

Boosting Alpha Factors in Chinese A Share Stock Market

Reporter : Zhou Kaiwen

Report Date : 2017.04.17

● Multi-Factor model Background

A multi-factor model is a financial model that employs multiple factors in its computations to explain market phenomena and/or equilibrium asset prices. The multi-factor model can be used to explain either an individual security or a portfolio of securities. It does so by comparing two or more factors to analyze relationships between variables and the resulting performance.

Multi-factor models are used to construct portfolios with certain characteristics, such as risk, or to track indexes. When constructing a multi-factor model, it is difficult to decide how many and which factors to include. Also, models are judged on historical numbers, which might not accurately predict future values.

Multi-factor models can be divided into three categories: macroeconomic models, fundamental models and statistical models. Macroeconomic models compare a security's return to such factors as employment, inflation and interest. Fundamental models analyze the relationship between a security's return and its underlying financials, such as earnings. Statistical models are used to compare the returns of different securities based on the statistical performance of each security in and of itself.

Factors are compared using the following formula:

$$R_i = a_i + \beta_{i(m)} * R_m + \beta_{i(1)} * F_1 + \beta_{i(2)} * F_2 + \dots + \beta_{i(N)} * F_N + e_i$$

Where:

R_i is the returns of security i

R_m is the market return

$F(1, 2, 3 \dots N)$ is each of the factors used

β is the beta with respect to each factor including the market (m)

e is the error term

a is the intercept

● Previous Work

As we know, the essence of multi-factor models is single factor, find a good factor can obviously improve the performance of multi-factor model, I have found several un-public good single factors for Chinese A share stock markets. These single factor's IC (Information Coefficient, defined as the correlation between factor value in time t with stock return in time $t+1$) is above 10% (traditionally criterion of a good single factor is $IC > 3\%$), I will not show my factor construction formula but I can say that some of the factors are derivatives of momentum factors, reversal factors.

■ Why single factor works ?

For example, Momentum is the tendency for assets that have performed well (poorly) in the recent past to continue to perform well (poorly) in the future, at least for a short period of time. Traditionally momentum factor is defined as the last 12 months of returns excluding the most recent month (in other words, month 2-12).

Logic is more important than number. The intuition behinds momentum factors is summarized as following:

■ Financial institutions, in particular mutual funds and foreign investors (which generally

are also institutional investors), are momentum traders, while private households are instead contrarians taking the other side. The data were statistically significant at the 1 percent confidence level.

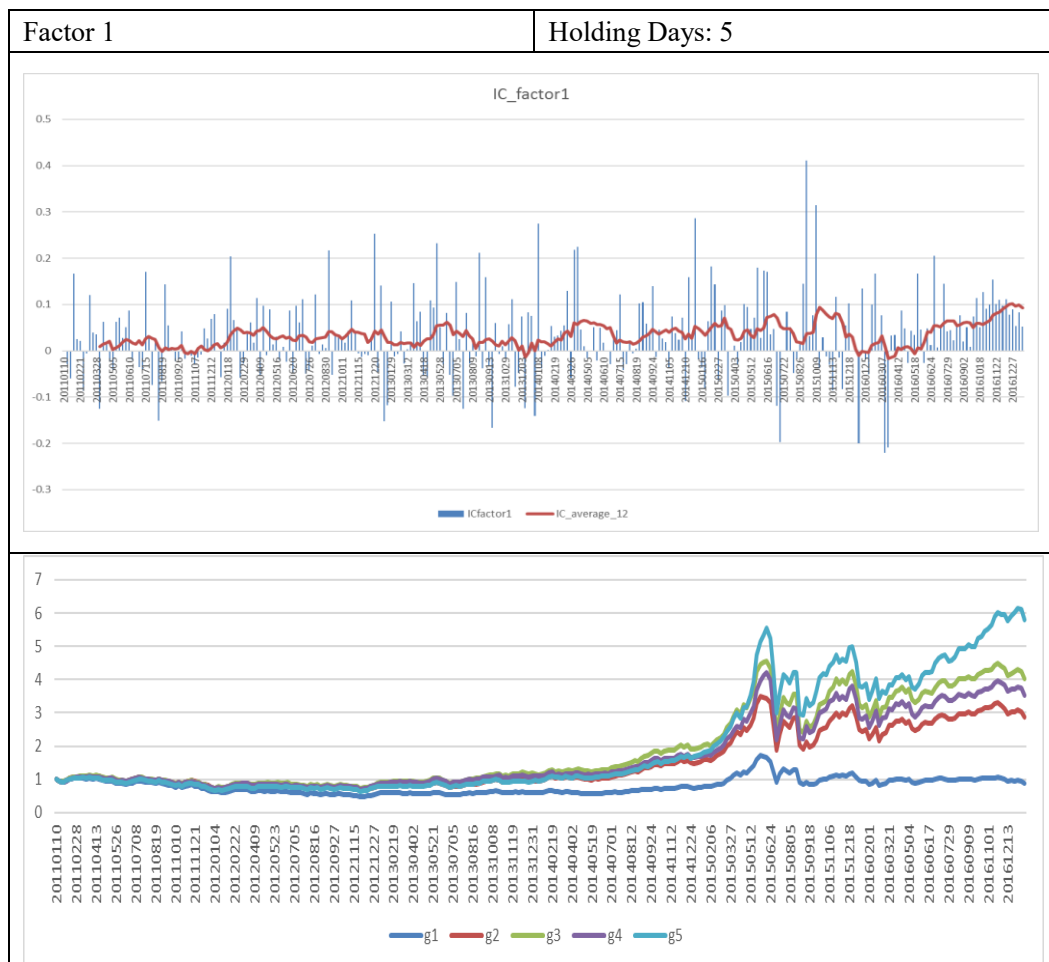
- Aggregate momentum trading increases during market downturns and in highvolatility phases.
- When looking at winner and loser stocks separately, momentum trading is particularly strong among losers.
- Another behavioral explanation arises from what is called the disposition effect. Investors tend to sell winning investments prematurely to lock in gains and hold on to losing investments too long in the hope of breaking even.
- Yale University professor Tobias Moskowitz points out: “Underreaction results from information travelling slowly into prices. That causes momentum.

● Factors Historical Performance

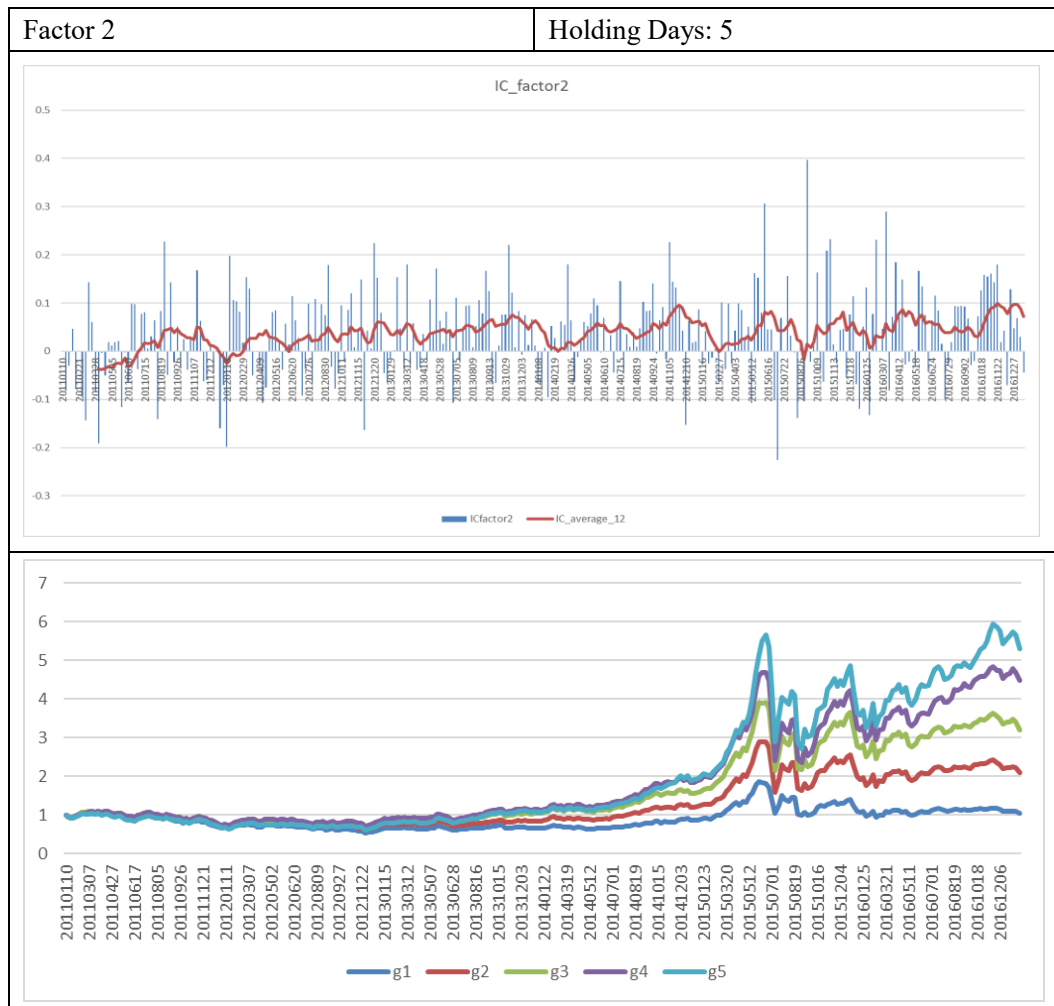
■ **Backtest Date :** 2011.01.01 – 2016.12.31

Backtest Market : Chinese A share market

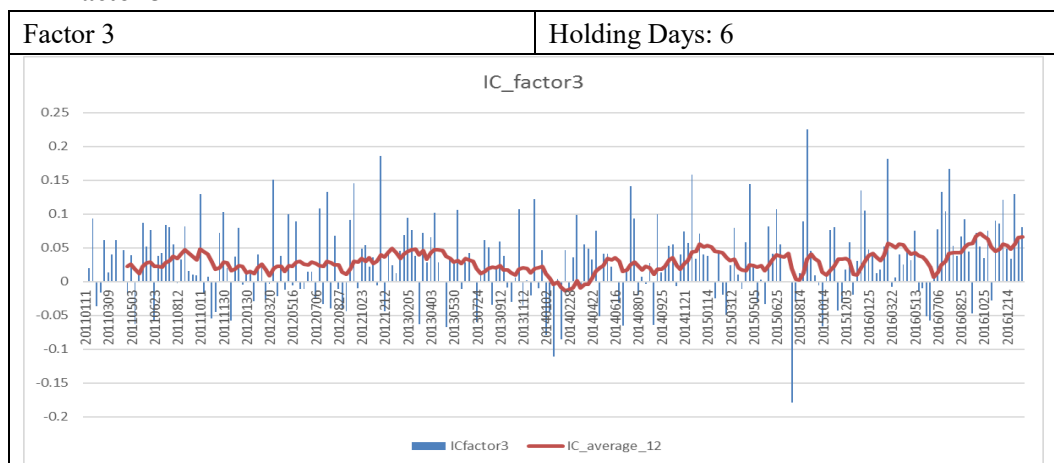
■ Factor 1

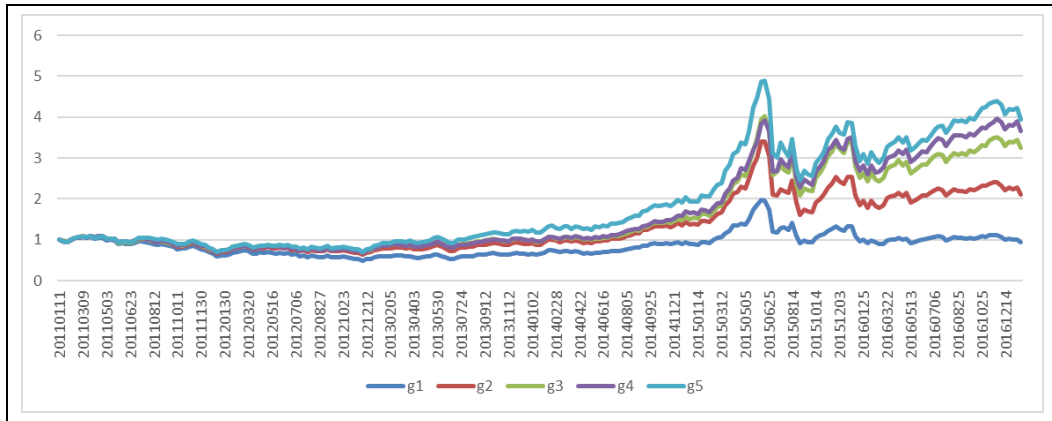


Factor 2

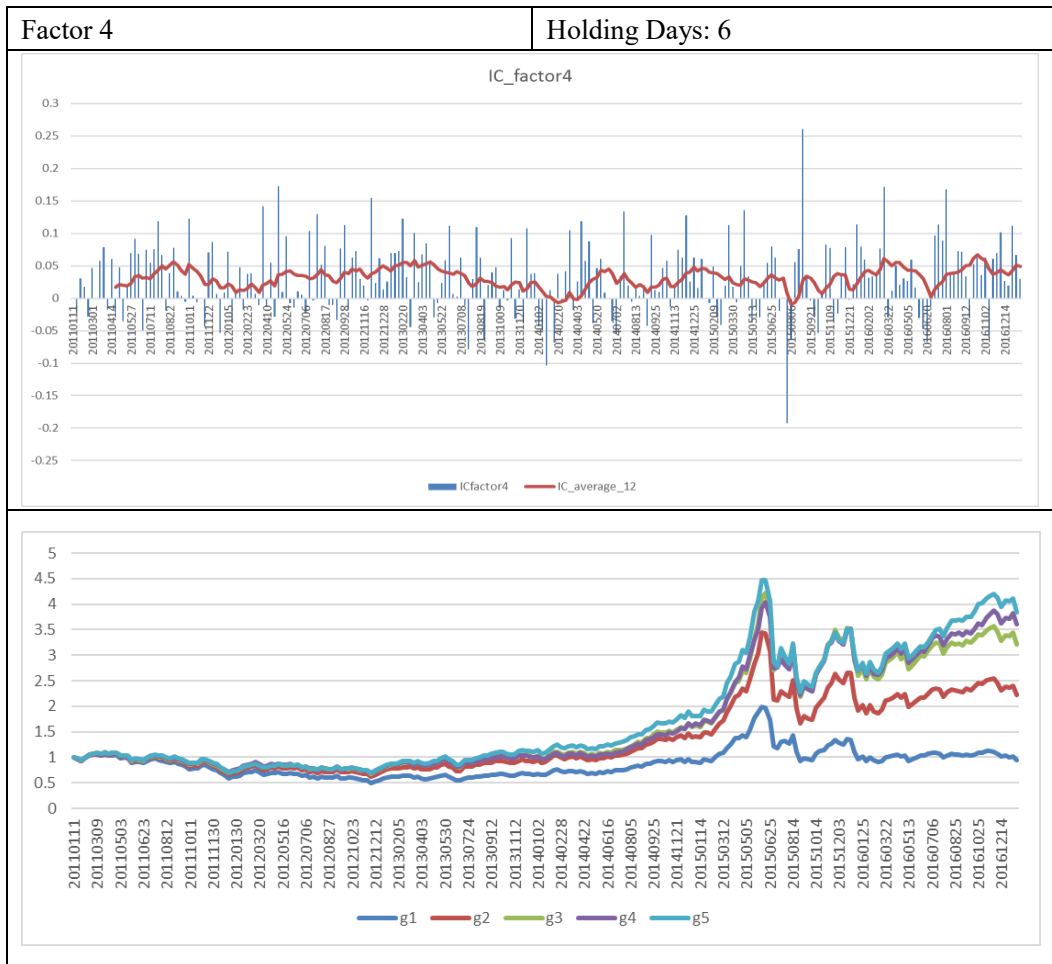


Factor 3

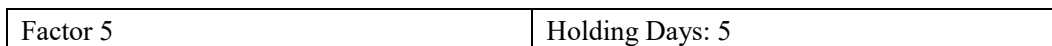


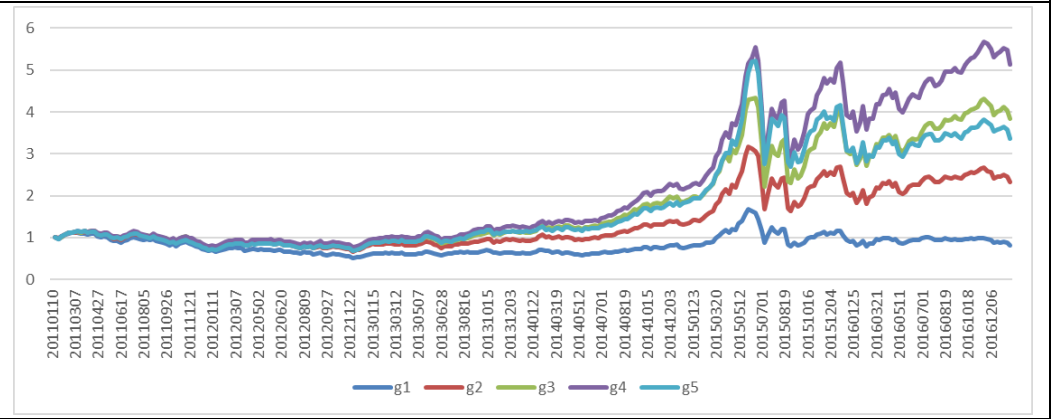
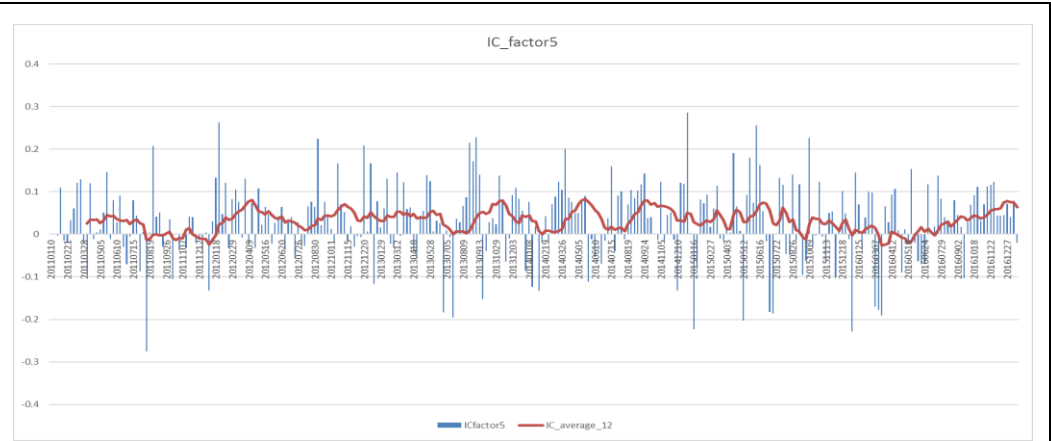


Factor 4

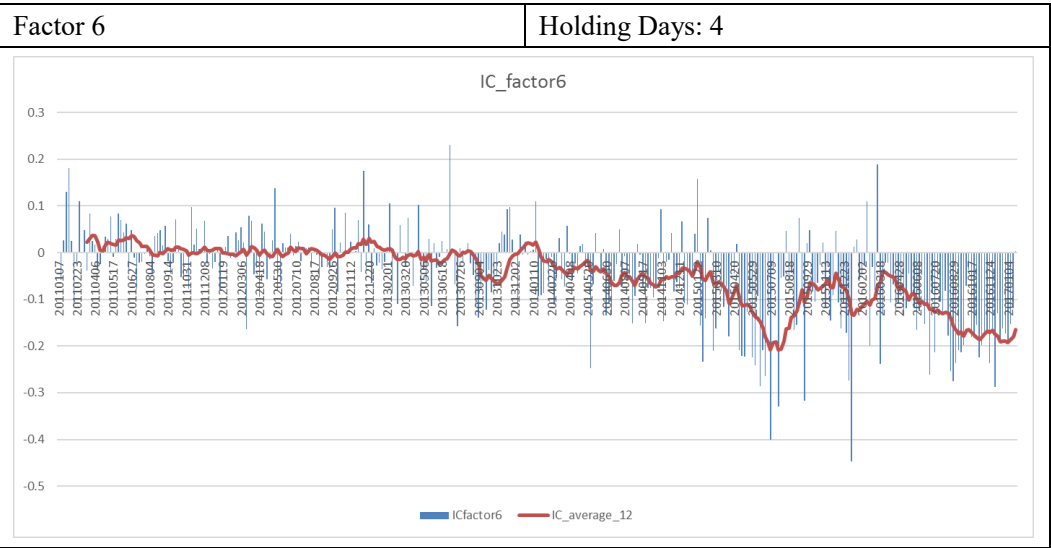


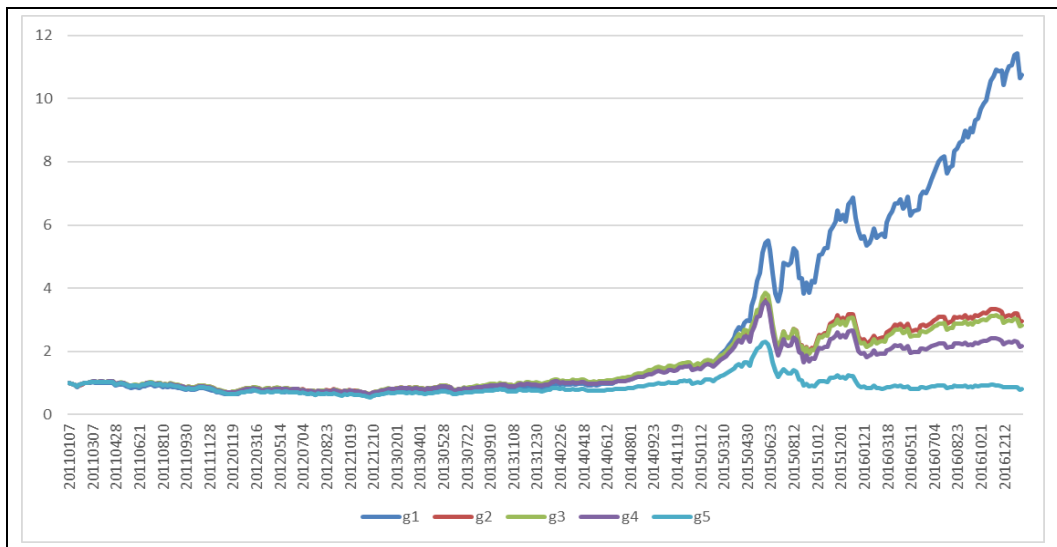
Factor 5



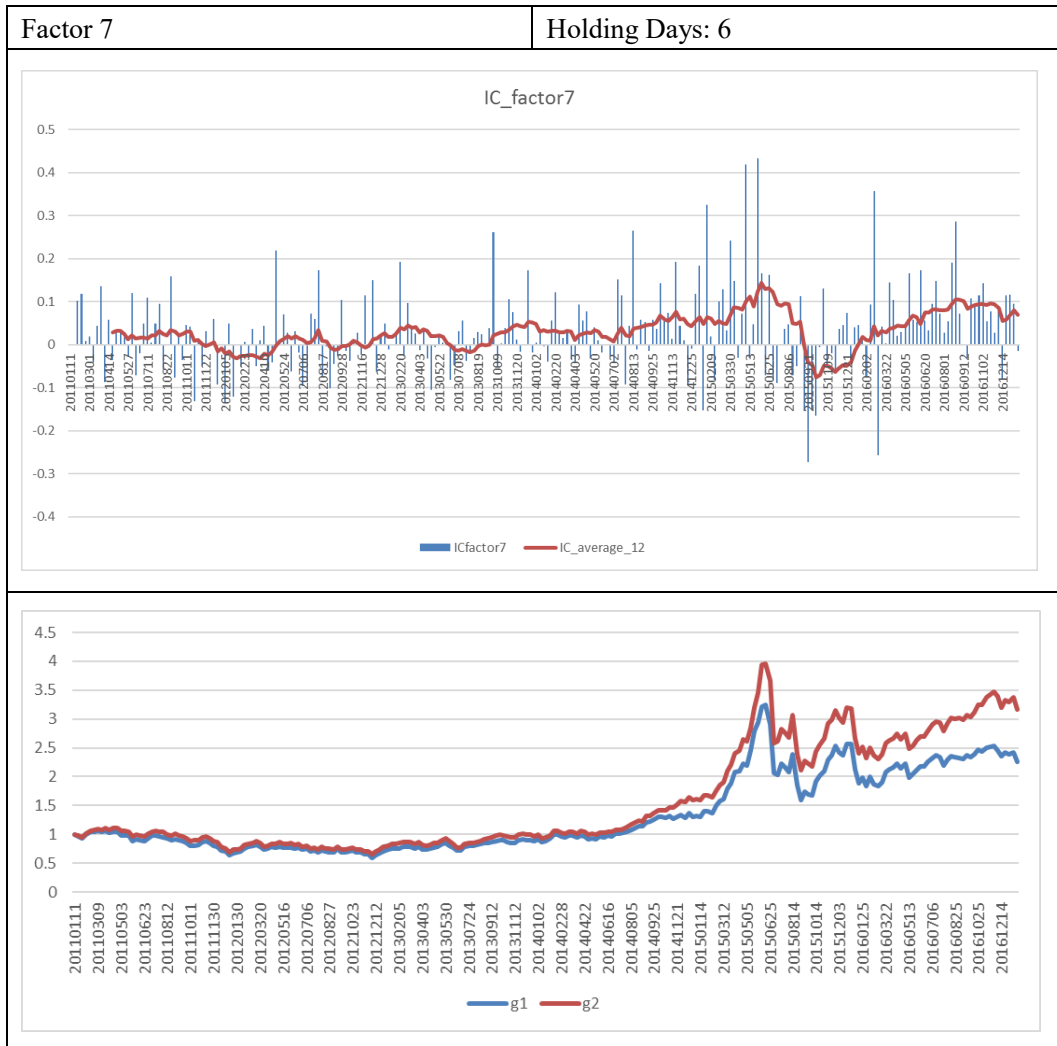


Factor 6





Factor 7



- **What are the problems within these single factors ?**

All these single factors are good factors, but all these factors have a high turnover rate. So there are two things to solve: (1) how to combine these single factors together to a single strong factor? (2) how to make the new strong factor's turnover rate as low as much possible?

- **Different methods to solve the above problems**

To solve the problem1, methods are including: static score method, dynamic score method and regression method etc. To solve the problem2, we generally use the optimization methods to add some constraints such as turnover rate restriction, industry restriction etc.

Here, I want to solve the two problems simultaneously via machine learning methods, i.e. adaboosting methods.

Adaboosting methods can boost several weak classifiers into a strong classifier, so it can solve the problem1 easily, we also notice that adaboosting method can hugely decrease the error rate, so less error rate means less wrong classified rate and momentum effect makes the low turnover rate possible.

- **Algorithm Description**

- **Algorithm 1 : Naïve Boosting**

(1) Initialize $w_i = 1/N$, $i = 1, 2, \dots, N$

(2) For $m = 1$ to M :

Fit a classifier $G(x)$ to training data using weights w_i

(3) Output $G(x) = \text{sign}[w_i * G_m(x)]$

- **Algorithm 2 : Standard Boosting (Adaboosting)**

(1) Initialize $w_i = 1/N$, $i = 1, 2, \dots, N$

(2) For $m = 1$ to M :

Fit a classifier $G(x)$ to training data using weights w_i

Compute:

$$err_m = \frac{\sum_{i=1}^N w_i I(y_i \neq G_m(x))}{\sum_{i=1}^N w_i}$$

$$\text{Compute } \alpha_m = \log\left(\frac{1 - err_m}{err_m}\right) \text{ update } w_i = w_i \exp(\alpha_m I(y_i \neq G_m(x)))$$

(3) Output $G(x) = \text{sign}[\sum_{m=1}^M \alpha_m G_m(x)]$

- **Algorithm 3 : Revised Boosting**

Note: algorithm 2 is free of order of weak classifier $G_m(x)$, because of the penalty coefficient α_m , here I

want to revise the boosting method to be order dependence, i.e. exclude the penalty coefficient α_m , the order

of the weak classifier $G_m(x)$ is determined by the err_m , the less err_m the higher priority, but one thing should be noted, some 0-1 factor $Y_m(x)$ are not suitable for the situation, I still let them free of order and has the penalty coefficient α_m .

(1) Initialize $w_i = 1/N$, $i = 1, 2, \dots, N$

(2) For $m = 1$ to M :

Fit a classifier $G_m(x)$ to training data using weights w_i

Compute:

$$err_m = \frac{\sum_{i=1}^N w_i I(y_i \neq G_m(x))}{\sum_{i=1}^N w_i}$$

Compute $\alpha_m = \log\left(\frac{1 - err_m}{err_m}\right)$ update $w_i = w_i \exp(\alpha_m I(y_i \neq G_m(x)))$

(3) $K = \operatorname{argmin}(err_m)$

(4) $G(x) = \operatorname{sign}\left[\sum_{k=1}^{K1} G_k(x) + \sum_{m=1}^{K2} \alpha_m Y_m(x)\right]$, $K1$ is the normal factor number and $K2$ is the 0-1 factor number

■ Algorithm Footnotes:

Q: Why use $\alpha_m = \log\left(\frac{1 - err_m}{err_m}\right)$?

A: It is derived from sigmoid function $err_m = \frac{1}{1 + \exp^\alpha} \rightarrow \alpha_m = \log\left(\frac{1 - err_m}{err_m}\right)$

● Algorithm Performance

■ Backtest Information

(1) **Backtest Market:** Chinese A share market

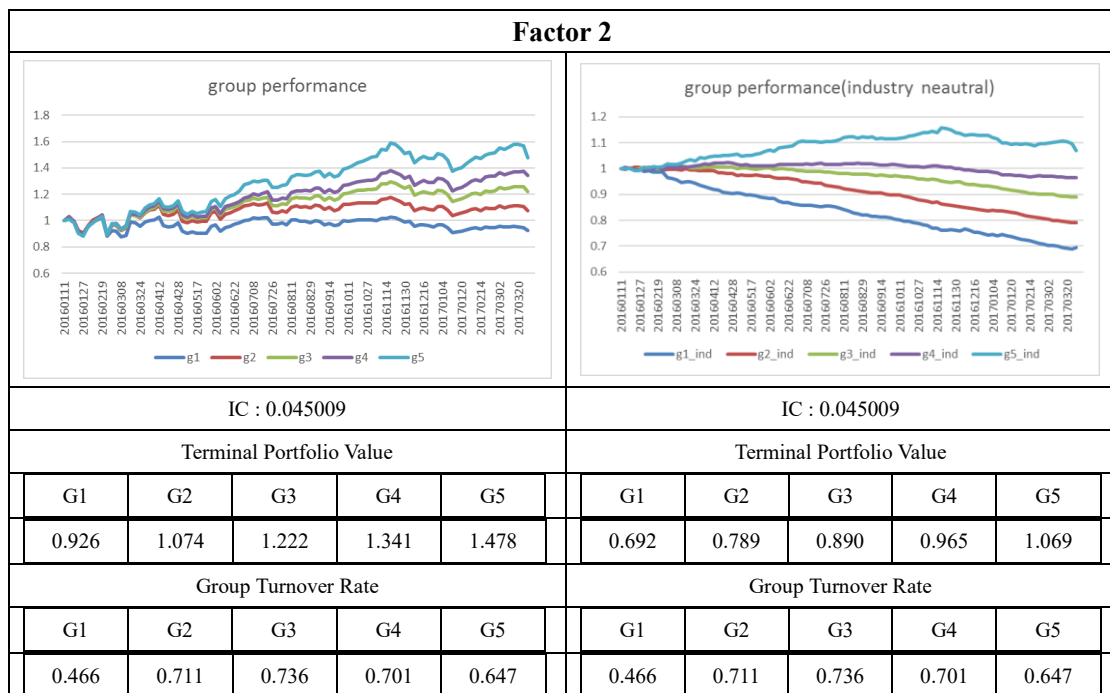
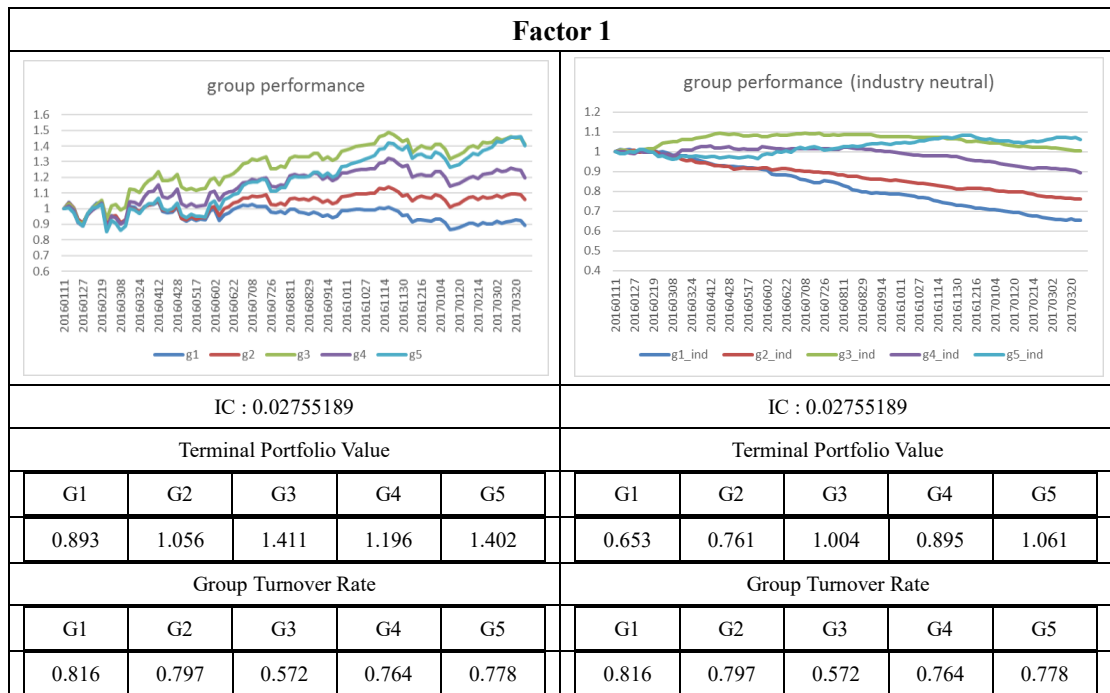
(2) **Backtest Date:** 2016.01.01 - 2017.03.31

(3) **Backtest Holding Day:** 3 day

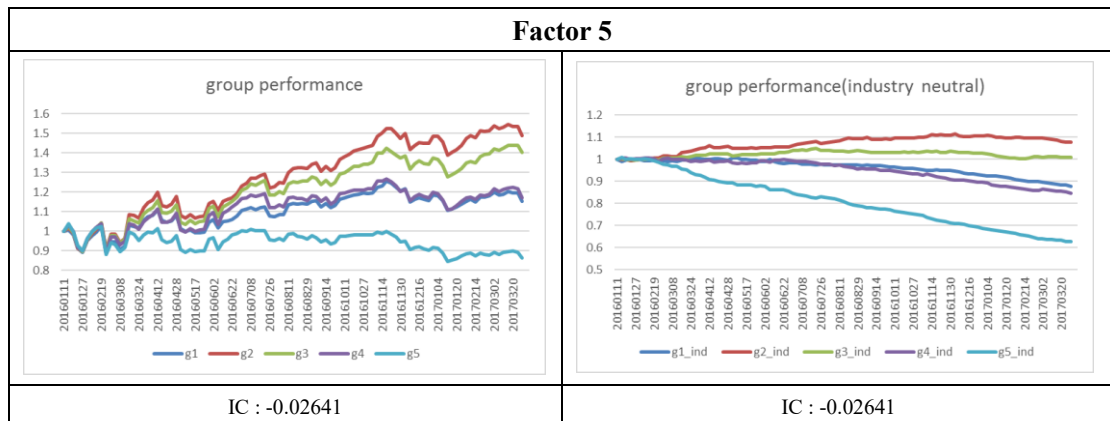
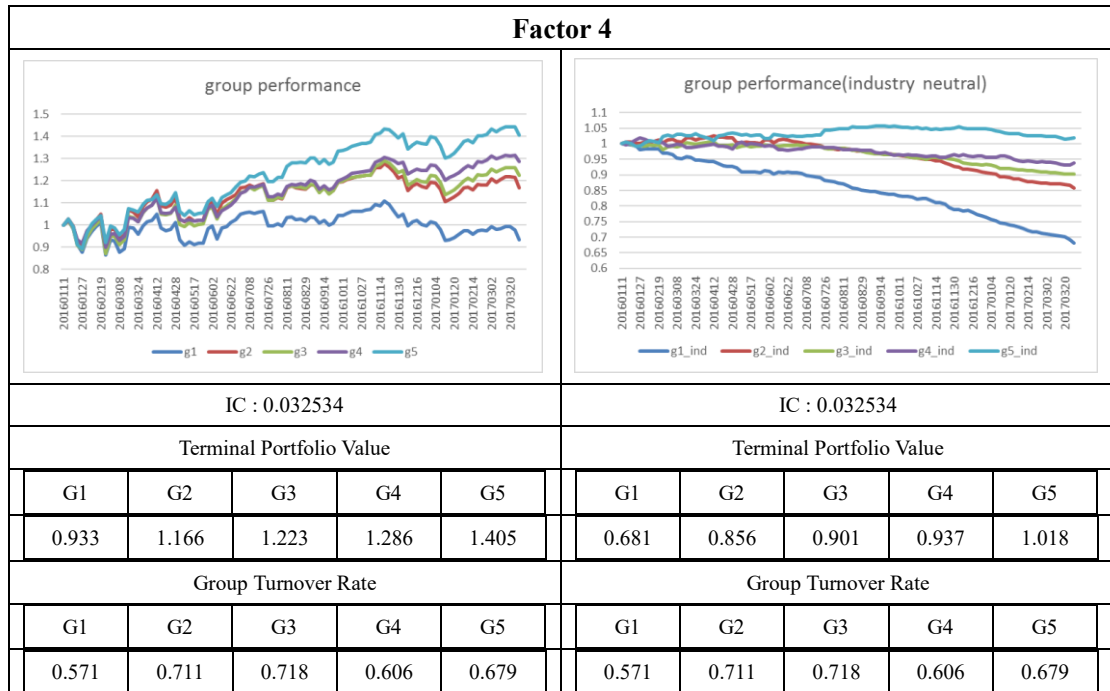
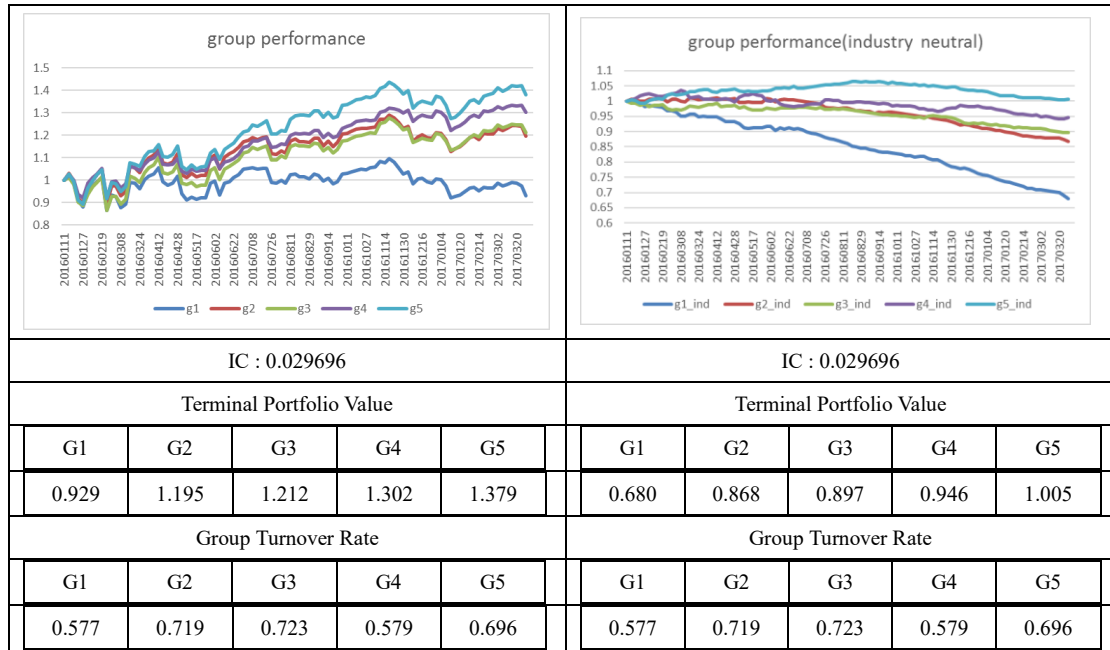
Note: the reason that I set the holding day as 3 just because I wish the all factors (classifiers) to become a weak classifier and to test and compare the pure performance of different boosting algorithms.

■ Single Factors Performance Profile

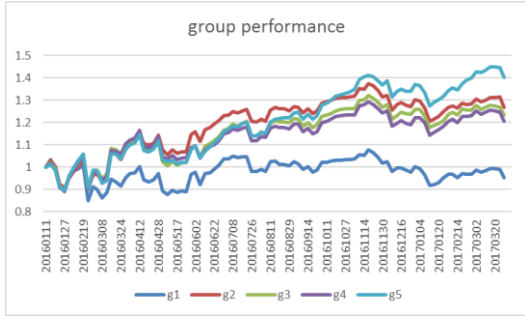
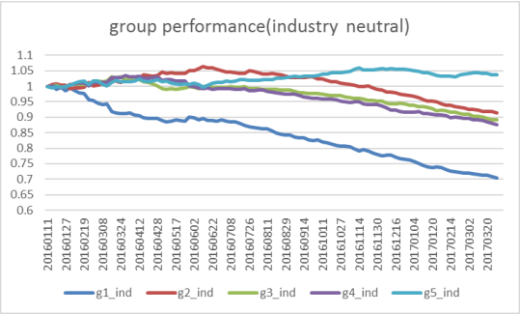
Most single factors during this test are still have an average IC about 3% (still good factors), compared with the historical performance, these factors have a underperformance largely due to the backtest holding day and partly due to the new dataset. And all these factors' groups have a high turnover rate about 60% - 70%.

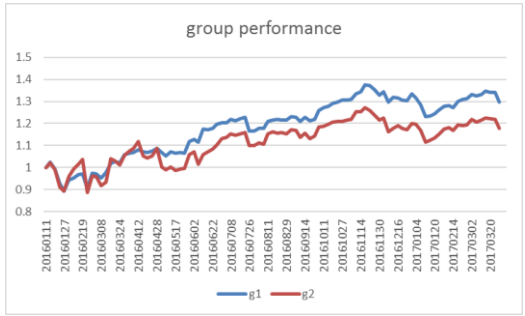
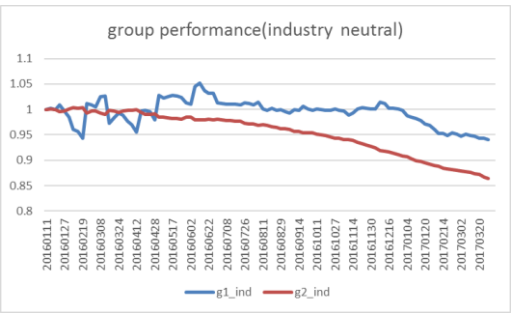


Factor 3



Terminal Portfolio Value					Terminal Portfolio Value				
G1	G2	G3	G4	G5	G1	G2	G3	G4	G5
1.150	1.489	1.403	1.172	0.864	0.876	1.076	1.009	0.845	0.627
Group Turnover Rate					Group Turnover Rate				
G1	G2	G3	G4	G5	G1	G2	G3	G4	G5
0.633	0.711	0.731	0.746	0.556	0.633	0.711	0.731	0.746	0.556

Factor 6									
									
IC : 0.0495					IC : 0.0495				
Terminal Portfolio Value					Terminal Portfolio Value				
G1	G2	G3	G4	G5	G1	G2	G3	G4	G5
0.952	1.267	1.232	1.206	1.402	0.704	0.913	0.891	0.876	1.036
Group Turnover Rate					Group Turnover Rate				
G1	G2	G3	G4	G5	G1	G2	G3	G4	G5
0.673	0.778	0.772	0.775	0.783	0.673	0.778	0.772	0.775	0.783

Factor 7									
									
IC : -0.00187					IC : -0.00187				
Terminal Portfolio Value					Terminal Portfolio Value				
	G1	G2				G1	G2		
	1.298	1.178				0.941	0.863		
Group Turnover Rate					Group Turnover Rate				
	G1	G2				G1	G2		
	0.575	0.140				0.575	0.140		

■ Adaboosting Algorithm Performance profile

From the table, we can find that (1) all boosting methods can increase the IC a lot, nearly twice time (from 3.x% to 6.x%). (2) all boosting methods have little use of decreasing group turnover rate (about 70% - 80%). (3) algorithm 2 has the best performance , i.e. algorithm 2 has the highest terminal portfolio value (G5 and G5(industry neutral)) ,highest IC (6.66579%) and lowest turnover rate (74.3%,G5).

Algorithm 1

group performance

Algorithm 1

group performance (industry neutral)

IC : 0.0637067					IC : 0.0637067				
Terminal Portfolio Value					Terminal Portfolio Value				
G1	G2	G3	G4	G5	G1	G2	G3	G4	G5
1.007	1.109	1.168	1.221	1.518	0.737	0.811	0.851	0.886	1.116
Group Turnover Rate					Group Turnover Rate				
G1	G2	G3	G4	G5	G1	G2	G3	G4	G5
0.697	0.783	0.785	0.765	0.744	0.697	0.783	0.785	0.765	0.744

Algorithm 2

group performance

Algorithm 2

group performance (industry neutral)

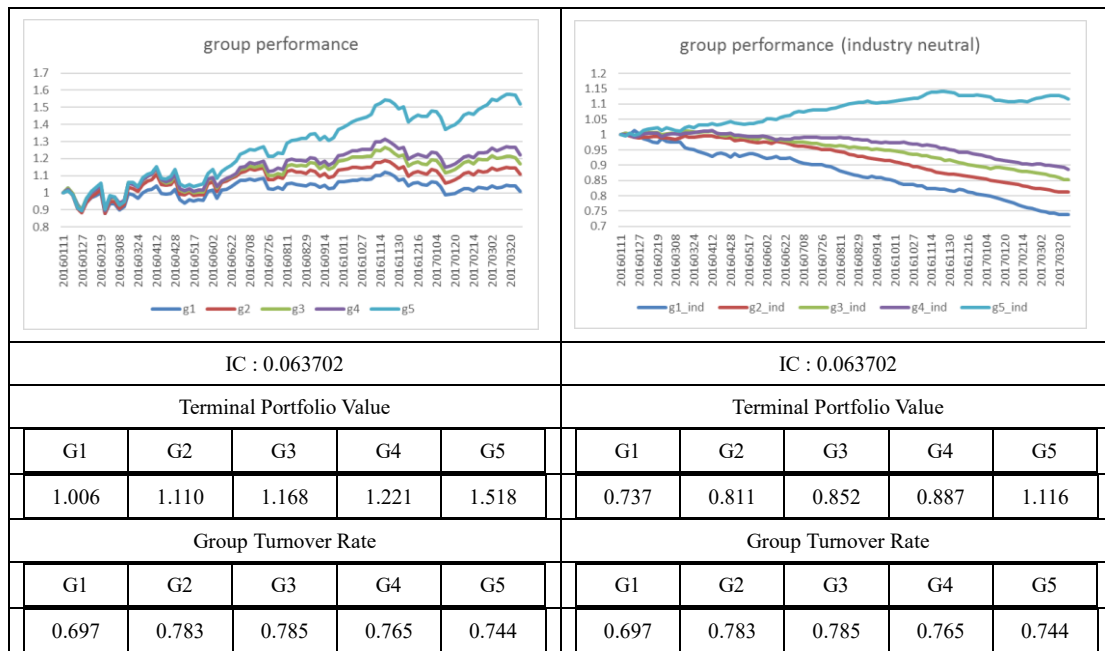
IC : 0.066579					IC : 0.066579				
Terminal Portfolio Value					Terminal Portfolio Value				
G1	G2	G3	G4	G5	G1	G2	G3	G4	G5
0.986	1.118	1.198	1.204	1.522	0.719	0.822	0.871	0.875	1.119
Group Turnover Rate					Group Turnover Rate				
G1	G2	G3	G4	G5	G1	G2	G3	G4	G5
0.701	0.784	0.786	0.766	0.743	0.701	0.784	0.786	0.766	0.743

Algorithm 3

group performance

Algorithm 3

group performance (industry neutral)



■ Statistical Table of Backtest Result

Legend: min: max:

Name	Best Group Value	BGV Turnover Rate	Worst Group Value	WGV Turnover Rate	IC(%)
Single Factors Profile					
Factor1	1.411	0.572	0.893	0.816	2.755189
Factor2	1.478	0.647	0.926	0.466	4.5009
Factor3	1.379	0.696	0.929	0.577	2.9696
Factor4	1.405	0.679	0.933	0.571	3.2534
Factor5	1.489	0.711	0.864	0.556	-2.641
Factor6	1.402	0.783	0.952	0.673	4.95
Factor7	1.298	0.575	1.178	0.140	-0.187
Algorithms Profile					
Algo 1	1.518	0.744	1.007	0.697	6.37067
Algo 2	1.522	0.743	1.118	0.784	6.6579
Algo 3	1.518	0.744	1.006	0.697	6.3702

From the above table, we can get some conclusions : (1) boosting methods can really improve the factor IC and portfolio performance. (2) boosting methods have little use of reducing turnover rate. (3) differences between different boosting methods are small. (4) find the good single factors is the most important thing to improve the portfolio performance.

● Algorithm Codes (in python)

```

1. # adaboosting kernal methods (combine the old factors into one new factor)
2. # method 1
3. # method description : just bumping method , equal weighted all factors to generate a new factor
4. finalresult['adaboost1'] = 0

```

```

5. for j in np.arange(factornamelen):
6.     finalresult['adaboost1'] = finalresult['adaboost1'] + finalresult[factornames[j]]
7.
8.     # method 2
9.     # method description : use the normal adaboosting method (factors order is random , so there is a penalty on the coefficient)
10.    backtestdate = finalresult['date'].drop_duplicates().reset_index(drop = True)
11.    # backtestdate = backtestdate.drop([backtestdate.index.max()]) # delete the last date due to no period data
12.    backtestdatelen = len(backtestdate)
13.
14.    dateparts = locals() # generate the local var sets
15.    for j in np.arange(backtestdatelen): # generate the different var name date'j'
16.        dateparts['date%s'% j] = finalresult[finalresult['date']==backtestdate[j]]
17.
18.    def renewweight(df):
19.        if df['factorgroup'] != df['returngroup']:
20.            df['weight'] = df['weight'] * math.exp(errorrate)
21.        else:
22.            df['weight'] = df['weight'] * math.exp(-errorrate)
23.        return df
24.
25.    for j in np.arange(backtestdatelen-1):
26.        locals()['date'+str(j+1)]['adaboost2'] = 0
27.        stocknumber = locals()['date'+str(j)].shape[0]
28.        locals()['date'+str(j)]['weight'] = 1 / stocknumber # initial the weight
29.        for i in np.arange(factornamelen):
30.            factor = factornames[i]
31.            locals()['date'+str(j)] = locals()['date'+str(j)][locals()['date'+str(j)][factor].notnull()] # delete the null factor
32.            locals()['date'+str(j)]['factorrank'] = locals()['date'+str(j)][factor].rank(method='dense')
33.            if i<6 : # these factors are normal factors (have lots of values)
34.                factorgroup = locals()['date'+str(j)][['factorrank']].drop_duplicates('factorrank').rank(method='dense').sort('factorrank').shape[0]
35.                locals()['date'+str(j)]['factorgroup'] = locals()['date'+str(j)]['factorrank'].map(lambda x: math.ceil(x/factorgroup*5)) # group 5 according to the factor
36.                locals()['date'+str(j)]['shift_periodreturn'] = locals()['date'+str(j)][['shift_periodreturn']].fillna(value = 0) # fill the periodreturn nan = 0
37.                locals()['date'+str(j)]['returnrank'] = locals()['date'+str(j)]['shift_periodreturn'].rank(method='dense')
38.                locals()['date'+str(j)]['returngroup'] = locals()['date'+str(j)]['returnrank'].map(lambda x: math.ceil(x/factorgroup*5)) # group 5 according to the next period return
39.            else : # this factor('alpha021') is a 0-1 factor (only have two values, so can only be divided into 2 groups)

```

```

40.     factorgroup = locals()['date'+str(j)][['factorrank']].drop_duplicates('factorrank').rank(method='dense').sort('factorrank').shape[0]

41.     locals()['date'+str(j)]['factorgroup'] = locals()['date'+str(j)]['factorrank'].map(lambda x: math.ceil(x/factorgroup*2)) # group 2 according to the factor

42.     locals()['date'+str(j)]['shift_periodreturn'] = locals()['date'+str(j)]['shift_periodreturn'].fillna(value = 0) # fill the periodreturn nan = 0

43.     locals()['date'+str(j)]['returnrank'] = locals()['date'+str(j)]['shift_periodreturn'].rank(method='dense')

44.     locals()['date'+str(j)]['returngroup'] = locals()['date'+str(j)]['returnrank'].map(lambda x: math.ceil(x/factorgroup*2)) # group 2 according to the next period return

45.     errorrate = locals()['date'+str(j)].where(locals()['date'+str(j)]['factorgroup'] != locals()['date'+str(j)]['returngroup']).dropna().shape[0]/stocknumber # to renew the weight

46.     weighterrorrate = locals()['date'+str(j)].where(locals()['date'+str(j)]['factorgroup'] != locals()['date'+str(j)]['returngroup']).dropna()['weight'].sum() # to renew the adaboost factor

47.

48.     locals()['date'+str(j)] = locals()['date'+str(j)].apply(renewweight,axis =1) # renew the weight

49.     locals()['date'+str(j)]['weight'] = locals()['date'+str(j)]['weight'] / locals()['date'+str(j)]['weight'].sum() # weight uniform

50.

51.     locals()['date'+str(j+1)]['adaboost2'] =locals()['date'+str(j+1)]['adaboost2'] + 1/weighterrorrate * 1
        ocalas()['date'+str(j+1)]['factor'] # renew the adaboost

52.

53.     locals()['date'+str(j)] = locals()['date'+str(j)].drop(['weight','factorrank','factorgroup','returnrank','returngroup'],axis =1)

54.

55. finalresult2 = locals()['date'+str(1)] # append all data to become finalresult2 and delete all the temporary datej

56. for j in np.arange(backtestdatelen-2):

57.     finalresult2 = finalresult2.append(locals()['date'+str(j+2)])

58.     del locals()['date'+str(j+2)]

59. del locals()['date'+str(0)]

60. del locals()['date'+str(1)]

61.

62.

63. # method 3

64. # method description : use the factor according to the minimum error rate ( nonrandom order , so there is no penalty coefficient )

65. backtestdate = finalresult2['date'].drop_duplicates().reset_index(drop = True)

66. #backtestdate = backtestdate.drop([backtestdate.index.max()]) # delete the last date due to no period data

67. backtestdatelen = len(backtestdate)

68.

69. dateparts = locals() # generate the local var sets

70. for j in np.arange(backtestdatelen): # generate the different var name date'j'

```

```

71.     dateparts['date%s'% j] = finalresult2[finalresult2['date']==backtestdate[j]]
72.
73.     def renewweight(df):
74.         if df['factorgroup'] != df['returngroup']:
75.             df['weight'] = df['weight'] * math.exp(errorrate)
76.         else:
77.             df['weight'] = df['weight'] * math.exp(-errorrate)
78.         return df
79.
80.
81.     for j in np.arange(backtestdatelen-1): # j represents backtestdate index
82.
83.         locals()['date'+str(j+1)]['adaboost3'] = 0
84.         stocknumber = locals()['date'+str(j)].shape[0]
85.         locals()['date'+str(j)]['weight'] = 1 / stocknumber # initial the weight
86.         G1G5weightsum = [] # group1 + group5 weights
87.         index = [] # record the factor has been used
88.         k = 0
89.
90.         # generate G1G5weightsum and index and update adaboost3
91.         while k <= factornamelen-1: # k represents G1G5weightsum index
92.             if k < factornamelen-1:
93.
94.                 i = 0 # i represents factor index
95.                 tempweightsum = [] # record all the weightsum
96.                 tempindex = [] # record all the index
97.                 while i < factornamelen-1:
98.                     if i in index:
99.                         i = i+1 # i has been recorded in index
100.                    else :
101.                        factor = factornames[i]
102.                        locals()['date'+str(j)] = locals()['date'+str(j)][locals()['date'+str(j)][factor].notnull()] # delete the null factor
103.                        locals()['date'+str(j)]['factorrank'] = locals()['date'+str(j)][factor].rank(method='dense')
104.                        factorgroup = locals()['date'+str(j)][['factorrank']].drop_duplicates('factorrank').rank(method='dense').sort('factorrank').shape[0]
105.                        locals()['date'+str(j)]['factorgroup'] = locals()['date'+str(j)]['factorrank'].map(lambda x: math.ceil(x/factorgroup*5)) # group 5 according to the factor
106.                        G1G5weightsum_i = locals()['date'+str(j)].where( (locals()['date'+str(j)]['factorgroup']==1) | (
                            locals()['date'+str(j)]['factorgroup']==5)).dropna()['weight'].sum()
107.                        tempweightsum.append(G1G5weightsum_i)
108.                        tempindex.append(i)
109.                        i = i+1
110.                    # find the best factor

```



```

111.     tempweightsumindex = np.where(pd.DataFrame(tempweightsum) == pd.DataFrame(tempweightsum).min())[0][0]
112.     factorindex = tempindex[tempweightsumindex]
113.     GIG5weightsum.append(tempweightsum[tempweightsumindex])
114.     index.append(factorindex)
115.     # update the weight
116.     factor = factornames[factorindex]
117.     locals()['date'+str(j)] = locals()['date'+str(j)][locals()['date'+str(j)][factor].notnull()] # delete the null factor
118.     locals()['date'+str(j)][factorrank] = locals()['date'+str(j)][factor].rank(method='dense')
119.     factorgroup = locals()['date'+str(j)][factorrank].drop_duplicates(factorrank).rank(method='dense').sort(factorrank).shape[0]
120.     locals()['date'+str(j)][factorgroup] = locals()['date'+str(j)][factorrank].map(lambda x: math.ceil(x/factorgroup*5)) # group 5 according to the factor
121.     locals()['date'+str(j)][shift_periodreturn] = locals()['date'+str(j)][shift_periodreturn].fillna(value = 0) # fill the periodreturn nan = 0
122.     locals()['date'+str(j)][returnrank] = locals()['date'+str(j)][shift_periodreturn].rank(method='dense')
123.     locals()['date'+str(j)][returngroup] = locals()['date'+str(j)][returnrank].map(lambda x: math.ceil(x/factorgroup*5)) # group 5 according to the next period return
124.     errorrate = locals()['date'+str(j)].where(locals()['date'+str(j)][factorgroup] != locals()['date'+str(j)][returngroup]).dropna().shape[0]/stocknumber # to renew the weight
125.     locals()['date'+str(j)] = locals()['date'+str(j)].apply(renewweight,axis =1) # renew the weight
126.     locals()['date'+str(j)][weight] = locals()['date'+str(j)][weight] / locals()['date'+str(j)][weight].sum() # weight uniform
127.     # update the adaboost3
128.     locals()['date'+str(j+1)][adaboost3] =locals()['date'+str(j+1)][adaboost3] + locals()['date'+str(j+1)][factor] # renew the adaboost
129.
130.     k = k+1
131.
132.     else: # k==6 , to deal with the 0-1 variable
133.         factor = factornames[k]
134.         locals()['date'+str(j)] = locals()['date'+str(j)][locals()['date'+str(j)][factor].notnull()] # delete the null factor
135.         locals()['date'+str(j)][factorrank] = locals()['date'+str(j)][factor].rank(method='dense')
136.         factorgroup = locals()['date'+str(j)][factorrank].drop_duplicates(factorrank).rank(method='dense').sort(factorrank).shape[0]
137.         locals()['date'+str(j)][factorgroup] = locals()['date'+str(j)][factorrank].map(lambda x: math.ceil(x/factorgroup*2)) # group 2 according to the factor
138.         locals()['date'+str(j)][shift_periodreturn] = locals()['date'+str(j)][shift_periodreturn].fillna(value = 0) # fill the periodreturn nan = 0
139.         locals()['date'+str(j)][returnrank] = locals()['date'+str(j)][shift_periodreturn].rank(method='dense')

```

```

140.     locals()['date'+str(j)]['returngroup'] = locals()['date'+str(j)]['returnrank'].map(lambda x: math.ceil(x/factorgroup*2)) # group 2 according to the next period return
141.
142.     errorrate = locals()['date'+str(j)].where(locals()['date'+str(j)]['factorgroup'] != locals()['date'+str(j)]['returngroup']).dropna().shape[0]/stocknumber # to renew the weight
143.     locals()['date'+str(j+1)]['adaboost3'] = locals()['date'+str(j+1)]['adaboost3'] + 1/errorrate * locals()['date'+str(j+1)]['factor'] # renew the adaboost
144.
145.     k = k+1
146.
147.     #print(j)
148.     locals()['date'+str(j)] = locals()['date'+str(j)].drop(['weight', 'factorrank', 'factorgroup', 'returnrank', 'returngroup'],axis =1)
149.
150. finalresult3 = locals()['date'+str(1)] # append all data to become finalresult2 and delete all the temporary datej
151. for j in np.arange(backtestdatelen-2):
152.     finalresult3 = finalresult3.append(locals()['date'+str(j+2)])
153.     del locals()['date'+str(j+2)]
154. del locals()['date'+str(0)]
155. del locals()['date'+str(1)]

```