Machine Learning Algorithm for Predicting Stock Price Based on Historical Time Series Price Data

Introduction and Overview

This document contains a detailed description of how the machine learning algorithm for stock price prediction works. Within the Django framework, the algorithm's code resides in the Django View (`views.py` file). The historical data and predictions data is then passed through to the Template and rendered visually using Google Charts.

What is LSTM?

LSTM (Long-Short Term Memory) is a type of Recurrent Neural Network. A neural network can be defined mathematically as a differentiable function which maps a certain variable to some other variable. LSTM is useful for time series predictions in that it can take in a certain length data sequence and predict a sequence of unequal length to the input sequence. LSTMs are well suited for making predictions based on time series data.

Code Walkthrough

The algorithm uses the following Python libraries shown:

```
import math
import pandas_datareader as web
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense, LSTM
```

First, the stock price data for some symbol is queried within a time range using `pandas_datareader`. Next, the data set is manipulated and scaled down from 0 to 1. It is good practice to scale data down when training the neural networks

```
df=web.DataReader('AAPL',data_source='yahoo',start='2012-01-01',end=`2020-12-1
8`)
#Create a new dataframe with only the 'Close' column
data = df.filter(['Close'])
#Convert the dataframe to a numpy array
```

```
dataset = data.values
#Define the number of rows to train the LSTM model on.
training_data_len = math.ceil(len(dataset)*.9)
#Scale the data.
scaler = MinMaxScaler(feature_range=(0,1))
scaled_data = scaler.fit_transform(dataset)
```

With the scaled data, the train data set is formulated. Next, the dependent training variable and independent training variables are created. The dependent variable `x_train` is the historical time series data. The independent variable `y_train` will be the predictions.

```
#Create the scaled training data set
train_data = scaled_data[0:training_data_len , :]
#Split the data into x_train and y_train data sets
#Independent Training Variable is x_train, this is the historical data.
x_train = []
#Dependent variable or Target Variable is y_train, this is the prediction.
y_train = []
```

The for loop iteration sets the predictions to be performed for the last sixty days, it takes the 60 day delimiter as the bounds for the predictions and the historical time series data.

```
for i in range(60, len(train_data)):
    x_train.append(train_data[i-60:i, 0])
    y_train.append(train_data[i,0])
    if i <= 60:
        print("x_train:", x_train)
        print("y_train:", y_train)
        print()
#Convert the x_training and y_training to numpy
x_train, y_train = np.array(x_train), np.array(y_train)
#Make the data two dimensional. LSTM needs three dimensions
x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))
x_train.shape
```

The next code block builts the LSTM model with 50 neurons and a second layer.

```
#Build the LSTM model
model = Sequential()
#LSTM has 50 neurons, second LSTM layer = True, input shape is the # of time
steps (60) and # features (1)
model.add(LSTM(50,return_sequences = True, input_shape =
(x_train.shape[1],1)))
#This is the second layer. It also has 50 neurons, but the return is false (no
other/more LSTM #layers)
model.add(LSTM(50, return_sequences = False))
#This is the densely connected neural network layer with 2 neurons
model.add(Dense(25))
model.add(Dense(1))
```

After the model is formulated and trained, a loss function is computed to evaluate how well the model did. The epochs is the number of times the data is passed through and back.

```
#Compile the model
#An optimizer is used to improve upon the loss function
#The loss function quantifies how well it did
model.compile(optimizer = 'adam', loss='mean_squared_error')
#Train the model
#Epochs is number of times data set is passed forward and backwards #through
the model
model.fit(x_train, y_train, batch_size = 1, epochs = 2)
#Create the testing data set
test_data = scaled_data[training_data_len - 60: , :]
#Create the data sets x_test and y_test
x_test = []
#prediction values are y_test
y_test = dataset[training_data_len: , :]
```

The next code block simply reshapes the data and sets it ready for it to pass through to the Template for visualization using Google Charts.

```
for i in range(60 , len(test_data)):
    x_test.append(test_data[i-60:i,0])
#Convert the data to a numpy array
x_test = np.array(x_test)
x_test.shape
#Reshape the data to be three dimensional
x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1))
x_test.shape
#Get the models predicted price values
predictions = model.predict(x_test)
predictions = scaler.inverse_transform(predictions)
#Get the root mean squared error (RMSE)
rsme = np.sqrt( np.mean(predictions - y_test )**2 )
```