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

In [2]:
```
start = datetime.datetime(1990, 2, 16) # or start = '1/1/2016'
end = datetime.date.today()
df = web.DataReader('CSCO', 'yahoo', start, end)
print (df.head())  # print first rows of the prices data
```

```
                Open      High       Low     Close  Adj Close     Volume
Date
1990-02-16  0.073785  0.079861  0.073785  0.077257   0.063830  940636800
1990-02-20  0.077257  0.079861  0.074653  0.079861   0.065982  151862400
1990-02-21  0.078993  0.078993  0.075521  0.078125   0.064547   70531200
1990-02-22  0.078993  0.081597  0.078993  0.078993   0.065265   45216000
1990-02-23  0.078993  0.079861  0.078125  0.078559   0.064906   44697600
```

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 [4]:
```
#import os
#s=os.getcwd()
#s
```

In [5]:
```
#df = pd.read_csv('/Users/Yuffie/USA/SCU/COEN281DataMining/TermProject/data/GOOG.csv')
#df.head()
```

In [6]:
```
# Data preparation
col_list = df.columns.tolist()
col_list
```

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

In [7]:
```
col_list.remove('Close')
col_list.append('Close')
#col_list.remove('Date')
col_list.remove('Volume')
col_list
```

Out[7]:
```
['Open', 'High', 'Low', 'Adj Close', 'Close']
```

In [8]:
```
df = df[col_list]
df.head()
```

Out[8]:

	Open	High	Low	Adj Close	Close
Date					
1990-02-16	0.073785	0.079861	0.073785	0.063830	0.077257
1990-02-20	0.077257	0.079861	0.074653	0.065982	0.079861
1990-02-21	0.078993	0.078993	0.075521	0.064547	0.078125
1990-02-22	0.078993	0.081597	0.078993	0.065265	0.078993
1990-02-23	0.078993	0.079861	0.078125	0.064906	0.078559

In [9]:
```
# Save data
df.to_csv('CSCO-adjust.csv', index=False)
validate_df = pd.read_csv('CSCO-adjust.csv')
validate_df.head()
```

Out[9]:

	Open	High	Low	Adj Close	Close
0	0.073785	0.079861	0.073785	0.063830	0.077257
1	0.077257	0.079861	0.074653	0.065982	0.079861
2	0.078993	0.078993	0.075521	0.064547	0.078125
3	0.078993	0.081597	0.078993	0.065265	0.078993
4	0.078993	0.079861	0.078125	0.064906	0.078559

In [10]:
```
# 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 [11]:
```
# 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(0.99 * 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 [12]:
```
# 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 [13]:
```
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 (6914, 20, 5)
y_train (6914,)
X_test (70, 20, 5)
y_test (70,)
```

In [14]:
```
model = build_model([X_train.shape[2], window, 100, 1])
```

```
Compilation Time :  0.034848928451538086
```

In [15]:
```
# Training the model
model.fit(
    X_train,
    y_train,
    batch_size=768,
    epochs=300,
    validation_split=0.1,
    verbose=0)
```

Out[15]:
```
<keras.callbacks.History at 0x1243a95f8>
```

In [16]:
```
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: 0.00 MSE (0.05 RMSE)
['loss', 'acc']
Test Score: 0.01 MSE (0.10 RMSE)
```

In [17]:
```
#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)
```

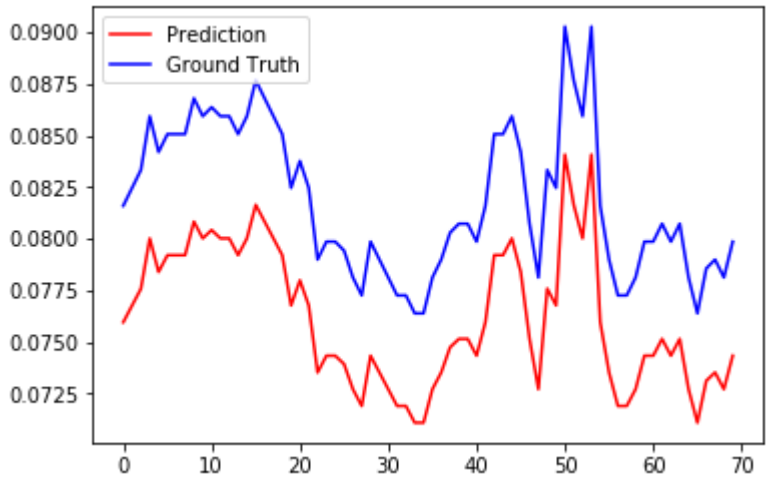
```
error_ratio [0.074267868306719276, 0.074213219559673194, 0.074160161259755553, 0.07400212108337478
9948186023834, 0.074042220330739372, 0.074038841830492697, 0.074035562718226311, 0.07393078154509780
98161806056932, 0.073952143178539265, 0.073977578420702406, 0.073977280338576756, 0.07402850772638358
7397668417482150 5, 0.073875638721128345, 0.073927304068290178, 0.073978273946306183, 0.0740297994785219
52,
0.074184597317554779, 0.074107794578500474, 0.074185392359239932, 0.07438847703619933, 0.0743353204514
98266,
0.074335320451498266, 0.074357773782972147, 0.074430600088587484, 0.074478388560166975, 0.0743230951468660
81, 0.074371480082490615, 0.074418671977168893, 0.074466360550694777, 0.074464471879455152, 0.0745126571136
88485, 0.07451007251799302, 0.074405849552777514, 0.074352396424487432, 0.074272912869329799, 0.0742457890259
52691, 0.074244795206769743, 0.074295862541654545, 0.074194522889289072, 0.073992936080757632, 0.0739940290291
95163, 0.073942505226268107, 0.074046487676047157, 0.074253540878687518, 0.074409527271106723, 0.0741066972
97785074, 0.0741609453657186, 0.073700279362251431, 0.07385507339102193, 0.073955918217074368, 0.07370345802
927547, 0.074219367776167999, 0.074372474041586534, 0.074478587372503124, 0.07448017787384198, 0.07442831384
6715666, 0.074323492713371575, 0.074320610362874406, 0.074265069481972867, 0.074313156079862441, 0.0742595030
918458813, 0.074412310426037909, 0.074512657113688485, 0.074384987288272253, 0.074359155142494915, 0.0744083
34494835859, 0.074301825767799645]
```

In [18]:
```
# 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)
```

```
[0.075955916031087006, 0.076767813408405938, 0.07757967852975349, 0.080016601748711655, 0.0783921460402279,
0.079204521377012491, 0.07920477052302527, 0.079205012341214159, 0.080830162885460355, 0.080018237818734475,
0.080424440277541379, 0.0800184302975607, 0.080018452506656013, 0.079205532616711455, 0.080018496924846694,
0.081642600724615016, 0.080830424621069208, 0.080018378476338234, 0.079205437355000671, 0.07676985892921099
6, 0.07798751710285437, 0.076769802109188628, 0.073523685043523121, 0.074335264306899781, 0.0743352643068997
81, 0.07392974848622896 3, 0.072712932779053899, 0.07190186496307914, 0.074336110208151632, 0.073524848215381
591, 0.072713740032302909, 0.071902669861533486, 0.07190279625054 6963, 0.071091763781724993, 0.0710919347838
13621, 0.072714607829545613, 0.073526154232906898, 0.074743576830524405, 0.075149468422118865, 0.07514953794
5363137, 0.074337994573541871, 0.075961102259697266, 0.079208155977669614, 0.079208075371606665, 0.080021043
567778144, 0.078396047998060783, 0.075148926140813616, 0.072714358926460498, 0.077583540080000804, 0.0767715
49324876273, 0.084081192615152026, 0.081644164256860885, 0.080020044158488166, 0.084080943695292162, 0.07595
9345407187018, 0.073524780193635481, 0.071901851658972463, 0.071901745226119002, 0.072713087502593282, 0.074
336082699167819, 0.074336282139300353, 0.075148119671180 18, 0.074336797932746612, 0.075148509001348049, 0.07
2714170567369063, 0.071091763781724993, 0.07311997182525927, 0.073525691685033354, 0.072714439651785395, 0.0
74337581938784883]
[ 0.081597  0.082465  0.083333  0.085938  0.084201  0.085069  0.085069
  0.085069  0.086806  0.085938  0.086372  0.085938  0.085938  0.085069
  0.085938  0.087674  0.086806  0.085938  0.085069  0.082465  0.083767
  0.082465  0.078993  0.079861  0.079861  0.079427  0.078125  0.077257
  0.079861  0.078993  0.078125  0.077257  0.077257  0.076389  0.076389
  0.078125  0.078993  0.080295  0.080729  0.080729  0.079861  0.081597
  0.085069  0.085069  0.085938  0.084201  0.080729  0.078125  0.083333
  0.082465  0.090278  0.087674  0.085938  0.090278  0.081597  0.078993
  0.077257  0.077257  0.078125  0.079861  0.079861  0.080729  0.079861
  0.080729  0.078125  0.076389  0.078559  0.078993  0.078125  0.079861]
```

In [19]:
```
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 [ ]: