In [1]:

```
import numpy as np
import pandas as pd
import random
from datetime import datetime, timedelta
from bs4 import BeautifulSoup
from selenium import webdriver
import requests
from selenium.webdriver.common.by import By
from textblob import TextBlob
from selenium.common.exceptions import StaleElementReferenceException, Timeou
import quandl
import requests
import time

import tensorflow as tf
from tensorflow import keras
from tensorboard.plugins.hparams import api as hp
from tensorboard import notebook
from tensorflow.contrib.tensor_forest.python import tensor_forest
import ast
from sklearn import preprocessing
import os
```

In [2]:

```
tf.enable_eager_execution()
```

In [3]:

```
%load_ext tensorboard
```

In [4]:

```
csv = pd.read_csv("C:/Users/Adhithya/Desktop/sector_output.csv")
```

In [4]:

```
def train_model(input_neurons, hidden_layers, output_neurons, opt, dropout):
    print(input_neurons)
    nn_model = tf.keras.models.Sequential()
    nn_model.add(keras.layers.InputLayer(input_shape=(1,input_neurons)))
    for neurons in hidden_layers:
        print(neurons)
        nn_model.add(tf.keras.layers.Dense(neurons, activation=tf.nn.relu))
    print(output_neurons)
    nn_model.add(tf.keras.layers.Dropout(dropout))
    nn_model.add(keras.layers.Dense(output_neurons, activation=tf.nn.softmax)

    nn_model.compile(optimizer=opt,
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

    return nn_model
```

In [5]:

```
def populate_training_testing(train_data, test_data):

    testing_inputs = []
    testing_outputs = []
    training_inputs = []
    training_outputs = []

    for i,row in test_data.iterrows():
        features = populate_features(row)
#         this_test = [row['token']]
#         this_test.extend([features, row['labels'], row['bounding box']])
#         test_array.append(this_test)

#         this_output = ast.literal_eval(row['labels'])
#         this_output = ast.literal_eval(row['labels'])
        target = row['Prev Target']
        testing_outputs.append(target)

        testing_inputs.append(features)

    for i,row in train_data.iterrows():
        features = populate_features(row)
#         this_output = ast.literal_eval(row['labels'])
        target = row['Prev Target']
        training_outputs.append(target)

        training_inputs.append(features)

    return training_inputs, training_outputs, testing_inputs, testing_outputs
```

In [140]:

```
def scale_average(starting, average):
    starting = float(starting)
    average = float(average)
    perc_change = (average - starting)/average
    return perc_change

csv["Scaled 200"] = list(map(scale_average, csv["200 Day Moving Average"], cs
csv["Scaled 90"] = list(map(scale_average, csv["90 Day Moving Average"], csv[
csv["Scaled 30"] = list(map(scale_average, csv["30 Day Moving Average"], csv[
```

In [141]:

```
def target_func(end, start):
    if float(end) > float(start):
        return 1
    else:
        return 0


csv["target"] = list(map(target_func, csv["Closing"], csv["Starting"]))
```

In [142]:

```
volumes_list = []
for symbol in csv["Symbol"].unique():
    volumes = preprocessing.scale(csv[csv["Symbol"] == symbol]["Volume"].valu
    volumes_list.extend(volumes)

csv["Scaled Volumes"] = volumes_list
cleaned_csv = csv.dropna(axis=0)
```

In [6]:

```
def populate_features(row):
    features = []
    volume = row['Scaled Volumes']
    snp = row['snp_change']
    nyse = row['nyse_change']
    nasdaq = row['nasdaq_change']
    two_hundred = row['Scaled 200']
    ninety = row['Scaled 90']
    thirty = row['Scaled 30']
    sector = row['Sector Change']
#     features.extend([volume, snp, nyse, nasdaq, two_hundred, ninety, thirty
    features.extend([volume, two_hundred, ninety, thirty, snp, nyse, nasdaq]
    return features
```

In [7]:

```
def reshape(input_data):
    return np.array(input_data).reshape(len(input_data),1, len(input_data[0]
```

In [ ]:

In [146]:

```
cleaned_csv["Prev Target"] = cleaned_csv['target'].shift(-1)
```

```
C:\Users\Adhithya\Anaconda\Anaconda3\lib\site-packages\ipykernel_launcher.p
y:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/
stable/indexing.html#indexing-view-versus-copy (http://pandas.pydata.org/pa
ndas-docs/stable/indexing.html#indexing-view-versus-copy)
  """Entry point for launching an IPython kernel.
```

In [147]:

```
cleaned_csv
```

Out[147]:

	Unnamed: 0	Symbol	Date	Starting	High	Low	Closing	Volume	Week
143	143	A	2015-05-11	42.39	42.8300	42.390	42.62	975400	201520
144	144	A	2015-05-12	42.36	42.5000	41.500	41.91	2617800	201520
145	145	A	2015-05-13	42.05	42.3900	41.710	41.81	1992700	201520
146	146	A	2015-05-14	42.18	42.2300	41.870	42.05	2883600	201520
148	148	A	2015-05-18	42.05	42.7000	41.980	42.63	1966100	201521
149	149	A	2015-05-19	41.57	42.8300	41.500	42.37	5277300	201521

In [148]:

```
cleaned_csv.drop(cleaned_csv.columns[0], axis=1)
```

Out[148]:

	Symbol	Date	Starting	High	Low	Closing	Volume	Week	snp_change
143	A	2015-05-11	42.39	42.8300	42.390	42.62	975400	201520	-0.004836
144	A	2015-05-12	42.36	42.5000	41.500	41.91	2617800	201520	-0.001783
145	A	2015-05-13	42.05	42.3900	41.710	41.81	1992700	201520	-0.000543
146	A	2015-05-14	42.18	42.2300	41.870	42.05	2883600	201520	0.009841
148	A	2015-05-18	42.05	42.7000	41.980	42.63	1966100	201521	0.003724
149	A	2015-05-19	41.57	42.8300	41.500	42.37	5277300	201521	-0.000761

In [152]:

```
cleaned_csv = cleaned_csv.set_index("Date")
```

In [154]:
```
cleaned_csv
```

Out[154]:
```
            Unnamed:  Symbol  Starting     High     Low  Closing   Volume    Week  snp_cha
                   0
      Date
2015-05-11       143       A     42.39  42.8300  42.390    42.62   975400  201520   -0.004
2015-05-12       144       A     42.36  42.5000  41.500    41.91  2617800  201520   -0.001
2015-05-13       145       A     42.05  42.3900  41.710    41.81  1992700  201520   -0.000
2015-05-14       146       A     42.18  42.2300  41.870    42.05  2883600  201520    0.009
2015-05-18       148       A     42.05  42.7000  41.980    42.63  1966100  201521    0.003
```

In [8]:
```
cleaned_csv = pd.read_csv("C:/Users/Adhithya/Desktop/cleaned.csv")
```

In [9]:
```
cleaned_csv = cleaned_csv.dropna()
```

In [10]:
```
train_data = cleaned_csv[cleaned_csv['Symbol'] != 'A']
test_data = cleaned_csv[cleaned_csv['Symbol'] == 'A']
```

In [11]:
```
training_inputs, training_outputs, testing_inputs, testing_outputs  = populat
```

In [12]:
```
training_inputs, testing_inputs = reshape(training_inputs), reshape(testing_i
model_data = training_inputs,  testing_inputs, training_outputs, testing_outp
```

In [13]:
```
cleaned_csv
```

Out[13]:

	Date	Unnamed: 0	Symbol	Starting	High	Low	Closing	Volume	Week
0	2015-05-11	143	A	42.39	42.8300	42.390	42.62	975400	201520
1	2015-05-12	144	A	42.36	42.5000	41.500	41.91	2617800	201520
2	2015-05-13	145	A	42.05	42.3900	41.710	41.81	1992700	201520
3	2015-05-14	146	A	42.18	42.2300	41.870	42.05	2883600	201520
4	2015-05-18	148	A	42.05	42.7000	41.980	42.63	1966100	201521
5	2015-05-19	149	A	41.57	42.8300	41.500	42.37	5277300	201521

In [21]:
```
testing_inputs.shape
```

Out[21]:
```
(270, 1, 7)
```

In [55]:
```
nn_model = train_model(7, [10, 6], 2, "adam", 0.2)
```

```
7
10
6
2
```

In [56]:

```
nn_model.fit(training_inputs, np.array(training_outputs), epochs=5)
test_loss, test_acc = nn_model.evaluate(testing_inputs, testing_outputs)
print('Test accuracy:', test_acc)
```

```
Epoch 1/5
288433/288433 [==============================] - 17s 59us/sample - loss: 0.
6931 - acc: 0.5094
Epoch 2/5
288433/288433 [==============================] - 17s 58us/sample - loss: 0.
6930 - acc: 0.5100
Epoch 3/5
288433/288433 [==============================] - 17s 59us/sample - loss: 0.
6930 - acc: 0.5103
Epoch 4/5
288433/288433 [==============================] - 16s 57us/sample - loss: 0.
6930 - acc: 0.5104
Epoch 5/5
288433/288433 [==============================] - 17s 57us/sample - loss: 0.
6930 - acc: 0.5103
270/270 [==============================] - 0s 1ms/sample - loss: 0.6918 - a
cc: 0.5284
Test accuracy: 0.5283739
```

In [103]:

```
predictions = nn_model.predict(testing_inputs)
```

In [48]:

```
predictions
```

```
       [[0.2575083 , 0.7424918 ]],

       [[0.22941078, 0.77058923]],

       [[0.5453895 , 0.45461056]],

       [[0.40323377, 0.59676623]],

       [[0.5544524 , 0.44554755]],

       [[0.33468476, 0.6653152 ]],

       [[0.7776768 , 0.22232313]],

       [[0.19776514, 0.8022348 ]],

       [[0.48313162, 0.5168684 ]],

       [[0.5508936 , 0.44910643]],
```

In [14]:

```
cleaned_csv['3 Month Future'] =  cleaned_csv['Closing'].shift(-63)
cleaned_csv['1 Month Future'] =  cleaned_csv['Closing'].shift(-21)
cleaned_csv['1 Week Future'] =  cleaned_csv['Closing'].shift(-4)
```

In [15]:

```
cleaned_csv
```

Out[15]:

	Date	Unnamed: 0	Symbol	Starting	High	Low	Closing	Volume	Week
0	2015-05-11	143	A	42.39	42.8300	42.390	42.62	975400	201520
1	2015-05-12	144	A	42.36	42.5000	41.500	41.91	2617800	201520
2	2015-05-13	145	A	42.05	42.3900	41.710	41.81	1992700	201520
3	2015-05-14	146	A	42.18	42.2300	41.870	42.05	2883600	201520
4	2015-05-18	148	A	42.05	42.7000	41.980	42.63	1966100	201521
5	2015-05-19	149	A	41.57	42.8300	41.500	42.37	5277300	201521

In [16]:

```
def improvement_perc(average, start, future):
    start = float(start)
    future = float(future)
    if (average > 0 and future > start) or (average < 0 and future < start):
        return 1
    else:
        return 0

week_improvement_200 = list(map(improvement_perc, cleaned_csv["Scaled 200"],
month_improvement_200 = list(map(improvement_perc, cleaned_csv["Scaled 200"],
month3_improvement_200 = list(map(improvement_perc, cleaned_csv["Scaled 200"]

week_improvement_90 = list(map(improvement_perc, cleaned_csv["Scaled 90"], cl
month_improvement_90 = list(map(improvement_perc, cleaned_csv["Scaled 90"], c
month3_improvement_90 = list(map(improvement_perc, cleaned_csv["Scaled 90"],

week_improvement_30 = list(map(improvement_perc, cleaned_csv["Scaled 30"], cl
month_improvement_30 = list(map(improvement_perc, cleaned_csv["Scaled 30"], c
month3_improvement_30 = list(map(improvement_perc, cleaned_csv["Scaled 30"],
```

In [17]:

```
week_perc_200 = sum(week_improvement_200) / len(cleaned_csv)
month_perc_200 = sum(month_improvement_200) / len(cleaned_csv)
month3_perc_200 = sum(month3_improvement_200) / len(cleaned_csv)

week_perc_90 = sum(week_improvement_90) / len(cleaned_csv)
month_perc_90 = sum(month_improvement_90) / len(cleaned_csv)
month3_perc_90 = sum(month3_improvement_90) / len(cleaned_csv)

week_perc_30 = sum(week_improvement_30) / len(cleaned_csv)
month_perc_30 = sum(month_improvement_30) / len(cleaned_csv)
month3_perc_30 = sum(month3_improvement_30) / len(cleaned_csv)
```

In [18]:

```
print(week_perc_200, week_perc_90, week_perc_30)
```

```
0.5042829482201432 0.503576339698583 0.5027484993228335
```

In [19]:

```
print(month_perc_200, month_perc_90, month_perc_30)
```

```
0.4977676019992865 0.49819364537258015 0.49734155862599283
```

In [20]:

```
print(month3_perc_200, month3_perc_90, month3_perc_30)
```

```
0.4748547815575176 0.47555792631181526 0.4750695351277957
```

In [22]:

```
count = 0
for i, row in cleaned_csv.iterrows():
    if (row["Sector Change"] < 0 and row['target'] == 0) or (row["Sector Char
         count += 1

count/len(cleaned_csv)
```

Out[22]:

```
0.6901521632958438
```

In [23]:

```
cleaned_csv
```

25	06-17	170	A	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	201525
26	2015-06-18	171	A	39.80	40.0500	39.720	39.90	1865200	201525
27	2015-06-19	172	A	39.80	39.9400	39.490	39.49	2658700	201525
28	2015-06-22	173	A	39.81	40.0100	39.730	39.81	3909200	201526
29	2015-06-23	174	A	39.89	39.9500	39.420	39.60	2053500	201526
...	...	...	...	...	...	...	...	...	...
288673	2019-03-22	519743	KR	24.37	24.6600	24.150	24.34	6415100	201912
288674	2019-03-25	519744	KR	24.30	24.3800	24.040	24.15	5714600	201913
288675	2019-	519745	KR	24.23	24.4900	24.190	24.47	4715500	201913

In [135]:

```
HP_NUM_UNITS_1 = hp.HParam('num_units_1', hp.Discrete([8,12,16]))
HP_DROPOUT = hp.HParam('dropout', hp.RealInterval(0.1, 0.2))
HP_NUM_UNITS_2 = hp.HParam('num_units_2', hp.Discrete([4,6,8]))
# HP_NUM_LAYERS = hp.HParam('num_layers', hp.Discrete([1,2,3]))
HP_DROPOUT = hp.HParam('dropout', hp.RealInterval(0.1, 0.2))
HP_OPTIMIZER = hp.HParam('optimizer', hp.Discrete(['adam', 'sgd']))

METRIC_ACCURACY = 'accuracy'

with tf.contrib.summary.create_file_writer('logs/hparam_tuning').as_default()
    hp.hparams_config(
        hparams=[HP_NUM_UNITS_1, HP_NUM_UNITS_2, HP_DROPOUT, HP_OPTIMIZER],
        metrics=[hp.Metric(METRIC_ACCURACY, display_name='Accuracy')],
    )
```

In [141]:

```
def train_test_model(hparams, data):
    training_inputs, testing_inputs, training_outputs, testing_outputs = data
    nn_model = tf.keras.models.Sequential()
    nn_model.add(keras.layers.InputLayer(input_shape=(1,training_inputs.shape
#     for x in range(hparams[HP_NUM_LAYERS]):
#         nn_model.add(tf.keras.layers.Dense(hparams[HP_NUM_UNITS], activatio
    nn_model.add(tf.keras.layers.Dense(hparams[HP_NUM_UNITS_1], activation=tf
    nn_model.add(tf.keras.layers.Dropout(hparams[HP_DROPOUT]))
    nn_model.add(tf.keras.layers.Dense(hparams[HP_NUM_UNITS_2], activation=tf
    nn_model.add(keras.layers.Dense(2, activation=tf.nn.softmax))

    nn_model.compile(
      optimizer=hparams[HP_OPTIMIZER],
      loss='sparse_categorical_crossentropy',
      metrics=['accuracy'],
    )

    nn_model.fit(training_inputs, training_outputs, epochs=5)
    _, accuracy = nn_model.evaluate(testing_inputs, testing_outputs)
    return accuracy
```

In [142]:

```
def run(run_dir, hparams, model_data):
    with tf.contrib.summary.create_file_writer(run_dir).as_default():
        hp.hparams(hparams)
        accuracy = train_test_model(hparams, model_data)
        tf.summary.scalar(METRIC_ACCURACY, accuracy)
```

In [143]:

```
def run_hyperparameter_tuning(model_data):
    session_num = 0
    for num_units_1 in HP_NUM_UNITS_1.domain.values:
        for num_units_2 in HP_NUM_UNITS_2.domain.values:
            for dropout_rate in (HP_DROPOUT.domain.min_value, HP_DROPOUT.doma
                for optimizer in HP_OPTIMIZER.domain.values:
                    hparams = {
    #                   HP_NUM_LAYERS: num_layers,
                      HP_NUM_UNITS_1: num_units_1,
                      HP_NUM_UNITS_2: num_units_2,
                      HP_DROPOUT: dropout_rate,
                      HP_OPTIMIZER: optimizer,
                    }
                    run_name = "run-%d" % session_num
                    print('--- Starting trial: %s' % run_name)
                    print({h.name: hparams[h] for h in hparams})
                    run("logs/hparam_tuning/" + run_name, hparams, model_data
                    session_num += 1
```

In [144]:

```
run_hyperparameter_tuning(model_data)
```

```
--- Starting trial: run-0
{'num_units_1': 8, 'num_units_2': 4, 'dropout': 0.1, 'optimizer': 'ada
m'}
./NeuralLogs\Layer_1:8Layer_2:4Optimizer:adam

WARNING: Logging before flag parsing goes to stderr.
W0823 12:29:36.823054 20104 deprecation.py:323] From C:\Users\Adhithya\A
naconda\Anaconda3\lib\site-packages\tensorflow\python\ops\math_grad.py:1
250: add_dispatch_support.<locals>.wrapper (from tensorflow.python.ops.a
rray_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
```

In [ ]:

In [ ]: