: 0.5769230769230769
Precision: 0.8165680473372782
Recall: 0.5769230769230769
```



Netflix

```
netflix_path = test_data_base_path + 'netflix.csv'
test_on_transcripts(netflix_path)
```

```
Confusion Matrix:
|TP:38 | FN:23|
|FP:6 | TN:7|

Accuracy: 0.6081081081081081
Precision: 0.7529074529074529
Recall: 0.6081081081081081
```



Microsoft ¶

```
microsoft_path = test_data_base_path + 'microsoft.csv'
test_on_transcripts(microsoft_path)
```

```
Confusion Matrix:
|TP:55 | FN:36|
|FP:2 | TN:1|

Accuracy: 0.5957446808510638
Precision: 0.9349797726057524
Recall: 0.5957446808510638
```

References

1. https://github.com/fchollet/deep-learning-with-pythonnotebooks
2. https://cloud.google.com/natural-language/docs/sentiment-tutorial
3. https://www.ibm.com/watson/services/natural-language-understanding/
4. https://azure.microsoft.com/en-us/services/cognitive-services/text-analytics/
5. https://docs.aws.amazon.com/comprehend/latest/dg/API_Reference.html
6. http://ai.stanford.edu/~amaas/data/sentiment/