Big Data Intelligence and Analytics, Volume 1, Issue 1, April 2018
BDIA 0000-0000

Stock Prediction using Machine Learning

Mohit Ramani, Soumiya Roy

Big Data Intelligence and Analytics
College of Engineering, Northeastern University

***Abstract*-** Stock market prediction is the act of trying to determine the future value of a company stock or other financial instrument traded on an exchange. The successful prediction of a stock's future price could yield significant profit. The efficient-market hypothesis suggests that stock prices reflect all currently available information and any price changes that are not based on newly revealed information thus are inherently unpredictable. Others disagree and those with this viewpoint possess myriad methods and technologies which purportedly allow them to gain future price information.

***Index Terms*-** Stock, Prediction, Machine Learning, Python

I. INTRODUCTION

The objective of this paper is to predict the stock price at (t+1) day utilizing the historical closing price of the stock. We are calculating Market Sentiment by calculating moving averages with window of 50 trading days. After checking the correlation between the input features, we're able to say that both features are highly correlated. Since, market sentiment is nothing but moving average of closing prices.
Feeding input features into Machine learning model popular for time-series analysis like LSTM and RNN, our output is only one feature which is closing price for the (t+1) day.

Step 1: Calculate Market Sentiment by calculating moving averages with window of 50 trading days. Checking the correlation between input features.
Step 2: Our input features are closing price and its moving average.
Step 3: Feeding input features into LSTM and regression analysis.
Step 4: Output is prediction of closing prices.
Step 5: Calculating r2score(variance) for each model that is- LSTM and Regression.

Machine learning Model utilized in the project:
Long Short Term Memory:
For understanding Long Short Term Memory, we need to understand Recurrent Neural Network, the chain-like nature reveals that recurrent neural networks are intimately related to sequences and lists. They are the natural architecture of neural network to use for such data.

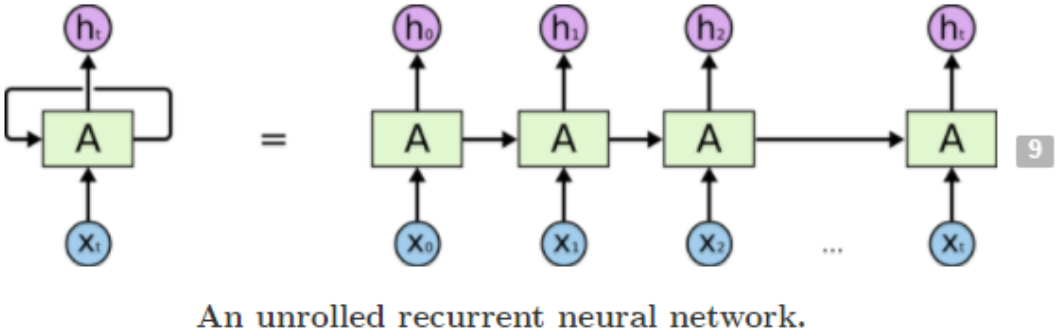

Fig 1: An unrolled recurrent neural network [CC By SA 4.0 (https://www.datasciencecentral.com/profiles/blogs/machine-learning-translation-and-the-google-translate-algorithm)]

LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior.
Ordinary least-squares (OLS) regression
Regression is used to evaluate relationships between two or more feature attributes. Identifying and measuring relationships lets you better understand what's going on in a place, predict where something is likely to occur, or begin to examine causes of why things occur where they do.

Ordinary Least Squares (OLS) is the best known of all regression techniques. It is also the proper starting point for all spatial regression analyses. It provides a global model of the variable or process you are trying to understand or predict; it creates a single regression equation to represent that process.

Machine Learning Techniques:

- Long Short Term Memory (LSTM)
- Recurrent Neural Network (RNN)
- Ordinary Least Square Regression (OLS)

II. IMPLEMENTATION OF THE MODEL

(i) Importing the necessary libraries

MODEL IMPLEMENTATION:

Import statements

In [3]:
```
import pandas as pd
import quandl
import matplotlib.pyplot as plt
from sklearn import preprocessing
from numpy import concatenate
from sklearn import linear_model
from sklearn.metrics import mean_squared_error
from math import sqrt
from keras.models import Sequential
from keras.layers import LSTM, Dense
import xgboost
import math
import statsmodels.api as sm
from sklearn.metrics import r2_score
```

(ii) Preprocessing data
Utilizing Quandl API to get the data # calculating correlation between closing prices and market sentiment Why we have utilized moving averages as a way to determine the Market Sentiment?
A moving average (MA) is a widely used indicator in technical analysis that helps smooth out price action by filtering out the “noise” from random price fluctuations. It is a trend-following, or lagging, indicator because it is based on past prices.

The most common applications of moving averages are to identify the trend direction and to determine support and resistance levels. Moving averages also impart important trading signals on their own, or when two averages cross over. A rising moving average indicates that the security is in an uptrend, while a declining moving average indicates that it is in a downtrend

```
def dataPreProcessing():
    quandl.ApiConfig.api_key = 'T6mx2AnLLbGx7N7hUz8s'

    selected = ['MSFT', 'MS']
    data = quandl.get_table('WIKI/PRICES', ticker=selected,
                            qopts={'columns': ['date', 'ticker', 'adj_close']},
                            date={'gte': '1998-01-24', 'lte': '2018-03-26'}, paginate=True)
    clean = data.set_index('date')
    tableDf = clean.pivot(columns='ticker')

    tableDf = tableDf.reset_index()
    tableDf.columns = tableDf.columns.droplevel(0)
    tableDf.columns = ['Date', 'MSFT', 'MS']
    morganStanleySeries = tableDf[['MS']]
    morganStanleySeriesMA = morganStanleySeries.rolling(window=50).mean()
    microsoftSeries = tableDf[['MSFT']]
    MicrosoftSeriesMA = microsoftSeries.rolling(window=50).mean()

    tableDf['MS_MA_50'] = morganStanleySeriesMA
    tableDf['MSFT_MA_50'] = MicrosoftSeriesMA
    #correlations for validation
    corr1 = tableDf['MSFT_MA_50'].corr(tableDf['MSFT'])
    print("correlation between MSFT_MA_50 and MSFT closing prices: ", corr1)
    corr2 = tableDf['MS_MA_50'].corr(tableDf['MS'])
    print("correlation between MS_MA_50 and MS closing prices: ", corr2)
    return tableDf[50:]
```

(iii) Function for plotting the Microsoft closing prices Vs Microsoft Market sentiment (Moving average with window of 50)

```
def plotting(df):
    plt.plot(df['Date'], df['MSFT_MA_50'])
    plt.plot(df['Date'], df['MSFT'])
    plt.legend(loc='best')
    plt.show()
```

(iv) Scaling the data (Pre-processing) for feeding into LSTM

```
def scalingMicroSoft(df):
    dfMicrosoft=df[['Date','MSFT','MSFT_MA_50']]
    dfMicrosoft=dfMicrosoft.set_index('Date')
    dfMicrosoftValues=dfMicrosoft.values
    encoder=preprocessing.LabelEncoder()
    dfMicrosoftValues[:,1] = encoder.fit_transform(dfMicrosoftValues[:,1])
    # ensure all data is float
    dfMicrosoftValues = dfMicrosoftValues.astype('float32')
    # normalize features
    scaler = preprocessing.MinMaxScaler(feature_range=(0, 1))
    scaled = scaler.fit_transform(dfMicrosoftValues)
    # frame as supervised learning
    reframedDf = series_to_supervised(scaled, 1, 1)
    # drop columns we don't want to predict
    reframedDf.drop(reframedDf.columns[[3]], axis=1, inplace=True)
    return reframedDf
```

(v) Function for splitting the data into train and test

```
def splitData(df):
    break_point=int(0.8 * len(df))
    test_size=0.2*len(df)
    values=df.values

    train=values[:break_point, :]
    test=values[break_point:,:]
    # split into input and outputs
    train_X, train_y = train[:, :-1], train[:, -1]
    test_X, test_y = test[:, :-1], test[:, -1]
    # reshape input to be 3D [samples, timesteps, features]
    train_X = train_X.reshape((train_X.shape[0], 1, train_X.shape[1]))
    test_X = test_X.reshape((test_X.shape[0], 1, test_X.shape[1]))
    #print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)
    return train_X,train_y, test_X, test_y
```

(vi) Logic for converting series to supervised # shifting the data frame

```
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
    n_vars = 1 if type(data) is list else data.shape[1]
    df = pd.DataFrame(data)
    cols, names = list(), list()
    # input sequence (t-n, ... t-1)
    for i in range(n_in, 0, -1):
        cols.append(df.shift(i))
        names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]
    # forecast sequence (t, t+1, ... t+n)
    for i in range(0, n_out):
        cols.append(df.shift(-i))
        if i == 0:
            names += [('var%d(t)' % (j+1)) for j in range(n_vars)]
        else:
            names += [('var%d(t+%d)' % (j+1, i)) for j in range(n_vars)]
    agg = pd.concat(cols, axis=1)
    agg.columns = names
    # drop rows with NaN values
    if dropnan:
        agg.dropna(inplace=True)
    return agg
```

(vii) LSTM Neural Network Model applied on two input features (closing price and moving average of the closing price)

```
def lstm_network(train_X, train_y, test_X, test_y, df):
    # design network
    model = Sequential()
    model.add(LSTM(50, input_shape=(train_X.shape[1], train_X.shape[2])))
    model.add(Dense(1))
    model.compile(loss='mae', optimizer='adam')
    # fit network
    history = model.fit(train_X, train_y, epochs=50, batch_size=72, validation_data=(test_X, test
                        shuffle=False)

    # plot history
    plt.plot(history.history['loss'], label='train')
    plt.plot(history.history['val_loss'], label='test')
    plt.legend()
    #plt.show()
    # make a prediction
    yhat = model.predict(test_X)
    test_X = test_X.reshape((test_X.shape[0], test_X.shape[2]))
    # invert scaling for forecast

    scaler = preprocessing.MinMaxScaler(feature_range=(0, 1))
    values=df.values
    scaled=scaler.fit_transform(values)

    inv_yhat = concatenate((yhat, test_X[:, 1:]), axis=1)
```

```
    inv_yhat = scaler.inverse_transform(inv_yhat)
    inv_yhat = inv_yhat[:, 0]
    # invert scaling for actual
    test_y = test_y.reshape((len(test_y), 1))
    inv_y = concatenate((test_y, test_X[:, 1:]), axis=1)
    inv_y = scaler.inverse_transform(inv_y)
    inv_y = inv_y[:, 0]
    # calculate RMSE
    rmse = sqrt(mean_squared_error(inv_y, inv_yhat))
    predictedList=inv_yhat.tolist()
    actualList=inv_y.tolist()
    print("r2score/variance of LSTM model", r2_score(actualList,predictedList))
    return actualList,predictedList
```

(viii) OLS regression Model

```
def OLSregression(df):
    X = df["MSFT"]  ## X usually means our input variables (or independent variables)
    y = df["MSFT_MA_50"]  ## Y usually means our output/dependent variable
    X = sm.add_constant(X)  ## let's add an intercept (beta_0) to our model
    model = sm.OLS(y, X).fit()  ## sm.OLS(output, input)
    predictions = model.predict(X)
#   Print out the statistics
#   print(model.summary())
#   print("actual vs pridected values using OLS Regression")
#     for item in zip(df["MSFT"], predictions):
#         print(item)
    score=r2_score(df["MSFT"], predictions)
    print("r2score/variance of the OLSRegression model", score)
    plotEndResult(df["MSFT"],predictions)
```

(ix) Function to plot two list

```
def plotEndResult(list1, list2):
    plt.plot(list1, label='actual values')
    plt.plot(list2, label='predicted values')
    plt.legend(loc='best')
    plt.show()
```

(x) Main function

```
def main():
    df=dataPreProcessing()
    plotting(df)
    dfMicrosoft = df[['Date', 'MSFT', 'MSFT_MA_50']]
    dfMicrosoft = dfMicrosoft.set_index('Date')
    scalingMicroSoftDf=scalingMicroSoft(df)
#   print(scalingMicroSoftDf.head(10))
    train_X, train_y, test_X, test_y=splitData(scalingMicroSoftDf)
    print("OLS Regression")
    OLSregression(dfMicrosoft)
    print("LSTM :-")
    actualList, predictedList=lstm_network(train_X, train_y, test_X, test_y, dfMicrosoft)
    print("LSTM Plot output")
    plotEndResult(actualList, predictedList)
```

Time series analysis: plotted correlation between closing prices and its moving average. After applying regression and LSTM, we are predicting the closing price for (t+1) day.
R2score/variance is one of the methods provided by scikit-learn to calculate the accuracy of the model. It ranges from 0-1, we are getting values of it above 0.96.
The time series analysis of test values- actual and predicted have been plotted.

III. RESULTS

Correlation between closing price and its market sentiment for Microsoft= 0.96.
We are utilizing R2score for calculating the accuracy of the model.
Definition of r2score: R^2 (coefficient of determination) regression score function. Best possible score is 1.0 and it can be negative (because the model can be arbitrarily worse). A constant model that always predicts the expected value of y, disregarding the input features, would get a R^2 score of 0.0.

Regression	
'explained_variance'	metrics.explained_variance_score
'neg_mean_absolute_error'	metrics.mean_absolute_error
'neg_mean_squared_error'	metrics.mean_squared_error
'neg_mean_squared_log_error'	metrics.mean_squared_log_error
'neg_median_absolute_error'	metrics.median_absolute_error
'r2'	metrics.r2_score

Fig 2: Regression Outcome [CC By SA 4.0 (Own Work)]

R2score/variance of the model (OLS Regression): 0.99
R2score/variance of the model (LSTM): 0.973

IV. DISCUSSION

We are able to get better accuracy using regression model. However, accuracy increases for LSTM with the increase in the data available. For regression model, variance doesn't change a lot. So, it can be implied that increasing the size of the data, we can increase the efficiency in the machine learning model like LSTM.

ACKNOWLEDGMENT

This research was supported by Northeastern University. We would like to acknowledge and extend our gratitude to our beloved Professor Nik Brown who helped us steer through the project with his valuable insights and knowledge on the subject. We thank our colleagues from Northeastern University who provided insight and expertise that greatly assisted the research, although they may not agree with all of the interpretations/conclusions of this paper.

REFERENCES

[1] https://link.springer.com/article/10.1007/s10898-011-9692-3
[2] https://ocw.mit.edu/courses/mathematics/18-s096-topics-in-mathematics-with-applications-in-finance-fall-2013/video-lectures/lecture-14-portfolio-theory/
[3] https://www.investopedia.com/walkthrough/fund-guide/introduction/1/modern-portfolio-theory-mpt.aspx
[4] https://www.dezyre.com/article/top-10-machine-learning-algorithms/202
[5] http://www.optimization-online.org/DB_FILE/2014/11/4625.pdf
[6] https://en.wikipedia.org/wiki/Black%E2%80%93Litterman_model

LICENSING

AUTHORS
First Author – Mohit Ramani, Graduate Student, Northeastern University, ramani.m@husky.neu.edu
Second Author – Soumiya Roy, Graduate Student, Northeastern University, roy.so@husky.neu.edu