Uncovering Sentiments using EDGAR Datasets

Group 3 Report

Experiment 1

- Web scraping [1]:
  - Queried the facebook earnings call website to obtain HTML
  - Used Beautiful soup python package to parse HTML content
  - Obtained element information of page

```
[bs4.element.Doctype,
 bs4.element.NavigableString,
 bs4.element.Tag,
 bs4.element.NavigableString]
```

  - Obtained tags
  - Obtained paragraphs

- Creating json file in specified format [2]:
  - Created a list of numbers for each paragraph
  - Created json from this dictionary of numbers and parahs
  - Labelled the sentiments of paragraphs manually

- Consolidated json files into single dataset [3]:
  - Collected all 12 json files
  - Ran a loop to create dictionary
  - Converted dictionary to dataframe

- Preprocessing text data [4]:
  - Implemented NLTK package to remove:
    - Stop words - the, is, are
    - Stemmer to remove inflicted and similar words to roots - like, likes, respons, responsive
    - Non-alphabetic characters - !, ?
    - Changed to lower case
  - Tokenized

- Bag of Words - CountVectorizer [5] :
    - Sklearn’s CountVectorizer package to create a Bag of Words
    - Includes maximum 2000 features
    - Minimum number of occurance of a word to be included in Bag is 3
    - Maximum frequency is 0.6
    - Stop words from English language
    - Total number of words contained = 1649
    - Created a Logistic Regression model

- Balancing uneven distribution of classes [a, b, c, d, e, f, g]:
    - Oversampled: RandomOverSampler - Increased the proportion of negative and positive classes to match that of positive class on the training dataset
    - Downsampled: RandomUnderSampler - Decreased the proportion of neutral class to match that of positive and negative classes on the training dataset
    - NearMiss1 - Downsampling with 1 nearest neighbour [ref]
    - NearMiss2 - Downsampling with 2 nearest neighbour
    - NearMiss3 - Downsampling with 3 nearest neighbour
    - SMOTE - Synthetic minority over sampling technique - increase negative and positive class for nearest neighbours to match neutral class
    - Results -

Model	f1-score	accuracy
ROS	49	62
Original	52	65
RUS	52	58
Smote	48	59
NearMiss1	45	48
NearMiss2	42	44
NearMiss3	46	52

- Keras model [7]:
    - Input size of bag of words - 1649, F1 score = 0.48

- Grid search CV [a , b]:
    - batch_size = [10, 20, 40, 60, 80, 100] and epochs = [2,5,10, 20], **f1 score = 0.49, Best: 0.692949 using {'batch_size': 80, 'epochs': 5}**

- optimizer = ['SGD', 'RMSprop', 'Adagrad', 'Adadelta', 'Adam', 'Adamax', 'Nadam'], **f1 score = 0.52, Best: 0.687642 using {'optimizer': 'Adagrad'}**

- RNN:

1. Data Preprocessing

```
In [3]: data = pd.read_csv("cleanedfinancial_data.csv")
        data.head()
```

Out[3]:

	sentiment	text
0	neutral	Good day, ladies and gentlemen, and welcome to...
1	negative	I'm not sure. I think Model T was a little bit...
2	negative	Well, we need to bring the Shanghai factory on...
3	neutral	So it's - it is eligible for that. But it soun...
4	positive	The demand for - the demand for Model 3 is ins...

```
In [0]: data_inputs =  data["text"].get_values()
        # Convert sentiments into 0,1,2
        sent = {'positive': 1,'negative': 0,'neutral': 2}
        data.sentiment = [sent[item] for item in data.sentiment]
```

```
In [0]: data_labels = data.sentiment
```

```
In [8]: tokenizer = Tokenizer(nb_words=2000)
        tokenizer.fit_on_texts(data_inputs)
        sequences = tokenizer.texts_to_sequences(data_inputs)

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

        data = pad_sequences(sequences, maxlen=1000)

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

```
/usr/local/lib/python3.6/dist-packages/keras_preprocessing/text.py:178: UserWarning: The `nb_words` argume
nt in `Tokenizer` has been renamed `num_words`.
  warnings.warn('The `nb_words` argument in `Tokenizer` '
Found 6376 unique tokens.
Shape of data tensor: (1649, 1000)
Shape of label tensor: (1649, 3)
```

2.1 Modeling - 1st Try

```python
print('Build model...')
model = Sequential()

model.add(Embedding(nb_words,
                    embedding_dims,
                    input_length=maxlen))
model.add(Dropout(0.2))

# model.add(LSTM(lstm_units))
# running on a GPU:
model.add(CuDNNLSTM(lstm_units))

# To stack multiple RNN layers, all RNN layers except the last one need
# to have "return_sequences=True".  An example of using two RNN layers:
#model.add(LSTM(lstm_units, return_sequences=True))
#model.add(LSTM(lstm_units))

model.add(Dense(3, activation='softmax'))

model.compile(optimizer='adadelta',
              loss='poisson',
              metrics=['accuracy'])

print(model.summary())
```

```
Build model...
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_3 (Embedding)      (None, 1000, 50)          500000    
_________________________________________________________________
dropout_3 (Dropout)          (None, 1000, 50)          0         
_________________________________________________________________
cu_dnnlstm_3 (CuDNNLSTM)     (None, 32)                10752     
_________________________________________________________________
dense_3 (Dense)              (None, 3)                 99        
=================================================================
Total params: 510,851
Trainable params: 510,851
Non-trainable params: 0
```

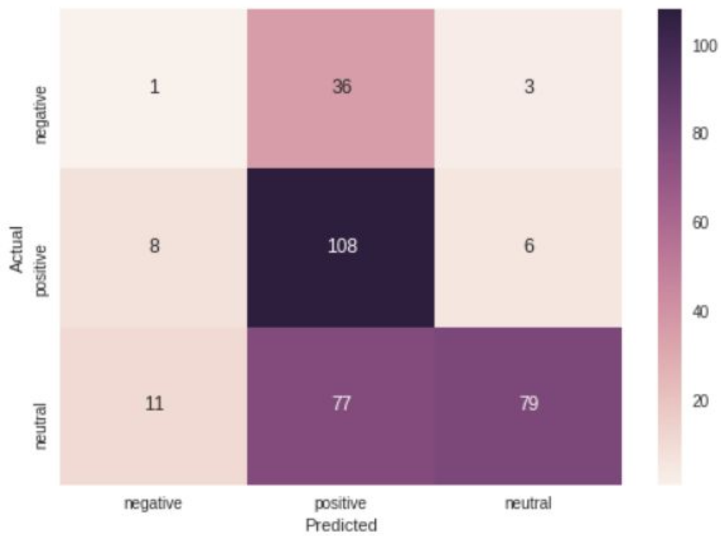
2.2. Accuracy Score and Confusion Matrix

```python
loss, acc = model.evaluate(x_val,y_val, verbose = 2, batch_size = 128)
print("Validation Loss: %.2f" % (loss))
print("Validation Accuracy: %.2f" % (acc*100))
```

```
Validation Loss: 0.62
Validation Accuracy: 59.88
```

```python
# Confusion matrix
cm = confusion_matrix(y_val.argmax(axis=1), y_pred.argmax(axis=1))
print(cm)

LABELS = ['negative', 'positive','neutral']
sns.heatmap(cm, annot=True, xticklabels=LABELS, yticklabels=LABELS, fmt='g')
xl = plt.xlabel("Predicted")
yl = plt.ylabel("Actual")
```

```
[[  1  36   3]
 [  8 108   6]
 [ 11  77  79]]
```



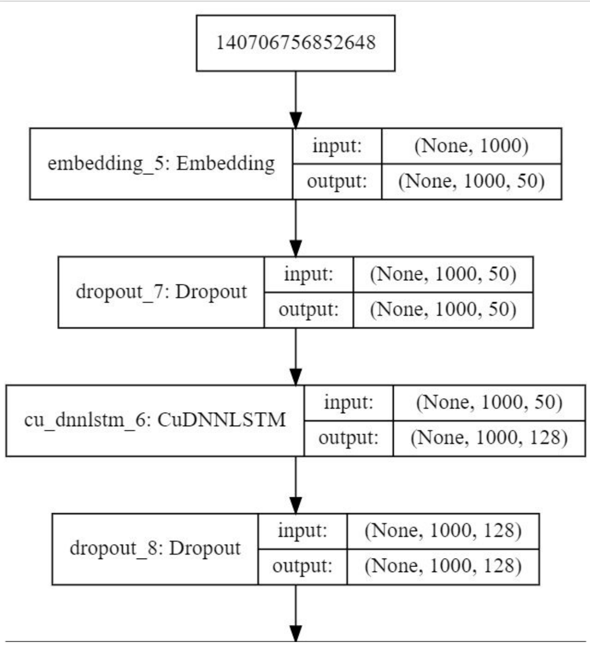
3.1 Modeling - 2nd Try

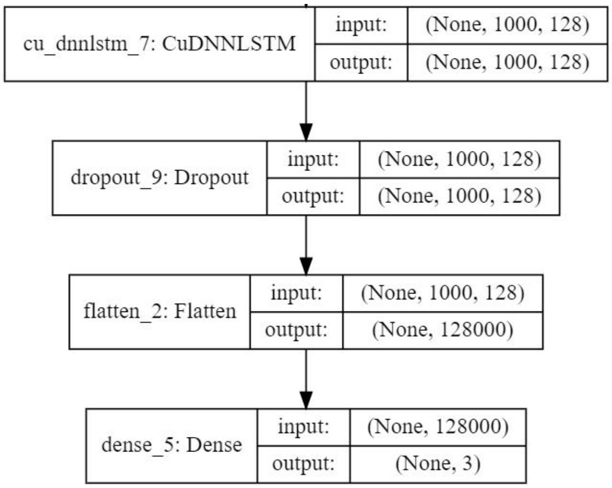
Since the first model is a bit easy thus did not result in a good performance for accuracy score, so I tried second modeling with one more LSTM layers and a Flatten layer, which shows the performance has been improved:

```python
print('Build 2nd model...')
model = Sequential()

model.add(Embedding(nb_words,
                    embedding_dims,
                    input_length=maxlen))
model.add(Dropout(0.2))
model.add(CuDNNLSTM(lstm_units,return_sequences=True))
model.add(Dropout(0.2))
model.add(CuDNNLSTM(lstm_units,return_sequences=True))
model.add(Dropout(0.3))
model.add(Flatten())

model.add(Dense(3, activation='softmax'))

model.compile(optimizer='adadelta',
              loss='poisson',
              metrics=['accuracy'])

print(model.summary())
```

```
Build 2nd model...
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_5 (Embedding)      (None, 1000, 50)          500000    
_________________________________________________________________
dropout_7 (Dropout)          (None, 1000, 50)          0         
_________________________________________________________________
cu_dnnlstm_6 (CuDNNLSTM)     (None, 1000, 128)         92160     
_________________________________________________________________
dropout_8 (Dropout)          (None, 1000, 128)         0         
_________________________________________________________________
cu_dnnlstm_7 (CuDNNLSTM)     (None, 1000, 128)         132096    
_________________________________________________________________
dropout_9 (Dropout)          (None, 1000, 128)         0         
_________________________________________________________________
flatten_2 (Flatten)          (None, 128000)            0         
_________________________________________________________________
dense_5 (Dense)              (None, 3)                 384003    
=================================================================
Total params: 1,108,259
Trainable params: 1,108,259
Non-trainable params: 0
```

```python
SVG(model_to_dot(model, show_shapes=True).create(prog='dot', format='svg'))
```



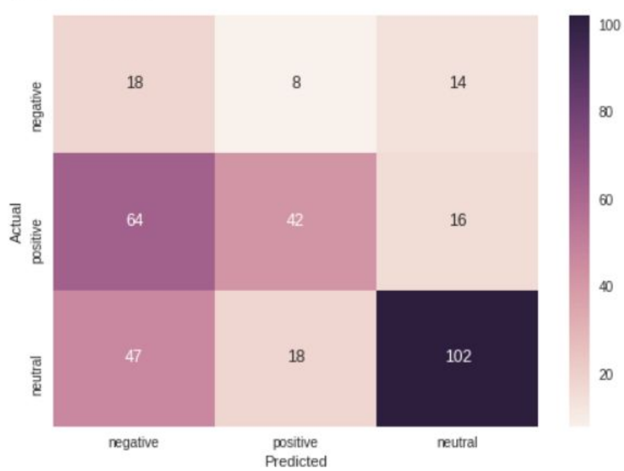


3.2 Accuracy Score and Confusion Matrix

```
Validation Loss: 0.61
Validation Accuracy: 64.44
```

```python
# Predicting the Test set results
y_pred = model.predict(x_val)
# cutoff 0.5
y_pred = (y_pred > 0.5)

y_pred = y_pred.astype(int)


# Confusion matrix
cm = confusion_matrix(y_val.argmax(axis=1), y_pred.argmax(axis=1))
print(cm)

LABELS = ['negative', 'positive','neutral']
sns.heatmap(cm, annot=True, xticklabels=LABELS, yticklabels=LABELS, fmt='g')
xl = plt.xlabel("Predicted")
yl = plt.ylabel("Actual")
```

```
[[ 18   8  14]
 [ 64  42  16]
 [ 47  18 102]]
```



4. RNN Best Accuracy: 64.44%

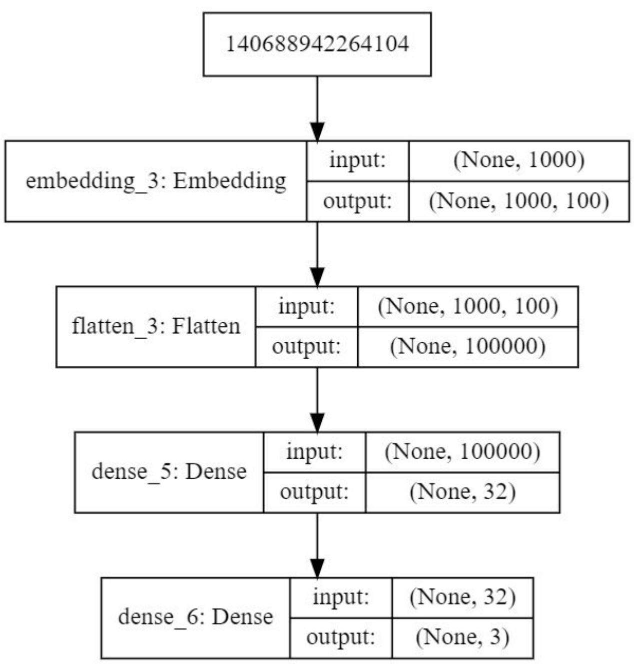
- GloVe

1. Modeling

```
model = Sequential()
model.add(Embedding(max_words, embedding_dim, weights = [embedding_matrix], trainable = False, input_lengt
h=maxlen))
model.add(Flatten())
model.add(Dense(32, activation='relu'))
model.add(Dense(3, activation='softmax'))
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_3 (Embedding)      (None, 1000, 100)         500000    
_________________________________________________________________
flatten_3 (Flatten)          (None, 100000)            0         
_________________________________________________________________
dense_5 (Dense)              (None, 32)                3200032   
_________________________________________________________________
dense_6 (Dense)              (None, 3)                 99        
=================================================================
Total params: 3,700,131
Trainable params: 3,200,131
Non-trainable params: 500,000
_________________________________________________________________
```

```
from IPython.display import SVG
from keras.utils.vis_utils import model_to_dot

SVG(model_to_dot(model, show_shapes=True).create(prog='dot', format='svg'))
```



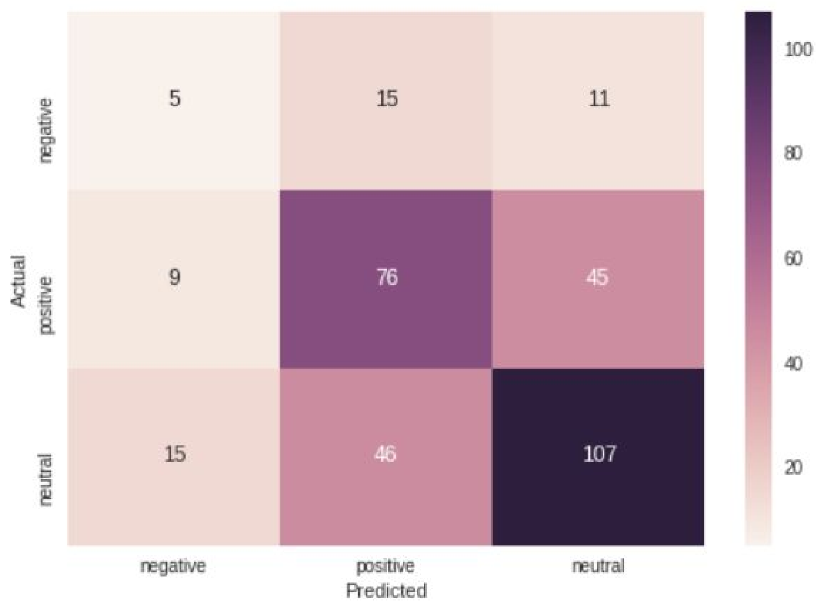
2. Accuracy Score and Confusion Matrix

```python
loss, acc = model.evaluate(x_val,y_val, verbose = 2, batch_size = batch_size)
print("Validation Loss: %.2f" % (loss))
print("Validation Accuracy: %.2f" % (acc*100))
```

```
Validation Loss: 1.10
Validation Accuracy: 59.27
```

```python
# Confusion matrix
cm = confusion_matrix(y_val.argmax(axis=1), y_pred.argmax(axis=1))
print(cm)

LABELS = ['negative', 'positive','neutral']
sns.heatmap(cm, annot=True, xticklabels=LABELS, yticklabels=LABELS, fmt='g')
xl = plt.xlabel("Predicted")
yl = plt.ylabel("Actual")
```

```
[[  5  15  11]
 [  9  76  45]
 [ 15  46 107]]
```



3. GloVe Best Accuracy: 59.27%

Experiment 2

- Bag of Words model
- Transfer learning [i , ii , iii , iv , v]:
    - Created a bag of words with max features of 2000 on the IMDB dataset (**25000**)
    - Created bag of words of financial dataset and predicted on the keras model
    - Created a bag of words with max features of 2000 on the IMDB dataset (**5000**)
    - Created a bag of words with max features of 2000 on the IMDB dataset (**88585 - all words**)
    - Max features - 5000

- Tests on GE [a , b]:
    - Trained on financial dataset with BoW keras model and tested on GE dataset
    - Learned on IMDB dataset, tested on whole GE dataset

- Observations:
    - Transfer learning gives very bad results - 0.16 f1 score be it on GE or Finance dataset
    - Adagrad optimizer gives best performance with batch size of 80 and 5 epochs
    - Balancing does not impact the results a lot, they are very similar to unbalanced results

- RNN:

1. Data Preprocessing for IMDB Dataset

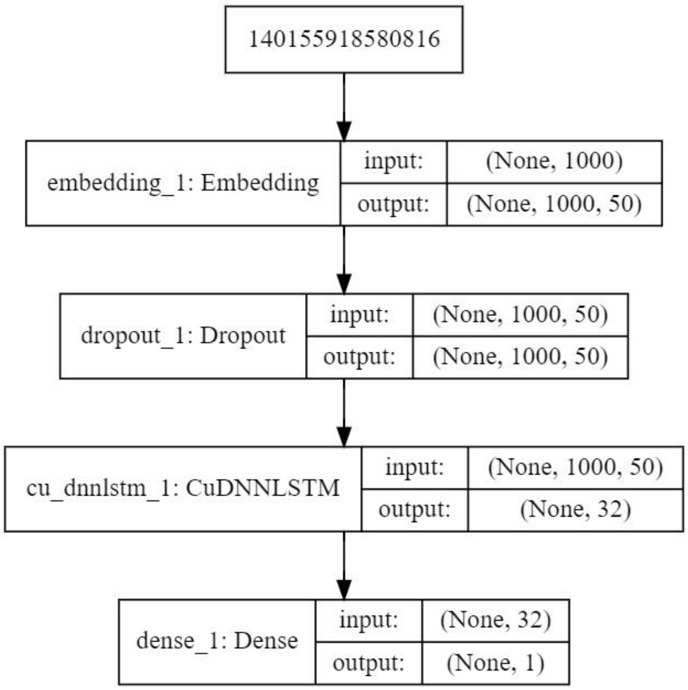
```python
# number of most-frequent words to use
nb_words = 10000
# cut texts after this number of words
maxlen = 1000

print('Loading data...')
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=nb_words)
print('x_train:', x_train.shape)
print('x_test:', x_test.shape)
print()

print('Pad sequences (samples x time)')
x_train = sequence.pad_sequences(x_train, maxlen=maxlen)
x_test = sequence.pad_sequences(x_test, maxlen=maxlen)
print('x_train shape:', x_train.shape)
print('x_test shape:', x_test.shape)
```

```
Loading data...
Downloading data from https://s3.amazonaws.com/text-datasets/imdb.npz
17465344/17464789 [==============================] - 3s 0us/step
x_train: (25000,)
x_test: (25000,)

Pad sequences (samples x time)
x_train shape: (25000, 1000)
x_test shape: (25000, 1000)
```

2. Modeling for IMDB Dataset

```
Build model...
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/op_def_library.
py:263: colocate_with (from tensorflow.python.framework.ops) is deprecated and will be removed in a future
version.
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:3445: c
alling dropout (from tensorflow.python.ops.nn_ops) with keep_prob is deprecated and will be removed in a f
uture version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob`.
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 1000, 50)          500000
_________________________________________________________________
dropout_1 (Dropout)          (None, 1000, 50)          0
_________________________________________________________________
cu_dnnlstm_1 (CuDNNLSTM)     (None, 32)                10752
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 33
=================================================================
Total params: 510,785
Trainable params: 510,785
Non-trainable params: 0
```

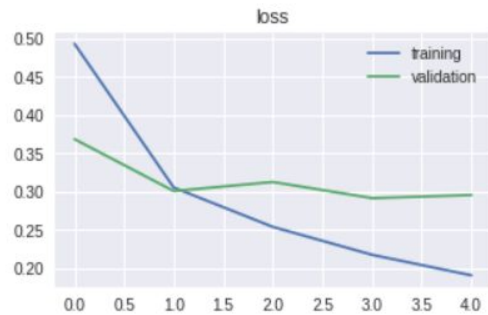
```
SVG(model_to_dot(model, show_shapes=True).create(prog='dot', format='svg'))
```



3. Training the Model

- Plot the data to see how the training progressed.

```
plt.figure(figsize=(5,3))
plt.plot(history.epoch,history.history['loss'], label='training')
plt.plot(history.epoch,history.history['val_loss'], label='validation')
plt.title('loss')
plt.legend(loc='best')

plt.figure(figsize=(5,3))
plt.plot(history.epoch,history.history['acc'], label='training')
plt.plot(history.epoch,history.history['val_acc'], label='validation')
plt.title('accuracy')
plt.legend(loc='best');
```



A big gap between training and validation accuracies would suggest overfitting.

4. Accuracy Score and Confusion Matrix

```
score, acc = model.evaluate(x_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (acc*100))
print("Test score: %.2f%%" % (score*100))
```

```
Accuracy: 87.40%
Test score: 30.93%
```

```
sns.set(font_scale=1.4)
sns.heatmap(df_cm, annot=True, fmt='d')
```

```
0.87396
              precision    recall  f1-score   support

         pos       0.89      0.86      0.87     12500
         neg       0.86      0.89      0.88     12500

   micro avg       0.87      0.87      0.87     25000
   macro avg       0.87      0.87      0.87     25000
weighted avg       0.87      0.87      0.87     25000

[[10691  1809]
 [ 1342 11158]]
<matplotlib.axes._subplots.AxesSubplot at 0x7f788f0a82e8>
```

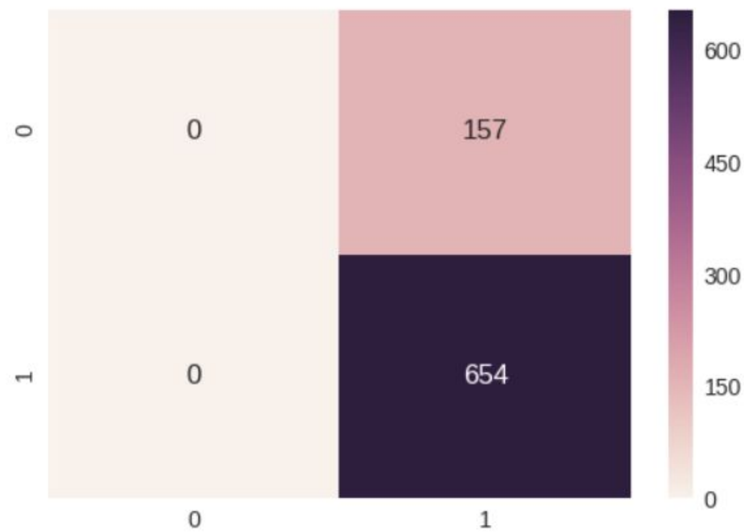

```
According to the confusion matrix:
10230 positive reviews were correctly predicted (True Positive)
10058 negative samples were correctly predicted (True Negative)
Number of incorrect predictions are 2270 (False Positive) and 2442 (False Negative).
```

5. Transfer Learning

```
score, acc = model.evaluate(financial_x_test, financial_y_test, verbose=1)
print("Accuracy: %.2f%%" % (acc*100))
print("Test score: %.2f%%" % (score*100))
```

```
811/811 [==============================] - 1s 2ms/step
Accuracy: 80.64%
Test score: 60.56%
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f788f0b9c18>



Since no neutral sentiment in IMDB Dataset, so we removed all those neutral sentiment in financial dataset

6. Transfer Learning for Financial Dataset Accuracy: 80.64%

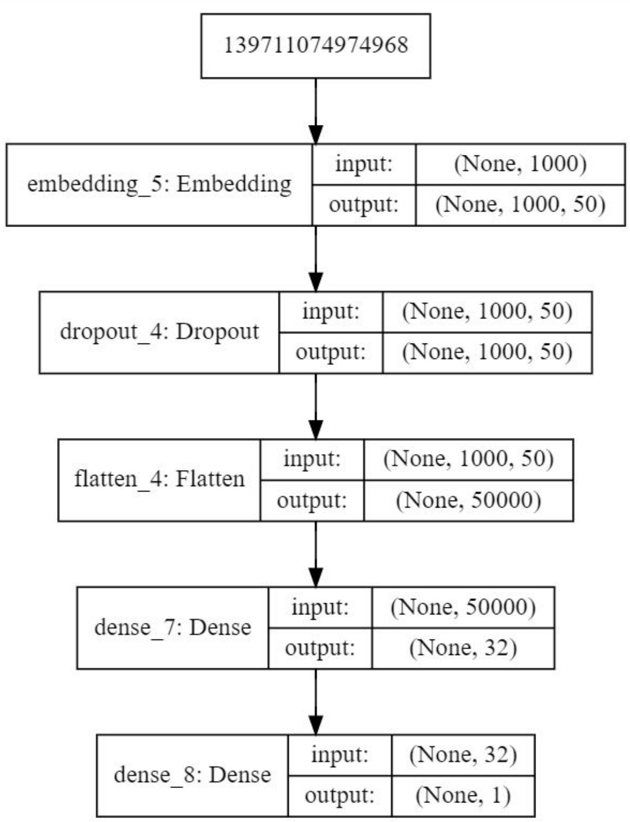
- GloVe:

1. Modeling on IMDB Dataset

```python
model = Sequential()
model.add(Embedding(max_words,
                    embedding_dims,
                    weights = [embedding_matrix],
                    trainable = False,
                    input_length=maxlen))
model.add(Dropout(0.2))
model.add(Flatten())
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

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

model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_5 (Embedding)      (None, 1000, 50)          250000    
_________________________________________________________________
dropout_4 (Dropout)          (None, 1000, 50)          0         
_________________________________________________________________
flatten_4 (Flatten)          (None, 50000)             0         
_________________________________________________________________
dense_7 (Dense)              (None, 32)                1600032   
_________________________________________________________________
dense_8 (Dense)              (None, 1)                 33        
=================================================================
Total params: 1,850,065
Trainable params: 1,600,065
Non-trainable params: 250,000
```

```python
SVG(model_to_dot(model, show_shapes=True).create(prog='dot', format='svg'))
```

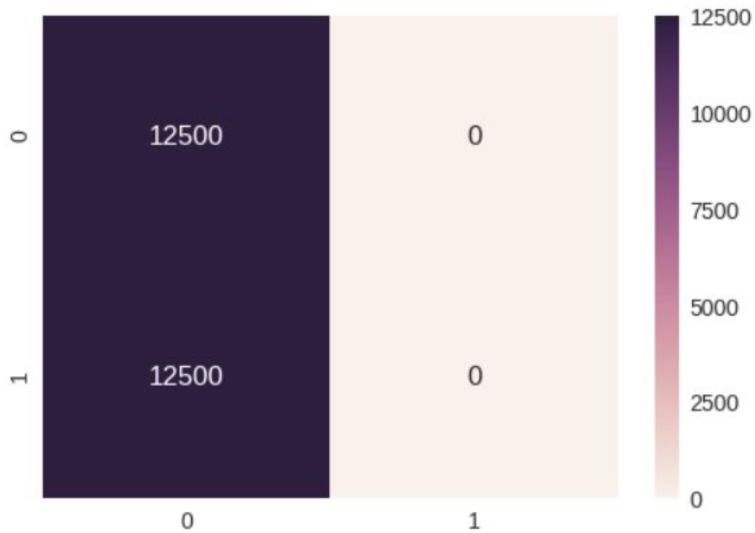


1.2. Accuracy Score and Confusion Matrix

```
score, acc = model.evaluate(x_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (acc*100))
```

```
Accuracy: 50.00%
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f110209f0f0>
```



2.1 CNN With GloVe Modeling

```python
model = Sequential()
model.add(Embedding(max_words,
                    embedding_dims,
                    weights = [embedding_matrix],
                    input_length=maxlen))
model.add(Conv1D(num_filters, 7, activation='relu', padding='same'))
model.add(MaxPooling1D(2))
model.add(Conv1D(num_filters, 7, activation='relu', padding='same'))
model.add(GlobalMaxPooling1D())
model.add(Dropout(0.5))
model.add(Dense(32, activation='relu', kernel_regularizer=regularizers.l2(weight_decay)))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy', optimizer=optimizers.Adam(lr=0.001), metrics=['accuracy'])
model.summary()
```

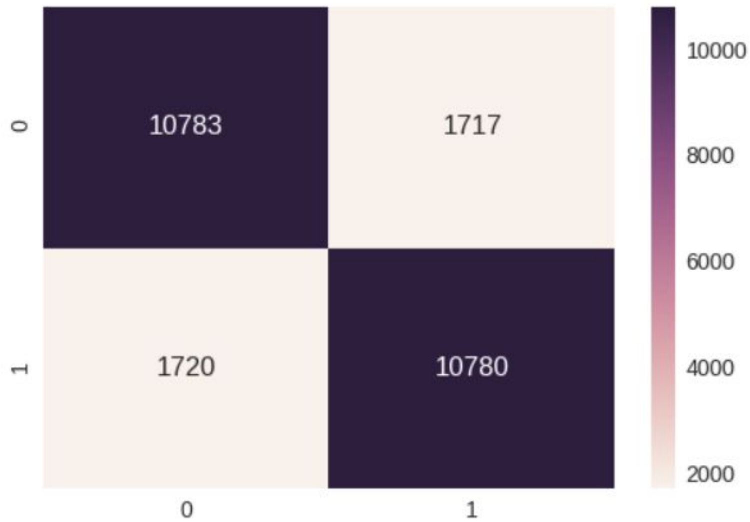
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_9 (Embedding)      (None, 1000, 50)          250000
_________________________________________________________________
conv1d_3 (Conv1D)            (None, 1000, 64)          22464
_________________________________________________________________
max_pooling1d_2 (MaxPooling1 (None, 500, 64)           0
_________________________________________________________________
conv1d_4 (Conv1D)            (None, 500, 64)           28736
_________________________________________________________________
global_max_pooling1d_2 (Glob (None, 64)                0
_________________________________________________________________
dropout_6 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_11 (Dense)             (None, 32)                2080
_________________________________________________________________
dense_12 (Dense)             (None, 1)                 33
=================================================================
Total params: 303,313
Trainable params: 303,313
Non-trainable params: 0
```

2.2 Accuracy Score and Confusion Matrix

```python
score, acc = model.evaluate(x_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (acc*100))
print("Test score: %.2f%%" % (score*100))
```

```
Accuracy: 86.25%
Test score: 37.34%
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f10f7d588d0>

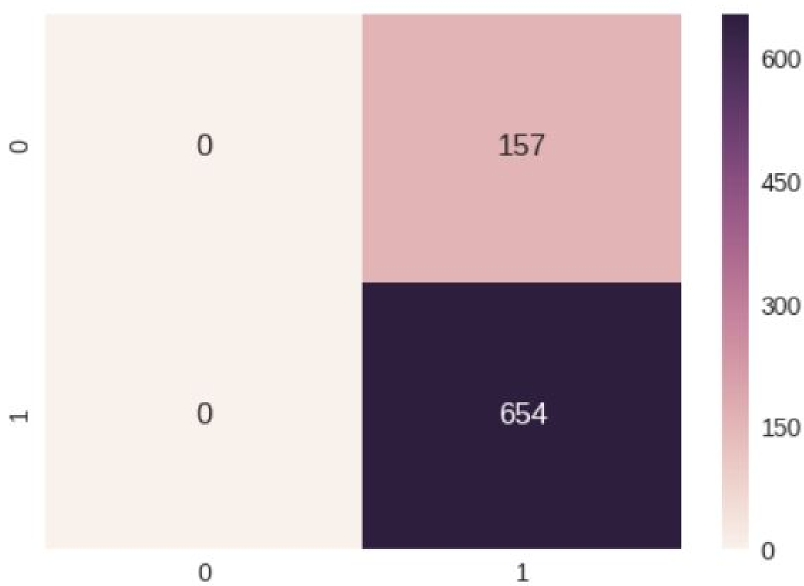


3. Transfer Learning on Financial Dataset

```
score, acc = model.evaluate(financial_x_test, financial_y_test, verbose=1)
print("Accuracy: %.2f%%" % (acc*100))
print("Test score: %.2f%%" % (score*100))
```

```
811/811 [==============================] - 0s 189us/step
Accuracy: 80.64%
Test score: 53.70%
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f1101d97390>



4. Transfer Learning for Financial Dataset Accuracy: 80.64%

Experiment 3

- Azure API:

1. Preprocess the data

```
financial_data.text=financial_data.text.astype(str)
text = list(financial_data.text)

id_list = [i for i in range(1, len(text)+1)]

analyze_text = []
for i in range(len(text)):
    dict_doc = {"language": "en",
                "id": str(id_list[i]),
                "text": text[i]}
    analyze_text.append((dict_doc))
```

```
analyze_text
```

```
[{'id': '1',
  'language': 'en',
  'text': 'Good day, ladies and gentlemen, and welcome to the Tesla, Inc. Q4 2018 Financial Results and Q&
A Webcast. At this time, all participants are in a listen-only mode. Later, we will conduct a question-and
-answer session, and instructions will follow at that time. [Operator Instructions] As a reminder, this co
nference is being recorded.'},
 {'id': '2',
```

2. Using the Azure API for sentiment analysis

```
import requests
from pprint import pprint
headers   = {"Ocp-Apim-Subscription-Key": subscription_key}
response  = requests.post(sentiment_api_url, headers=headers, json=document)
languages = response.json()
pprint(languages)
```

```
{'code': 'RequestEntityTooLarge',
 'innerError': {'code': 'InvalidRequestContent',
                'maxCharactersPerRequest': 524288,
                'message': 'Request Payload sent is too large to be processed. '
                           'Limit request size to: 524288 characters.'},
 'message': 'Invalid request'}
```

```
document_partA = {"documents": analyze_text[:1000]}

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

```
{'documents': [{'id': '1', 'score': 0.8151894807815552},
               {'id': '2', 'score': 0.1584499478340149},
               {'id': '3', 'score': 0.5},
               {'id': '4', 'score': 0.7589907646179199},
               {'id': '5', 'score': 0.14238208532333374},
```

```
document_partB = {"documents": analyze_text[1000:]}

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


```
{'documents': [{'id': '1001', 'score': 0.5},
               {'id': '1002', 'score': 0.5},
               {'id': '1003', 'score': 0.5},
               {'id': '1004', 'score': 0.5},
               {'id': '1005', 'score': 0.5},
               {'id': '1006', 'score': 0.7688121795654297},
```


3. Combining the Azure API Score

```
res = json.dumps(languages)
res = json.loads(res)
score_1 = []

for i in range (len(res['documents'])):
    score_1.append(res['documents'][i]['score'])
```

```
res = json.dumps(languages_B)
res = json.loads(res)
score_2 = []

for i in range (len(res['documents'])):
    score_2.append(res['documents'][i]['score'])
```

```
score = score_1 + score_2
len(score)
```

```
1649
```

4. Save into the azure_df.csv file

```
score_labels = []

for i in range(len(score)):
    if score[i] > 0.6:
        score_labels.append ('positive')
    elif score[i] < 0.4:
        score_labels.append ('negative')
    elif score[i] < 0.6 and score[i] > 0.4:
        score_labels.append ('neutral')
```

```
dic = {
    'text': text,
    'sentiment_predicted': score_labels,
    'score': score
}

azure_df = pd.DataFrame.from_dict(dic)
azure_df.to_csv('azure_df.csv')
```

```
from google.colab import files

files.download("azure_df.csv")
```

- Google API:

1. Get the credentials

```
cred_file_loc = r'My First Project-8ef1ccd74268.json'
```

```
cred = service_account.Credentials.from_service_account_file(cred_file_loc)
client = language.LanguageServiceClient(credentials=cred)
```

2. Preprocess the data

```
import pandas as pd
import io
import numpy as np

data = pd.read_csv(io.StringIO(uploaded['cleanedfinancial_data.csv'].decode('utf-8')))
data.text=data.text.astype(str)
texts = list(data['text'])
```

```
sentiment = np.array(data['sentiment'])
```

3. Using Google API to get the sentiment score

```
all_scores = []
for text in texts:
    client = language.LanguageServiceClient(credentials=cred)

    document = types.Document(
        content=text,
        type=enums.Document.Type.PLAIN_TEXT)
    annotations = client.analyze_sentiment(document=document)
    score = annotations.document_sentiment.score
    magnitude = annotations.document_sentiment.magnitude

    for index, sentence in enumerate(annotations.sentences):
        sentence_sentiment = sentence.sentiment.score

    all_scores.append(score)
```

4. Save into the google_result.csv file

```
score_text_form = []
for score in all_scores:
  if score>0:
    score_text_form.append("positive")
  elif score ==0:
    score_text_form.append("neutral")
  else:
    score_text_form.append("negative")
```

```
google = {
         'text': texts,
         'predict': score_text_form,
         'score': all_scores,
         'label': sentiment
      }

google_result = pd.DataFrame.from_dict(google)
google_result.to_csv('google_result.csv')
```

```
files.download("google_result.csv")
```

- IBM Watson API:

1. Preprocess the data

```
import pandas as pd
import io
data = pd.read_csv(io.StringIO(uploaded['cleanedfinancial_data.csv'].decode('utf-8')))

data.text=data.text.astype(str)
text = np.array(data.text)
```

```
labels = np.array(data.sentiment)
```

2. Using IBM API to get the sentiment score

```
def watson(texts):
  label = []
  score = []
  for i in range(len(texts)):
    try:
      response = natural_language_understanding.analyze(
                 text = texts[i],
                 features=Features(entities=EntitiesOptions(sentiment = True,  limit = 1))).get_result()
      if response['entities']:
        label.append(response['entities'][0]['sentiment']['label'])
        score.append(response['entities'][0]['sentiment']['score'])
      else:
        label.append('NA')
        score.append('NA')
    except:
      label.append('NA')
      score.append('NA')
  return label, score
```

```
pred, score = watson(text)
```

3. Obtain the score and save into ibm_result.csv file

```
ibm = {
         'text': text,
         'predict': pred,
         'score': score,
         'label': labels
      }

df_ibm = pd.DataFrame.from_dict(ibm)
df_ibm.to_csv('ibm_result.csv')
```

```
from google.colab import files

files.download("ibm_result.csv")
```

Experiment 4

- Merge All API's Score

```
dict_score = {'azure': np.array(azure.score),
              'google': (np.array(google.score,dtype=np.float64)),
              'ibm': (np.array(ibm.score,dtype=np.float64)),
              'label': list(google['label'].replace({'positive':'0','negative':'1','neutral':'2'})) }
df_scores = pd.DataFrame(dict_score)
df_scores.head()
```

	azure	google	ibm	label
0	0.815189	0.2	0.000000	2
1	0.158450	0.0	0.000000	1
2	0.500000	0.0	-0.183405	1
3	0.758991	0.2	0.554778	2
4	0.142382	0.0	0.000000	0

```
df_scores.to_csv('merged_api_score.csv', index=False)
# Check with the "algo.ipynb" to read the clean data
files.download("merged_api_score.csv")
```

- TPOT:

1. Preprocess the merged data

```
from tpot import TPOTClassifier
from sklearn.model_selection import train_test_split
X = df_scores.iloc[:,:-1]
y = df_scores.iloc[:,-1]
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8, test_size=0.2)
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

```
((1319, 3), (330, 3), (1319,), (330,))
```

2. Build TPOTClassifier

```
tpot = TPOTClassifier(verbosity=2, max_time_mins=20, population_size=40)
tpot.fit(X_train, y_train)
print(tpot.score(X_test, y_test))
```

```
HBox(children=(IntProgress(value=0, description='Optimization Progress', max=40, style=ProgressStyle(descr
ipti…
Generation 1 - Current best internal CV score: 0.617054308056532
Generation 2 - Current best internal CV score: 0.6170772652270607
Generation 3 - Current best internal CV score: 0.6178147643302963
Generation 4 - Current best internal CV score: 0.6178319822081929
Generation 5 - Current best internal CV score: 0.6223947198507784
Generation 6 - Current best internal CV score: 0.6224004591434105
Generation 7 - Current best internal CV score: 0.6224004591434105
Generation 8 - Current best internal CV score: 0.6238697180572494
Generation 9 - Current best internal CV score: 0.6238697180572494
Generation 10 - Current best internal CV score: 0.6238697180572494
Generation 11 - Current best internal CV score: 0.6238697180572494
Generation 12 - Current best internal CV score: 0.6238697180572494
Generation 13 - Current best internal CV score: 0.6268885859817778
Generation 14 - Current best internal CV score: 0.6268885859817778
Generation 15 - Current best internal CV score: 0.6268885859817778
Generation 16 - Current best internal CV score: 0.6268885859817778
Generation 17 - Current best internal CV score: 0.6268885859817778
Generation 18 - Current best internal CV score: 0.6268885859817778
Generation 19 - Current best internal CV score: 0.6268885859817778
Generation 20 - Current best internal CV score: 0.6268885859817778
Generation 21 - Current best internal CV score: 0.6268885859817778
Generation 22 - Current best internal CV score: 0.6268885859817778

20.0065854 minutes have elapsed. TPOT will close down.
TPOT closed during evaluation in one generation.
WARNING: TPOT may not provide a good pipeline if TPOT is stopped/interrupted in a early generation.


TPOT closed prematurely. Will use the current best pipeline.

Best pipeline: RandomForestClassifier(PolynomialFeatures(Normalizer(input_matrix, norm=max), degree=2, inc
lude_bias=False, interaction_only=False), bootstrap=True, criterion=entropy, max_features=0.35000000000000
003, min_samples_leaf=15, min_samples_split=18, n_estimators=100)
```

3. TPOT Best Score: 0.627

- AutoSKLearn

1. Preprocess the merged data

```
from sklearn.model_selection import train_test_split
X = df_scores.iloc[:,:-1]
y = df_scores.iloc[:,-1]
```

```
import autosklearn.metrics
from sklearn.metrics import accuracy_score,confusion_matrix
from sklearn.metrics import confusion_matrix

X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2)
```

2. Build the AutoSklearnClassifier

```
cls = autosklearn.classification.AutoSklearnClassifier()
cls.fit(X_train, y_train)
```

```
/usr/local/lib/python3.6/dist-packages/autosklearn/evaluation/train_evaluator.py:197: RuntimeWarning: Mean
of empty slice
  Y_train_pred = np.nanmean(Y_train_pred_full, axis=0)
[WARNING] [2019-04-02 17:57:09,293:EnsembleBuilder(1):7b2a97035e7de8c9013cedc0f1cc61ff] No models better t
han random - using Dummy Score!
[WARNING] [2019-04-02 17:57:09,317:EnsembleBuilder(1):7b2a97035e7de8c9013cedc0f1cc61ff] No models better t
han random - using Dummy Score!
/usr/local/lib/python3.6/dist-packages/autosklearn/evaluation/train_evaluator.py:197: RuntimeWarning: Mean
of empty slice
  Y_train_pred = np.nanmean(Y_train_pred_full, axis=0)
/usr/local/lib/python3.6/dist-packages/autosklearn/evaluation/train_evaluator.py:197: RuntimeWarning: Mean
of empty slice
  Y_train_pred = np.nanmean(Y_train_pred_full, axis=0)
/usr/local/lib/python3.6/dist-packages/autosklearn/evaluation/train_evaluator.py:197: RuntimeWarning: Mean
of empty slice
```

```
AutoSklearnClassifier(delete_output_folder_after_terminate=True,
           delete_tmp_folder_after_terminate=True,
           disable_evaluator_output=False, ensemble_memory_limit=1024,
           ensemble_nbest=50, ensemble_size=50, exclude_estimators=None,
           exclude_preprocessors=None, get_smac_object_callback=None,
           include_estimators=None, include_preprocessors=None,
           initial_configurations_via_metalearning=25, logging_config=None,
           metadata_directory=None, ml_memory_limit=3072, n_jobs=None,
           output_folder=None, per_run_time_limit=360,
           resampling_strategy='holdout',
           resampling_strategy_arguments=None, seed=1, shared_mode=False,
           smac_scenario_args=None, time_left_for_this_task=3600,
           tmp_folder=None)
```

3. Obtain the AutoSklearn Score

```
pred_train = cls.predict(X_train)
print("Accuracy score", accuracy_score(y_train, pred_train))
```

```
Accuracy score 0.6580742987111448
```

```
pred_test = cls.predict(X_test)
print("Accuracy score", accuracy_score(y_test, pred_test))
```

```
Accuracy score 0.6181818181818182
```

4. AutoSklearn Best Score: 0.618

- H2O

1. Import H2O and get the init

```
import h2o
h2o.init()
from h2o.estimators.word2vec import H2OWord2vecEstimator
from h2o.estimators.gbm import H2OGradientBoostingEstimator
```

```
Connecting to H2O server at http://127.0.0.1:54321 ... successful.
```

H2O cluster uptime:	02 secs
H2O cluster timezone:	Etc/UTC
H2O data parsing timezone:	UTC
H2O cluster version:	3.24.0.1
H2O cluster version age:	1 day
H2O cluster name:	H2O_from_python_unknownUser_dvad9d
H2O cluster total nodes:	1
H2O cluster free memory:	2.938 Gb
H2O cluster total cores:	2
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:	Amazon S3, XGBoost, Algos, AutoML, Core V3, Core V4
Python version:	3.6.7 final

2. Import the merged dataset

```
h2o_data = h2o.import_file("merged_api_score.csv")
h2o_data.head()
```

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

azure	google	ibm	label
0.815189	0.2	0	2
0.15845	0	0	1
0.5	0	-0.183405	1
0.758991	0.2	0.554778	2
0.142382	0	0	0
0.5	0	0.82987	2
0.803309	0	0	2
0.910819	0.2	0	0
0.183081	0.1	0	0
0.5	0.4	0.38213	0

3. Build the H2OAutoML

```
train,test,valid= h2o_data.split_frame(ratios=[.7, .15])
```

```
# Train the model
from h2o.automl import H2OAutoML

aml = H2OAutoML(max_models=30, seed=99,max_runtime_secs =6000)

x = train.columns
y = "label"
x.remove(y)

train[y] = train[y].asfactor()
aml.train(x=x, y=y, training_frame=train)
```

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

4. View the AutoML Leaderboard

```
# View the AutoML Leaderboard
lb = aml.leaderboard
print (lb)
```

model_id	**mean_per_class_error**	**logloss**	**rmse**	**mse**
DeepLearning_grid_1_AutoML_20190402_180520_model_5	0.551064	0.900581	0.564019	0.318118
StackedEnsemble_BestOfFamily_AutoML_20190402_180520	0.552608	0.86372	0.553188	0.306017
DeepLearning_grid_1_AutoML_20190402_180520_model_3	0.55383	0.951389	0.555784	0.308896
GBM_grid_1_AutoML_20190402_180520_model_2	0.554632	0.870487	0.553314	0.306157
DeepLearning_grid_1_AutoML_20190402_180520_model_2	0.554921	0.9314	0.552502	0.305259
GBM_grid_1_AutoML_20190402_180520_model_1	0.556476	1.08477	0.66199	0.43823
DeepLearning_grid_1_AutoML_20190402_180520_model_4	0.557485	0.899835	0.555174	0.308219
XRT_1_AutoML_20190402_180520	0.558329	1.0895	0.557862	0.31121
DeepLearning_grid_1_AutoML_20190402_180520_model_1	0.558538	0.947142	0.566306	0.320703
GBM_1_AutoML_20190402_180520	0.55995	0.869403	0.557416	0.310713

5. Get the H2OAutoML's Score

```
# Predict
test_y = aml.leader.predict(test)
```

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

```
from sklearn.metrics import accuracy_score

print("H2O Auto ML Accuracy Score is "+str(accuracy_score(preds,test['label'])))
```

```
Export File progress: |███████████████████████████████████████████████████| 100%
Export File progress: |███████████████████████████████████████████████████| 100%
H2O Auto ML Accuracy Score is 0.6
```

6. H2OAutoML Best Score: 0.6