Stock Market Prediction

CAPSTONE PROPOSAL

PREPARED BY

Kavya Kushnoor

Udacity - Machine Learning Nanodegree

Project Domain Background:

The domain of the project is financial trading data using publicly traded stocks. Trading data is well documented and made available by several online resources like Quandl, Google finance, yahoo finance, etc. However, cleaning this data and making it uniform for analysis and prediction is the challenge. The field of stock market predictions and investing is valued for its immense business potential.

Related Academic Research:

- Machine Learning for Trading- Udacity Course <https://www.udacity.com/course/machine-learning-for-trading--ud501>
- Vatsal H. Shah. (2007). “Machine Learning Techniques for Stock Prediction” <http://artent.net/2012/08/30/machine-learning-techniques-for-stock-prediction/>
- Gurav, Uma & Sidnal, Nandini. (2018). Predict Stock Market Behavior: Role of Machine Learning Algorithms. 10.1007/978-981-10-7245-1_38. <https://www.researchgate.net/publication/322611175_Predict_Stock_Market_Behavior_Role_of_Machine_Learning_Algorithms>
- Hegazy, Osman & Soliman, Omar S. & Abdul Salam, Mustafa. (2013). A Machine Learning Model for Stock Market Prediction. International Journal of Computer Science and Telecommunications. 4. 17-23. <https://www.researchgate.net/publication/259240183_A_Machine_Learning_Model_for_Stock_Market_Prediction>

Problem Statement:

There are several challenges in analyzing stocks data; viz:

- Discrepancy and accuracy across varied online resources
- Metrics and statistical methods to use for portfolio valuation
- Missing values in the historic datasets
- Since various companies trade at varying rates, normalizing data is essential for a fair comparison

In order to address these discrepancies, pandas dataframes have been used to clean and tabulate data for analysis. The steps taken to solve this issue have the script in the attached Jupyter Notebook. The steps are:

Step 1: Normalize stock prices in the portfolio by dividing each value with the first one.
Step 2: Assign the allocated amount of investment by multiplying it with each of the normalized values.
Step 3: Calculate possible values by multiplying allocated amount with the starting value
Step 4: Get the sum of all the values in each stock to find the daily return value of the portfolio.

Datasets and Inputs:

Quandl is the major source of stocks data for the portfolio. However, market data like S&P 500 is taken from investing.com as it required premium membership access in Quandl. The cleaned data files can be found in the Github repository.

For market data, visit: Market data
For portfolio data, visit: Portfolio data

The calculation script and explanations regarding the input, dimensionality, normalization and pre-processing of data can be found in the attached Jupyter Notebook. The data collected is over a period of 3.5 years, viz; July 1, 2015 to Dec 31, 2018.

Pre-Process the Data:

The /quandl dataframes have the following attributes:

- Date
- Open
- High
- Low
- Close
- Volume
- Ex-Dividend
- Split Ratio
- Adj. Open
- Adj. High
- Adj. Low
- Adj. Close
- Adj. Volume

A sample dataframe with all values for a single stock EXC is here:

Date	Open	High	Low	Close	Volume	Ex-Dividend	Split Ratio	Adj. Open	Adj. High	Adj. Low	EXC	Adj. Volume
2006-03-27	54.820	54.83	53.95	53.97	1652300.0	0.0	1.0	34.533570	34.539870	33.985518	33.998117	1652300.0
2006-03-28	53.975	54.41	53.30	53.65	2818800.0	0.0	1.0	34.001267	34.275293	33.576054	33.796535	2818800.0
2006-03-29	53.800	54.31	53.60	54.17	2045700.0	0.0	1.0	33.891027	34.212298	33.765038	34.124106	2045700.0
2006-03-30	53.800	54.03	52.79	53.19	3466700.0	0.0	1.0	33.891027	34.035914	33.254782	33.506760	3466700.0
2006-03-31	54.070	54.07	52.80	52.90	3627100.0	0.0	1.0	34.061112	34.061112	33.261082	33.324076	3627100.0

Of all these columns, we will use only ***Adj. Close***. This is because the trading data is very volatile and changes several times within a day. Adjusted closing price factors a stock's value to reflect any corporate actions such as dividends, rights offerings and stock splits.

Investopedia defines Adjusted Close as:

“The adjusted closing price amends a stock's closing price to reflect that stock's value after accounting for any corporate actions. It is often used when examining historical returns or doing a detailed analysis of past performance.”

For more information, visit this post.

This is the reason why you will notice that each of the stock's adjusted price has been renamed to its ticker symbol in the cell above.

On joining the individual stocks, we obtain a new dataframe. For sample data see table below:

Date	AAPL	AMZN	EXC	NDAQ	RGLD
2015-07-01	121.243068	437.39	29.107162	46.691199	58.987619
2015-07-02	121.089838	437.71	29.400710	47.093792	60.214716
2015-07-06	120.668456	436.04	29.299803	46.739127	61.282387
2015-07-07	120.371574	436.72	30.061194	46.815812	59.548025
2015-07-08	117.383593	429.70	29.905246	45.876428	59.678465

A summary of this entire dataframe is below:

```
df.describe()
```

	AAPL	AMZN	EXC	NDAQ	RGLD
count	629.000000	629.000000	629.000000	629.000000	629.000000
mean	122.975465	778.805509	33.152518	64.564981	64.829244
std	24.597165	186.382364	4.108439	8.047604	17.093386
min	88.288161	429.700000	23.831237	45.876428	24.741683
25%	105.059007	625.900000	30.232317	59.958759	49.692324
50%	113.767235	766.770000	33.346851	66.043219	67.648195
75%	144.473805	948.430000	35.500436	69.816866	79.215683
max	176.420000	1195.830000	42.390000	79.270000	93.978947

On plotting this entire dataframe, we obtain the following results:

We notice the discrepancy since Amazon is trading at a higher range compared to the other stocks. To overcome this and make a fair comparison, we use data normalization. We will make them all begin at one fair point so that the comparison is on an equal footing. We will normalize price data, so that each value begins at 1.

```
def normalize_data(df):
    df = df/df.iloc[0, :]
    return df
df = normalize_data(df)
df.head()
```

Date	AAPL	AMZN	EXC	NDAQ	RGLD
2015-07-01	1.000000	1.000000	1.000000	1.000000	1.000000
2015-07-02	0.998736	1.000732	1.010085	1.008622	1.020803
2015-07-06	0.995261	0.996914	1.006618	1.001026	1.038903
2015-07-07	0.992812	0.998468	1.032777	1.002669	1.009500
2015-07-08	0.968167	0.982418	1.027419	0.982550	1.011712

On plotting the normalized data, we can observe that the stock prices move more closely together. See image below for visualization of the stocks movement.

Assign Weightages:

We will now use an investment value and assign weights to each stock in the portfolio. Assuming the ***investment value is $10,000*** and we assign weightages in the following order:

Stock	**Weights**
Apple	25%
Amazon	25%
Exelon Corporation	10%
Nasdaq	30%
Royal Gold Inc.	10%

We now assign a hypothetical investment amount of $10000 to this portfolio. The sum of all the stocks will provide a portfolio value at the end of each day. See table below for sample:

Date	AAPL	AMZN	EXC	NDAQ	RGLD	Portfolio Val
2015-07-01	2500.000000	2500.000000	1000.000000	3000.000000	1000.000000	10000.000000
2015-07-02	2496.840442	2501.829031	1010.085093	3025.867378	1020.802621	10055.424566
2015-07-06	2488.151659	2492.283774	1006.618342	3003.079450	1038.902539	10029.035764
2015-07-07	2482.030016	2496.170466	1032.776552	3008.006569	1009.500410	10028.484013
2015-07-08	2420.418641	2456.046092	1027.418847	2947.649353	1011.711712	9863.244645

A summary of this entire dataframe is below:

```
df.describe()
```

	AAPL	AMZN	EXC	NDAQ	RGLD	Portfolio Val
count	629.000000	629.000000	629.000000	629.000000	629.000000	629.000000
mean	2535.721572	4451.436411	1138.981482	4148.425086	1099.031372	13373.595923
std	507.187034	1065.309928	141.148745	517.074148	289.779223	2352.693243
min	1820.478525	2456.046092	818.741340	2947.649353	419.438576	9500.286457
25%	2166.288949	3577.470907	1038.655613	3852.466429	842.419570	11281.620234
50%	2345.850303	4382.644779	1145.657948	4243.404754	1146.820239	13302.268737
75%	2979.011651	5420.962985	1219.646102	4485.868853	1342.920516	15365.576956
max	3637.733744	6835.032808	1456.342620	5093.251034	1593.197851	18309.025416

Mean value of portfolio is: 13373.59. ***10% of mean portfolio value is: 1337.36.*** We will use this value to check the accuracy of the model by comparing it with RMSE later in the report.

We now have all the dependent variables ready in the form of the column 'Portfolio Val'. For the independent variable, we will use market data that is derived from S&P500 daily values. This data is obtained from investing.com.

Solution Statement:

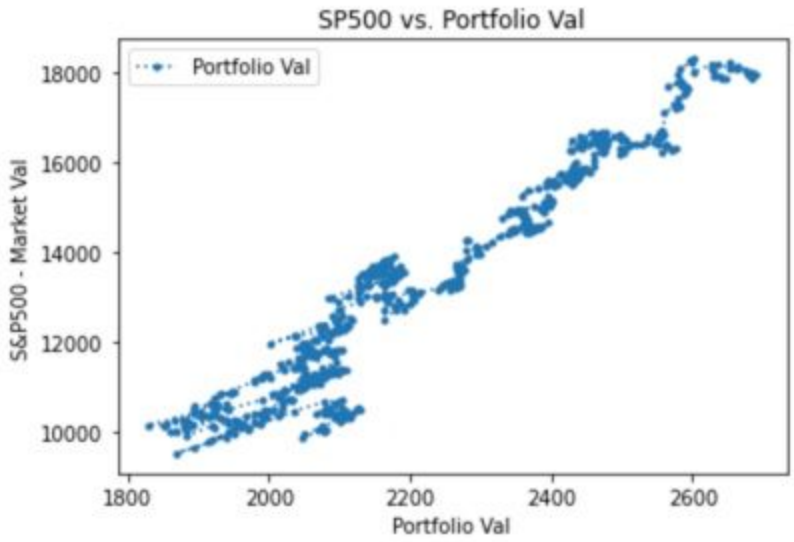
A regression analysis has been performed to make the predictions in the benchmark model. The measure is univariate regression analysis, which is the same as the Capital Asset Pricing Model (CAPM) equation. The predictions made using the model are compared with the test values. The regression library used is:

- Linear Regression

Preparing the Data:
In the next step, we will assign the independent variables array to 'y' and dependent variables array to 'x'. In this univariate simple regression analysis, we have two columns. We predict the return on the portfolio using the market fluctuations and relevant data. Hence, the attribute set (y-variable) is S&P 500 data and the label set (x-variable) is Portfolio value. We can execute the following script to extract these two variables:

```
y = dfXY[["Portfolio Val"]].to_numpy()
x = dfXY[["SP500"]].to_numpy()
```

Plotting these arrays on a scatter plot, we get the following chart:



We will now split this data into two sets for training and testing. We will use Scikit-Learn library and the built-in train_test_split() method. Execute script below to obtain the required results:

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=0)
```

```
(array([-11065.28056245]), array([[10.99550251]]))
```

We have the model coefficient = 10.99 and intercept = -11065.28. This means that for every one unit of change in the S&P 500 data, we can expect a 10.99% corresponding change in the portfolio value. By specifying test_size = 0.2, we split the data, 80% into the train set and 20% into the test set.

Train the Algorithm:

We train the algorithm using the following script:

```
model = LinearRegression().fit(x_train, y_train)
```

After training the algorithm, we will verify the accuracy of the model's predictions.

```
y_pred=model.predict(x_test)
```

We will compare the predicted values y_pred with the actual output values y_test and plot them in a table. A sample of the table is below:

	y_test	y_pred	difference
0	11880.345449	11821.088231	59.257218
1	13544.718360	12682.036078	862.682283
2	15872.732578	15799.480950	73.251627
3	13639.776538	12972.647209	667.129328
4	13325.314189	13779.387229	-454.073040
...	...	...	...
121	13708.398660	12684.235178	1024.163482
122	10856.664474	10354.508106	502.156368
123	15157.826232	15290.059319	-132.233087
124	10190.723381	12048.915043	-1858.191662
125	13679.571030	12532.167379	1147.403651

We will evaluate the accuracy of the model using three commonly used methods:

1. Mean Absolute Error : It is the mean of the absolute value of the errors.

$$\frac{1}{n}\sum_{i=1}^{n}|Actual - Predicted|$$

2. Mean Squared Error: It is the sum of the mean of the squared errors.

$$\frac{1}{n}\sum_{i=1}^{n}|Actual - Predicted|^2$$

3. Root Mean Squared Error: It is the square-root of the sum of the mean of the squared errors.

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}|Actual - Predicted|^2}$$

We find these values by executing this script:

```
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
Mean Absolute Error: 496.2985370004204
Mean Squared Error: 435980.02522676234
Root Mean Squared Error: 660.2878351346194
```

RMSE is 660.28. The 10% of mean portfolio value is: 1337.36. Since RMSE is lower that 10% of the mean portfolio value, we can conclude that this model did a fairly good job of predicting values.

Benchmark Model:

A benchmark model is attached as a Jupyter Notebook with this submission. The final project will follow this benchmark model as a foundation and enhance it.

Simple Linear regression has been used for predictive analysis. It is examining two things:

a. Is the independent variable (predictor) accurate in predicting the dependent variable (outcome)?
b. Depending on the beta, how strong is the fit?

The regression estimates clarify the relationship between the two variables and the relationship between them.

Simple Linear regression is defined by the formula:

$$y = c + b*x,$$

where,
y = estimated dependent variable score
c = constant
b = regression coefficient
x = score on the independent variable

The linear regression algorithm fits a line on the data points which has the least resulting error.

Evaluation Metrics:
The major evaluation metric used is Root mean square error.

Project Design:

The reason for selecting Regression Analysis is because it uses the same principles of Capital Asset Pricing Model to predict stock prices.

Investopedia defines CAPM as:

"The Capital Asset Pricing Model (CAPM) describes the relationship between systematic risk and expected return for assets, particularly stocks. CAPM is widely used throughout finance for pricing risky securities and generating expected returns for assets given the risk of those assets and cost of capital."

For more information, visit this post.

Capital Asset Pricing Model (CAPM): Regression Analysis:

In formal financial language, the regression model works with the same underlying principles as the Capital Asset Pricing Model (CAPM) equation. This is the reason why Regression analysis has been used as the model of choice.

$$r_p(t) = \beta_p r_m(t) + \alpha_p(t)$$

where,

$r_p(t)$: Return on portfolio for a given day *t*
β_p: Co-efficient
r_m: Market returns; viz S&P 500 values
$\alpha_p(t)$: Intercept

The CAPM emphasizes that the return on any stock or portfolio is due to the markets. This is why $r_m(t)$ is the independent x-variable in this equation. Since we are predicting the returns on the portfolio value, it is the dependent y-variable in this equation, represented by $r_P(t)$.

The final project will build on this attached benchmark project. It will beat the benchmark project by using the following features:

- Multivariate Regression Analysis:
  The additional independent variables using stock prices will be:
  - Bollinger Bands

$$\textbf{Bollinger Bands} = \frac{\textbf{Price(t) - Simple Moving Average (t)}}{\textbf{2 * Standard Deviation (t)}}$$

where each of these values are taken in a given timeframe window of 20-21 days.

  - Simple Moving Average

$$\textbf{Simple Moving Average} = \frac{\textbf{Price(t)}}{\textbf{Price[t - n:t].mean()}} \textbf{ -1}$$

where each of these values are at a given time t, and n which is the time frame window.

  - Momentum

$$\textbf{Momentum} = \frac{\textbf{Price(t)}}{\textbf{Price (t-n) -1}}$$

where each of these values are at a given time t, and time n which is the beginning of the time frame window.

These additional variables will be applied to the final portfolio value. The additional variables will be used on top of the market-returns variable that is already used in the benchmark's univariate regression analysis. The final project will also be deployed on AWS Sagemaker using XGBoost.