QuantFy: Predictive Market Behavior App using Machine Learning

Balagangadhar Bathula

March 17, 2017

Abstract

Investment firms, hedge funds and even individuals have been using financial models to better understand market behavior and make profitable investments and trades. To maximize return of investment these firms use large amounts of historical data and apply machine learning algorithms to the process. QuantFy is an application (app) that would build predictive/optimized models using machine learning models such as linear regression, random forests and k-nearest neighbour.

Contents

List of Figures

List of Tables

This is description document for the Capstone project for Machine Learning Nanodegree program of Udacity.

1 Domain Background

Many financial, high frequency trading (HFT), and hedge fund companies are analyzing trading strategeis including algorithmic steps from information gathering to market orders using machine-learning approaches. This project proposes to apply these probabilistic techniques to make trading decisions. Using approaches such as linear regression, Q-learning, k-NN and regression trees and apply them to the actual trading situations.

This application software will be build using Python tools [Hil]. The book [RB] provides a comprehensive overview of domain knowledge that would be required to understand the financial market world. Also the speicifications given [Bal16] will be used as guidelines to build this application.

2 Problem Statement

Using the historical data for prices and performance statistics as features, we can get predicted future price of the stock. For instance we can set of performance statistics (such as Bollinger bands, price-to-earning (P/E) ratio) as features of the present date mapped to price (say one week forecast which is an input) of price. This creates a data set $< X, Y >$, where X is the set of features and Y as trading days. Features used are the measurable quantities that a particular stock could use in the predicting things such as change in price, market relative change in price or simply future stock price. Few questions that one would ask, before getting started are-

1. Breath and depth of data: How much historical data one would like to consider?
2. What ticker symbols are you going to use?

General machine learning models that can be used for this problem are:

1. Regression
2. k-nearest neighbors
3. Random Forests
4. Time-series analysis: This analysis will be used to estimate the commodity prices such as oil/gas etc. For instance we can observe long term trend in the time-series (such as exponential) and fit an exponential model to predict the future price. Using the concepts of drift and seasonality, build models for commodity stocks.

3 Project Design

Design phase of the QuantFy application consists of following phases:

- Time-series plots for the various stocks can be visualized at this link: Price plot [Bat17b]. This link also computes various other technical indicators such as the SMA, Bollinger bands, etc.
- Portfolio optimizer: Analyzing using stock price data, and using numerical optimization techniques, this has already been implemented in the app. Link to the app is **Portfolio optimizer** [Bat17a]
- Investment and Trading using machine learning: This was be built using the machine-learning stratagies such as (1) k-nearest neighbors (2) Random forests (3) Linear regression and (4) Ridge regression. One can compare the performance of any two algorithms in the application. For more details, visit the application at **ML models**

3.1 Software

1. Front-end: Python `flask` based web application
2. Back-end: Python libraries, such as `scikit-learn, pandas, numpy, scipy, bokeh`.
3. Hosting platform: Heroku

This application can be accessed at: **QuantFy** [Bat17c]. The current version of the source code for this application can be found at this GitHub link: **Source code** [Bat17d]

4 Datasets and Inputs

There are many open APIs to extract data:

1. Yahoo Finance:
   (a) Using an API call `http://ichart.finance.yahoo.com/table.csv?s={YOUR_SYMBOL}` we can query and get historical data.
   (b) We can also use `yahoo_finance` python library or,
   (c) We can also use `pandas_datareader`
2. Quandl: Using `quandl` python API we can get the historical data
3. Bloomberg API

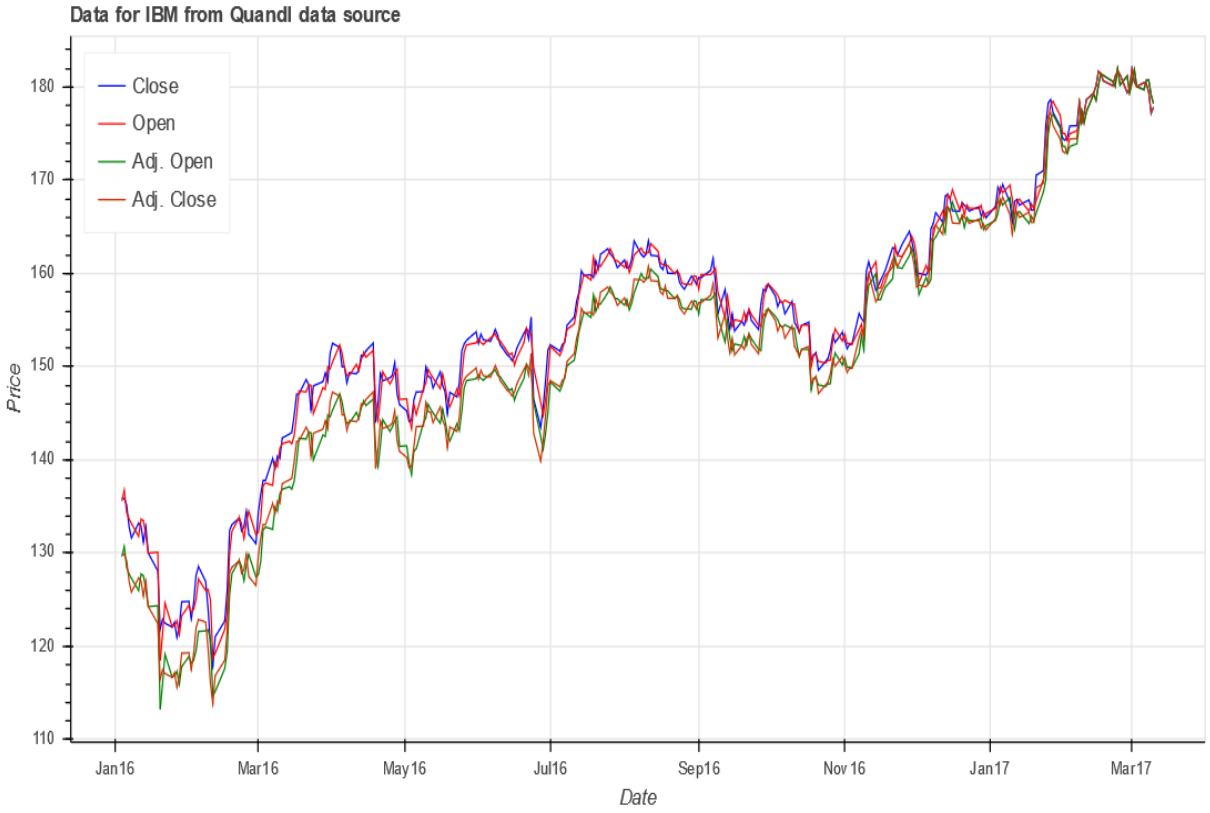

Figure 1: IBM prices from Jan 1 2016, from Quandl data source

Metric	IBM	TSLA	AAPL
Count	300	300	300
Mean	151.4008	216.3058	107.5923
Standard deviation	15.7737	25.3987	12.2136
Minimum	113.8011	143.67	89.47
25% percentile	143.5718	199	96.9126
50% percentile	151.7834	215.205	107.1624
75% percentile	151.7834	215.205	107.1624
Maximum	151.7834	215.205	107.1624

Table 1: Description of the data for the stock symbols IBM, TSLA, AAPL

Current version of QuantFy application proposes to uses Quandl to get historical data and Yahoo Finance source will be added in the subsequent versions. One can extract more than 10 years of historical stock prices for many ticker symbols.

For instance user can specify the date range for viewing the stock prices of ticker symbols. Then an request call will be made and acquired data will be converted into a `pandas` dataframe. This data will also be stored locally, using python libraries such as `ediblepickle`, so when the same request is made (no change in the parameters), data is retrieved locally. Since the data is generated online (rather than locally available data), we are not particularly concerned about the storing the data. At least from the past experience, the stock price data can be easily stored in the dataframe without any memory errors. We anticipate that the historical prices for the each ticker symbol will be in the order of less than 1 MB.

1. Close price and/or Adjusted close price
2. Start price and/or Adjusted start price,
3. Opening price and/or adjusted opening price

The data from these sources consists of features like:

Snapshot of the plot prices for IBM, TSLA (Tesla) and AAPL (Apple) are shown in Figure 1, 2, and 3 respectively. For more detailed refer to the application website **QuantFy** [Bat17c]. Table 1, describe the parameters for the data, such as count, mean, standard deviation etc. This can is achieved using the `describe()` option in pandas.

5 Analyzing the stock price data

When getting the stock price data for the ticker symbols, one need to clean the data to see if there are any un-available data points. For instance the stock has not traded

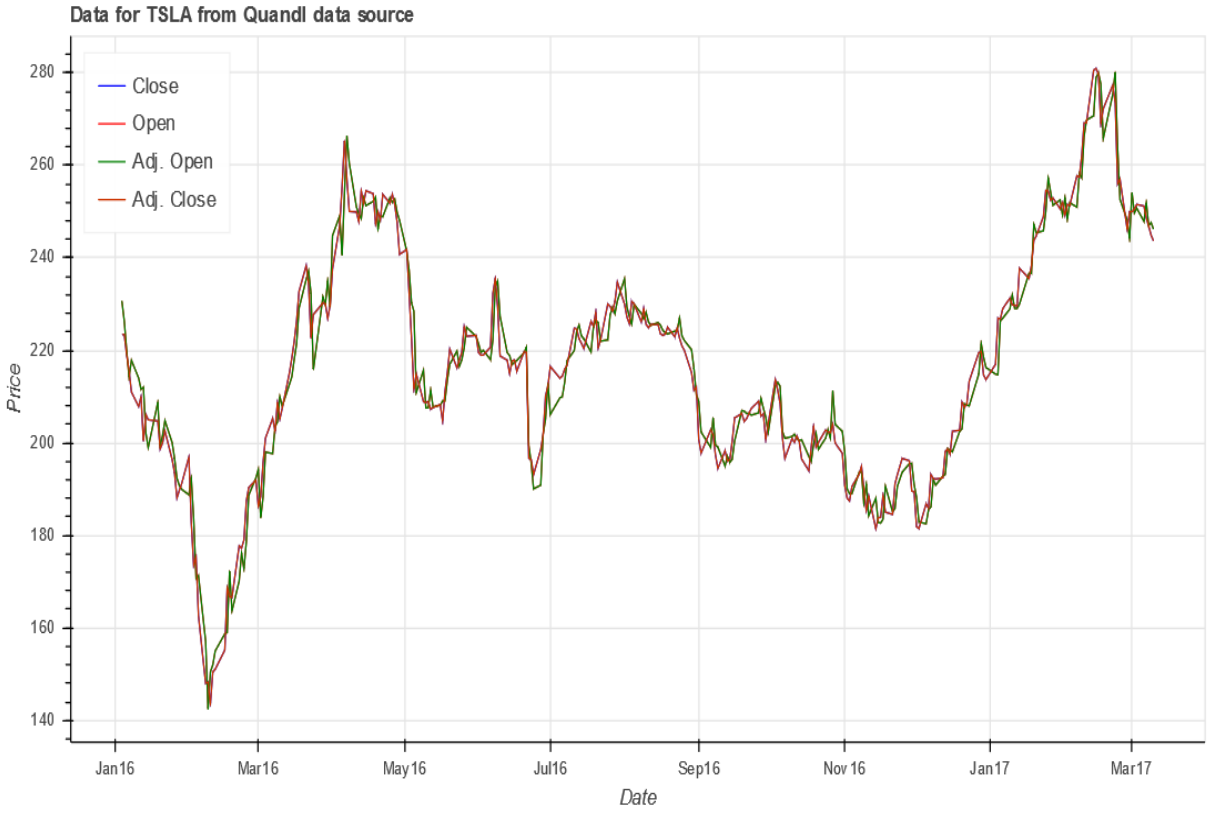

Figure 2: Tesla prices from Jan 1 2016, Quandl data source

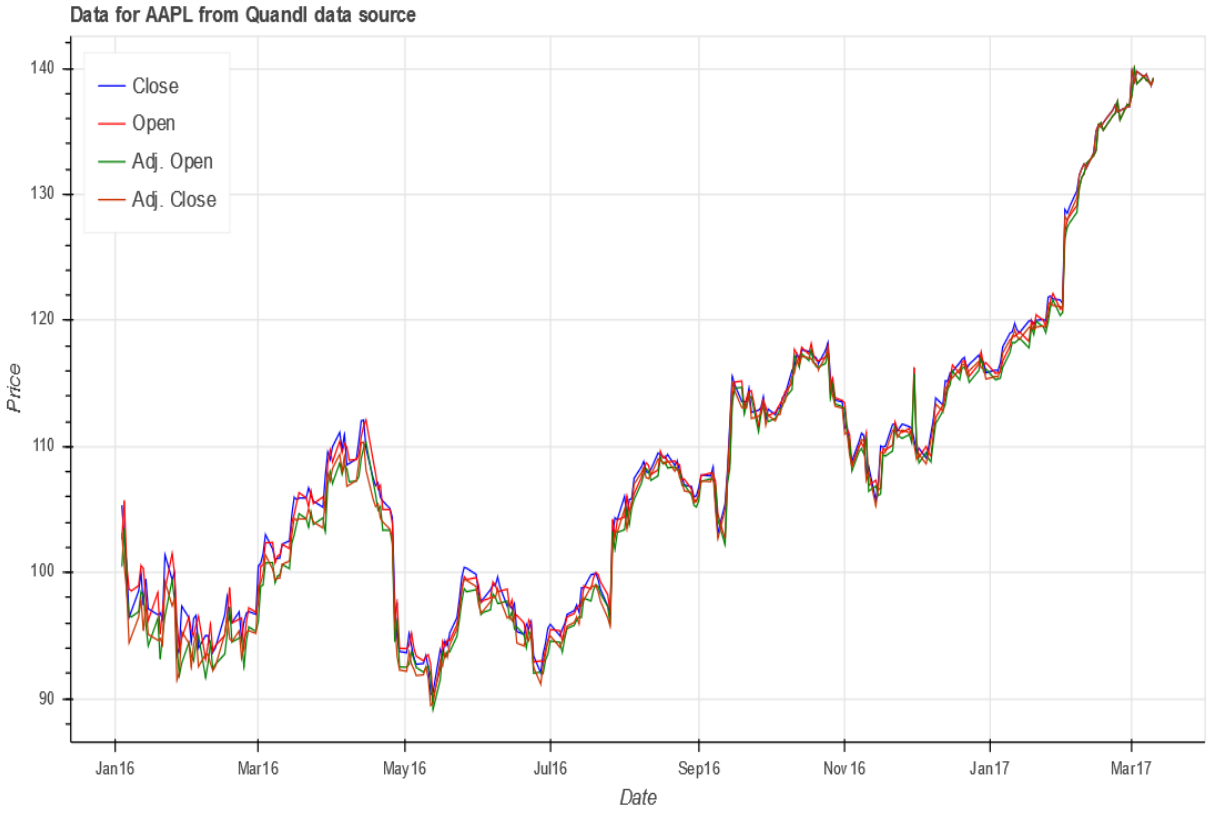

Figure 3: Apple prices from Jan 1 2016, Quandl data source

a day, then it is mostly likely the exchange is closed. These `NaN` values have to be removed from the data-set. The most common way to do it this is to align the ticker symbol with the a benchmark symbol, say S&P500 company and drop any `NaN` values in the data set. Also, we drop dates at which the S&P500 has not traded. Sometimes the company's stock price might not be available, as it did not have an IPO (initial public offering) for the date. So we need to remove such data points as well. For example if we get the stock price of the `GOOG` from year 2010, then there will be `NaN` until year 2014. In `pandas` we can accomplish this with `df=df.dropna(axis=0)`, where `df` is the dataframe, and `dropna()` drops any values with `NaN` along the rows (`axis=0`).

We use adjusted close price is what we use for the rest of the discussion. Along with these features, one could compute other features like:

1. Daily returns (DR): How much the price will go up or down on a particular instance of time:
$$r(t) = \frac{p(t)}{p(t-1)} - 1, \tag{1}$$
where $p(t)$ is the price at time t.

2. Average daily return (ADR): This is simply the average value of the the daily returns.
$$adr = \frac{\Sigma_{t=1}^{N} r(t)}{N} \tag{2}$$

3. Cumulative daily return (CDR): Defined as the ratio of the final value of the the daily return to the initial value of the daily return
$$cdr = \frac{r(N)}{r(0)}, \tag{3}$$
where N is the last sample point in the series.

4. Standard deviation of DR: It is simply the standard deviation of the daily return $(r(t))$, which is the square-root of the variance.

5. Cumulative return is given by,
$$cr(t) = \frac{p(t)}{p(0)} - 1 \tag{4}$$

6. Simple moving average (SMA): Also know as moving average
```
rolling_df=pd.DataFrame(index=df.index)
rolling_df =pd.Series.rolling(df,window=window).mean().to_frame()
rolling_df.ix[:window, :] = 0
```
Table 2 shows the computed parameters for the data. Also, the other parameters are plotted in Fig. 4, 5 and 6 respectively.

Metric	IBM	TSLA	AAPL
Cummulative daily return	0.368	0.091	0.348
Average daily return	0.0011	0.00056	0.0011
Standard deviation of daily return	0.012	0.023	0.014
Sharpe ratio	1.49	0.38	1.23

Table 2: Parameters computed for the stock symbols IBM, TSLA, AAPL

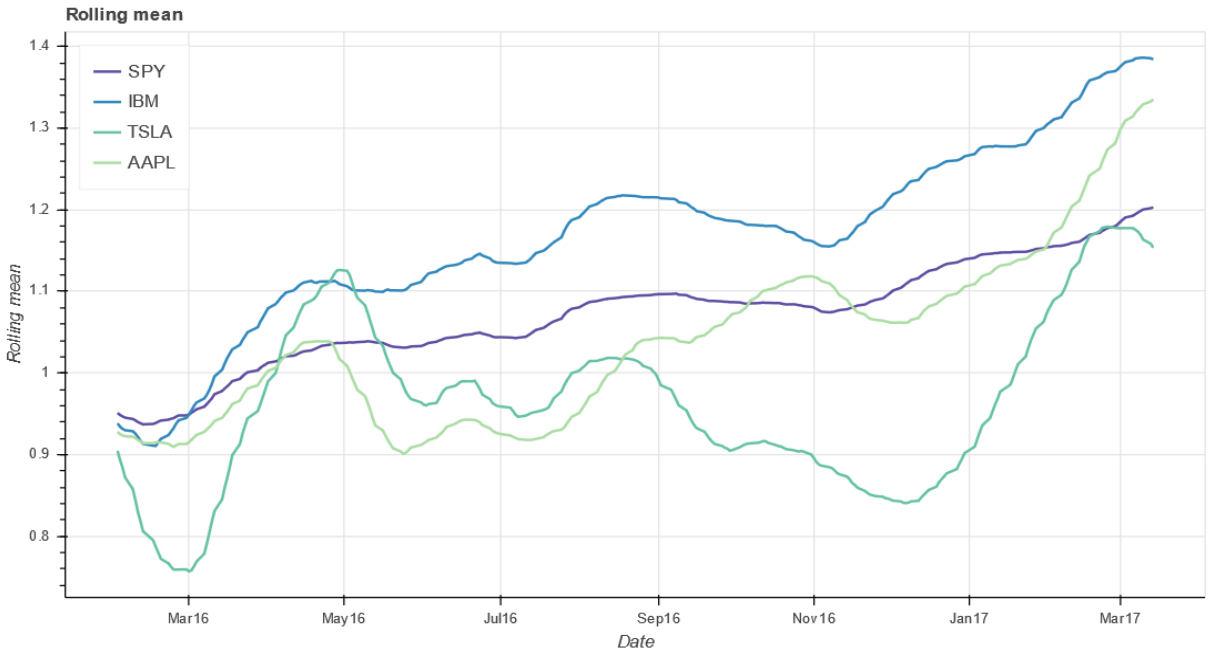

Figure 4: Rolling mean for the symbols IBM, TSLA and AAPL

7. Bollinger bands:

$$\begin{aligned} U_{band} &= m_r + 2 \times \sigma_r, \\ L_{band} &= m_r - 2 \times \sigma_r \end{aligned} \tag{5}$$

where m_r is the rolling mean and σ_r is the rolling standard-deviation of the adjusted close price.

8. Normalized Bollinger bands: Value varies between -1.0 and 1.0.

$$bb_{norm} = \frac{price[t] - SMA[t]}{2 \times \sigma[t]} \tag{6}$$

where σ is the rolling standard deviation and SMA is the simple moving average.

9. Sharpe ratio This metric adjusts the return for risk, also know as risk adjusted reward.

$$sr = \frac{R_p - R_f}{\sigma_p}, \tag{7}$$

where R_p is the portfolio return, R_f is risk-free return and σ_p is the standard deviation of portfolio return.

$$sr = \frac{E[R_p - R_f]}{\sigma[R_p - R_f]} \tag{8}$$

Sharpe ratio varies depending on how frequently you sample the data. $sr_f = k \times sr$, where $k = \sqrt{\#\text{ samples/year}}$

10. Volatility is nothing but the standard deviation of the daily returns.

$$v(t) = \sqrt{var[r(t)]} \tag{9}$$

11. Momentum is defined as:

$$r(t) = \frac{p(t)}{p(t-N)} - 1 \tag{10}$$

where N is the size of the window.

Plots for rolling mean, rolling standard deviation and bollinger-bands are show in Fig. 4, 5, and 6 for IBM, TSLA and AAPL. For more details, refer to the application website [Bat17b]

6 Algorithms used in the project

This section provides brief description of the algorithms used in this project.

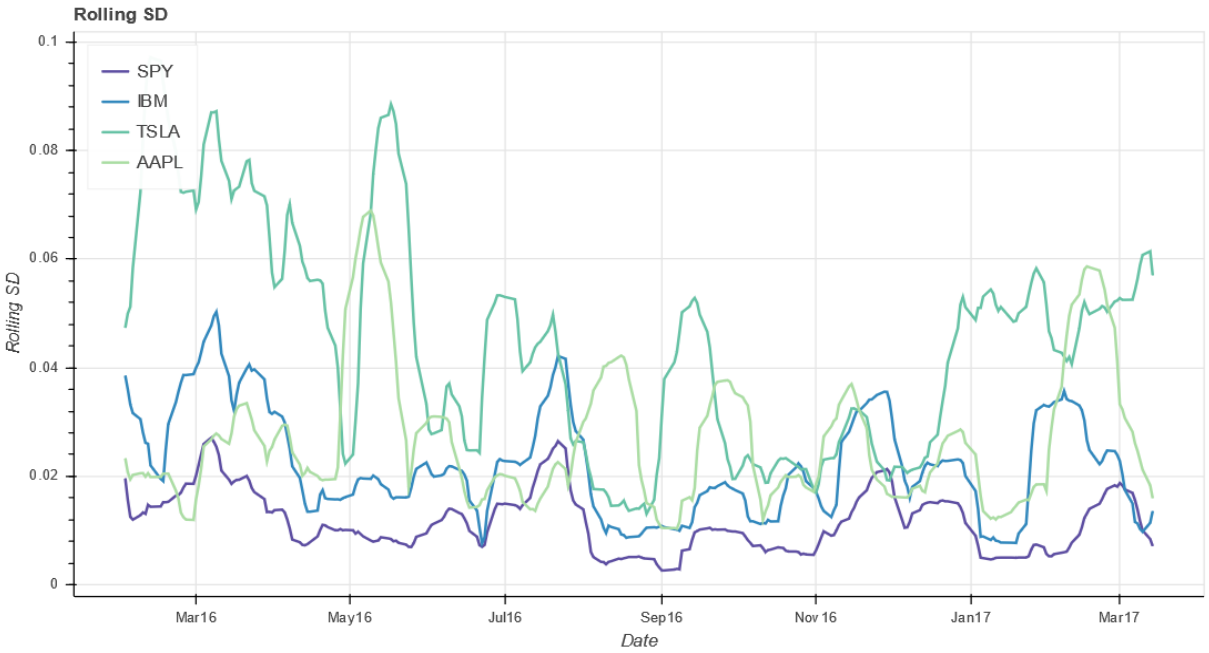

Figure 5: Rolling standard deviation (σ) for the symbols IBM, TSLA and AAPL

6.1 Convex Optimization

A real-valued function defined on an interval is said to be convex if the line-segment between two points on the graph of the function lies above or on the graph. More formally, let $\mathbf{X}$ be a convex set in a real vector space and let $f : \mathbf{X} \to \mathbb{R}$ be a function, then f is called **convex** if:

$$\forall x_1, x_2 \in \mathbf{X}, \forall t \in [0, 1] : f(tx_1 + (1-t)x_2) \leq tf(x_1) + (1-t)f(x_2) \tag{11}$$

In finance and economics, *convex optimization* plays an important role. Convex optimization algorithms are easy to solve, since they have one local minimum, which is also a global minimum. They can be solved using gradient-descent or other numerical optimizations as Sequential Least SQuare Programming (SLSQP). Fig. 7 shows an example of the convex function.

- Given set of assets and time-period, find allocations of funds to assets that maximizes performance.
- Performance metrics can be:
  - $\max cr(t)$,
  - $\min \sigma(t)$, i.e., minimizing volatility.
  - $\max sr$

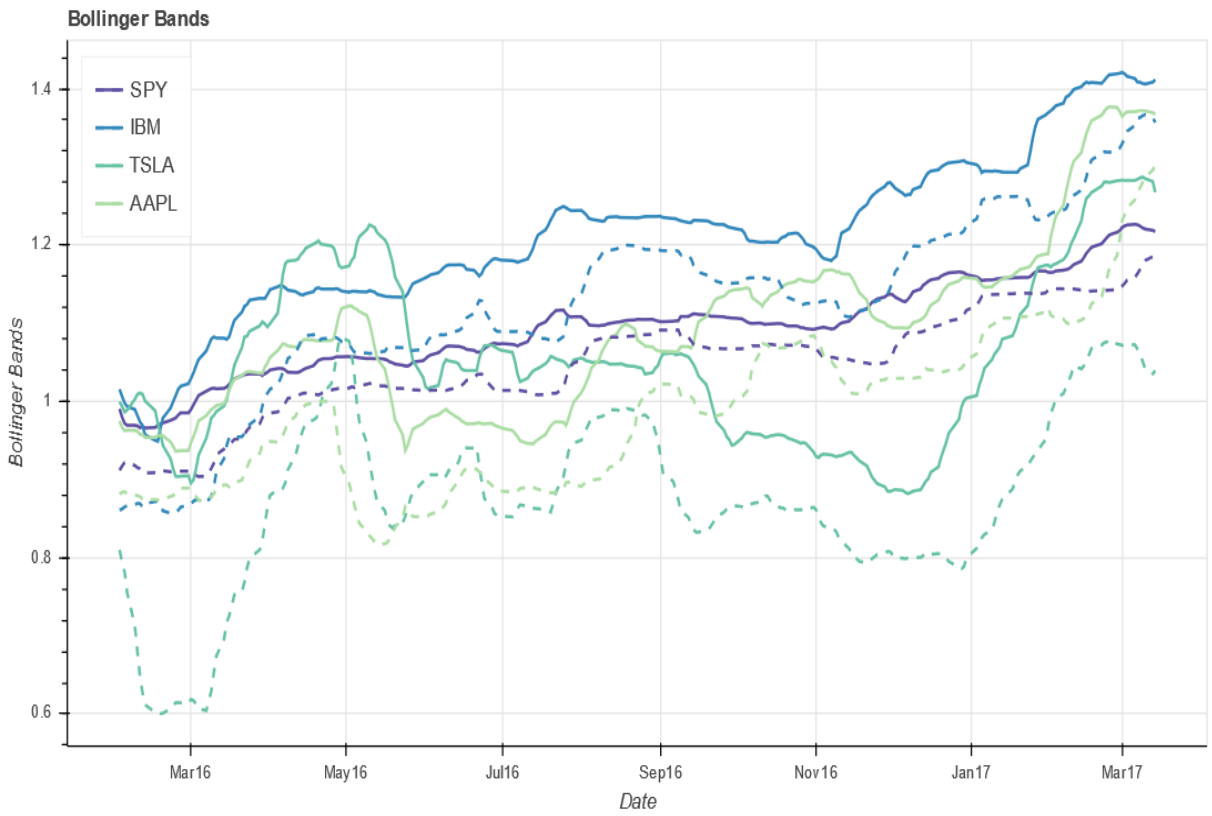

Figure 6: Bollinger bands IBM, TSLA and AAPL; upper are shown in solid lines and lower bands are shown in dotted lines

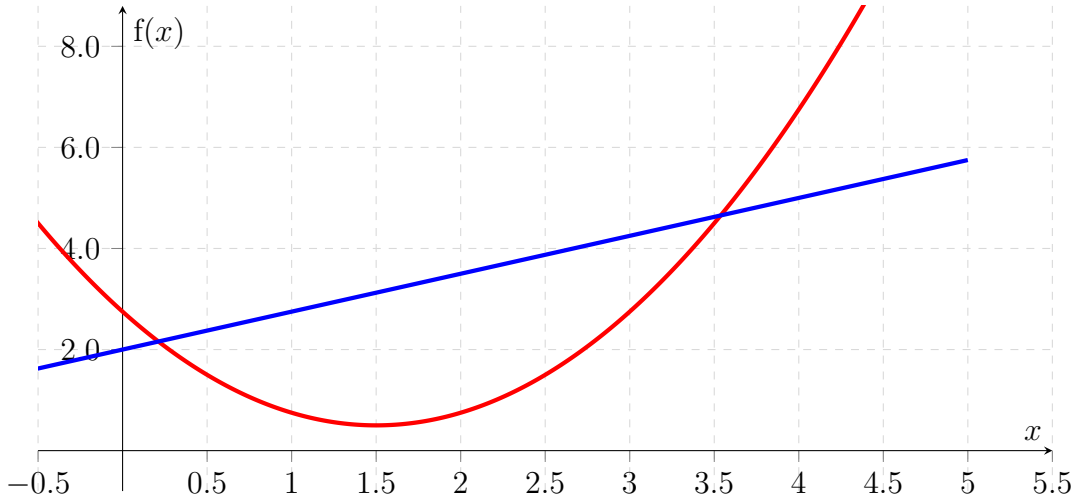

Figure 7: Example of the convex function $f(x) = (x - 1.5)^2 + 0.5$. The blue line describes the property of convex function in Eq. (11)

– Framing the problem:
  1. Provide a function $f(X) = cX + b$ to minimize.
  2. Provide an initial guess for allocations.
  3. Call the optimizer.
– Constraints: We know that allocations for each stock symbol in the portfolio should be ≤ 1, i.e.,
$$\Sigma_{i=0}^{k}|X_i| = 1.0,$$
where X_i's are allocations for each stock.

Below is the code snippet of the optimization in `scipy`.

```
import scipy.optimize as spo
initial_alloc=[(1.0/len(sym))]*len(sym)
bounds=((0,1),)*len(sym)
opt_allocs=spo.minimize(sharpe_function*-1, initial_alloc,
        args=df, method='SLSQP',bounds=bounds,
        constraints=({'type':'eq','fun':lambda
        opt_allocs: 1-np.sum(np.abs(opt_allocs))}
        )
        );
optimal_allocs=opt_allocs.x
```

In this project, we are trying to predict the stock- future prices values, using certain features such as Volatility Eq. (9), Momentum Eq. (10) and Normalized Bollinger Bands (BB) Eq. (6). This problem will be a regression problem, as the output that we are trying to predict is a real-number. The next part of the section describes the details of the algorithms that we use in this project. Also the problem of stock prediction falls under the category of supervised learning problem, as we are given with the y_i label set. In other words we trying to build a model f, while mapping the feature row X_j to each label y_j, so that:

$$f(X_j) \approx y_j$$

Regression algorithms can be of two types:

1. Parametric: In this type of algorithms we use data to calculate/estimate the parameters of an equation such as $y = m \times x + c$, where m is the slope of the line and c is the y-intercept. Linear regression, Random-forest are examples of parametric regression algorithms.
   Pros and Cons:

- We do not have to store the original data, so these algorithms are space efficient.
- We cannot easily update the model as more data is gathered. We have to do complete rerun of the learning algorithm to update. Thus training usually is slow, but querying (or predicting) is fast.

2. Non-parametric: In this type of algorithms we take the data to construct the information rather than using the predetermined model. Algorithms such as k-nearest neighbor is an example.
   Pros and Cons:
   - We have to store all the data points. So for huge datasets it is inefficient
   - But as the new evidence comes, the model can be easily trained.

6.2 Linear regression

In the simplest form the Linear regression can be expressed as :

$$f(\mathbf{X_j}) = \Sigma_i \beta_i \mathbf{X_{ji}}, \tag{12}$$

where β_j are the co-efficients.

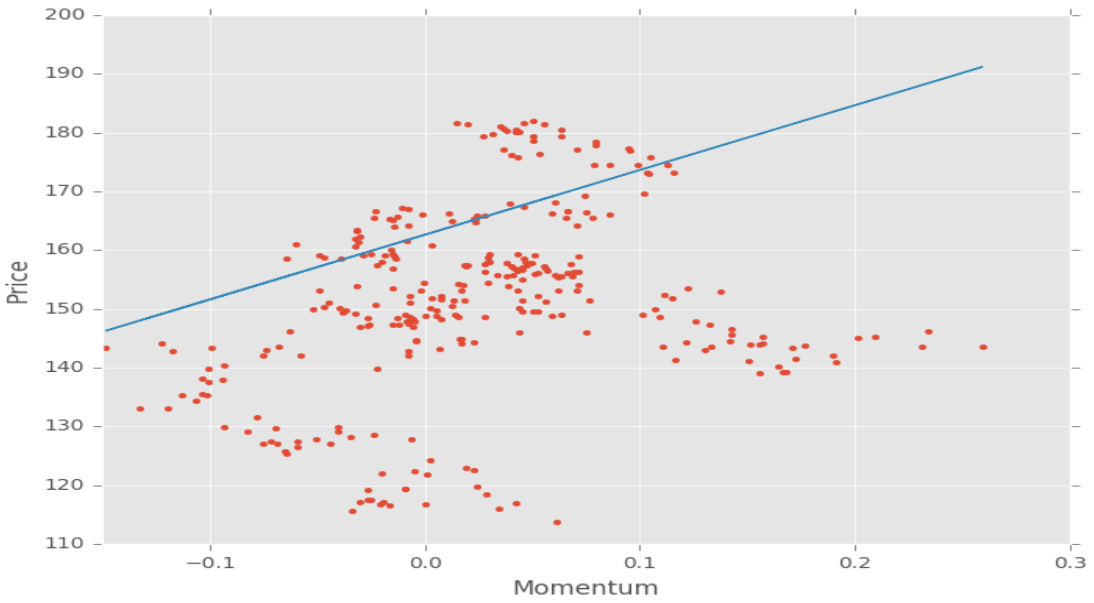

Figure 8: Scatter plot of y_{true} values vs predicted values of y using linear regression for IBM stock.

The Figure 8 shows the example of linear regression fit for the IBM ticket symbol using the momentum feature. The linear fit shown in Figure 8 indicates that the data

exhibits a lot of non-linearity and Linear regression might not give a good estimate of the stock prediction. We will discuss the metrics used for model evaluation in Section 7.

6.3 Ridge regression

Ridge regression shrinks the regression coefficients by imposing a penalty on the their size [H$^+$]. The coefficients minimize a penalized residual sum squares. Ridge model solves the regression model where the loss function is the linear least-squares function and the regularization is given by the L^2 norm [sci17b].

6.4 Random forest regression

Random forests are ensemble (or collection) of decision trees, that output mean prediction (regression) of the individual tree for regression problems. The idea is to average many noisy but approximately unbiased models and thus reduce the variance [H$^+$]. Random-forests regressions are quite efficient especially where the data is non-linear. In the the Fig. 8, we have seen that the momentum is quite non-linear and thus not surprisingly, the linear-regression and so does the ridge did not work well. Figure 9 shows the fit for Random-forests, and this clearly indicates that predictions are clearly close enough to the true values.

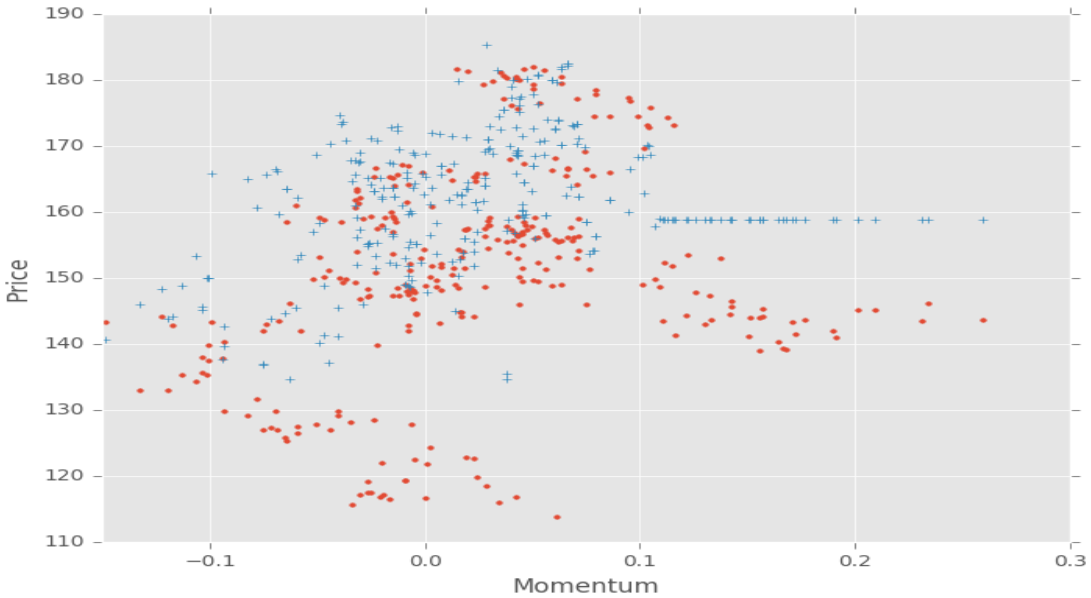

Figure 9: Scatter plot of y_{true} values vs predicted values of y using Random forest regression for IBM stock. Circles indicate true values and + indicates the predicated values.

6.5 k-nearest neighbors (kNN)

As the name implies, the prediction for a new point is based on the closest k-points in the training set, where closest is usually defined in terms of Euclidiean distance on the **p** dimensional feature space. As the regression problem, it take the average label of the nearest k-neighbors. This is a non-parametric algorithm as discussed in the earlier part of the section. From the Fig. 10, we see that kNN performs almost similar to Random forest regression, capturing the non-linearities in the data.

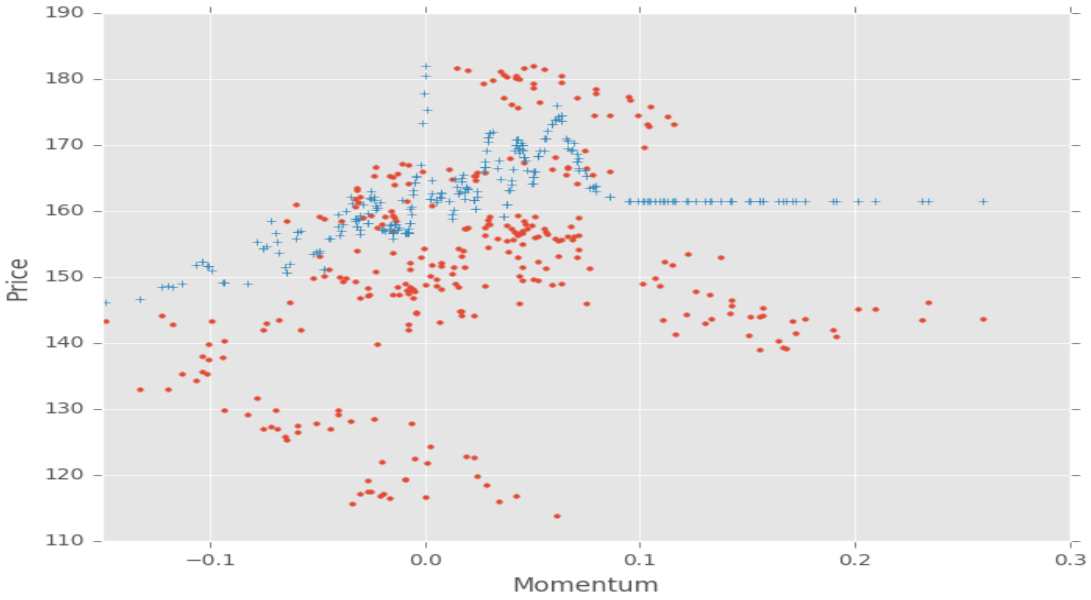

Figure 10: Scatter plot of y_{true} values vs predicted values of y using k nearest neighbors for IBM stock. Circles indicate true values and + indicates the predicated values.

6.6 Benchmark Model

As a benchmark model, one could use a simple mean-model. For instance, a particular stock price can be predicted using mean of the adjusted close price. This is the simplistic baseline model. This can be implemented using the `DummyRegressor` [sci17a] in `scikit-learn`.

7 Evaluation Metrics

In the previous section, we talked about the different models that can be used for the problem of stock-price prediction. The problem of stock price prediction is not just limited to the list mentioned in Section 6, there are many other algorithms that can be applied. But, the important question is, how can one test and compare the performance of these algorithms/models. For this problem of regression we choose two important evaluation metrics:

- **Root-mean-square error (RMSE)**: Mean Absolute Error (MAE), is calculated as the ratio of sum of absolute values of difference between actual value and the predicted value to number of samples. Clearly this metric indicates the average deviation of a predicted value from its actual value. This is a linear function and all errors are weighted equally.

  On the other hand mean-square-error (MSE), is ratio of sum of squared deviations to the number of samples, i.e., it is the mean of the squared deviation. This is a quadratic function, and if the errors are large, they are weighted more while the small errors are weighted less (say if absolute error for a sample point is less than 1, then square of the number is much smaller). Thus in a MSE based estimator, we can clearly see how the predicted values compared to the true values. MSE would be preferred performance metric over MAE especially if you want to have a model, where large errors are particularly undesirable. Also by taking the square-root of the MSE value we get the Standard deviation, which is again helps to predict the variability of the data.

  For this problem, minimizing the metric RMSE makes the stock price prediction accurate.

- **Correlation coefficient**: This metric can be used to evaluate ML algorithm is to look at the relationship between predicted value vs. actual value. Let say we have a test data, and we run our model again test data to get $Y_{predict}$, which is the stock price in this case. We can now compare the Y_{true} against $Y_{predict}$. By looking at the relationship between Y_{true} and $Y_{predict}$ we say if the model is correctly predicting the values. This metric is the correction function ρ and it ranges between $-1 \leq \rho \leq +1$. If $\rho = 1$, then they are strongly correlated; if $\rho = -1$ they are negatively correlated and if $\rho = 0$, they are not correlated.

8 Methodology

Financial stock, commodity stock come in the category of time-series data. Usually time-series data has characteristics such as drift, seasonality and stationarity associated with it. We will talk briefly these set of characteristics:

1. **Drift**: Prices of commodity stocks such as oil, gas, gold, etc., tend to inflate as the time progresses. Usually we see and exponential fit for such data. This property is called drift and price can be mathematically represented as:

$$X_t = e^{\mu t} \tag{13}$$

   We can use log of the above equation as:

$$X_t = \mu t + \epsilon \tag{14}$$

2. **Seasonality**: Time-series data can posses seasonal component, such as temperature data, where temperature increases in summer and decreases in winter. There is certain periodicity associated with the data. Such data can be represented using Fourier series.

$$X_t = \alpha \sin(\omega t) + \beta \cos(\omega t) \tag{15}$$

3. **Stationarity**: This means the the distribution of X_t is independent of time.

These characteristics of the time-series data may not be particularly relevant for the financial stock, but they can be applied for the commodity stock (which is the future extension of this work).

Another important concern for the time-series is whether it is predictable. We generally assume that financial data is ergodic in nature (i.e., there is a non-zero probability that state will recur). In these time-series data we generate features with lagged data. For instance, we generate the features described in Eq. (6), (9), and (10) for a particular stock-price with a lag of say 5 days.

After we generate the feature data, we then split the data into *train* and *cross-validation* set. Time-series again has a very different way of generating cross-validation set. The next section describes the cross-validation methodology in the time-series data.

8.1 Cross-validation of time-series data

In time-series we are trying to predict the value in future. Thus the validation data has to occur **after** the training data. We just cannot pick data point at random, because there might be seasonal effects. Also we cannot have our testing set occur before the training set.

The Fig. 11 shows the general two methods of handling the time-series cross validation data. The module `TimeSeriesSplit` in `sklearn` creates cross-validation sets using Forward chaining method in Fig. 11.

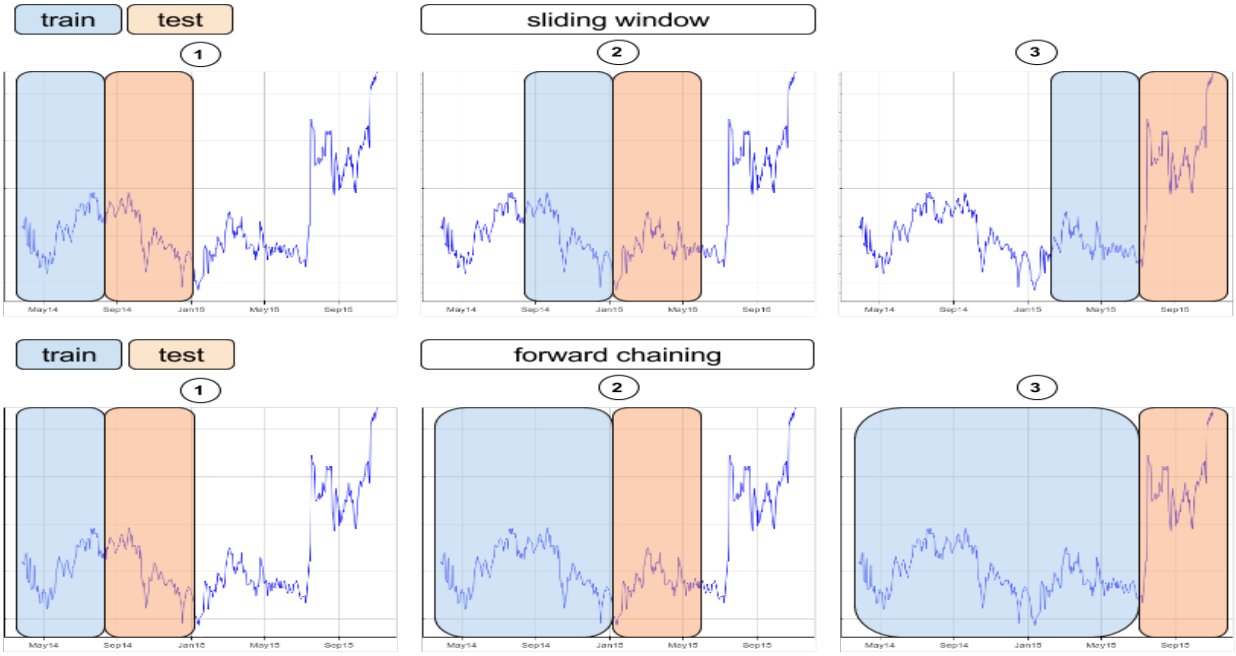

Figure 11: Sliding window Vs. Forward chaining method

8.2 Tuning of hyperparameters

Another important aspect of the machine learning algorithms is tuning of hyper-parameters. Every algorithm has certain set of input arguments (or parameters) that needs to be tuned in order to improve the performance of the model. In the regression problem, we choose to minimize the mean-square error (Section 7). The mimimum value of MSE can be achieved cross-validating and carefully tuning the parameters of the regressor. The optimal parameter (or the best estimator) can be obtained using `GridSearchCV` [gri17] module in `sklearn`. In this section we describe the performance of the machine-learning algorithm for different values hyper-parameters.

8.2.1 k-nearest neighbors

In kNN algorithm, the important parameter that needs to be tuned is number of nearest neighbors that we want to choose in the algorithm. We cannot fix number of nearest neighbors, as it can change depending of the what stock (or ticker symbol) we choose. For instance IBM ticker symbol may give minimum MSE at different number of neighbors than TSLA (Tesla). So, instead of hard coding the regressor with the number of neighbors, we use `GridSearchCV` with parameter grid, so that regressor picks an optimal hyper-parameter for any given ticker symbol. The Fig. 12 shows the optimal value of the nearest neighbor (i.e., when the MSE is at its minimum value). The MSE plotted

in the Fig. 12 is the mean MSE for all the cross-validation sets. In this and subsequent examples we have chosen 5 cross-validation folds (`TimeSeriesSplit(n_splits=5)`).

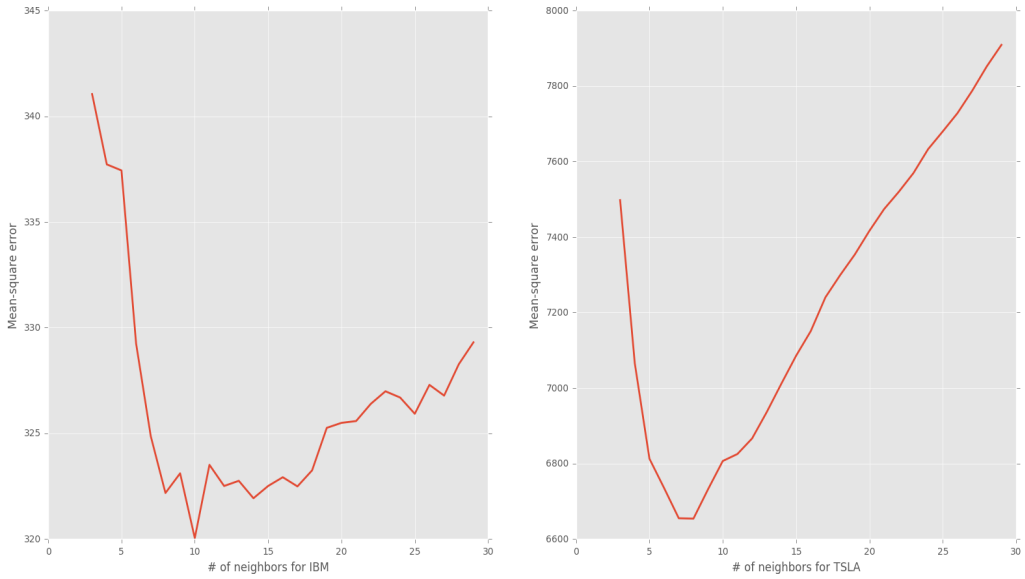

Figure 12: For IBM stock the minimum value of the MSE occurs at 10 while for TSLA it occurs at ≈ 6

8.2.2 Random forest regressor

In Random forest regressor, the parameter that can be tuned to improve the performance of the algorithm is number of trees or number of estimators `n_estimators`. The Fig. 13 shows the variation of MSE as the number of estimators is changed in the model.

8.2.3 Ridge regression

In Ridge regression, the parameter that needs to be tuned is the regularization term α. We have considered α in the range of $10^{-4} \leq \alpha \leq 1$. The Fig. 14, shows the variation of MSE with respect to α. The regularization term is varied using the log-scale, `np.logspace(-4., 0, 20)`.

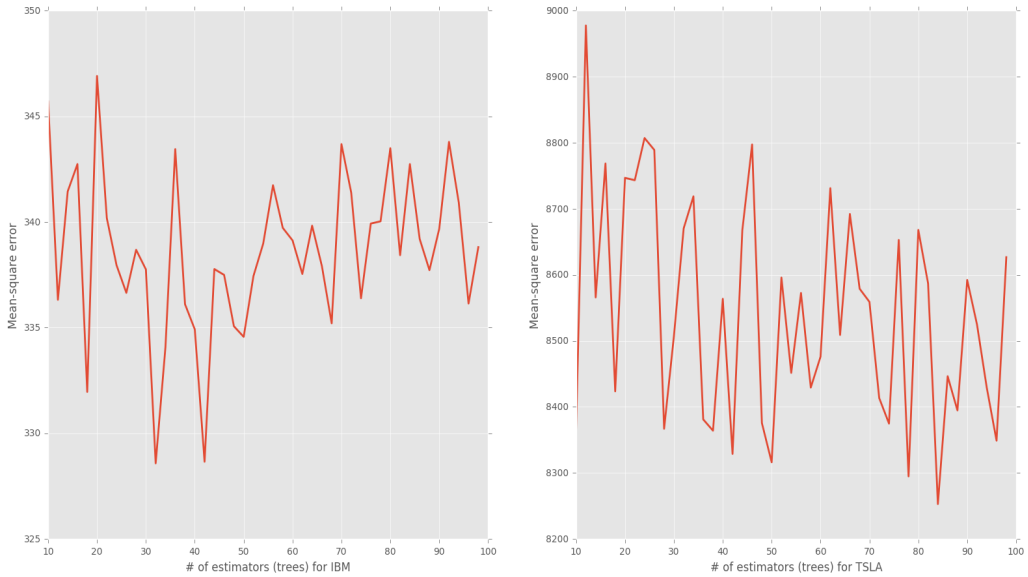

Figure 13: For IBM stock the minimum value of the MSE occurs at 32, while for TSLA it occurs at 84 estimators

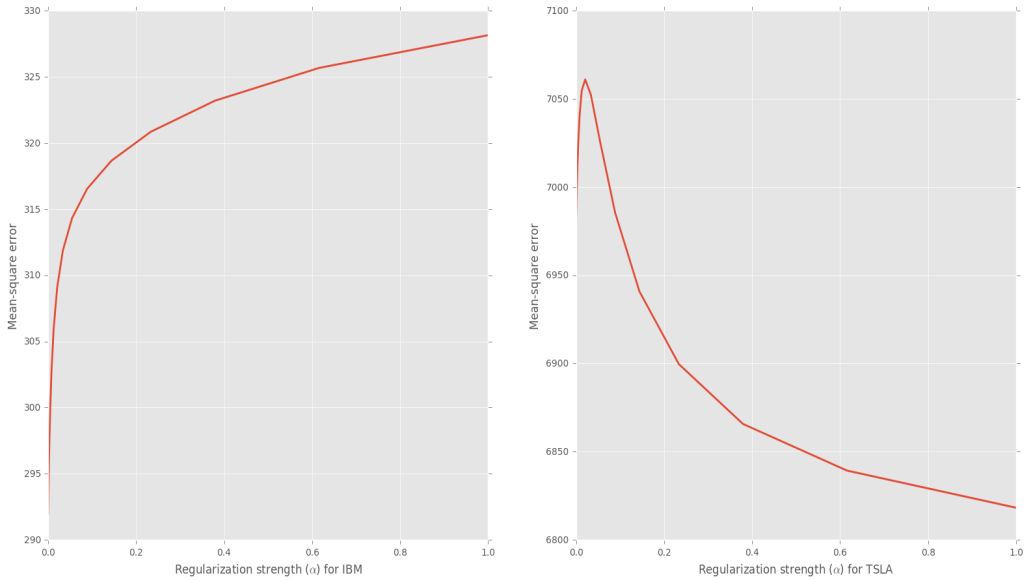

Figure 14: For IBM stock the minimum value of the MSE occurs at 10^{-4}, while for TSLA it occurs at 1.0 value of α

8.3 Coding implementation

This section we briefly describe the coding implementation/challenges of machine learning algorithms using all the processes described so far in the data wrangling, calculating feature sets, cross-validation, machine-learning algorithm, hyper-parameter tunning and finally computing metrics.

1. We start with collecting data from Quandl or other sources. The module `apicall_data.py` consists of all code related to the querying the data from various sources. In this work, we have only enabled the Quandl data-sources.

   ```
   dataframe=quandl.get('WIKI/%s'%(symbol),start_date=
   start_dt.strftime('%Y-%m-%d'), end_date=end_dt.strftime('%Y-%m-%d'))
   ```

   `symbol` is the ticker symbol and the start and the end dates for the which the date is collected and stored as `pandas` dataframe object.

2. However it is possible that the dataframe consists of `NaN` values, i.e., the missing data. We use a S&P500 as benchmark symbol to clean up the data.

   ```
   df_temp = pd.DataFrame(tpl[1]['data']['Adj. Close'],
                   index=tpl[1]['data'].index)
   df_temp = df_temp.rename(columns={'Adj. Close': tpl[0]})
   df = df.join(df_temp)
   if tpl[0] == benchmark_symbol:  # drop dates SPY did not trade
     df = df.dropna(subset=[benchmark_symbol])
   df=df.dropna(axis=0) # This drops any NaN
   ```

3. After cleaning up the data, the next step is to compute the feature set for the machine learning algorithm, i.e., the X_i, where i is the i^{th} feature. We compute the features using the equations described in (6), (9), and (10). Below are the coding details of these features:

   ```
   rm[sym]=pd.Series.rolling(df[sym],window=window)
                .mean().to_frame() # Rolling mean
   rstd[sym]=pd.Series.rolling(df[sym],window=window).std()
                 .to_frame() # Rolling std
   normalized_bb[sym]=(df[sym] - rm[sym])/(2 * rstd[sym])
   momentum[window:] = (df[window:]/df[:-window].values) - 1
   daily_returns=compute_daily_returns(df)
   volatility=get_rolling_std(daily_returns,window)
   ```

4. Since we are using the back-testing method, we need to shift (lag) the features, before we split the data in train (train + cross-validation) and test sets (will be used to test the algorithm for unseen data).

   ```
   # shift is the number of days in the future
   Y_shifted=df[sym].shift(-shift);
   ```

Now we split the data as follows:

```
# split_number is how much we want the data to be test set
X_train=X_data.ix[:-shift,:].ix[:-split_number,:]
X_test=X_data.ix[:-shift,:].ix[-split_number:,:]
Y_train=Y_shifted.ix[:-shift].ix[:-split_number]
Y_test=Y_shifted.ix[:-shift].ix[-split_number:]
```

5. Cross-validation set: Training data can be further split into train and cross-validation set, as follows:

   ```
   from sklearn.model_selection import TimeSeriesSplit
   tscv=TimeSeriesSplit(n_splits=5)
   ```

   Time series split creates 5 sets of train and cross-validation using the forward chaining method described in Fig 11.

6. We have considered few regression algorithms to test the performance of predictions. Depending on the behavior of stock, we can have choose appropriate algorithm of a particular stock symbol.

   ```
   from sklearn.dummy import DummyRegressor # Used for benchmark
   from sklearn.ensemble import RandomForestRegressor
   from sklearn.neighbors import KNeighborsRegressor
   from sklearn.linear_model import LinearRegression,Ridge
   ```

7. We use MSE as the metric, that should be minimized for picking the right parameters for the regression algorithm.

   ```
   from sklearn.metrics import mean_squared_error,make_scorer
   scorer=make_scorer(mean_squared_error,greater_is_better=False)
   ```

8. For hyper-parameter tunning we use `GridSearchCV`. Here we describe the kNN algorithm using grid-search cross validation. For other algorithms, refer to the code-base [Bat17d].

   ```
   param_grid={'n_neighbors':range(3,20,1)}
   n_neighbours_cv=GridSearchCV(KNeighborsRegressor(),
           param_grid=param_grid, cv=tscv, scoring=scorer)
   ```

   The number of nearest neighbors is varied from 3 to 20 in steps of 1, and cross-validated such that MSE is minimized.

9. Pick the best estimator, predict the y and compute the metrics on the test set.

   ```
   # Return the optimal model
   bestEst= n_neighbours_cv.best_estimator_
   # Predict
   Y_pred_test=bestEst.predict(X_test)
   # Compute metrics
   mse= mean_squared_error(Y_test, Y_pred_test)
   ```

```
correlation_mat= np.corrcoef(Y_pred_test,Y_test)
correction_coff=correlation_mat[0,1];
```

10. Finally, apply the model to the future prediction of the stock symbol

```
X_pred=X_pred=X_data.ix[-shift:,:]
Y_future=bestEst.predict(X_pred)
```

The most challenging part of the coding is picking up the right hyper-parameters for the machine-learning models. We have seen in Section 8.2 if the right value of the parameter is not chosen, then the MSE could be very high and model might be significantly inaccurate. Also due high variability in behavior across various stocks, the parameter chosen for one stock might not apply for other stock (example Fig. 12). One has to carefully choose the parameters such that model should not be unbiased and overfitted. Tunning of hyper-parameters with cross-validation can take long time to run. So, we need to be cautious about that and choose the range wisely !

9 Results

In this section we discuss various results obtained for portfolio optimization, market simulator, and machine learning models. The first two subsections 9.1 and 9.2, though not directly related to the the concepts of machine-learning. These two sections will be particularly important to make the trading strategies, based on the prediction values obtained from the machine learning models. For instance if the machine learning model forecasts with certain accuracy that the stock price is going to increase in next 5 days, then the trader can make decisions to buy and those improve the portfolio performance of the client. This will be the future extensions of this work.

9.1 Portfolio optimization

Portfolios in comparison with SP&500 for the symbols, GOOG, AAPL, MSFT and IBM for duration between Jan 1 2013 to March 14 2017, for both unoptimized and optimized are shown in Fig. 15 and 16 respectively.

In Fig. 15, the allocations for the ticker symbols were chosen between an uniform random variables $\sim U[0, 1]$, such that the sum is equals to 1. The first row in the Table 3, gives the random allocations. In the app http://quantfy.herokuapp.com/portfolio user can also pick his/her choice of the allocations. If none were chosen then the random allocations are made. From Fig. 15, we see that in the first half the graph, the portfolio is performing close S&P500 (SPY). The parameter for unoptimized portfolio are given in Table 4.

When we optimize the portfolio, via maximizing the Sharpe ratio, we can do a better allocation for the chosen ticker symbols. Figure 16 clearly improves the performance

Figure 15: Unoptimized portfolio for GOOG, AAPL, MSFT and IBM in comparison with SP&500, with a start value of $10000

Metric	GOOG	AAPL	MSFT	IBM
Random allocations	0.4682	0.1894	0.2449	0.0975
Optimized allocations	0	0.539	0.461	0

Table 3: Parameters computed for un-optimized and optimized portfolio

over the SPY. The optimized portfolio only allocates assets to AAPL and MSFT as given in Table 3. The parameters for the optimized portfolio are given in Table 4. This shows an improvement in the end value over the un-optimized portfolio. Also, since we are maximizing the Sharpe ratio, we see an improvement in the value.

Figure 16: Optimized portfolio for GOOG, AAPL, MSFT and IBM in comparison with SP&500

9.2 Market Simulator

Based on the orders file, the portfolio value can be analyzed. For instance if BUY or SELL order comes, depending on the stock price value of the order, the portfolio value changes. Using the leverage threshold, orders can be executed or fail to execute if the leverage value exceeds the threshold.

Leverage value is defined as:

$$leverage = \frac{\Sigma|sp|}{(\Sigma(sp) + cash)}$$

,

Metric	Un-optimized	Optimized
Cumulative daily return	0.6065	0.8507
Average daily return	0.0007	0.0009
Standard deviation of daily return	0.0116	0.0126
Sharpe ratio	0.9624	1.1424
End value	$160651.061	$185068.0823

Table 4: Parameters computed for un-optimized and optimized portfolio

Figure 17: Market simulator for a order file. The green vertical lines indicate a BUY order and RED vertical lines indicate a SELL order

where *sp* is defined as all-stock positions [Bal16].

The Fig. 9 is generated using orders file located at:
https://github.com/beegeesquare/QuantFy/blob/master/orders/orders_demo.csv

9.3 Machine-learning model comparison

Predicts the stock prices and also measures the accuracy of these forecasts. Using back-testing a method where we roll back time and measure the accuracy of these forecasts. Slice the sample data from (historical) training set and apply the machine learning models to get the forecast. By comparing the predictive results of the model against the historical results, back-testing can determine whether the model has predictive value.

The QuantFy application at: http://quantfy.herokuapp.com/mlModels, where user can test any other ticker symbols. But for this discussion we have chosen, Apple (AAPL), Microsoft (MSFT) and IBM. These are the technology companies and the price behavior might be similar. However the code will generate a separate model for each ticker symbol.

User can select two regression algorithms from:

- Mean-model,
- k-nearest neighbors,
- Random forest regressors,
- Linear regression, and
- Ridge regression.

Left side of the RED dotted line indicates the past values, while the right side are the predicted values of the stock.

Table 5 shows the Mean square error for the three ticker symbols chosen. From the Table 5 we clearly see that each ticker symbol (or stock) has different behavior. For instance AAPL stock show a linear behavior, with MSE being lower than benchmark (mean model) and showing a positive correlation coefficient for both Linear and Ridge algorithms as in Table 6.

Typically, as the MSE decreases the correlation-coefficient increases (close to 1). The k-NN seem to perform well for IBM with lowest MSE and high ρ.

However for the Microsoft (MSFT), the MSE seem to be consistently high for all the algorithms used in the project. The number of features used in the project are less. Adding more features such as P/E ratio, volume of trades, can definitely improve the performance of the algorithm and hence correctly capture the behavior of the stock.

We clearly see that all the algorithms used in this project provide a significant improvement over the benchmark algorithm.

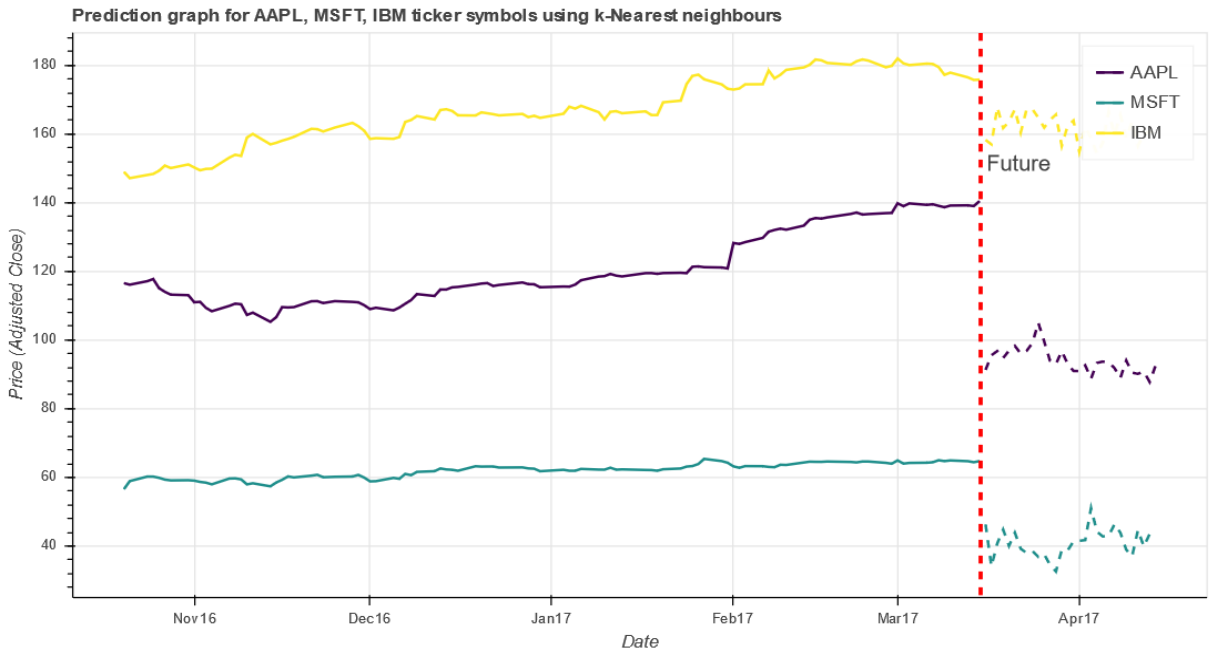

Figure 18: Prediction values for 15 days for AAPL, MSFT and, IBM using k-nearest neighbors

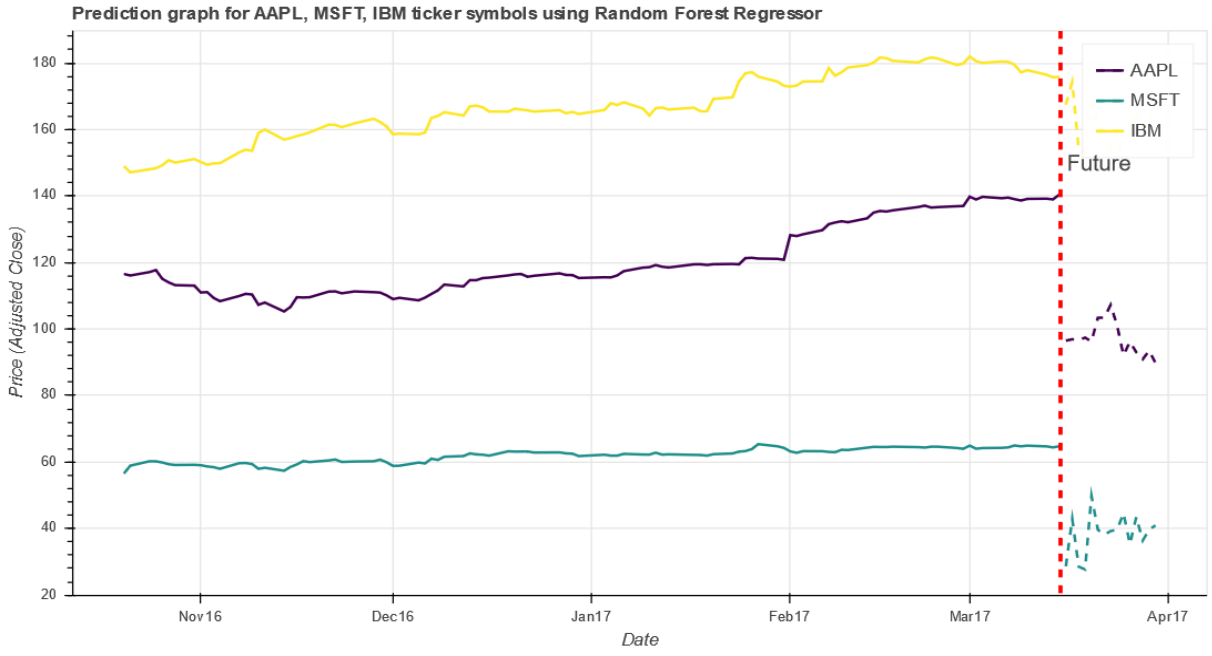

Figure 19: Prediction values for 15 days for AAPL, MSFT and, IBM using Random forest regressor.

ML algorithm	AAPL	MSFT	IBM
k-nearest neighbors	862.082	469.81	136.86
Random forest regessor	1390.155	468.274	164.709
Linear regression	875.99	428.24	132.94
Ridge regression	876.146	437.069	132.92
Mean model	897.55	423.58	183.39

Table 5: Mean square error

ML algorithm	AAPL	MSFT	IBM
k-nearest neighbors	0.252	-0.023	0.293
Random forest regressor	-0.33044	-0.0761	0.0614
Linear regression	0.649	-0.1706	0.196
Ridge regression	0.677	-0.145	0.197
Mean model	NaN	NaN	NaN

Table 6: Correlation coefficient (ρ)

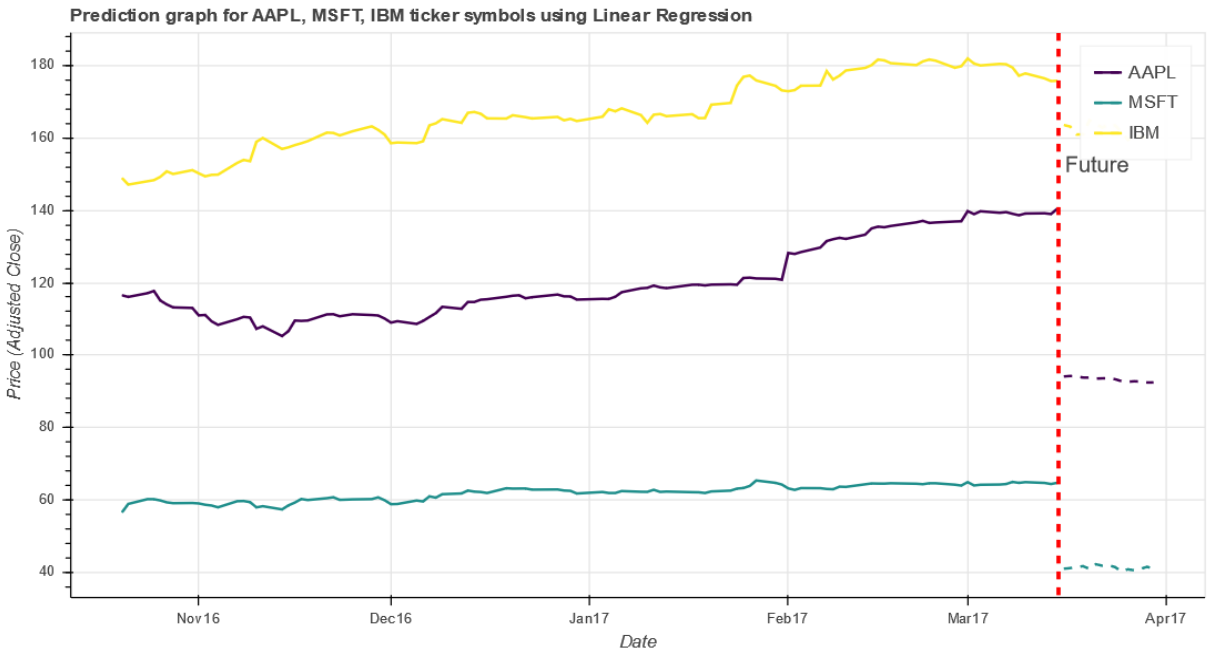

Figure 20: Prediction values for 15 days for AAPL, MSFT and, IBM using Linear regression.

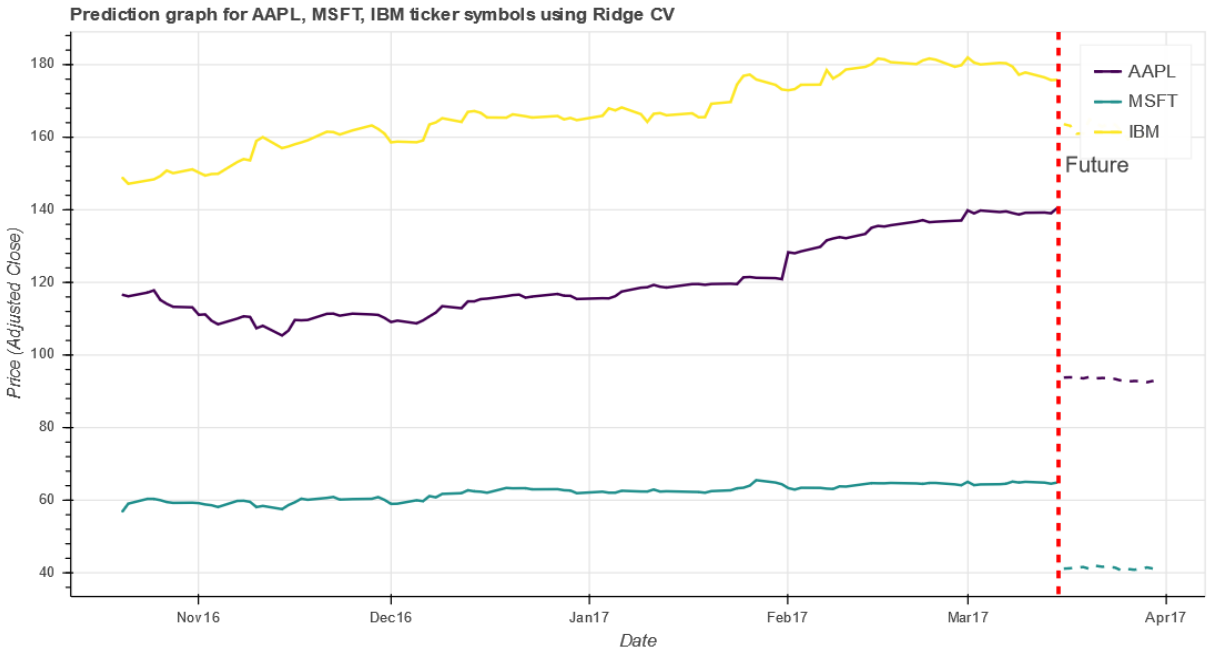

Figure 21: Prediction values for 15 days for AAPL, MSFT and, IBM using Ridge regression.

10 Conclusion

This project we have provided analysis for the stock price data. Various visualizations are provided to see the behavior of particular stock price. Many parameters are computed to understand the dynamic nature of the data. Since stock price data is a time-series, we have provided the characteristics of time-series data and how it is different from any other data set.

Using convex optimization techniques, the portfolio optimization can be done and the allocations of assets to the stocks can be choose optimally.

Various machine-learning algorithms has been analyzed for the ticker symbols. We have seen that every ticker symbol has an unique behavior. For some symbols we see linear relationship and for other we see non-linear or stochastic behavior. Using the metrics such as MSE and correlation-coefficient one can assess the quality of the prediction.

Using cross-validation and parameter tunning, we can minimize the errors, i.e., select the parameters such that the model is neither biased nor overfitted. Thus the model works well for the unseen data.

Regression algorithms possess certain drawback for the stock price predictions such as:

- Noisy and uncertain.
- It is challenging to estimate the confidence in the prediction.
- Does not capture the stochastic behavior well.

We can overcome of these issues in regression using Reinforcement learning. As the next steps to improve the project, we will use Q-learning algorithms for the stock price predictions. Using advanced algorithms for feature selection, one can further improve the accuracy of models.

As the further steps this application could be extended to make trading strategies based on the predictions.

Also we will extend the work for estimating the prices for commodity stocks such oil, gas, gold, etc.

References

[Bal16] Tucker Balch. Machine learning for trading. Udacity course, 2016.

[Bat17a] Balagangadhar Bathula. Portfolio optimizer: QuantFy. `http://quantfy.herokuapp.com/portfolio`, 2017. [Online].

[Bat17b] Balagangadhar Bathula. Price Plot: QuantFy. `http://quantfy.herokuapp.com/price_plot`, 2017. [Online].

[Bat17c] Balagangadhar Bathula. QuantFy. `http://quantfy.herokuapp.com/`, 2017. [Online].

[Bat17d] Balagangadhar Bathula. Source code: QuantFy. `https://github.com/beegeesquare/QuantFy`, 2017. [Online].

[gri17] Grid search cross-validation. `http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html`, 2017. [Online].

[H+] Trevor Hasti et al. *The Elements of Statistical Learning*. Springer.

[Hil] Yves Hilpisch. *Python for Finance*. O'Reilly.

[RB] Philip J. Romero and Tucker Balch. *What Hedge Funds Really Do: An Introduction to Portfolio Management*. Business Expert Press.

[sci17a] Dummy Regressor. `http://scikit-learn.org/stable/modules/generated/sklearn.dummy.DummyRegressor.html`, 2017. [Online].

[sci17b] Ridge regression. `http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html`, 2017. [Online].