1. Initial Asset=100M (parameter)

2. For each buying period (e.g. first trading day in May, September and November), divide current asset into parts (assetPiece) equal to the number of files in the related folder.

3. read the related folder and extract opening price for every stock in the folder, buy amount of stock (stored in a numpy array: openingPrice)

stockAmount = FLOOR(assetPiece/(100\*dailyOpeningPrice)). The remainder is assetPiece-stockAmount\*100\*dailyOpeningPrice

4. For each selling period (e.g. last trading day before each buying period), get the current asset by adding up the products of each stockAmount\*dailyClosingPrice + remainders

(may get current asset each day later on)

5. put the current asset in step 2 until running out of the outer folder

6. get the final asset (or net value)

time span: 2005.5.1(first trading day:5.9)-2014.4.30 (set a variable for current date)

code: driver for processing data, trading for buying and selling (passing cash: total asset for buying and cash remainders for selling, number of stocks, opening/closing price vectors and returning asset and stockAmount (for buying))

Added Features:

Daily net value

Deal with issues of available trading days

Use 000001index to determine all the available trading days in the market

Change num parameter every day in the trading month, increase num in the loop to look for available trading stocks for opening prices

In either April, September or November, the buying date could be any time from the first available trading day to the last one, (e.g. 09/13 in Sep), so each stock could have different buying date in the month.

As for the selling date, the last trading day before each selected month (4,9,11) is going to be found. For example, if the last trading day for a stock is 6/7 with closing price of 2.81, so for each trading day after 6/7, we calculate net value still with 2.81 until we find the next available trading day.

Pass dates vectors in each iteration for all the directories

Directory for each period:

Name of each directory should be the first trading day of the starting month of the period

There won’t be a folder for the very end of the last period: 20140501 since we only need to sell all the stocks in the last trading day in 201404, so we set the end date manually in a variable.

Trading and get daily net value:

a vector for cash remaining (asset allocation initially) of each stock

a vector for flags of each stock if it is bought

divide the dates, opening and closing for the time span given in the timeVector, count the start of each stock (dates[i]>dates[i+1])

a vector for number of shares of each stock

iterate through the timeVector:

check through the flag vector, if it is False (wasn’t bought), see if it has an opening price for that day, buy if so, stay otherwise; if it is True (was bought), calculate its daily value by multiplying its number and its closing price on that day. If it doesn’t have a closing price, use the previous available closing price and so on.

singlePortfolio():

inputs:

#path:(string) the current portfolio folder, which gets changed in \_\_main\_\_

#timeVector:(datetime list) time span of current period

#cash:(float) available cash to invest

outputs:

assetNum: assetVector[-1]

assetVector: the vector of daily total asset

Error log:

path:/Users/xinyuan/Desktop/codeForPaper/data/20050901

driver.py:90: RuntimeWarning: invalid value encountered in double\_scalars

allocationVector[i]=allocationVector[i]-np.sum(amountVector[i]\*100\*bid)

20050509 000728

1890: 5/9/05 0

opening price is zero

Net value of the first day should be calculated by the closing prices of the first day over the opening prices (i.e. initial asset) instead of just 1, all the other net values are the asset with the closing prices of the corresponding day over the initial asset

And the net value of the benchmark is the closing price of the first day over the opening price of that day

Ascending & Descending: modify the code to select double-level factors. Select the first factor with top 6 ascending or descending, and select the second one with top 3 after the first level.

Timing strategy:

#Set a flag for the position (0 or 1), 0 initially and finally

Count the number of changing positions in signal function, since in trading function, some stocks may have staggering on the trading day (i.e. not available for trading on the signal day)

If not\_last\_day\_of\_period:

if buyFlags[i]==0 and not selling\_signal:

buy

get daily asset

elif buyFlags[i]==0 and selling\_signal:

stop buying, and sell all other holding assets to cash

get daily asset

#elif buyFlags[i]!=0 and not selling\_signal:

#get daily asset

elif buyFlags[i]!=0 and selling\_signal and pos==1:

sell all holding assets to cash

get daily asset

#elif buyFlags[i]!=0 and not buying\_signal:

#get daily asset

elif buyFlags[i]!=0 and buying\_signal and pos==0:

buy all holding assets with cash

get daily asset

else:

get daily asset

else:

sell all holding assets to cash

get daily asset