# BIG DATA IN FINANCE PROJECT DESCRIPTION

Haorong Chen
Jun Ouyang
Yixiao Zhao

IEOR, Columbia University
Dec. 2017

## 1. Project Objectives

The thriving of machine learning has empowered not only internet technology industries but also ultimately evolved the traditional banking industries. There are increasingly interests raising from the pioneer investors to understand the magic of computing intelligence. Among all great discussions, using machine learning to predict pricing moving trend is absolutely at spotlight. Predicting prices is not a new topic. Looking back in the history, people have always been trying to development methods to predict stock price movements. However, attempts to predict stock price are seen to be impossible previously, but with the machine learning techniques, it becomes an achievable task lately.

Decision tree is a method commonly used in data mining. In this project, the goal is to create a model that predicts the stock price based on the historical data and some technical indicators. Each leaf represents a value of the stock price given the values of the input variables represented by the path from the root to the leaf. Each internal node denotes a test on an attribute, and each branch represents the outcome of a test. A tree can be improved by splitting the source set into subsets based on attribute value tests. This is a recursion process and it stops when splitting no longer adds value to the target variable (in this project, the target variable is the accuracy of stock price predictions). Decision tree is simple to understand and interpret, because it mirrors human decision making closely.

The prediction target of this project is to construct a trading strategy to maximize the return of our portfolio over a time period using decision tree. The stocks in the portfolio will be selected from a certain stock pool (e.g. S&P 500, Dow Jones Industrial Average). We will use the performance of the stock pool as a benchmark to evaluate the effectiveness of the strategy constructed from decision tree. We will also observe the volatility of the portfolio and use it as a risk criterion. Another key target is to test the correlation of the performance of the portfolio to the market. We expect our portfolio not only outperform the benchmarks in the upside, but also systematically hedge the risk on the downside.

We intend to use decision tree to construct and manage a portfolio based on historical KPIs. To begin with, we follow the methodology on "Use Decision Trees in Machine Learning to Predict Stock Movements" to build our first model. The model would be implemented and back-tested on Quantopian platform as it is easy to retrieving many indicators

## 2. Acquire and Explore Data

In Quantopian, there are various data sources to acquire and the libraries we are going to involve in our project include numpy, pandas, sklearn, talib and quantopian pipelines.

In Quantopian Pipeline, we can retrieve the historical data for all the stocks including open price, high, low, close price, volume and so on. Here we have attached an example below to get and plot the historical price for Apple(AAPL).

```
In [3]: sample = get_pricing('AAPL');
        sample
```

Out[3]:

| | open_price | high | low | close_price | volume | price |
|---|---|---|---|---|---|---|
| **2013-01-03 00:00:00+00:00** | 534.453 | 536.199 | 527.742 | 529.195 | 11788310.0 | 529.195 |
| **2013-01-04 00:00:00+00:00** | 523.805 | 525.430 | 513.236 | 514.094 | 20125970.0 | 514.094 |
| **2013-01-07 00:00:00+00:00** | 509.207 | 516.328 | 502.574 | 511.129 | 16312375.0 | 511.129 |
| **2013-01-08 00:00:00+00:00** | 516.240 | 518.855 | 508.476 | 512.319 | 15004532.0 | 512.319 |
| **2013-01-09 00:00:00+00:00** | 509.695 | 512.124 | 503.344 | 504.427 | 13208269.0 | 504.427 |
| **2013-01-10 00:00:00+00:00** | 515.597 | 515.762 | 502.886 | 510.787 | 19713820.0 | 510.787 |
| **2013-01-11 00:00:00+00:00** | 508.232 | 512.446 | 506.300 | 507.373 | 11202506.0 | 507.373 |
| **2013-01-14 00:00:00+00:00** | 490.361 | 495.063 | 486.293 | 489.600 | 23251599.0 | 489.600 |
| **2013-01-15 00:00:00+00:00** | 486.088 | 486.761 | 471.534 | 473.904 | 29639268.0 | 473.904 |
| **2013-01-16 00:00:00+00:00** | 482.518 | 496.955 | 480.430 | 493.677 | 22482784.0 | 493.677 |

We can filter the stocks list by average daily volumes in order to maintain market liquidity and price variances.

## 3. Model Data and Tools

In terms of variable selection, we are going to implement our models using Average Directional Movement Index, Bollinger Bands, Relative Strength Index, Simple Moving Average etc. We are also going to implement some feature selection criterion using Scikit-learn.

From talib, we can derive the technical analysis on all the assets using the built-in functions such as RSI, SMA, ADX, LMA etc. Scikit-learn is a powerful machine learning tool for data mining and data analysis, where we are planning to generate decision tree classifiers and training mechanisms.

In the model selection stage, we are going to implement the back test as below and try to pick the best-performing model from the test set.


## 4. Model Implementation

### *Phase 1: Stock Selection (Decision Tree vs Random Forest vs AdaBoost)*

At the current stage, we have chosen Relative Strength Index (RSI), Moving Average Convergence Divergence(MACD), Exponential Moving Average(EMA), Simple Moving Average (SMA) and Average

Directional Index(ADX) as stock performance indicators, and classify the stock movement responses into two sub-classes: up or down.

Using the historical data and preferred classification method, we could predict if a stock will increase in one week. Based on the prediction, we could select a group of stocks from a certain pool. We could easily add or remove the performance indicators, and also modify the length of predicting period while currently everything is still on a testing phase.

Currently, we are comparing the three different classifiers in terms of prediction success ratios and L2 errors (false-negatives). We have splitted the data into 80% training set and 20% test set. Here we have attached some sample code in order to demonstrate the potential preference in these three most popular classifiers:

Let's pick the relevant features in terms of predicting

```
52]: feature_columns = ['RSI', 'MACD', 'EMA','SMA_5','SMA_10','ADX']
     response_column = ['ClassValue']
```

```
53]: clf = tree.DecisionTreeClassifier()
     clf.fit(training_set[feature_columns].values, training_set[response_column].values)

     training_response = clf.predict(training_set[feature_columns].values)
     training_error = np.sum(np.square(training_response - training_set[response_column].values.flatten()))
     training_error_ratio = float(training_error.item())/int(training_size)*100
     print('Error on training set %.8f %%' % training_error_ratio)

     test_response = clf.predict(test_set[feature_columns].values)
     test_error = np.sum(np.square(test_response - test_set[response_column].values.flatten()))
     test_error_ratio = float(test_error.item())/int(test_size)*100
     print('Error on test set %.8f %%' % test_error_ratio)
```

```
Error on training set 0.00000000 %
Error on test set 36.13231552 %
```

Now let's try the random forest classifier

```
:  clf_2 = ensemble.RandomForestClassifier()
   clf_2.fit(training_set[feature_columns].values, training_set[response_column].values)

   training_response = clf_2.predict(training_set[feature_columns].values)
   training_error = np.sum(np.square(training_response - training_set[response_column].values.flatten()))
   training_error_ratio = float(training_error.item())/int(training_size)*100
   print('Error on training set %.8f %%' % training_error_ratio)

   test_response = clf_2.predict(test_set[feature_columns].values)
   test_error = np.sum(np.square(test_response - test_set[response_column].values.flatten()))
   test_error_ratio = float(test_error.item())/int(test_size)*100
   print('Error on test set %.8f %%' % test_error_ratio)
```

```
Error on training set 2.03951562 %
Error on test set 27.98982188 %
```

As we can see from the comparison between decision tree and random forest classifiers, decision tree is achieving 36% error ratio while random forest is achieving 8% lower! Similarly, let’s look at the popular Adaboost classifier:

```
clf_3 = ensemble.AdaBoostClassifier()
clf_3.fit(training_set[feature_columns].values, training_set[response_column].values)

training_response = clf_3.predict(training_set[feature_columns].values)
training_error = np.sum(np.square(training_response - training_set[response_column].values.flatten()))
training_error_ratio = float(training_error.item())/int(training_size)*100
print('Error on training set %.8f %%' % training_error_ratio)

test_response = clf_3.predict(test_set[feature_columns].values)
test_error = np.sum(np.square(test_response - test_set[response_column].values.flatten()))
test_error_ratio = float(test_error.item())/int(test_size)*100
print('Error on test set %.8f %%' % test_error_ratio)
```

```
Error on training set 19.50286807 %
Error on test set 32.56997455 %
```

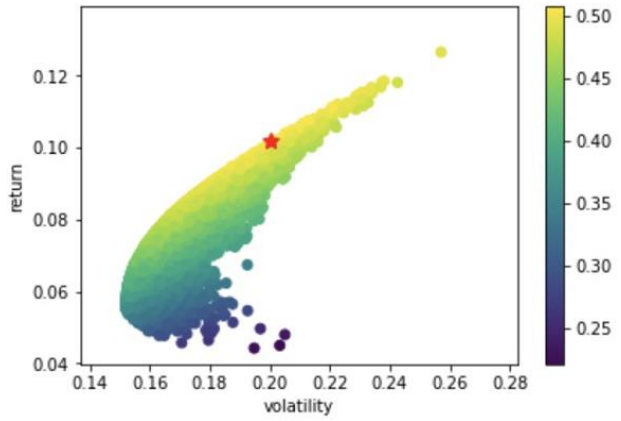
AdaBoost still seems to work better than Decision Tree while worse than Random Forest.

Temporarily, we are picking Random Forest to be our ideal classification methods while more validation and testing need to be done in future in terms of different market scenarios and different equities.

### *Phase 2: Mean Variance Portfolio Optimization*

We have decided to use Markowitz Mean Variance analysis to construct a one-week portfolio after selecting desired stocks by decision tree method. The fundamental goal of Mean Variance Portfolio Optimization is to optimally allocate wealth between different assets, such that the portfolio will achieve highest expected return for a given variance level.

We tested the Mean Variance Portfolio Optimization by constructing a portfolio with the eight indexes provided in the first assignment. We obtained the expected return and volatility of these eight indexes from the historical data. The following graph shows expected return versus volatility. Every possible combination of risky assets is plotted on on the graph, and the collection of such possible portfolios defines a region in this space. The upper edge of this region is the efficient frontier. Every portfolio on the efficient frontier is the portfolio with highest expected return for a given volatility level. The red star on the graph represents the tangency portfolio, which is the best possible point on the capital allocation line. We will choose the tangency portfolio as the optimal portfolio.

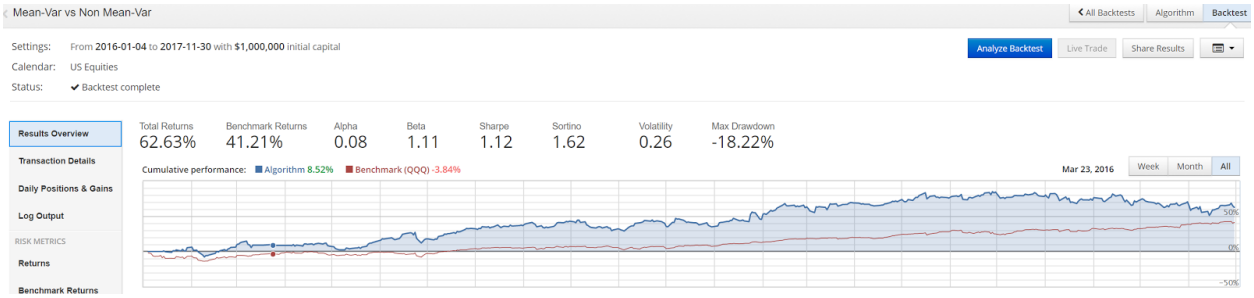



Once we finish selecting stocks, we could extract historical returns from Quantopian and implement the above Mean Variance Portfolio Optimization method to construct the optimal portfolio.

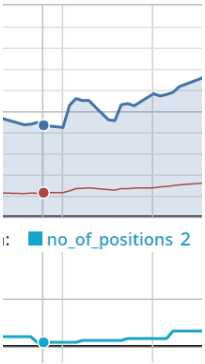
## 5. Result and Analysis

Here we have tested two strategies using decision trees, with and without mean variance optimization.


The total return provides a clear view that the decision trees method contributes unique alpha to the portfolio. However, it is noticeable that the maximum drawdown is over -15%, which is not an acceptable risk control. In contrast, we also implemented the mean variance optimization based on the decision tree strategy:



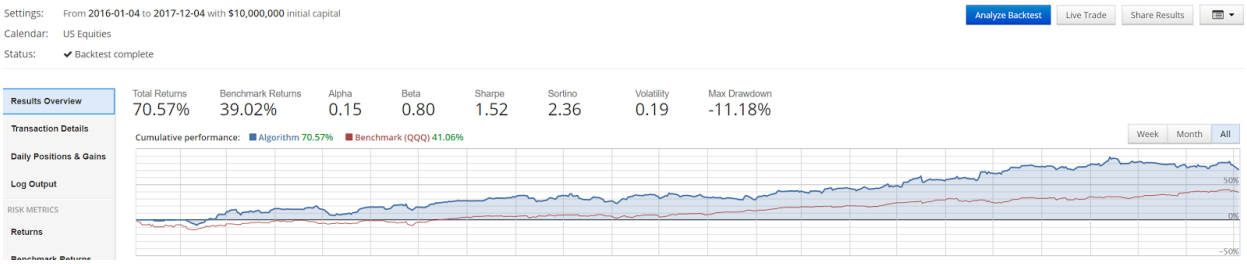

It is quite astonishing at the first glance that the mean variance method did not improve much on the maximum drawdown and the total volatility. In fact, the maximum drawdown even decreases to -18%. This is an unexpected result as we all know mean variance is a well-behaved risk control measure. We took a long time understanding why this is happening and finally find the cue.




The key is that, in many times, the classifier would only predict few potential stocks, sometimes even less than 5 stocks. The plot shows on Jan.10, 2017, the decision tree can only recommend two stocks to take a long position. In such a small sample space, again applying mean variance optimization would randomly eliminate the ones

potentially gain profit. Therefore, a threshold on the size of the sample space should be posted against mean variance.

Here is the result after we apply the threshold technique:




Now we can see both the maximum drawdown and the volatility are improved significantly. Based on a lower volatility, the Sharpe ratio also increases to around 1.5.

## 6. Conclusion

In this project we mainly focused on how to implement decision tree to create an auto-trading strategy. The back-test results show strong potential for this method although it should me more carefully examined when apply to reality. One of the key drawback for this method is sometimes the predictor only has few stock recommendations which conflicts the ideas of diversification as a fund. We need to develop more advanced risk control method based on this to gain more confidence on this strategy.

Also, as time constrained, we could not include more factors to analyze. Including fundamental performance indicators for other sectors (except for technologies) may also boost the prediction accuracy and the total performance of the portfolio.

It is also possible to include macro economic data as systematic factors, such as inflation rate, GDP. This could also provide the model more alpha.

https://www.quantopian.com/posts/big-data-in-finance-course-project-decision-tree-application-to-auto-trading