**FINAL PROJECT**

**MULTI FACTOR PORTFOLIO TRADING STRATEGY**

**DESIGN AND IMPLEMENTATION OF SYSTEMS TO SUPPORT COMPUTATIONAL FINANCE**

SUBMITTED BY-

YASHOVARDHAN JALLAN

GT ID#: 903206747

(I have chosen to exercise the late submission option. I did not use this for md-term project submission).

PROBLEM DESCRIPTION

The task at hand is to invest $10 million in a large universe of Chinese Stocks traded on the Shanghai Stock Exchange and Schengen Stock Exchange. A universe of total 2466 securities to choose from.

The constraint is that the Market Cap of each security must be atleast $500 million RMB and their average daily trading volume over the last 15 days needs to be no less than 1 million.

This project was to identify the best combination of factors and the hyperparameters associated with the factors to come up with a winning strategy. The factors chosen in this project are:

1. Factor F1: Price to Book ratio (PB)
2. Factor F2: Price to Cash Flow ratio (PCF)
3. Factor F3: Price to Earnings ratio (PE)
4. Factor F4: Price to Sales ratio: (PS)
5. Factor F5: n-period momentum factor (PM)
6. Factor F6: m-period reversion factor (PRev)
7. Factor F7: L-period log-return volatility (Vol)

Based on these factors, we calculate a weighted M-score for each security and then make trades on the top performing 100 stocks with the highest M-score values.

The rebalancing period for making trades can be between 1 to 20 days. Another constraint is that each stock must have 1% of the portfolio value and the transaction cost are 0.1% of the dollar amount traded.

The project uses stock market data from 2011-01-01 to 2014-10-31 for back-testing. And, stock market data from 2014-11-01 to 2015-07-31 as the out of sample test period.

DETAILS ON IMPLEMENTATION AND CODE STRUCTURE DESCRIPTION

I have run the code on Python 3.5. using Spyder IDE with inbuilt IPython Kernel.

Files Submitted in the Zip folder:

1. Data directory: this contains all the downloaded stock data from yahoo Finance. I have added this because if you run the code to download the data from the internet, it takes between 90 to 120 minutes.
2. Mkt\_data directory: this folder has the market data (000300.SS)
3. Ticker\_universe.csv
4. optimization\_df\_insample.csv: the csv file I generated after running the parameter testing on insample period for varying N,L,M and U for the 192 test cases.
5. optimization\_df\_outsample.csv: the csv file I generated after running the parameter testing on outsample period for varying N,L,M and U for the 192 test cases.
6. yjallan3\_code.py: my python code

I have not included the szss directory (which is 302MB in size). To run the code, this must be copied into the working directory.

Code Structure Description:

* Libraries used

1. Datetime: this is used to create date as an object.
2. Numpy: Library in Python for performing numeric operations
3. Pandas: Library which provides easy use of dataframes to deal with time-series data.
4. pandas\_datareader: this is used to retrieve stock data from Yahoo Finance
5. sklearn (linear\_model) : used to run cross-sectional regression on the data.

* Classes

1. Security class:
   1. \_\_init\_\_ function: constructor of this class
   2. Store\_data function: this is used to download all the stock data from Yahoo Finance and store it into the data folder in the working directory
   3. get\_stock\_df\_dict function: this reads the stored stock data and stores them in stock dictionary which contains adj close price of the different tickers with the ticker name as the key.
2. Security\_Market Class:
   1. \_\_init\_\_ function: constructor of this class
   2. store\_data function: this is to read the market data (000300.SS) and store it into the mkt\_data folder.
3. Strategy Class:
   1. \_\_init\_\_ function: constructor of the class
   2. get\_M\_score\_df\_and\_optimal\_weight\_list function: this is a crucial function. This function reads all the stock time series. Creates a dataframe of all stocks and calculates the 3 factors (PM, PRev and Vol). Along with the 4 accounting factors, this dataframe now contains all the seven factors. I have then run cross-sectional regression on all these 7 factors to identify which factors should be assigned what weights. Based on the calculate weights, I have calculated the M-Score values for each stock for each date. This function returns the dataframe containing M-score by dates for all stocks and the optimal weights per factor as a list.
   3. Top\_100\_stocks function: this function is used to choose the top 100 stocks for each trading day.
   4. portfolio\_value function: this is used to update the new portfolio value
   5. sellEquities function: this is used to determine which equities we need to sell at every rebalancing period.
   6. buyEquities function: this is used to determine which equities we need to buy at every rebalancing period.
   7. Rebalance function: this is used to determine the total transaction involved in rebalancing the equities that we are holding on to, to make sure that they satisfy the 1% portfolio value criteria.
   8. get\_performance function: this function is used to get all the performance measures of the portfolio. Every performance measure required as a part of this project is calculated in it.
   9. run\_strategy function: this is the main executing function in this class which calls the other functions and computes the required performance values.

* Main Function

This is the main function of the mile where I have instantiated all the classes. I have commented parts which reads data from the internet as this takes a lot of time. Instead, I am reading the downloaded data from the directory itself to ensure that the code runs fast.

For one set of parameters, the code executes in less than 5 mins depending on the system that we run it in.

STRATEGY OPTIMIZATION

I experimented with a large pool of hyper parameters to get the best values for my out-sample performance. I have commented this part of the code but I basically ran 192 iterations with a list of momentum window size n\_window\_list=[2,5,10,20], a list of Reversion window size m\_window\_list=[10,15,20,30], a list of volatility window size l\_window\_list=[15,20,30,40] and finally a list of different rebalancing periods u\_freq\_list=[20,15,10].

In total this was 4\*4\*4\*3 =192 iterations.

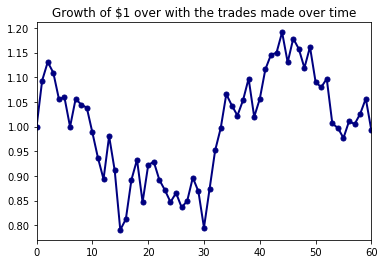
In the final submitted code, I am running it for only the most optimized iteration that I obtained.

Performance Results:

Best InSample performance:

N=20, M=15, L=20 and U =15. The way I have written the code, I do not need to include weights as parameter because my program calculates the optimal weight vector based on each group of N,M,L and U.

|  |  |
| --- | --- |
| Total Profit/Loss (in $): | $ -36,634.18 |
| Average Profit/Loss per trade (in $): | $ -610.57 |
| Annualized Return on Portfolio (%): | -0.1% |
| Portfolio Sharpe Ratio: | -0.44 |
| Maximum Drawdown ($): | $ 3,404,015.87 |
| Portfolio Volatility (Annualized) (in %) | 21.68% |
| Percentage of winning trades(%) | 49.18% |



Best OutSample performance:

N=20, M=15, L=20 and U =15. The way I have written the code, I do not need to include weights as parameter because my program calculates the optimal weight vector based on each group of N,M,L and U.

|  |  |
| --- | --- |
| Total Profit/Loss (in $): | $575,150.92 |
| Average Profit/Loss per trade (in $): | $47,929.24 |
| Annualized Return on Portfolio (%): | 36.18% |
| Portfolio Sharpe Ratio: | 2.02 |
| Maximum Drawdown ($): | $1,012,414.57 |
| Portfolio Volatility (Annualized) (in %) | 17.91% |
| Percentage of winning trades(%) | 50% |

