In [1]:
```
import datetime
import pandas as pd
import pandas_datareader.data as web
```

In [2]:
```
start = datetime.datetime(2012, 5, 18) # or start = '1/1/2016'
end = datetime.date.today()
df = web.DataReader('FB', 'yahoo', start, end)
print (df.head())  # print first rows of the prices data
```

```
                 Open       High        Low      Close  Adj Close     Volume
Date
2012-05-18  42.049999  45.000000  38.000000  38.230000  38.230000  573576400
2012-05-21  36.529999  36.660000  33.000000  34.029999  34.029999  168192700
2012-05-22  32.610001  33.590000  30.940001  31.000000  31.000000  101786600
2012-05-23  31.370001  32.500000  31.360001  32.000000  32.000000   73600000
2012-05-24  32.950001  33.209999  31.770000  33.029999  33.029999   50237200
```

In [3]:
```
import time
import math
from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Activation
from keras.layers.recurrent import LSTM
import numpy as np
import pandas as pd
import tensorflow as tf
import sklearn.preprocessing as prep
from keras import backend
```

```
Using TensorFlow backend.
```

In [ ]:
```
import os
s=os.getcwd()
s
```

In [ ]:
```
df = pd.read_csv('/Users/Yuffie/USA/SCU/COEN281DataMining/TermProject/data/GOOG.csv')
df.head()
```

In [4]:
```
# Data preparation
col_list = df.columns.tolist()
col_list
```

Out[4]: ['Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume']

In [6]:
```
col_list.remove('Close')
col_list.append('Close')
#col_list.remove('Date')
col_list.remove('Volume')
col_list
```

Out[6]: ['Open', 'High', 'Low', 'Adj Close', 'Close']

In [7]:
```
df = df[col_list]
df.head()
```

Out[7]:

	Open	High	Low	Adj Close	Close
Date					
2012-05-18	42.049999	45.000000	38.000000	38.230000	38.230000
2012-05-21	36.529999	36.660000	33.000000	34.029999	34.029999
2012-05-22	32.610001	33.590000	30.940001	31.000000	31.000000
2012-05-23	31.370001	32.500000	31.360001	32.000000	32.000000
2012-05-24	32.950001	33.209999	31.770000	33.029999	33.029999

In [8]:

```
# Save data
df.to_csv('GOOG-adjust.csv', index=False)
validate_df = pd.read_csv('GOOG-adjust.csv')
validate_df.head()
```

Out[8]:

	Open	High	Low	Adj Close	Close
0	42.049999	45.000000	38.000000	38.230000	38.230000
1	36.529999	36.660000	33.000000	34.029999	34.029999
2	32.610001	33.590000	30.940001	31.000000	31.000000
3	31.370001	32.500000	31.360001	32.000000	32.000000
4	32.950001	33.209999	31.770000	33.029999	33.029999

In [9]:

```
# Standardization the dataset
def standard_scaler(X_train, X_test):
    train_samples, train_nx, train_ny = X_train.shape
    test_samples, test_nx, test_ny = X_test.shape

    X_train = X_train.reshape((train_samples, train_nx * train_ny))
    X_test = X_test.reshape((test_samples, test_nx * test_ny))

    preprocessor = prep.StandardScaler().fit(X_train)
    X_train = preprocessor.transform(X_train)
    X_test = preprocessor.transform(X_test)

    X_train = X_train.reshape((train_samples, train_nx, train_ny))
    X_test = X_test.reshape((test_samples, test_nx, test_ny))

    return X_train, X_test
```

In [10]:
```
# Split the data to X_train, y_train, X_test, y_test
def preprocess_data(stock, seq_len):
    amount_of_features = len(stock.columns)
    data = stock.as_matrix()

    sequence_length = seq_len + 1
    result = []
    for index in range(len(data) - sequence_length):
        result.append(data[index : index + sequence_length])

    result = np.array(result)
    row = round(0.9 * result.shape[0])
    train = result[: int(row), :]
    y_test_org = result[int(row) :, -1][ : ,-1]

    train, result = standard_scaler(train, result)

    X_train = train[:, : -1]
    y_train = train[:, -1][: ,-1]
    X_test = result[int(row) :, : -1]
    y_test = result[int(row) :, -1][ : ,-1]

    X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], amount_of_features))
    X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], amount_of_features))

    return [X_train, y_train, X_test, y_test, y_test_org]
```

In [11]:
```
# Build LSTM Neural Network
# LSTM --> Dropout --> LSTM --> Dropout --> Fully-Conneted(Dense)
def build_model(layers):
    model = Sequential()

    # By setting return_sequences to True we are able to stack another LSTM layer
    model.add(LSTM(
        return_sequences=True,
        input_shape=(None, 5), units=20))
    model.add(Dropout(0.4))

    model.add(LSTM(
        layers[2],
        return_sequences=False))
    model.add(Dropout(0.3))

    model.add(Dense(
        units=1))
    model.add(Activation("linear"))

    start = time.time()
    model.compile(loss="mse", optimizer="rmsprop", metrics=['accuracy'])
    print("Compilation Time : ", time.time() - start)
    return model
```

In [12]:
```
window = 20
X_train, y_train, X_test, y_test, y_test_org = preprocess_data(df[:: -1], window)
print("X_train", X_train.shape)
print("y_train", y_train.shape)
print("X_test", X_test.shape)
print("y_test", y_test.shape)
```

```
X_train (1238, 20, 5)
y_train (1238,)
X_test (137, 20, 5)
y_test (137,)
```

In [13]:
```
model = build_model([X_train.shape[2], window, 100, 1])
```

```
Compilation Time :  0.022166967391967773
```

In [14]:
```
# Training the model
model.fit(
    X_train,
    y_train,
    batch_size=768,
    epochs=300,
    validation_split=0.1,
    verbose=0)
```

Out[14]:
```
<keras.callbacks.History at 0x11e40ae10>
```

In [15]:
```
trainScore = model.evaluate(X_train, y_train, verbose=0)
print('Train Score: %.2f MSE (%.2f RMSE)' % (trainScore[0], math.sqrt(trainScore[0])))
print(model.metrics_names)
testScore = model.evaluate(X_test, y_test, verbose=0)
print('Test Score: %.2f MSE (%.2f RMSE)' % (testScore[0], math.sqrt(testScore[0])))
```

```
Train Score: 0.01 MSE (0.11 RMSE)
['loss', 'acc']
Test Score: 0.02 MSE (0.15 RMSE)
```

In [16]:

```python
#Visualize the Prediction
diff = []
ratio = []
pred = model.predict(X_test)
for u in range(len(y_test)):
    pr = pred[u][0]
    ratio.append((y_test[u] / pr) - 1)
    diff.append(abs(y_test[u] - pr))
print('error_ratio', ratio)
#print('error_abs', diff)
#print(pred)
```

```
error_ratio [-0.098042958775797606, -0.094387898152137217, -0.088177836205370785, -0.1016463994996889, -0.09
1531150874285205, -0.077665662356397824, -0.075558213633958315, -0.074090689005970711, -0.0480034589274903б
1, -0.055358434837225б9, -0.041504548351936177, -0.043161685013260009, -0.056630789900777967, -0.04099218598
4557117, -0.048125189397700052, -0.017690875810006501, -0.026214407917106608, -0.020459438781791683, -0.0371
5887691878339, -0.048873779787159122, -0.062470022008482062, -0.066564632619159458, -0.067597965841832863, -
0.069324303897041l4, -0.068893295978076807, -0.080800825611690885, -0.089356994003521706, -0.097671302640438
995, -0.045262016884400125, -0.041384407926094657, -0.038249635124750658, -0.040201666241083678, -0.05479043
8263231045, -0.051445192951791729, -0.053749324562053502, -0.055228623280366262, -0.059511721066375811, -0.0
59027507873875229, -0.067678306476899763, -0.070482866664272725, -0.077388068925051789, -0.09092801456512078
5, -0.088251372855242982, -0.092377271630725954, -0.086719958194268365, -0.080296905513316919, -0.0606420041
42723582, -0.064005651854373236, -0.059500800489382111, -0.066554521755501916, -0.094461636164366647, -0.090
27008141529147, -0.098153502032053197, -0.077158117664345549, -0.071066438645468133, -0.076555486560161312,
-0.058349958697675675, -0.061250534332721984, -0.041456865712741098, -0.034342892261972735, -0.0390729044197
46535, -0.04157597905638255, -0.039183366198515923, -0.030471826703060367, -0.037660710138708975, -0.0541713
74429546448, -0.056439224633190177, -0.060974657668029342, -0.059406195792016003, -0.063507403502283566, -0.
064298017492259385, -0.064488795125987597, -0.060881670796506193, -0.072077696717143591, -0.0592297983964134
02, -0.069862282371802475, -0.087171694883671935, -0.075452816267847278, -0.090167939214506831, -0.091104215
473806982, -0.078293364512044O5, -0.072587767541385628, -0.071298539039252651, -0.0860708773689286, -0.07358
2521944453028, -0.058030134365244956, -0.068867632322571537, -0.079960276821142928, -0.098536712792807846, -
0.10435300656826441, -0.14451208721681663, -0.17496548260613853, -0.15698017292516819, -0.15378928760956334,
-0.14667011588933698, -0.14304993518387377, -0.13794697331067929, -0.11738518681447685, -0.114761863648429,
-0.1457483069465505, -0.14223198697307271, -0.13907987732577731, -0.14100514072681258, -0.14577343321333958,
-0.13343105032821512, -0.12432633405143267, -0.11610597906342912, -0.10647052501904675, -0.1090882222808399
2, -0.11111165180479887, -0.12253897150824244, -0.13400379299851262, -0.1157596877612429, -0.128360785046430
45, -0.10741282654824669, -0.10188104361804218, -0.10592306027277598, -0.098020071723493918, -0.076819489484
665215, -0.052600162340873169, -0.041381547936191З1, -0.049000854650854797, -0.049245607042324835, -0.056442
498177195155, -0.050232403142573823, -0.062975038334869171, -0.054135269132665864, -0.073652229171729755, -
0.089491897588440894, -0.11928540042127478, -0.099638885094949226, -0.10896307162069874, -0.1523249228589583
8, -0.16558657379577535, -0.14557842603101012, -0.12534141302742363, -0.16524105015003765]
```

In [19]:

```
# Scale the representation back
y_test = y_test_org
pred_org = []
for u in range(len(y_test)):
    pred_org.append(y_test[u]/(ratio[u] + 1))
    #print(ratio[u])
print(pred_org)
print(y_test)
```

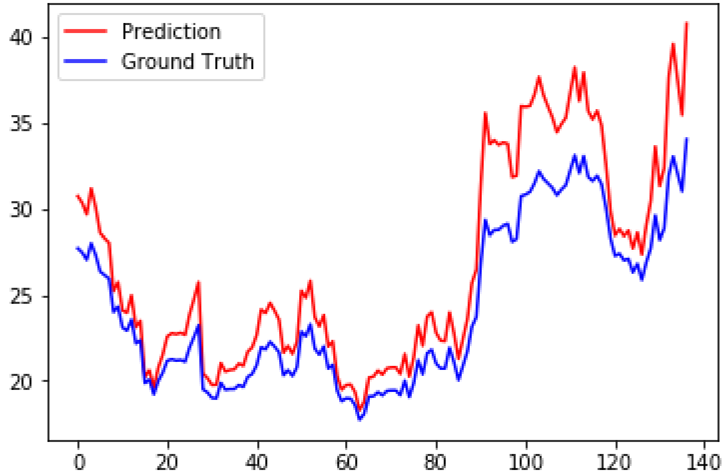
```
[30.722082908061733, 30.322031854443029, 29.654906486886244, 31.168127989253051, 30.07257764126091, 28.57965
9158462047, 28.287340950688758, 28.015703797331483, 25.210175630430172, 25.745214795634514, 24.1002708570825
42, 23.953890266525992, 24.974314136796977, 23.117642709471184, 23.490485041674422, 20.217669276335435, 20.6
10286456458009, 19.611233838146969, 20.761473020626084, 21.521853319761565, 22.58061128386823, 22.7653683828
43828, 22.7155231585511, 22.789892428493779, 22.671946092553274, 23.868603901434323, 24.773702594150539, 25.
744498726436142, 20.42445188612438, 20.154064006201253, 19.755646261142338, 19.774987445193279, 21.032371872
615876, 20.536504433117031, 20.628783161465851, 20.661083179484034, 20.999730078926241, 20.872022470735011,
21.698519020354382, 21.9468789421785, 22.663916751691541, 24.145503713327635, 23.943002873898575, 24.5366266
22399059, 24.078047251006939, 23.551078744699851, 21.6317954279567, 22.030048622464371, 21.563017821336327,
22.272324934391573, 25.244652146123077, 24.831545647244265, 25.824795075966207, 23.698535381433288, 23.16634
9990219366, 23.823846132400323, 21.993307589470906, 22.295618549431207, 20.270344969345075, 19.4789629251120
82, 19.751758574919748, 19.782474756144897, 19.337716840400624, 18.287245784419092, 18.766768841583012, 20.1
83360371955683, 20.242469270275532, 20.595823273492432, 20.35947920805733, 20.726271699946462, 20.7758467582
80416, 20.780083550809032, 20.402114839192215, 21.564305469550071, 20.249364794429489, 21.362429050471647, 2
3.22452194040843, 22.043222194167882, 23.7406450387689, 23.996149362019036, 22.794671526778373, 22.341735718
829462, 22.310720797795486, 23.9843544288155, 22.765114540224019, 21.274566980437172, 22.424307998316134, 2
3.596806152009531, 25.680468998044962, 26.472482098280082, 31.385598322071115, 35.56214846947239, 33.7477246
51646415, 33.974989419343252, 33.703261230530508, 33.840944986943271, 33.768225502085137, 31.82588778293658
6, 31.91231696866317, 35.961297179516251, 35.918801508203131, 35.973138720233202, 36.635840902036819, 37.659
795715571939, 36.615666891847226, 35.938044300910136, 35.298350549922937, 34.436468926395406, 34.90805798933
2789, 35.280022585146206, 36.730976024541185, 38.221874105671631, 36.25711309047778, 37.917062969408171, 35.
671585865244381, 35.184648732167446, 35.690440701597218, 34.823391314281132, 32.507185385930555, 29.86067748
3226151, 28.44718870296148, 28.811802969539468, 28.409019406132167, 28.721090074157704, 27.701512545862848,
28.611830097202063, 27.350634985911636, 29.038770154267279, 30.444538523689349, 33.609071558662315, 31.30966
0683172474, 32.366784227963265, 37.644140851259934, 39.584692626836791, 37.452237835419403, 35.4424005683167
92, 40.766258338549676]
[ 27.709999  27.459999  27.040001  28.        27.32      26.360001  26.15
  25.940001  24.        24.32      23.1       22.92      23.559999  22.17
  22.360001  19.860001  20.07      19.209999  19.99      20.469999  21.17
  21.25      21.18      21.209999  21.110001  21.940001  22.559999  23.23
  19.5       19.32      19.        18.98      19.879999  19.48      19.52
  19.52      19.75      19.639999  20.23      20.4      20.91      21.950001
  21.83      22.27      21.99      21.66      20.32      20.620001
  20.280001  20.790001  22.860001  22.59      23.290001  21.870001  21.52
  22.        20.709999  20.93      19.43      18.809999  18.98      18.959999
  18.58      17.73      18.059999  19.09      19.1       19.34      19.15
  19.41      19.440001  19.440001  19.16      20.01      19.049999
  19.870001  21.200001  20.379999  21.6       21.809999  21.01      20.719999
  20.719999  21.92      21.09      20.040001  20.879999  21.709999  23.15
  23.709999  26.85      29.34      28.450001  28.75      28.76      29.
  29.110001  28.09      28.25      30.719999  30.809999  30.969999
  31.469999  32.169998  31.73      31.469999  31.200001  30.77      31.1
```

```
 31.360001  32.23      33.099998  32.060001  33.049999  31.84      31.6
 31.91      31.41      30.01      28.290001  27.27      27.4       27.01
 27.1       26.309999  26.809999  25.870001  26.9       27.719999  29.6
 28.190001  28.84      31.91      33.029999  32.        31.        34.029999]
```

In [18]:

```python
import matplotlib.pyplot as plt2

plt2.plot(pred_org, color='red', label='Prediction')
plt2.plot(y_test, color='blue', label='Ground Truth')
plt2.legend(loc='upper left')
plt2.show()
```



In [ ]: