

# 《机器学习驱动的基本面量化投资研究》

## 程序列表

李 斌，邵新月，李玥阳

武汉大学经济与管理学院

1. 数据及代码说明 .....	错误!未定义书签。
1) 数据说明: .....	1
2) 代码说明: .....	1
2. 代码整理 .....	4
a) MainFile.py .....	4
b) DataTransfrom.py .....	9
c) NWttest.py .....	12
d) ReturnSeriesTest.py .....	14
e) StrategyConstruct.py .....	16
f) FactorTest.py .....	27
g) DFN.py .....	45
h) RNNMODEL.py .....	48
i) Ensembleall.py .....	51
j) transecfee.py .....	52
k) selectFactor.py .....	54

## 1. 数据及代码说明

### 1) 数据说明：

- a) factor 文件夹中包含初始 96 项因子数据
- b) factorselect 文件夹中包含筛选完成的 16 项因子数据
- c) returnseries 文件夹中包含 3/12/24/36 个月滑动窗口下各算法构建投资组合月度收益序列
- d) ff3/ff5 分别为 Fama-French3 因子数据，RF 为月度无风险利率数据，final\_return 为股票月度收益数据
- e) ff30 为去掉市值最小的 30%股票后分别根据市值 (size)和收益价格比 (EP) 分组后构建 MKT/SMB/VMG 3 因子数据
- f) factorEW/factorVW 分别为单因子检验 10-1 多空组合（等权重/市值加权）月度收益序列

### 2) 代码说明：

#### a) MainFile.py

主程序，运行该函数即可得到各主要结果

#### b) DataTransfrom.py

导入基础数据并进行预处理，最后将原始数据转换成每个截面一个 stocknumfactornum 的 Dataframe,列名为[各因子名称+' stock'+ 'ret'], 'stock'为股票代码，'ret' 为对应截面股票月度收益

#### c) NWttest.py

定义对某一序列是否异于 0 进行 Newey and West (1987) t 检验函数

#### d) ReturnSeriesTest.py

在获取各个算法构建多空组合月度收益序列后，对各个收益序列与 OLS 回归 (benchmark) 和 DFN（表现最好的深度算法）序列是否存在显著差异进行 NW-T 检验

#### e) StrategyConstruct.py

- i. 投资组合构建通用函数 (output),
- ii. FC 和 ensemble 无内置算法包, 故单独构建 FC 和 ensemblenn

**f) FactorTest.py**

因子检验补充结果, 内容包含:

- i. 在去掉市值最小的 30%股票后分别根据市值 (size)和收益价格比 (EP) 分组后构建 MKT/SMB/VMG 3 因子
- ii. 单因子 10 分组 10-1/1-10 多空组合因子调整收益
- iii. 单因子 10 分组检验各组因子调整收益
- iv. 各因子与 size 因子独立双变量分组检验结果
- v. 各因子与 BM 因子独立双变量分组检验结果
- vi. 6.96 项因子 fama macbeth 回归检验结果

**g) DFN.py**

深度前馈网络核心库文件, 包含深度前馈网络核心函数 DFN(), 需在 GPU 环境下运行

**h) RNNMODEL.py**

循环神经网络库文件, 包含如下内容:

- i. 基础循环神经网络单元: BaseRnn()
- ii. 可用于训练的循环神经网络整体架构: lstmmodule(), 该架构基于 BaseRnn()构建循环神经网络。

该文件需在 GPU 环境下运行

**i) Ensembleall.py**

集成深度学习模型库文件, 包含集成模型: Ensemblelr()。

因其中集成深度学习模型, 需在 GPU 环境下使用, 如不集成深度学习模型, 则无 GPU 限制。

**j) transecfee.py**

计算不同交易成本下的投资组合绩效变化, 包含核心函数 transecfee() 以及 showtransecfee()

直接调用 showtransecfee()即可

**k) selectFactor.py**

用于筛选重要因子，包含：

- i. 非循环神经网络模型所用的 dropimportant()函数
- ii. 循环神经网络模型所用的 dropimportant2()函数
- iii. FC 方法的筛选函数 FCselect()函数

**注：**获取结果需要运行的 python 文件仅为：**MainFile.py** 和 **FactorTest.py**

## 2. 代码整理

### a) MainFile.py

```
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3  """
4  @description:
5      构建机器学习驱动多因子投资策略主函数
6      1. 首先输入各算法参数（参数根据第一个滑动窗口网格调参确定，此处直接输入）
7      2. 全样本 3/12/24/36 个月滑动窗口函数运行，最终直接输出多空组合月度收益，FF3/5-alpha, sharpe
8      ratio，并将收益序列保存在' output'文件夹内
9      3. 全样本 3/12/24/36 个月滑动窗口各个算法多空组合月度收益序列是否存在显著差异 NW-T 检验
10     4. 不同交易费率下的多空组合绩效结果
11     5. 全样本 12 个月华东窗口下各个算法的特征筛选
12     6. 特征筛选后 16 项因子 12 个月滑动窗口函数运行，最终输出多空组合月度收益，FF3/5-
13     alpha, sharpe ratio，并将收益序列保存在' output'文件夹内
14     注：深度学习算法要在使用 GPU 的环境下进行训练
15     """
16  from StrategyConstruct import FC, output, output2, comboutput, ensemblenn
17  from selectFactor import dropimportant, dropimportant2, FCselect
18  from DataTransfrom import datatransfrom, datatransfrom2
19  from xgboost.sklearn import XGBRegressor
20  from sklearn.ensemble import GradientBoostingRegressor
21  from sklearn.linear_model import LinearRegression, Lasso, ElasticNet, Ridge
22  from sklearn.cross_decomposition import PLSRegression
23  from sklearn.neural_network import MLPRegressor
24  from sklearn.svm import SVR
25  import DFN
26  import RNNMODEL as rm
27  import Ensembleall as ea
28  import warnings
29  from mxnet import gpu
30  import os
31  from transecfee import showtransecfee
32  from ReturnSeriesTest import returnseriestest
33  warnings.filterwarnings('ignore')
34
35
36  #各个算法参数（根据第一个窗口网格调参确定）
37  window=[3, 12, 24, 36]
38  PLS_params=[2, 2, 1, 1]
```

```

39 lasso_params=[1e-3, 5e-4, 0.01, 0.01]
40 ridge_params=[0.1, 0.005, 0.01, 0.005]
41 elasticnet_params={'alpha':[0.01, 1e-3, 0.01, 0.1], 'l1_ratio':[0.3, 0.3, 0.7, 0.3]}
42 SVR_params={'kernel':['linear', 'linear', 'rbf', 'rbf'], 'gamma':[1e-3, 1e-3, 1e-3, 1e-
43 4], 'C':[0.01, 0.001, 0.01, 1e-4]}
44 GBDT_params={'learning_rate':[0.1, 0.1, 0.1, 0.1], 'maxdepth':[2, 3, 2, 2], 'n_estimators':[100, 100,
45 100, 100]} #XGBOOST 与 GBDT 相同 此处共用
46 ENANN_params = {'max_iter':[100, 100, 200, 300], 'p':[0.3, 0.5, 0.7, 0.5]}
47 DFN_params = {'learning_rate':[0.1, 0.1, 0.1, 0.001], 'batch':[300, 400, 300, 400]}
48 LSTM_params = {'learning_rate':[1e-4, 1e-5, 1e-4, 1e-6], 'depth':[2, 2, 1, 2],
49 'hidden_number':[256]*4}
50 RNN_params = {'learning_rate':[0.1, 0.1, 0.1, 0.001], 'depth':[1, 1, 2, 1],
51 'hidden_number':[256]*4}
52
53
54 *****2. 全样本 3/12/24/36 个月滑动窗口函数运行
55 *****#
56 path = r'..\DataBase\factor' #96 项因子所在路径
57 factorname = [x[1:-4] for x in os.listdir(path)]
58 riskfree, timeseries, factor, timeseries2, index = datatransfrom(path)[0],
59 datatransfrom(path)[1], datatransfrom(path)[2], datatransfrom2(path)[0],
60 datatransfrom2(path)[1]
61 for i in range(4):
62     i= 0
63     output(window[i], LinearRegression(), 'OLS'+str(window[i]), riskfree[i], timeseries)
64     FC(window[i], riskfree[i], timeseries, 96, 'FC')
65     output(window[i], PLSRegression(PLS_params[i]), 'PLS' + str(window[i]), riskfree[i],
66 timeseries)
67     output(window[i], Lasso(alpha=lasso_params[i]), 'Lasso' + str(window[i]), riskfree[i],
68 timeseries)
69     output(window[i], Ridge(alpha=ridge_params[i]), 'Ridge'+str(window[i]), riskfree[i],
70 timeseries)
71     output(window[i], ElasticNet(alpha= elasticnet_params['alpha'] [i], l1_ratio=
72 elasticnet_params['l1_ratio'] [i]), 'ElasticNet'+str(window[i]), riskfree[i], timeseries)
73     output(window[i], SVR(kernel=SVR_params['kernel'] [i], gamma= SVR_params ['gamma'] [i], C=
74 SVR_params ['C'] [i] ), 'SVR'+str(window[i]), riskfree[i], timeseries)
75     output(window[i],
76 GradientBoostingRegressor(n_estimators=GBDT_params['n_estimators'] [i], max_depth=GBDT_params[
77 'maxdepth'] [i], learning_rate=GBDT_params['learning_rate'] [i]), 'GBDT' +
78 str(window[i]), riskfree[i], timeseries)
79     output(window[i],
80 XGBRegressor(n_estimators=GBDT_params['n_estimators'] [i], max_depth=GBDT_params['maxdepth'] [i]

```

```

81 ], learning_rate=GBDT_params['learning_rate'][i]), 'XGBOOST' + str(window[i]), riskfree[i],
82 timeseries)
83     output(window[i], ensemblenn(5,modeluse = MLPRegressor(solver = 'lbfgs',
84 max_iter=ENANN_params['max_iter'][i]), pickpercent=ENANN_params['p'][i]), 'ENANN' +
85 str(window[i]), riskfree[i], timeseries)
86     output(window[i], DFN.DFN(outputdim=1, neuralset=[96, 50, 25, 10, 5, 2], ctx=gpu(0),
87 epoch=10, batch_size=DFN_params['batch'][i], lr=DFN_params['learning_rate'][i]), 'DFN' +
88 str(window[i]), riskfree[i], timeseries)
89     output2(window[i], rm.lstmmodule(96, LSTM_params['hidden_number'][i],
90 LSTM_params['depth'][i], 100, 3571, lr=LSTM_params['learning_rate'][i]), 'LSTM' +
91 str(window[i]), riskfree[i], timeseries2)
92     output2(window[i], rm.lstmmodule(96, RNN_params['hidden_number'][i],
93 RNN_params['depth'][i], 100, 3571, lr=RNN_params['learning_rate'][i], ntype='RNN'), 'RNN' +
94 str(window[i]), riskfree[i], timeseries2)
95     modellist = [DFN.DFN(outputdim=1, neuralset=[96, 50, 25, 10, 5, 2], ctx=gpu(0),
96 epoch=10, batch_size=DFN_params['batch'][i], lr=DFN_params['learning_rate'][i]),
97     ensemblenn(5,modeluse = MLPRegressor(solver = 'lbfgs',
98 max_iter=ENANN_params['max_iter'][i]), pickpercent=ENANN_params['p'][i]),
99
100 XGBRegressor(n_estimators=GBDT_params['n_estimators'][i],max_depth=GBDT_params['maxdepth'][i],
101 learning_rate=GBDT_params['learning_rate'][i]),
102
103 GradientBoostingRegressor(n_estimators=GBDT_params['n_estimators'][i],max_depth=GBDT_params[
104 'maxdepth'][i],learning_rate=GBDT_params['learning_rate'][i]),
105     PLSRegression(PLS_params[i]),
106     Ridge(alpha=ridge_params[i]),
107     SVR(kernel=SVR_params['kernel'][i],gamma= SVR_params ['gamma'][i],C=
108 SVR_params ['C'][i]))# PLS 一定要放在倒数第三个 (PLS 输出形式为 list, 故进行了进一步处理)
109     nmolist = [rm.lstmmodule(96, LSTM_params['hidden_number'][i], LSTM_params['depth'][i],
110 100, 3571, lr=LSTM_params['learning_rate'][i]),
111     rm.lstmmodule(96, RNN_params['hidden_number'][i], RNN_params['depth'][i],
112 100, 3571, lr=RNN_params['learning_rate'][i], ntype='RNN'))# 循环神经网络模型
113     modelname = ['DFN', 'En-ann', 'xgboost', 'GBDT', 'lasso', 'Elasticnet', 'pls', 'Ridge',
114 'svm', 'LSTM', 'RNN']
115     ensemblemodel = ea.Ensemblelr(modellist, nmolist, modelname)
116     comboutput(window[i],ensemblemodel, 'Ensemble'+str(window[i]),riskfree[i], timeseries2,
117 index)
118     #####3.. 各算法收益序列差异 NW-t 检验
119     #####
120
121 for i in window:
122     returnseriestest(i)

```

```

123
124 #####4. 不同交易费率情形
125 #####
126
127 showtrasecfee(0.005)
128 showtrasecfee(0.0075)
129 showtrasecfee(0.01)
130
131 #####5. 全样本 12 个月特征筛选过程
132 #####
133
134 i = 1#选取 12 个月滑动窗口筛选因子
135 dropimportant(window[i], LinearRegression(), 'OLS'+str(window[i]), factorname,
136 timeseries, 0.0201)
137 FCselect(factorname, timeseries)
138 dropimportant(window[i], PLSRegression(PLS_params[i]), 'PLS', factorname, timeseries,
139 0.0230)
140 dropimportant(window[i], Lasso(alpha=lasso_params[i]), 'Lasso', factorname, timeseries,
141 0.0208)
142 dropimportant(window[i], Ridge(alpha=ridge_params[i]), 'Ridge', factorname, timeseries,
143 0.0208)
144 dropimportant(window[i], ElasticNet(alpha= elasticnet_params['alpha'] [i], l1_ratio=
145 elasticnet_params['l1_ratio'] [i]), 'ElasticNet', factorname, timeseries, 0.0212)
146 dropimportant(window[i], SVR(kernel=SVR_params['kernel'] [i], gamma= SVR_params
147 ['gamma'] [i], C= SVR_params ['C'] [i] ), 'SVR', factorname, timeseries, 0.0225)
148 dropimportant(window[i],
149 GradientBoostingRegressor(n_estimators=GBDT_params['n_estimators'] [i], max_depth=GBDT_params[
150 'maxdepth'] [i], learning_rate=GBDT_params['learning_rate'] [i]), 'GBDT', factorname,
151 timeseries, 0.0268)
152 dropimportant(window[i], XGBRegressor(n_estimators=GBDT_params['n_estimators'] [i], max_depth=G
153 BDT_params['maxdepth'] [i], learning_rate=GBDT_params['learning_rate'] [i]), 'XGBOOST',
154 factorname, timeseries, 0.0273)
155 dropimportant(window[i], ensablenn(5, modeluse = MLPRegressor(solver = 'lbfgs',
156 max_iter=ENANN_params['max_iter'] [i]), pickpercent=ENANN_params['p'] [i]), 'ENANN',
157 factorname, timeseries, 0.0234)
158 dropimportant(window[i], DFN.DFN(outputdim=1, neuralset=[96, 50, 25, 10, 5, 2], ctx=gpu(0),
159 epoch=10, batch_size=DFN_params['batch'] [i], lr=DFN_params['learning_rate'] [i]), 'DFN',
160 factorname, timeseries, 0.0278)
161 dropimportant2(window[i], rm.lstmmodule(95, LSTM_params['hidden_number'] [i],
162 LSTM_params['depth'] [i], 100, 3571, lr=LSTM_params['learning_rate'] [i]), 'LSTM', factorname,
163 timeseries2, 0.0257)
164 dropimportant2(window[i], rm.lstmmodule(95, RNN_params['hidden_number'] [i],

```



```

165 RNN_params['depth'][i], 100, 3571, lr=RNN_params['learning_rate'][i], ntype='RNN'), 'RNN',
166 factormname, timeseries2, 0.0210)
167
168
169 #####6. 特征筛选后 16 项因子 12 个月滑动窗口函数运行
170 #####
171 path = r'..\DataBase\factorselect' #经过筛选后因子集合所在路径
172 riskfree, timeseries, factor, timeseries2=datatransfrom(path)[0], datatransfrom(path)[1], datatransfrom(path)[2], datatransfrom2(path, after=True)[0]
173
174 i=1 #选取 12 个月滑动窗口测试筛选后因子集合绩效表现
175 output(window[i], LinearRegression(), 'OLS'+str(window[i]), riskfree[i], timeseries)
176 FC(window[i], riskfree[i], timeseries, 11, 'FC')
177 output(window[i], PLSRegression(PLS_params[i]), 'PLS' + str(window[i]), riskfree[i],
178 timeseries)
179 output(window[i], Lasso(alpha=lasso_params[i]), 'Lasso' + str(window[i]), riskfree[i],
180 timeseries)
181 output(window[i], Ridge(alpha=ridge_params[i]), 'Ridge'+str(window[i]), riskfree[i],
182 timeseries)
183 output(window[i], ElasticNet(alpha= elasticnet_params['alpha'] [i], l1_ratio=
184 elasticnet_params['l1_ratio'] [i]), 'ElasticNet'+str(window[i]), riskfree[i], timeseries)
185 output(window[i], SVR(kernel=SVR_params['kernel'] [i], gamma= SVR_params ['gamma'] [i], C=
186 SVR_params ['C'] [i] ), 'SVR'+str(window[i]), riskfree[i], timeseries)
187 output(window[i],
188 GradientBoostingRegressor(n_estimators=GBDT_params['n_estimators'] [i], max_depth=GBDT_params[
189 'maxdepth'] [i], learning_rate=GBDT_params['learning_rate'] [i]), 'GBDT' +
190 str(window[i]), riskfree[i], timeseries)
191 output(window[i],
192 XGBRegressor(n_estimators=GBDT_params['n_estimators'] [i], max_depth=GBDT_params['maxdepth'] [i]
193 ], learning_rate=GBDT_params['learning_rate'] [i]), 'XGBOOST' + str(window[i]), riskfree[i],
194 timeseries)
195 output(window[i], ensablenn(5, modeluse = MLPRegressor(solver = 'lbfgs',
196 max_iter=ENANN_params['max_iter'] [i]), pickpercent=ENANN_params['p'] [i]), 'ENANN' +
197 str(window[i]), riskfree[i], timeseries)
198 output(window[i], DFN.DFN(outputdim=1, neuralset=[16, 50, 25, 10, 5, 2], ctx=gpu(0),
199 epoch=10, batch_size=DFN_params['batch'] [i], lr=DFN_params['learning_rate'] [i]), 'DFN' +
200 str(window[i]), riskfree[i], timeseries)
201 output2(window[i], rm.lstmodule(11, LSTM_params['hidden_number'] [i],
202 LSTM_params['depth'] [i], 100, 3571, lr=LSTM_params['learning_rate'] [i]), 'LSTM' +
203 str(window[i]), riskfree[i], timeseries2)
204 output2(window[i], rm.lstmodule(11, RNN_params['hidden_number'] [i],
205 RNN_params['depth'] [i], 100, 3571, lr=RNN_params['learning_rate'] [i], ntype='RNN'), 'RNN' +
206 str(window[i]), riskfree[i], timeseries2)

```

## b) DataTransfrom.py

```
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3  """
4  @description:
5      导入基础数据并进行一些预处理, 最后变成每个截面一个 Dataframe, 列名为[各因子名称+'
6  stock'+ 'ret'], index 代表单只股票
7  """
8  import glob, os
9  import pandas as pd
10 import warnings
11
12 #####1. 导入因子数据, 无风险利率, 股票月度收益数据
13 #####
14 warnings.filterwarnings('ignore')
15 def datatransfrom(datapath):
16     path=datapath
17     file = glob.glob(os.path.join(path, "*.csv"))
18     k=[]
19     for i in range(len(file)):
20         k.append(pd.read_csv(file[i]))
21         #股票月度收益
22         ret=pd.read_csv('..\DataBase\\final_return.csv')
23         #无风险利率
24         rf=pd.read_csv('..\DataBase\\RF.csv')
25         rf3=rf.iloc[3:-1,:]
26         rf12=rf.iloc[12:-1,:]
27         rf24=rf.iloc[24:-1,:]
28         rf36=rf.iloc[36:-1,:]
29         riskfree = [rf3, rf12, rf24, rf36]
30         #因子名称
31         factor=[]
32         for i in range(len(file)):
33             factor.append(file[i][20:-4])
34             factor.append('stock')
35
36
37     #####对原始数据进行预处理, 每个截面一个 Dataframe, 列名为{96 因子名称+' stock'+ 'ret'},
38     index 代表单只股票#####
39     timeseries=[]
40     for i in range(len(ret.columns)-1):
```

```

41     kl=pd.concat([k[j].iloc[:,i+1] for j in range(len(file))],axis=1)
42     kl['stock'] = ret.iloc[:,0]
43     kl.columns = factor
44     kl=kl.iloc[:-2,: ]
45     timeseries.append(kl)
46     #删除月度收益不存在的数据条
47     for i in range(len(timeseries)):
48         timeseries[i]['ret']=ret.iloc[:,i+1]
49         timeseries[i]['ret']=timeseries[i]['ret'].fillna('null')
50         timeseries[i]=timeseries[i][~timeseries[i]['ret'].isin(['null'])]
51     return riskfree, timeseries, factor
52
53 ## 为 LSTM\RNN 设计的数据读取函数
54 def datatransfrom2(datapath, after=False):
55     path=datapath
56     file = glob.glob(os.path.join(path, "*.csv"))
57     k=[]
58     for i in range(len(file)):
59         k.append(pd.read_csv(file[i]))
60     #股票月度收益
61     ret=pd.read_csv('..\DataBase\\final_return.csv')
62     #因子名称
63     factor=[]
64     for i in range(len(file)):
65         factor.append(file[i][20:-4])
66     factor.append('stock')
67
68     *****对原始数据进行预处理, 每个截面一个 Dataframe, 列名为{96 因子名称+' stock'+'ret'},
69 index 代表单只股票*****#
70     timeseries2=[]
71     index = []
72     for i in range(len(ret.columns)-1):
73         kl=pd.concat([k[j].iloc[:,i+1] for j in range(len(file))],axis=1)
74         kl['stock'] = ret.iloc[:,0]
75         kl.columns = factor
76         if after:# 保证筛选后因子个数为 3571 个
77             kl = kl.iloc[:, :]
78         else:
79             kl = kl.iloc[:-2,: ]
80         timeseries2.append(kl)
81     # 加入月度收益, 令月度收益不存在为 null, 方便下一步函数处理
82     for i in range(len(timeseries2)):

```

```
83     timeseries2[i]['ret'] = ret.iloc[:, i + 1]
84     timeseries2[i]['ret'] = timeseries2[i]['ret'].fillna('null')
85     index.append(timeseries2[i]['ret'].isin(['null']))
86     return timeseries2, index
```

### c) NWttest.py

```
1  # -*- coding:utf-8 -*-
2  '''
3  @description:
4  NW-t 检验所用包
5  '''
6
7  import numpy as np
8  from collections import namedtuple
9  from scipy.stats import distributions
10
11  def _ttest_finish(df, t):
12      '''
13      :param df: 自由度
14      :param t: t 值
15      :return: 输出 t 和对应 p 值
16      '''
17      prob = distributions.t.sf(np.abs(t), df) * 2 # use np.abs to get upper tail
18      if t.ndim == 0:
19          t = t[()]
20      return t, prob
21
22  NWt_1sampleResult = namedtuple('NWT_1sampResult', ('statistic', 'pvalue'))
23  def nwttest_1samp(a, popmean, axis=0, L=1):
24      '''
25      主函数
26      :param a: 数据列表
27      :param popmean: 原假设值 u0
28      :param axis: 行还是列, 默认行
29      :param L: lag, 滞后多少, 默认 1
30      :return: 输出 nw-t 和对应 p 值
31      '''
32      a = np.array(a)
33      N = len(a)
34      df = N-1
35      e = a - np.mean(a)
36      residuals = np.sum(e**2)
37      Q = 0
38      for i in range(L):
39          w_l = 1 - (i+1)/(1+L)
40          for j in range(1, N):
41              Q += w_l*e[j]*e[j-(i+1)]
42      S = residuals + 2*Q
```

```
43     nw_var = S/N
44     d = np.mean(a,axis) - popmean
45     nw_sd = np.sqrt(nw_var / float(df))
46     with np.errstate(divide='ignore', invalid='ignore'):
47         t = np.divide(d, nw_sd)
48     t,prob = _ttest_finish(df,t)
49
50     return NWt_1sampleResult(t,prob)
```

#### d) ReturnSeriesTest.py

```
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3  """
4  @description:
5      在获取各个算法构建多空组合月度收益序列后, 对各个收益序列与 OLS 回归 (benchmark)
6      和 DFN (表现最好的深度算法) 序列是否存在显著差异进行 NW-T 检验
7  """
8
9  import glob, os
10 import pandas as pd
11 from NWttest import nwttest_1samp
12 import warnings
13 warnings.filterwarnings('ignore')
14
15 #
16 def returnseriestest(length):
17     path = r'..\DataBase\returnseries' + '\\' + str(length)
18     file = glob.glob(os.path.join(path, "*.csv"))
19     ols = pd.read_csv(path+'\\OLS'+str(length)+'.csv')
20     dfn=pd.read_csv(path+'\\DFN'+str(length)+'.csv')
21     k = [] # 每个算法一个 df
22     for i in range(len(file)):
23         k.append(pd.read_csv(file[i]))
24         #OLS 与其他算法区别
25         for i in range(len(k)):
26             t = []
27             t1 = nwttest_1samp(k[i].iloc[:, 1] - ols['long-short'], 0)
28             t.append(t1.statistic)
29             t2 = nwttest_1samp(k[i].iloc[:, 2] - ols['long'], 0)
30             t.append(t2.statistic)
31             t3 = nwttest_1samp(k[i].iloc[:, 3] - ols['short'], 0)
32             t.append(t3.statistic)
33             print('ols-' + file[i][27:-4], t)
34         #DFN 与其他算法区别
35         for i in range(len(k)):
36             t = []
37             t1 = nwttest_1samp(-k[i].iloc[:, 1] + dfn['long-short'], 0)
38             t.append(t1.statistic)
39             t2 = nwttest_1samp(-k[i].iloc[:, 2] + dfn['long'], 0)
40             t.append(t2.statistic)
41             t3 = nwttest_1samp(-k[i].iloc[:, 3] + dfn['short'], 0)
42             t.append(t3.statistic)
```

```
43     print('dfn-' + file[i][27:-4], t)
44     return
```



## e) StrategyConstruct.py

```
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3  """
4  @description:
5      1. 投资组合构建通用函数 (output), 函数最终输出多空组合月度收益, FF3/5-alpha, sharpe ratio
6      2. FC 和 ensemble 无内置算法包, 此处单独构建 FC 和 ensemblenn
7
8  """
9  import glob, os
10 import pandas as pd
11 import numpy as np
12 from scipy import stats
13 import statsmodels.api as sm
14 from xgboost.sklearn import XGBRegressor
15 from sklearn.ensemble import GradientBoostingRegressor
16 from sklearn.preprocessing import scale
17 from sklearn.linear_model import LinearRegression
18 from sklearn.cross_decomposition import PLSRegression
19 from sklearn.linear_model import Lasso, ElasticNet, Ridge
20 from NWttest import nwttest_lsamp
21 from sklearn.svm import SVR
22 import warnings
23 from sklearn.neural_network import MLPRegressor
24 import gc
25
26
27 """*****3. EN-ANN、FC 算法函数*****"""
28 #python 里没有 EN-ANN 和 FC 的对应算法包 此处先定义算法计算方式
29 #EN-ANN
30 class ensemblenn(object):
31     def __init__(self, ensemblennumbers, modeluse = MLPRegressor(solver = 'lbfgs'), pickpercent
32     = 0.5):
33         self.ensemblennumbers = ensemblennumbers
34         self.modellist = []
35         self.score = [0]*ensemblennumbers
36         for i in range(self.ensemblennumbers):
37             self.modellist.append(modeluse)
38             self.pickpercent = pickpercent
39     def fit(self, X, Y):
40         for i in range(self.ensemblennumbers):
41             self.modellist[i].fit(X, Y)
42             self.score[i] = self.modellist[i].loss_
```

```

43     def predict(self, xtest):
44         usemodel = np.array(self.modellist)[np.argsort(np.array(self.score))]
45         usemodel = usemodel[0:self.ensemblenumber//2]
46         predict = []
47         for i in range(self.ensemblenumber//2):
48             predict.append(usemodel[i].predict(xtest))
49         return list(np.mean(predict, axis=0))
50 #FC
51 def FC(length, rf, timeseries, lenn=96, na='FC'):
52     #length 为滑动窗口长度：取值{3, 12, 24, 36}
53     #na 为输出文件名称
54     #rf 为无风险利率，取值与 length 对应{rf3, rf12, rf24, rf36}
55     Long_Short = []
56     Long = []
57     Short = []
58     for i in range(len(timeseries) - length):
59         print(i)
60         FINALm = pd.concat(timeseries[i:i + (length + 1)], axis=0)
61         FINALm = FINALm.fillna(0)
62         FINAL_X = FINALm.iloc[:, :-2]
63         FINAL_x = scale(FINAL_X)
64         final = pd.concat(timeseries[i:i + length], axis=0)
65         x_train = FINAL_x[:len(final)]
66         x_test = FINAL_x[len(final):]
67         y_train = final.iloc[:, -1]
68         test = timeseries[i + length]
69         clf = LinearRegression()
70         k = []
71         for i in range(lenn):
72             x = x_train[:, i].reshape(-1, 1)
73             clf.fit(x, y_train)
74             k.append(clf.coef_[0])
75         PREDICTION = []
76         for i in range(len(x_test)):
77             test0 = np.array(x_test[i])
78             y = 0
79             for j in range(lenn):
80                 y = y + test0[j] * k[j]
81             PREDICTION.append(y)
82         y_test = test.iloc[:, -1]
83         # 构建投资组合
84         r_predict = pd.DataFrame(PREDICTION, columns=['predict'])
85         r_ture = pd.DataFrame(y_test)
86         r_ture.columns = ['ture']

```

```

87     r_ture.index = r_predict.index
88     FINAL = pd.concat([r_predict, r_ture], axis=1)
89     FINAL_sort = FINAL.sort_values(by='predict', axis=0)
90     r_final = np.array(FINAL_sort['ture'])
91     m = int(len(r_final) * 0.1) + 1
92     r_final = r_final.tolist()
93     long = r_final[-m:]
94     short = r_final[:m]
95     r_end = (np.sum(long) - np.sum(short)) / m
96     Long_Short.append(r_end)
97     Long.append(np.average(long))
98     Short.append(np.average(short))
99     T_value = []
100    Mean = []
101    p_value = []
102    sharpratio = []
103    Std = []
104    T0 = [Long_Short, Long, Short]
105    for l in T0:
106        t_test = nwttest_lsamp(l, 0)
107        mean = np.average(l) * 12
108        STD = np.std(l) * np.sqrt(12)
109        sharp = (mean - rf.mean().tolist()[0] * 12 / 100) / STD
110        T_value.append(t_test.statistic)
111        p_value.append(t_test.pvalue)
112        Mean.append(mean)
113        Std.append(STD)
114        sharpratio.append(sharp)
115    name = na
116    length = length
117    print(name, 'long-short', 'long', 'short')
118    print('mean', Mean[0] / 12, Mean[1] / 12, Mean[2] / 12)
119    print('t-statistic', '(' + str(round(T_value[0], 4)) + ')', '(' + str(round(T_value[1],
120    4)) + ')',
121          '(' + str(round(T_value[2], 4)) + ')')
122    A = pd.DataFrame(Long_Short, columns=['long-short'])
123    B = pd.DataFrame(Long, columns=['long'])
124    C = pd.DataFrame(Short, columns=['short'])
125    M = pd.concat([A, B, C], axis=1)
126    M.to_csv('..\output\\'+name + '.csv')
127    ff3 = pd.read_csv('..\DataBase\\ff3.csv')
128    ff5 = pd.read_csv('..\DataBase\\ff5.csv')
129    alpha3 = []
130    t3 = []

```

```

131     t5 = []
132     alpha5 = []
133     for i in range(3):
134         X1 = ff3.iloc[length:, 1:]
135         X2 = ff5.iloc[length:, 1:]
136         Y = M.iloc[: -2, i]
137         Y.index = X1.index
138         Y = Y - rf.RF[: -1] / 100
139         x1 = sm.add_constant(X1)
140         reg = sm.OLS(Y, x1).fit()
141         t3.append(reg.tvalues[0])
142         alpha3.append(reg.params[0] * 12)
143         x2 = sm.add_constant(X2)
144         reg = sm.OLS(Y, x2).fit()
145         t5.append(reg.tvalues[0])
146         alpha5.append(reg.params[0] * 12)
147         print('alpha-FF3', alpha3[0] / 12, alpha3[1] / 12, alpha3[2] / 12)
148         print('t-statistic', '(' + str(round(t3[0], 4)) + ')', '(' + str(round(t3[1], 4)) + ')',
149               '(' + str(round(t3[2], 4)) + ')')
150         print('alpha-FF5', alpha5[0] / 12, alpha5[1] / 12, alpha5[2] / 12)
151         print('t-statistic', '(' + str(round(t5[0], 4)) + ')', '(' + str(round(t5[1], 4)) + ')',
152               '(' + str(round(t5[2], 4)) + ')')
153         print('sharpe', sharpratio[0], sharpratio[1], sharpratio[2])
154
155
156     #####4. 投资组合构建主函数#####
157     def output(length, CLF, name, rf, timeseries):
158         #length 为滑动窗口长度: 取值 {3, 12, 24, 36}
159         #CLF 为预测选取的机器学习模型
160         #name 为输出文件名称 (type:string)
161         #rf 为无风险利率, 取值与 length 对应 {rf3, rf12, rf24, rf36}
162         Long_Short = []
163         Long = []
164         Short = []
165         for i in range(len(timeseries) - (length)):
166             print(i)
167             FINALm = pd.concat(timeseries[i:i + (length+1)], axis=0)
168             FINALm = FINALm.fillna(0) #因子缺失值以 0 填充
169             FINAL_X = FINALm.iloc[:, :-2]
170             FINAL_x = scale(FINAL_X)
171             final = pd.concat(timeseries[i:i + length], axis=0)
172             x_train = FINAL_x[:len(final)]
173             x_test = FINAL_x[len(final):]
174             y_train = final.iloc[:, -1]

```

```

175     test = timeseries[i + length]
176     y_test = test.iloc[:, -1]
177     # 基准-linear
178     clf = CLF
179     clf.fit(x_train, y_train)
180     PREDICTION = clf.predict(x_test)
181     # 构建投资组合
182     prediction = pd.DataFrame(PREDICTION)
183     r_predict = pd.DataFrame(PREDICTION, columns=['predict'])
184     r_ture = pd.DataFrame(y_test)
185     r_ture.columns = ['ture']
186     r_ture.index = r_predict.index
187     FINAL = pd.concat([r_predict, r_ture], axis=1)
188     FINAL_sort = FINAL.sort_values(by='predict', axis=0)
189     r_final = np.array(FINAL_sort['ture'])
190     m = int(len(r_final) * 0.1) + 1
191     r_final = r_final.tolist()
192     long = r_final[-m:]
193     short = r_final[:m]
194     r_end = (np.sum(long) - np.sum(short)) / m
195     Long_Short.append(r_end)
196     Long.append(np.average(long))
197     Short.append(np.average(short))
198     T_value = []
199     Mean = []
200     p_value = []
201     sharpratio = []
202     Std = []
203     T0 = [Long_Short, Long, Short]
204     for l in T0:
205         t_test = nwttest_1samp(1, 0)
206         mean = np.average(l) * 12
207         STD = np.std(l) * np.sqrt(12)
208         sharp = (mean - rf.mean().tolist()[0]*12/100) / STD
209         T_value.append(t_test.statistic)
210         p_value.append(t_test.pvalue)
211         Mean.append(mean)
212         Std.append(STD)
213         sharpratio.append(sharp)
214     print(name, 'long-short', 'long', 'short')
215     print('mean', Mean[0]/12, Mean[1]/12, Mean[2]/12)
216     print('t-statistic', '('+str(round(T_value[0], 4))+')', '('+str(round(T_value[1], 4))+')',
217           '('+str(round(T_value[2], 4))+')')
218     A = pd.DataFrame(Long_Short, columns=['long-short'])

```

```

219     B = pd.DataFrame(Long, columns=['long'])
220     C = pd.DataFrame(Short, columns=['short'])
221     M = pd.concat([A, B, C], axis=1)
222     M.to_csv('..\output\\'+name+'.csv')
223     ff3 = pd.read_csv('..\DataBase\\ff3.csv')
224     ff5 = pd.read_csv('..\DataBase\\ff5.csv')
225     alpha3 = []
226     t3 = []
227     t5 = []
228     alpha5 = []
229     for i in range(3):
230         X1 = ff3.iloc[length:, 1:]
231         X2 = ff5.iloc[length:, 1:]
232         Y = M.iloc[:length, i]
233         Y.index = X1.index
234         Y = Y - rf.RF[:length] / 100
235         x1 = sm.add_constant(X1)
236         reg = sm.OLS(Y, x1).fit()
237         t3.append(reg.tvalues[0])
238         alpha3.append(reg.params[0] * 12)
239         x2 = sm.add_constant(X2)
240         reg = sm.OLS(Y, x2).fit()
241         t5.append(reg.tvalues[0])
242         alpha5.append(reg.params[0] * 12)
243         print('alpha-FF3', alpha3[0]/12, alpha3[1]/12, alpha3[2]/12)
244         print('t-statistic', '('+str(round(t3[0], 4))+')', '('+str(round(t3[1], 4))+')',
245             '('+str(round(t3[2], 4))+')')
246         print('alpha-FF5', alpha5[0]/12, alpha5[1]/12, alpha5[2]/12)
247         print('t-statistic', '('+str(round(t5[0], 4))+')', '('+str(round(t5[1], 4))+')',
248             '('+str(round(t5[2], 4))+')')
249         print('sharpe', sharpratio[0], sharpratio[1], sharpratio[2])
250
251     # 因为 LSTM 与 RNN 一个步长内所用数据形状必须一致，设置专用的主函数供使用
252     def output2(length, CLF, name, rf, timeseries2):
253         # length 为滑动窗口长度：取值 {3, 12, 24, 36}
254         # CLF 为预测选取的机器学习模型
255         # name 为输出文件名称 (type:string)
256         # rf 为无风险利率，取值与 length 对应 {rf3, rf12, rf24, rf36}
257         Long_Short = []
258         Long = []
259         Short = []
260         for i in range(len(timeseries2) - length):
261             FINALm = pd.concat(timeseries2[i:(i + length + 1)], axis=0)
262             FINALm[~FINALm['ret'].isin(['null'])] =

```

```

263 FINALm[~FINALm['ret'].isin(['null'])].fillna(0)
264 FINAL_X = FINALm.iloc[:, :-2]
265 FINAL_x = FINAL_X
266 FINAL_x[~FINALm['ret'].isin(['null'])] =
267 scale(FINAL_X[~FINALm['ret'].isin(['null'])])
268 FINAL_x[FINALm['ret'].isin(['null'])] = 0
269 FINALm[FINALm['ret'].isin(['null'])] = 0
270 x_train = [FINAL_x.iloc[j * 3571:(j + 1) * 3571, :].values for j in range(length)]
271 y_train = [FINALm.iloc[j * 3571:(j + 1) * 3571, -1].values for j in range(length)]
272 x_test = np.array([FINAL_x.iloc[j * 3571:(j + 1) * 3571, :].values for j in range(1,
273 length + 1)])
274 y_test = list(FINALm.iloc[(length) * 3571:(length + 1) * 3571, -1].values)
275 # 基准-linear
276 clf = CLF
277 clf.fit(x_train, y_train)
278 PREDICTION = clf.predict(x_test)
279 PREDICTION = [PREDICTION[m][0] for m in range(3571 * (length - 1), 3571 * length)]
280 # 构建投资组合
281 r_predict = pd.DataFrame(PREDICTION, columns=['predict'])
282 r_ture = pd.DataFrame(y_test)
283 r_ture.columns = ['ture']
284 r_ture.index = r_predict.index
285 FINAL = pd.concat([r_predict, r_ture], axis=1)
286 FINAL = FINAL[~timeseries2[i + length]['ret'].isin(['null'])]
287 FINAL_sort = FINAL.sort_values(by='predict', axis=0)
288 r_final = np.array(FINAL_sort['ture'])
289 m = int(len(r_final) * 0.1) + 1
290 r_final = r_final.tolist()
291 long = r_final[-m:]
292 short = r_final[:m]
293 r_end = (np.sum(long) - np.sum(short)) / m
294 Long_Short.append(r_end)
295 Long.append(np.average(long))
296 Short.append(np.average(short))
297 gc.collect()
298 T_value = []
299 Mean = []
300 p_value = []
301 sharpratio = []
302 Std = []
303 T0 = [Long_Short, Long, Short]
304 for l in T0:
305     t_test = nwttest_1samp(1, 0, L=1)
306     mean = np.average(1 * 12 - rf.mean().tolist()[0] * 12 / 100

```

```

307     STD = np.std(l) * np.sqrt(12)
308     sharp = (mean ) / STD
309     T_value.append(t_test.statistic)
310     p_value.append(t_test.pvalue)
311     Mean.append(mean)
312     Std.append(STD)
313     sharpratio.append(sharp)
314     print(name, 'long-short', 'long', 'short')
315     print('mean', Mean[0] / 12 + rf.mean().tolist()[0] / 100, Mean[1] / 12 +
316 rf.mean().tolist()[0] / 100, Mean[2] / 12 + rf.mean().tolist()[0] / 100)
317     print('t-statistic', '('+str(round(T_value[0],4))+')', '('+str(round(T_value[1],4))+')',
318 '('+str(round(T_value[2],4))+')')
319     A = pd.DataFrame(Long_Short, columns=['long-short'])
320     B = pd.DataFrame(Long, columns=['long'])
321     C = pd.DataFrame(Short, columns=['short'])
322     M = pd.concat([A, B, C], axis=1)
323     M.to_csv('..\output\\' + name + '.csv')
324     ff3 = pd.read_csv('..\DataBase\\ff3.csv')
325     ff5 = pd.read_csv('..\DataBase\\ff5.csv')
326     alpha3 = []
327     t3 = []
328     t5 = []
329     alpha5 = []
330     for i in range(3):
331         X1 = ff3.iloc[length:, 1:]
332         X2 = ff5.iloc[length:, 1:]
333         Y = M.iloc[:-2, i]
334         Y.index = X1.index
335         Y = Y - rf.RF[:-1] / 100
336         x1 = sm.add_constant(X1)
337         reg = sm.OLS(Y, x1).fit()
338         t3.append(reg.tvalues[0])
339         alpha3.append(reg.params[0] * 12)
340         x2 = sm.add_constant(X2)
341         reg = sm.OLS(Y, x2).fit()
342         t5.append(reg.tvalues[0])
343         alpha5.append(reg.params[0] * 12)
344         print('alpha-FF3', alpha3[0] / 12, alpha3[1] / 12, alpha3[2] / 12)
345         print('t-statistic', '(' + str(round(t3[0], 4)) + ')', '(' + str(round(t3[1], 4)) + ')',
346             '(' + str(round(t3[2], 4)) + ')')
347         print('alpha-FF5', alpha5[0] / 12, alpha5[1] / 12, alpha5[2] / 12)
348         print('t-statistic', '(' + str(round(t5[0], 4)) + ')', '(' + str(round(t5[1], 4)) + ')',
349             '(' + str(round(t5[2], 4)) + ')')
350     print('sharp', sharpratio[0], sharpratio[1], sharpratio[2])

```



```

351
352 # 针对所有机器学习模型集成所设置的主函数
353 def comboutoutput(length, clf, name, rf,timeseries2, index):
354     Long_Short = []
355     Long = []
356     Short = []
357     for i in range(len(timeseries2) - length):
358         print(i)
359         # LSTM 数据
360         FINALm = pd.concat(timeseries2[i:(i + length + 1)], axis=0)
361         FINALm[~FINALm['ret'].isin(['null'])] =
362 FINALm[~FINALm['ret'].isin(['null'])].fillna(0)
363         FINAL_X = FINALm.iloc[:, :-2]
364         FINAL_x = FINAL_X
365         FINAL_x[~FINALm['ret'].isin(['null'])] =
366 scale(FINAL_X[~FINALm['ret'].isin(['null'])])
367         FINAL_x[FINALm['ret'].isin(['null'])] = 0
368         FINALm[FINALm['ret'].isin(['null'])] = 0
369         Nx_train = [FINAL_x.iloc[j * 3571:(j + 1) * 3571, :].values for j in range(length)]
370         Ny_train = [FINALm.iloc[j * 3571:(j + 1) * 3571, -1].values for j in range(length)]
371         Nx_test = [FINAL_x.iloc[j * 3571:(j + 1) * 3571, :].values for j in range(1, length
372 + 1)]
373         ## 传统数据
374         dl = timeseries2[i:i + (length + 1)]
375         p = [dl[j][~index[i+j]] for j in range(len(dl))]
376         TTX = pd.concat(p, axis=0)
377         TTX = TTX.fillna(0)
378         TXX = TTX.iloc[:, :-2]
379         TXx = scale(TXX)
380         final = pd.concat(p[:length], axis=0)
381         Tx_train = TXx[:len(final)]
382         Tx_test = TXx[len(final):]
383         Ty_train = final.iloc[:, -1]
384         test = p[-1]
385         Ty_test = test.iloc[:, -1]
386         # 基准-linear
387         clf = clf
388         clf.fit(Tx_train, Ty_train, Nx_train, Ny_train)
389         PREDICTION = clf.predict(Tx_test, Nx_test, index[i+length], length)
390         r_predict = pd.DataFrame(PREDICTION, columns=['predict'])
391         r_ture = pd.DataFrame(Ty_test)
392         r_ture.columns = ['ture']
393         r_ture.index = r_predict.index
394         FINAL = pd.concat([r_predict, r_ture], axis=1)

```

```

395     FINAL_sort = FINAL.sort_values(by='predict', axis=0)
396     r_final = np.array(FINAL_sort['ture'])
397     m = int(len(r_final) * 0.1) + 1
398     r_final = r_final.tolist()
399     long = r_final[-m:]
400     short = r_final[:m]
401     r_end = (np.sum(long) - np.sum(short)) / m
402     Long_Short.append(r_end)
403     Long.append(np.average(long))
404     Short.append(np.average(short))
405     gc.collect()
406     T_value = []
407     Mean = []
408     p_value = []
409     sharpratio = []
410     Std = []
411     T0 = [Long_Short, Long, Short]
412     for l in T0:
413         t_test = nwttest_lsamp(l, 0, L=1)
414         mean = np.average(l) * 12 - rf.mean().tolist()[0] * 12 / 100
415         STD = np.std(l) * np.sqrt(12)
416         sharp = (mean) / STD
417         T_value.append(t_test.statistic)
418         p_value.append(t_test.pvalue)
419         Mean.append(mean)
420         Std.append(STD)
421         sharpratio.append(sharp)
422     print(name, 'long-short', 'long', 'short')
423     print('mean', Mean[0] / 12, Mean[1] / 12, Mean[2] / 12)
424     print('t-statistic', '(' + str(round(T_value[0], 4)) + ')', '(' + str(round(T_value[1],
425 4)) + ')',
426         '(' + str(round(T_value[2], 4)) + ')')
427     A = pd.DataFrame(Long_Short, columns=['long-short'])
428     B = pd.DataFrame(Long, columns=['long'])
429     C = pd.DataFrame(Short, columns=['short'])
430     M = pd.concat([A, B, C], axis=1)
431     M.to_csv('..\output\\' + name + '.csv')
432     ff3 = pd.read_csv('..\DataBase\\ff3.csv')
433     ff5 = pd.read_csv('..\DataBase\\ff5.csv')
434     alpha3 = []
435     t3 = []
436     t5 = []
437     alpha5 = []
438     for i in range(3):

```

```

439     X1 = ff3.iloc[length:, 1:]
440     X2 = ff5.iloc[length:, 1:]
441     Y = M.iloc[:-2, i]
442     Y.index = X1.index
443     Y = Y - rf.RF[:-1] / 100
444     x1 = sm.add_constant(X1)
445     reg = sm.OLS(Y, x1).fit()
446     t3.append(reg.tvalues[0])
447     alpha3.append(reg.params[0] * 12)
448     x2 = sm.add_constant(X2)
449     reg = sm.OLS(Y, x2).fit()
450     t5.append(reg.tvalues[0])
451     alpha5.append(reg.params[0] * 12)
452     print('alpha-FF3', alpha3[0] / 12, alpha3[1] / 12, alpha3[2] / 12)
453     print('t-statistic', '(' + str(round(t3[0], 4)) + ')', '(' + str(round(t3[1], 4)) + ')',
454           '(' + str(round(t3[2], 4)) + ')')
455     print('alpha-FF5', alpha5[0] / 12, alpha5[1] / 12, alpha5[2] / 12)
456     print('t-statistic', '(' + str(round(t5[0], 4)) + ')', '(' + str(round(t5[1], 4)) + ')',
457           '(' + str(round(t5[2], 4)) + ')')
458     print('sharpe', sharpratio[0], sharpratio[1], sharpratio[2])

```

## f) FactorTest.py

```
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3  """
4  @description:
5  1. 在去掉市值最小的 30%股票后分别根据市值 (size)和收益价格比 (EP) 分组后构建 MKT/SMB/VMG 3 因
6  子
7  2. 单因子 10 分组 10-1/1-10 多空组合因子调整收益
8  3. 单因子 10 分组检验各组因子调整收益
9  4. 各因子与 size 因子独立双变量分组检验结果
10 5. 各因子与 BM 因子独立双变量分组检验结果
11 6. 96 项因子 fama macbeth 回归检验结果
12
13 """
14 *****1. 在去掉市值最小的 30%股票后分别根据市值 (size)和收益价格比 (EP) 分组后构建 MKT/SMB/VMG
15 3 因子*****#
16
17 import pandas as pd
18 import numpy as np
19 ret=pd.read_csv('..\DataBase\\final_return.csv')
20 size=pd.read_csv('..\DataBase\\factor\\01size.csv')
21 EP=pd.read_csv('..\DataBase\\factor\\32EP.csv')
22
23 faceq=pd.DataFrame(columns=['MKT', 'SMB', 'VMG'], index=ret.columns[1:])
24 facvw=pd.DataFrame(columns=['MKT', 'SMB', 'VMG'], index=ret.columns[1:])
25 rf=pd.read_csv('RF.csv')
26 for i in range(len(ret.columns)-1):
27     total=pd.concat([ret.iloc[:, i+1], size.iloc[:, i+1], EP.iloc[:, i+1]], axis=1)
28     total.columns=['ret', 'size', 'EP']
29     total=total.dropna()
30     total = total.sort_values(by='size')
31     final = total.iloc[int(len(total) * 0.3):, :]#去掉 30%小市值
32     #MKT
33     faceq.iloc[i, 0] = final.ret.mean()-rf.RF[i]/100
34     final['VW'] = final.apply(lambda x: x['size'] * x['ret'], axis=1)
35     facvw.iloc[i, 0] = final.VW.sum()/final.iloc[:, 1].sum()-rf.RF[i]/100
36     #SMB/VMG
37     SIZE1=final.iloc[:int(len(final)/2), :].#1/2
38     SIZE1=SIZE1.sort_values(by='EP')
39     SV=SIZE1.iloc[int(2*len(SIZE1)/3):, :]
40     sveq=SV['ret'].mean()
41     svvw=SV.VW.sum()/SV.iloc[:, 1].sum()
42     SM=SIZE1.iloc[int(1*len(SIZE1)/3):int(2*len(SIZE1)/3), :]
```

```

43     smeq=SM['ret'].mean()
44     smvw=SM.VW.sum()/SM.iloc[:,1].sum()
45     SG=SIZE1.iloc[:int(1*len(SIZE1)/3),:]
46     sgeq=SG['ret'].mean()
47     sgvw=SG.VW.sum()/SG.iloc[:,1].sum()
48
49     SIZE2 = final.iloc[int(len(final) / 2):, : ] #大
50     SIZE2 = SIZE2.sort_values(by='EP')
51     BV=SIZE2.iloc[int(2*len(SIZE2)/3):, :]
52     bveq=BV['ret'].mean()
53     bvvw=BV.VW.sum()/BV.iloc[:,1].sum()
54     BM=SIZE2.iloc[int(1*len(SIZE2)/3):int(2*len(SIZE2)/3),:]
55     bmeq=BM['ret'].mean()
56     bmvw=BM.VW.sum()/BM.iloc[:,1].sum()
57     BG=SIZE2.iloc[:int(1*len(SIZE2)/3),:]
58     bgeq=BG['ret'].mean()
59     bgvw=BG.VW.sum()/BG.iloc[:,1].sum()
60     faceq.iloc[i, 1] = (sveq +smeq +sgeq )/3-(bveq +bmeq +bgeq )/3
61     facvw.iloc[i, 1] = (svvw +smvw +sgvw )/3-(bvvw +bmvw +bgvw )/3
62     faceq.iloc[i, 2] = (sveq +bveq)/2-(sgeq+bgeq )/2
63     facvw.iloc[i, 2] = (svvw +bvvw)/2-(sgvw+bgvw )/2
64     print(i)
65
66     #####2. 单因子 10 分组 10-1 多空组合因子调整收益#####8#
67     import pandas as pd
68     import numpy as np
69     import statsmodels.api as sm
70     from scipy import stats
71     from NWttest import nwttest_1samp
72     factoreq=pd.read_csv('..\DataBase\\factorEW.csv')
73     factorvw=pd.read_csv('..\DataBase\\factorVW.csv')
74     resultew=pd.DataFrame(columns=factoreq.columns[1:], index=['ret', 't', 'exret', 't', 'capmret', 't',
75 'ff3ret', 't', 'ff30ret', 't'] )
76     resultvw=pd.DataFrame(columns=factoreq.columns[1:], index=['ret', 't', 'exret', 't', 'capmret', 't',
77 'ff3ret', 't', 'ff30ret', 't'] )
78     rf=pd.read_csv('RF.csv')
79     for i in range(len(factoreq.columns)-1):
80         if factoreq.iloc[:,i+1].mean()>0:
81             faceq=factoreq.iloc[:,i+1]
82         else:
83             faceq = -factoreq.iloc[:, i + 1]
84         if factorvw.iloc[:,i+1].mean()>0:
85             facvw=factorvw.iloc[:,i+1]
86         else:

```

```

87     facvw = -factorvw.iloc[:, i + 1]
88     ####return
89     reteq0 = faceq.dropna()
90     resultew.iloc[0, i] = reteq0.mean()
91     ttest = nwttest_lsamp(reteq0, 0)
92     resultew.iloc[1, i] = ttest.statistic
93     retvw0 = facvw.dropna()
94     resultvw.iloc[0, i] = retvw0.mean()
95     ttest = nwttest_lsamp(retvw0, 0)
96     resultvw.iloc[1, i] = ttest.statistic
97
98     ##excess return
99     exeq=faceq-rf.RF/100
100    exvw=facvw-rf.RF/100
101    exeq0=exeq.dropna()
102    resultew.iloc[2, i] = exeq0.mean()
103    #ttest=stats.ttest_lsamp(exeq0, 0)
104    ttest=nwttest_lsamp(exeq0, 0)
105    resultew.iloc[3, i]=ttest.statistic
106    exvw0 = exvw.dropna()
107    resultvw.iloc[2, i] = exvw0.mean()
108    # ttest=stats.ttest_lsamp(exvw0, 0)
109    ttest = nwttest_lsamp(exvw0, 0)
110    resultvw.iloc[3, i] = ttest.statistic
111    ##CAPM
112    ff3 = pd.read_csv('ff3.csv')
113    capm=ff3.iloc[:, 1]
114    x1 = sm.add_constant(capm)
115    X1=pd.concat([exeq[:-2], x1], axis=1)
116    X1 = X1.dropna()
117    X2 = pd.concat([exvw[:-2], x1], axis=1)
118    X2 = X2.dropna()
119    regeq = sm.OLS(X1.iloc[:, 0], X1.iloc[:, 1:]).fit()
120    regvw = sm.OLS(X2.iloc[:, 0], X2.iloc[:, 1:]).fit()
121    resultew.iloc[4, i]=regeq.params[0]
122    resultew.iloc[5, i]=regeq.tvalues[0]
123    resultvw.iloc[4, i] = regvw.params[0]
124    resultvw.iloc[5, i] = regvw.tvalues[0]
125
126    x1 = sm.add_constant(ff3.iloc[:, 1:])
127    X1 = pd.concat([exeq[:-2], x1], axis=1)
128    X1 = X1.dropna()
129    X2 = pd.concat([exvw[:-2], x1], axis=1)
130    X2 = X2.dropna()

```

```

131     regeq = sm.OLS(X1.iloc[:, 0], X1.iloc[:, 1:]).fit()
132     regvw = sm.OLS(X2.iloc[:, 0], X2.iloc[:, 1:]).fit()
133     resultew.iloc[6, i] = regeq.params[0]
134     resultew.iloc[7, i] = regeq.tvalues[0]
135     resultvw.iloc[6, i] = regvw.params[0]
136     resultvw.iloc[7, i] = regvw.tvalues[0]
137
138     ff30=pd.read_csv('..\DataBase\ff30.csv')
139     x1 = sm.add_constant(ff30.iloc[:, 1:])
140     X1 = pd.concat([exeq, x1], axis=1)
141     X1 = X1.dropna()
142     X2 = pd.concat([exvw, x1], axis=1)
143     X2 = X2.dropna()
144     regeq = sm.OLS(np.asarray(X1.iloc[:, 0]), np.asarray(X1.iloc[:, 1:])).fit()
145     regvw = sm.OLS(np.asarray(X2.iloc[:, 0]), np.asarray(X2.iloc[:, 1:])).fit()
146     resultew.iloc[8, i] = regeq.params[0]
147     resultew.iloc[9, i] = regeq.tvalues[0]
148     resultvw.iloc[8, i] = regvw.params[0]
149     resultvw.iloc[9, i] = regvw.tvalues[0]
150     print(i)
151
152     #####3. 单因子 10 分组检验各组因子调整收益#####
153     import pandas as pd
154     import numpy as np
155     import glob,os
156     import statsmodels.api as sm
157     from scipy import stats
158     from NWttest import nwttest_lsamp
159     ret=pd.read_csv('final_return.csv')
160     size=pd.read_csv('..\DataBase\factor\01size.csv')
161     path = r'..\DataBase\factor'
162     file = glob.glob(os.path.join(path, "*.csv"))
163     rf=pd.read_csv('RF.csv')
164     k = [] # 每个因子一个表
165     for i in range(96):
166         k.append(pd.read_csv(file[i]))
167     factor = []#因子名称
168     for i in range(len(file)):
169         factor.append(file[i][29:-4])
170     for i in range(10):
171         resultew=pd.DataFrame
172         (columns=factor, index=['ret', 't', 'exret', 't', 'capmret', 't', 'ff3ret', 't', 'ff30ret', 't'])
173         resultvw = pd.DataFrame(columns=factor, index=['ret', 't', 'exret', 't', 'capmret', 't',
174 'ff3ret', 't', 'ff30ret', 't'])

```

```

175     for j in range(96):
176         totalew = []
177         totalvw = []
178         for m in range(len(ret.columns)-1):
179             final=pd.concat([size.iloc[:,m+1],k[j].iloc[:,m+1],ret.iloc[:,m+1]],axis=1)
180             final.columns=['size','factor','ret']
181             final=final.dropna()
182             if len(final)==0:
183                 totalew.append(np.nan)
184                 totalvw.append(np.nan)
185             else:
186                 final = final.sort_values(by='factor')
187                 final['VW'] = final.apply(lambda x: x['size'] * x['ret'], axis=1)
188                 total = final.iloc[int(len(final) / 10) * i:int(len(final) / 10) * (i +
189 1), :]
190                 totalew.append(total['ret'].mean())
191                 totalvw.append(total['VW'].sum() / total['size'].sum())
192             faceq = pd.Series(totalew)
193             facvw = pd.Series(totalvw)
194
195             #####return
196             reteq0 = faceq.dropna()
197             resultew.iloc[0, j] = reteq0.mean()
198             ttest = nwttest_1samp(reteq0, 0)
199             resultew.iloc[1, j] = ttest.statistic
200             retvw0 = facvw.dropna()
201             resultvw.iloc[0, j] = retvw0.mean()
202             ttest = nwttest_1samp(retvw0, 0)
203             resultvw.iloc[1, j] = ttest.statistic
204
205             ##excess return
206             exeq = faceq - rf.RF / 100
207             exvw = facvw - rf.RF / 100
208             exeq0 = exeq.dropna()
209             resultew.iloc[2, j] = exeq0.mean()
210             # ttest=stats.ttest_1samp(exeq0, 0)
211             ttest = nwttest_1samp(exeq0, 0)
212             resultew.iloc[3, j] = ttest.statistic
213             exvw0 = exvw.dropna()
214             resultvw.iloc[2, j] = exvw0.mean()
215             # ttest=stats.ttest_1samp(exvw0, 0)
216             ttest = nwttest_1samp(exvw0, 0)
217             resultvw.iloc[3, j] = ttest.statistic
218             ##CAPM-alpha/FF3-alpha/去掉市值最小 30%股票后 adj-FF3-alpha

```



```

219     ff3 = pd.read_csv('ff3.csv')
220     capm = ff3.iloc[:, 1]
221     x1 = sm.add_constant(capm)
222     X1 = pd.concat([exeq[:-2], x1], axis=1)
223     X1 = X1.dropna()
224     X2 = pd.concat([exvw[:-2], x1], axis=1)
225     X2 = X2.dropna()
226     regeq = sm.OLS(X1.iloc[:, 0], X1.iloc[:, 1:]).fit()
227     regvw = sm.OLS(X2.iloc[:, 0], X2.iloc[:, 1:]).fit()
228     resultew.iloc[4, j] = regeq.params[0]
229     resultew.iloc[5, j] = regeq.tvalues[0]
230     resultvw.iloc[4, j] = regvw.params[0]
231     resultvw.iloc[5, j] = regvw.tvalues[0]
232
233     x1 = sm.add_constant(ff3.iloc[:, 1:])
234     X1 = pd.concat([exeq[:-2], x1], axis=1)
235     X1 = X1.dropna()
236     X2 = pd.concat([exvw[:-2], x1], axis=1)
237     X2 = X2.dropna()
238     regeq = sm.OLS(X1.iloc[:, 0], X1.iloc[:, 1:]).fit()
239     regvw = sm.OLS(X2.iloc[:, 0], X2.iloc[:, 1:]).fit()
240     resultew.iloc[6, j] = regeq.params[0]
241     resultew.iloc[7, j] = regeq.tvalues[0]
242     resultvw.iloc[6, j] = regvw.params[0]
243     resultvw.iloc[7, j] = regvw.tvalues[0]
244
245     ff30 = pd.read_csv('..\DataBase\\ff30.csv')
246     x1 = sm.add_constant(ff30.iloc[:, 1:])
247     X1 = pd.concat([exeq, x1], axis=1)
248     X1 = X1.dropna()
249     X2 = pd.concat([exvw, x1], axis=1)
250     X2 = X2.dropna()
251     regeq = sm.OLS(X1.iloc[:, 0], X1.iloc[:, 1:]).fit()
252     regvw = sm.OLS(X2.iloc[:, 0], X2.iloc[:, 1:]).fit()
253     resultew.iloc[8, j] = regeq.params[0]
254     resultew.iloc[9, j] = regeq.tvalues[0]
255     resultvw.iloc[8, j] = regvw.params[0]
256     resultvw.iloc[9, j] = regvw.tvalues[0]
257     resultew.to_csv('RESLew'+str(i)+' .csv')
258     resultvw.to_csv('RESLvw' + str(i) + ' .csv')
259
260     #####4. 各因子与 size 因子独立双变量分组检验结果
261     #####
262     import pandas as pd

```

```

263 import numpy as np
264 import glob, os
265 import statsmodels.api as sm
266 from NWttest import nwttest_lsamp
267 ret=pd.read_csv('..\DataBase\\final_return.csv')
268 size=pd.read_csv('..\DataBase\\factor\\01size.csv')
269 size=size.iloc[:, :-1]
270 size.columns=ret.columns
271 path = r'..\DataBase\\factor'
272 file = glob.glob(os.path.join(path, "*.csv"))
273 rf=pd.read_csv('..\DataBase\\RF.csv')
274 k = [] # 每个因子一个表
275 for i in range(96):
276     k.append(pd.read_csv(file[i]))
277 factor = [] #因子名称
278 for i in range(len(file)):
279     factor.append(file[i][29:-4])
280
281 def gendata(x1, x2, ret, size, m=5, n=5):
282     x1 = x1[~np.isnan(ret)]
283     x1=x1[~np.isnan(size)]
284     x2 = x2[~np.isnan(ret)]
285     x2 = x2[~np.isnan(size)]
286     x1sort = x1.apply(lambda x: np.argsort(x), axis=0)
287     x2sort = x2.apply(lambda x: np.argsort(x), axis=0)
288     datacolumn = []
289     for a in range(m):
290         for b in range(n):
291             datacolumn.append(str(a + 1) + 'X' + str(b + 1))
292     dfeq=pd.DataFrame(columns=datacolumn, index=[i for i in range(len(ret.columns)-1)])
293     dfvw = pd.DataFrame(columns=datacolumn, index=[i for i in range(len(ret.columns) - 1)])
294     for i in range(len(ret.columns)-1):
295         truelistx1 = x1sort.iloc[:, i+1 ]
296         truelistx2 = x2sort.iloc[:, i+1]
297         truelistx1 = truelistx1[~(truelistx1 == -1)].sort_values()
298         truelistx1=truelistx1.index
299         truelistx2 = truelistx2[~(truelistx2 == -1)].sort_values()
300         truelistx2 = truelistx2.index
301         if len(truelistx1)<5 or len(truelistx2)<5:
302             dfeq.iloc[i, :] =np.nan
303             dfvw.iloc[i, :] =np.nan
304         else:
305             x1lines = np.array_split(np.array(truelistx1), m)
306             x2lines = np.array_split(np.array(truelistx2), n)

```

```

307         for a in range(m):
308             xluse = x1lines[a]
309             for b in range(n):
310                 x2use = x2lines[b]
311                 tempindex = np.intersect1d(xluse, x2use)
312                 if len(tempindex)==0:
313                     dfeq.iloc[i, a * m + b]=np.nan
314                     dfvw.iloc[i, a * m + b] = np.nan
315                 else:
316                     dfeq.iloc[i, a * m + b] = ret.iloc[tempindex, i + 1].mean()
317                     total = pd.concat([ret.iloc[tempindex, i + 1], size.iloc[tempindex,
318 i + 1]], axis=1)
319                     total.columns = ['ret', 'size']
320                     total['vw'] = total.apply(lambda x: x['size'] * x['ret'], axis=1)
321                     dfvw.iloc[i, a * m + b] = total['vw'].sum() / total['size'].sum()
322         return dfeq, dfvw
323
324
325 RETEQ, RETVQ=[], []
326 for i in range(len(k)):
327     k[i] = k[i].iloc[:, :264]
328     k[i].columns = ret.columns
329     retev, retvw = gendata(size, k[i], ret, size)
330     RETEQ.append(retev)
331     RETVQ.append(retvw)
332
333 #删除 SIZE 因子本身 (不与自身分组)
334 del RETEQ[0]
335 del RETVQ[0]
336 del factor[0]
337
338 for l in range(5):
339     longew=pd.concat([i.iloc[:, 1*5+4] for i in RETEQ], axis=1)
340     shortew = pd.concat([i.iloc[:, 1 * 5 ] for i in RETEQ], axis=1)
341     longvw = pd.concat([i.iloc[:, 1 * 5 + 4] for i in RETVQ], axis=1)
342     shortvw = pd.concat([i.iloc[:, 1 * 5] for i in RETVQ], axis=1)
343     longew.columns = factor
344     longvw.columns = factor
345     shortew.columns = factor
346     shortvw.columns = factor
347     lsew=longew-shortew
348     lsvw=longvw-shortvw
349     resultew = pd.DataFrame(columns=factor,
350                             index=['ret', 't', 'exret', 't', 'capmret', 't', 'ff3ret', 't',

```

```

351 'ff30ret', 't'])
352 resultvw = pd.DataFrame(columns=factor,
353                           index=['ret', 't', 'exret', 't', 'capmret', 't', 'ff3ret', 't',
354 'ff30ret', 't'])
355 for j in range(len(factor)):
356     faceq, facvw = lsew.iloc[:, j], lsvw.iloc[:, j]
357     reteq0 = faceq.dropna()
358     resultew.iloc[0, j] = reteq0.mean()
359     ttest = nwttest_lsamp(reteq0, 0)
360     resultew.iloc[1, j] = ttest.statistic
361     retvw0 = facvw.dropna()
362     resultvw.iloc[0, j] = retvw0.mean()
363     ttest = nwttest_lsamp(retvw0, 0)
364     resultvw.iloc[1, j] = ttest.statistic
365     ##excess return
366     exeq = faceq - rf.RF / 100
367     exvw = facvw - rf.RF / 100
368     exeq0 = exeq.dropna()
369     resultew.iloc[2, j] = exeq0.mean()
370     # ttest=stats.ttest_lsamp(exeq0, 0)
371     ttest = nwttest_lsamp(exeq0, 0)
372     resultew.iloc[3, j] = ttest.statistic
373     exvw0 = exvw.dropna()
374     resultvw.iloc[2, j] = exvw0.mean()
375     # ttest=stats.ttest_lsamp(exvw0, 0)
376     ttest = nwttest_lsamp(exvw0, 0)
377     resultvw.iloc[3, j] = ttest.statistic
378     ##CAPM
379     ff3 = pd.read_csv('..\DataBase\\ff3.csv')
380     capm = ff3.iloc[:, 1]
381     x1 = sm.add_constant(capm)
382     X1 = pd.concat([exeq[:-2], x1], axis=1)
383     X1 = X1.dropna()
384     X2 = pd.concat([exvw[:-2], x1], axis=1)
385     X2 = X2.dropna()
386     regeq = sm.OLS(np.asarray(X1.iloc[:, 0]), np.asarray(X1.iloc[:, 1:])).fit()
387     regvw = sm.OLS(np.asarray(X2.iloc[:, 0]), np.asarray(X2.iloc[:, 1:])).fit()
388     resultew.iloc[4, j] = regeq.params[0]
389     resultew.iloc[5, j] = regeq.tvalues[0]
390     resultvw.iloc[4, j] = regvw.params[0]
391     resultvw.iloc[5, j] = regvw.tvalues[0]
392     x1 = sm.add_constant(ff3.iloc[:, 1])
393     X1 = pd.concat([exeq[:-2], x1], axis=1)
394     X1 = X1.dropna()

```

```

395     X2 = pd.concat([exvw[:-2], x1], axis=1)
396     X2 = X2.dropna()
397     regeq = sm.OLS(np.asarray(X1.iloc[:, 0]), np.asarray(X1.iloc[:, 1:])).fit()
398     regvw = sm.OLS(np.asarray(X2.iloc[:, 0]), np.asarray(X2.iloc[:, 1:])).fit()
399     resultew.iloc[6, j] = regeq.params[0]
400     resultew.iloc[7, j] = regeq.tvalues[0]
401     resultvw.iloc[6, j] = regvw.params[0]
402     resultvw.iloc[7, j] = regvw.tvalues[0]
403     ff30 = pd.read_csv('..\DataBase\\ff30.csv')
404     x1 = sm.add_constant(ff30.iloc[:, 1:])
405     X1 = pd.concat([exeq, x1], axis=1)
406     X1 = X1.dropna()
407     X2 = pd.concat([exvw, x1], axis=1)
408     X2 = X2.dropna()
409     regeq = sm.OLS(np.asarray(X1.iloc[:, 0]), np.asarray(X1.iloc[:, 1:])).fit()
410     regvw = sm.OLS(np.asarray(X2.iloc[:, 0]), np.asarray(X2.iloc[:, 1:])).fit()
411     resultew.iloc[8, j] = regeq.params[0]
412     resultew.iloc[9, j] = regeq.tvalues[0]
413     resultvw.iloc[8, j] = regvw.params[0]
414     resultvw.iloc[9, j] = regvw.tvalues[0]
415     resultew.to_csv('longshort55SIZEEW' + str(1) + '.csv')
416     resultvw.to_csv('longshort55SIZEVW' + str(1) + '.csv')
417
418 for u in range(25):
419     DATAEQ=pd.concat([i.iloc[:,u] for i in RETEQ],axis=1)
420     DATAVQ = pd.concat([i.iloc[:, u] for i in RETVQ],axis=1)
421     DATAEQ.columns=factor
422     DATAVQ.columns = factor
423     resultew = pd.DataFrame(columns=factor,
424                             index=['ret', 't', 'exret', 't', 'capmret', 't', 'ff3ret',
425 't', 'ff30ret', 't'])
426     resultvw = pd.DataFrame(columns=factor,
427                             index=['ret', 't', 'exret', 't', 'capmret', 't', 'ff3ret',
428 't', 'ff30ret', 't'])
429     for j in range(len(factor)):
430         faceq, facvw=DATAEQ.iloc[:,j],DATAVQ.iloc[:,j]
431         reteq0 = faceq.dropna()
432         resultew.iloc[0, j] = reteq0.mean()
433         ttest = nwttest_1samp(reteq0, 0)
434         resultew.iloc[1, j] = ttest.statistic
435         retvw0 = facvw.dropna()
436         resultvw.iloc[0, j] = retvw0.mean()
437         ttest = nwttest_1samp(retvw0, 0)
438         resultvw.iloc[1, j] = ttest.statistic

```

```

439
440     ##excess return
441     exeq = faceq - rf.RF / 100
442     exvw = facvw - rf.RF / 100
443     exeq0 = exeq.dropna()
444     resultew.iloc[2, j] = exeq0.mean()
445     # ttest=stats.ttest_1samp(exeq0, 0)
446     ttest = nwttest_1samp(exeq0, 0)
447     resultew.iloc[3, j] = ttest.statistic
448     exvw0 = exvw.dropna()
449     resultvw.iloc[2, j] = exvw0.mean()
450     # ttest=stats.ttest_1samp(exvw0, 0)
451     ttest = nwttest_1samp(exvw0, 0)
452     resultvw.iloc[3, j] = ttest.statistic
453     ##CAPM
454     ff3 = pd.read_csv('..\DataBase\ff3.csv')
455     capm = ff3.iloc[:, 1]
456     x1 = sm.add_constant(capm)
457     X1 = pd.concat([exeq[:-2], x1], axis=1)
458     X1 = X1.dropna()
459     X2 = pd.concat([exvw[:-2], x1], axis=1)
460     X2 = X2.dropna()
461     regeq = sm.OLS(np.asarray(X1.iloc[:, 0]), np.asarray(X1.iloc[:, 1:])).fit()
462     regvw = sm.OLS(np.asarray(X2.iloc[:, 0]), np.asarray(X2.iloc[:, 1:])).fit()
463     resultew.iloc[4, j] = regeq.params[0]
464     resultew.iloc[5, j] = regeq.tvalues[0]
465     resultvw.iloc[4, j] = regvw.params[0]
466     resultvw.iloc[5, j] = regvw.tvalues[0]
467
468     x1 = sm.add_constant(ff3.iloc[:, 1:])
469     X1 = pd.concat([exeq[:-2], x1], axis=1)
470     X1 = X1.dropna()
471     X2 = pd.concat([exvw[:-2], x1], axis=1)
472     X2 = X2.dropna()
473     regeq = sm.OLS(np.asarray(X1.iloc[:, 0]), np.asarray(X1.iloc[:, 1:])).fit()
474     regvw = sm.OLS(np.asarray(X2.iloc[:, 0]), np.asarray(X2.iloc[:, 1:])).fit()
475     resultew.iloc[6, j] = regeq.params[0]
476     resultew.iloc[7, j] = regeq.tvalues[0]
477     resultvw.iloc[6, j] = regvw.params[0]
478     resultvw.iloc[7, j] = regvw.tvalues[0]
479
480     ff30 = pd.read_csv('C:\Users\15083\Desktop\ff30.csv')
481     x1 = sm.add_constant(ff30.iloc[:, 1:])
482     X1 = pd.concat([exeq, x1], axis=1)

```

```

483     X1 = X1.dropna()
484     X2 = pd.concat([exvw, x1], axis=1)
485     X2 = X2.dropna()
486     regeq = sm.OLS(np.asarray(X1.iloc[:, 0]), np.asarray(X1.iloc[:, 1:])).fit()
487     regvw = sm.OLS(np.asarray(X2.iloc[:, 0]), np.asarray(X2.iloc[:, 1:])).fit()
488     resultew.iloc[8, j] = regeq.params[0]
489     resultew.iloc[9, j] = regeq.tvalues[0]
490     resultvw.iloc[8, j] = regvw.params[0]
491     resultvw.iloc[9, j] = regvw.tvalues[0]
492     resultew.to_csv('EW55size'+reteq.columns[u]+' .csv')
493     resultvw.to_csv('VW55size' + reteq.columns[u] + ' .csv')
494
495

```

```

496 #####5. 各因子与 BM 因子独立双变量分组检验结果

```

```

497 #####
498 import pandas as pd
499 import numpy as np
500 import glob, os
501 import statsmodels.api as sm
502 from NWttest import nwttest_lsamp
503
504 ret = pd.read_csv('..\DataBase\\final_return.csv')
505 size = pd.read_csv('..\DataBase\\factor\\01size.csv')
506 BM = pd.read_csv('..\DataBase\\factor\\28BM.csv')
507 size = size.iloc[:, :-1]
508 size.columns = ret.columns
509 path = r'..\DataBase\\factor'
510 file = glob.glob(os.path.join(path, "*.csv"))
511 rf = pd.read_csv('..\DataBase\\RF.csv')
512 k = [] # 每个因子一个表
513 for i in range(96):
514     k.append(pd.read_csv(file[i]))
515 factor = [] # 因子名称
516 for i in range(len(file)):
517     factor.append(file[i][29:-4])
518
519 BM = BM.iloc[:, :-1]
520 BM.columns = ret.columns
521
522
523 def gendata(x1, x2, ret, size, m=5, n=5):
524     x1 = x1[~np.isnan(ret)]
525     x1 = x1[~np.isnan(size)]
526     x2 = x2[~np.isnan(ret)]

```

```

527     x2 = x2[~np.isnan(size)]
528     x1sort = x1.apply(lambda x: np.argsort(x), axis=0)
529     x2sort = x2.apply(lambda x: np.argsort(x), axis=0)
530     datacolumn = []
531
532     for a in range(m):
533         for b in range(n):
534             datacolumn.append(str(a + 1) + 'X' + str(b + 1))
535     dfreq = pd.DataFrame(columns=datacolumn, index=[i for i in range(len(ret.columns) - 1)])
536     dfvw = pd.DataFrame(columns=datacolumn, index=[i for i in range(len(ret.columns) - 1)])
537     for i in range(len(ret.columns) - 1):
538         truelistx1 = x1sort.iloc[:, i + 1]
539         truelistx2 = x2sort.iloc[:, i + 1]
540         truelistx1 = truelistx1[~(truelistx1 == -1)].sort_values()
541         truelistx1 = truelistx1.index
542         truelistx2 = truelistx2[~(truelistx2 == -1)].sort_values()
543         truelistx2 = truelistx2.index
544         if len(truelistx1) < 5 or len(truelistx2) < 5:
545             dfreq.iloc[i, :] = np.nan
546             dfvw.iloc[i, :] = np.nan
547         else:
548             x1lines = np.array_split(np.array(truelistx1), m)
549             x2lines = np.array_split(np.array(truelistx2), n)
550             for a in range(m):
551                 xluse = x1lines[a]
552                 lsew=[]
553                 lsvw = []
554                 for b in range(n):
555                     x2use = x2lines[b]
556                     tempindex = np.intersect1d(xluse, x2use)
557                     if len(tempindex) == 0:
558                         dfreq.iloc[i, a * m + b] = np.nan
559                         dfvw.iloc[i, a * m + b] = np.nan
560                     lsew.append(0)
561                     lsvw.append(0)
562                 else:
563                     dfreq.iloc[i, a * m + b] = ret.iloc[tempindex, i + 1].mean()
564                     total = pd.concat([ret.iloc[tempindex, i + 1], size.iloc[tempindex,
565 i + 1]], axis=1)
566                     total.columns = ['ret', 'size']
567                     total['vw'] = total.apply(lambda x: x['size'] * x['ret'], axis=1)
568                     dfvw.iloc[i, a * m + b] = total['vw'].sum() / total['size'].sum()
569                     lsew.append(ret.iloc[tempindex, i + 1].mean())
570                     lsvw.append(total['vw'].sum() / total['size'].sum())

```



```

571     return dfeq, dfvw
572
573
574 RETEQ, RETVQ = [], []
575 for i in range(len(k)):
576     k[i] = k[i].iloc[:, :264]
577     k[i].columns = ret.columns
578     reteq, retvw = gendata(BM, k[i], ret, size)
579     RETEQ.append(reteq)
580     RETVQ.append(retvw)
581
582 #删除 BM 因子本身 (不与自身分组)
583 del RETEQ[27]
584 del RETVQ[27]
585 del factor[27]
586
587 for l in range(5):
588     longew=pd.concat([i.iloc[:, 1*5+4] for i in RETEQ], axis=1)
589     shortew = pd.concat([i.iloc[:, 1 * 5 ] for i in RETEQ], axis=1)
590     longvw = pd.concat([i.iloc[:, 1 * 5 + 4] for i in RETVQ], axis=1)
591     shortvw = pd.concat([i.iloc[:, 1 * 5] for i in RETVQ], axis=1)
592     longew.columns = factor
593     longvw.columns = factor
594     shortew.columns = factor
595     shortvw.columns = factor
596     lsew=longew-shortew
597     lsvw=longvw-shortvw
598     resultew = pd.DataFrame(columns=factor,
599                               index=['ret', 't', 'exret', 't', 'capmret', 't', 'ff3ret', 't',
600 'ff30ret', 't'])
601     resultvw = pd.DataFrame(columns=factor,
602                               index=['ret', 't', 'exret', 't', 'capmret', 't', 'ff3ret', 't',
603 'ff30ret', 't'])
604     for j in range(len(factor)):
605         faceq, facvw = lsew.iloc[:, j], lsvw.iloc[:, j]
606         reteq0 = faceq.dropna()
607         resultew.iloc[0, j] = reteq0.mean()
608         ttest = nwttest_lsamp(reteq0, 0)
609         resultew.iloc[1, j] = ttest.statistic
610         retvw0 = facvw.dropna()
611         resultvw.iloc[0, j] = retvw0.mean()
612         ttest = nwttest_lsamp(retvw0, 0)
613         resultvw.iloc[1, j] = ttest.statistic
614     ##excess return

```

```

615     exeq = faceq - rf.RF / 100
616     exvw = facvw - rf.RF / 100
617     exeq0 = exeq.dropna()
618     resultew.iloc[2, j] = exeq0.mean()
619     # ttest=stats.ttest_1samp(exeq0, 0)
620     ttest = nwttest_1samp(exeq0, 0)
621     resultew.iloc[3, j] = ttest.statistic
622     exvw0 = exvw.dropna()
623     resultvw.iloc[2, j] = exvw0.mean()
624     # ttest=stats.ttest_1samp(exvw0, 0)
625     ttest = nwttest_1samp(exvw0, 0)
626     resultvw.iloc[3, j] = ttest.statistic
627     ##CAPM
628     ff3 = pd.read_csv('..\DataBase\ff3.csv')
629     capm = ff3.iloc[:, 1]
630     x1 = sm.add_constant(capm)
631     X1 = pd.concat([exeq[:-2], x1], axis=1)
632     X1 = X1.dropna()
633     X2 = pd.concat([exvw[:-2], x1], axis=1)
634     X2 = X2.dropna()
635     regeq = sm.OLS(np.asarray(X1.iloc[:, 0]), np.asarray(X1.iloc[:, 1:])).fit()
636     regvw = sm.OLS(np.asarray(X2.iloc[:, 0]), np.asarray(X2.iloc[:, 1:])).fit()
637     resultew.iloc[4, j] = regeq.params[0]
638     resultew.iloc[5, j] = regeq.tvalues[0]
639     resultvw.iloc[4, j] = regvw.params[0]
640     resultvw.iloc[5, j] = regvw.tvalues[0]
641     x1 = sm.add_constant(ff3.iloc[:, 1:])
642     X1 = pd.concat([exeq[:-2], x1], axis=1)
643     X1 = X1.dropna()
644     X2 = pd.concat([exvw[:-2], x1], axis=1)
645     X2 = X2.dropna()
646     regeq = sm.OLS(np.asarray(X1.iloc[:, 0]), np.asarray(X1.iloc[:, 1:])).fit()
647     regvw = sm.OLS(np.asarray(X2.iloc[:, 0]), np.asarray(X2.iloc[:, 1:])).fit()
648     resultew.iloc[6, j] = regeq.params[0]
649     resultew.iloc[7, j] = regeq.tvalues[0]
650     resultvw.iloc[6, j] = regvw.params[0]
651     resultvw.iloc[7, j] = regvw.tvalues[0]
652     ff30 = pd.read_csv('..\DataBase\ff30.csv')
653     x1 = sm.add_constant(ff30.iloc[:, 1:])
654     X1 = pd.concat([exeq, x1], axis=1)
655     X1 = X1.dropna()
656     X2 = pd.concat([exvw, x1], axis=1)
657     X2 = X2.dropna()
658     regeq = sm.OLS(np.asarray(X1.iloc[:, 0]), np.asarray(X1.iloc[:, 1:])).fit()

```

```

659     regvw = sm.OLS(np.asarray(X2.iloc[:, 0]), np.asarray(X2.iloc[:, 1:])).fit()
660     resultew.iloc[8, j] = regeq.params[0]
661     resultew.iloc[9, j] = regeq.tvalues[0]
662     resultvw.iloc[8, j] = regvw.params[0]
663     resultvw.iloc[9, j] = regvw.tvalues[0]
664     resultew.to_csv('longshort55BMEW' + str(1) + '.csv')
665     resultvw.to_csv('longshort55BMVW' + str(1) + '.csv')
666
667 for u in range(25):
668     DATAEQ = pd.concat([i.iloc[:, u] for i in RETEQ], axis=1)
669     DATAVQ = pd.concat([i.iloc[:, u] for i in RETVQ], axis=1)
670     DATAEQ.columns = factor
671     DATAVQ.columns = factor
672     resultew = pd.DataFrame(columns=factor,
673                             index=['ret', 't', 'exret', 't', 'capmret', 't', 'ff3ret', 't',
674 'ff30ret', 't'])
675     resultvw = pd.DataFrame(columns=factor,
676                             index=['ret', 't', 'exret', 't', 'capmret', 't', 'ff3ret', 't',
677 'ff30ret', 't'])
678     for j in range(len(factor)):
679         faceq, facvw = DATAEQ.iloc[:, j], DATAVQ.iloc[:, j]
680         reteq0 = faceq.dropna()
681         resultew.iloc[0, j] = reteq0.mean()
682         ttest = nwttest_lsamp(reteq0, 0)
683         resultew.iloc[1, j] = ttest.statistic
684         retvw0 = facvw.dropna()
685         resultvw.iloc[0, j] = retvw0.mean()
686         ttest = nwttest_lsamp(retvw0, 0)
687         resultvw.iloc[1, j] = ttest.statistic
688         ##excess return
689         exeq = faceq - rf.RF / 100
690         exvw = facvw - rf.RF / 100
691         exeq0 = exeq.dropna()
692         resultew.iloc[2, j] = exeq0.mean()
693         # ttest=stats.ttest_lsamp(exeq0, 0)
694         ttest = nwttest_lsamp(exeq0, 0)
695         resultew.iloc[3, j] = ttest.statistic
696         exvw0 = exvw.dropna()
697         resultvw.iloc[2, j] = exvw0.mean()
698         # ttest=stats.ttest_lsamp(exvw0, 0)
699         ttest = nwttest_lsamp(exvw0, 0)
700         resultvw.iloc[3, j] = ttest.statistic
701         ##CAPM
702         ff3 = pd.read_csv('..\DataBase\\ff3.csv')

```

```

703     capm = ff3.iloc[:, 1]
704     x1 = sm.add_constant(capm)
705     X1 = pd.concat([exeq[:-2], x1], axis=1)
706     X1 = X1.dropna()
707     X2 = pd.concat([exvw[:-2], x1], axis=1)
708     X2 = X2.dropna()
709     regeq = sm.OLS(np.asarray(X1.iloc[:, 0]), np.asarray(X1.iloc[:, 1:])).fit()
710     regvw = sm.OLS(np.asarray(X2.iloc[:, 0]), np.asarray(X2.iloc[:, 1:])).fit()
711     resultew.iloc[4, j] = regeq.params[0]
712     resultew.iloc[5, j] = regeq.tvalues[0]
713     resultvw.iloc[4, j] = regvw.params[0]
714     resultvw.iloc[5, j] = regvw.tvalues[0]
715     x1 = sm.add_constant(ff3.iloc[:, 1:])
716     X1 = pd.concat([exeq[:-2], x1], axis=1)
717     X1 = X1.dropna()
718     X2 = pd.concat([exvw[:-2], x1], axis=1)
719     X2 = X2.dropna()
720     regeq = sm.OLS(np.asarray(X1.iloc[:, 0]), np.asarray(X1.iloc[:, 1:])).fit()
721     regvw = sm.OLS(np.asarray(X2.iloc[:, 0]), np.asarray(X2.iloc[:, 1:])).fit()
722     resultew.iloc[6, j] = regeq.params[0]
723     resultew.iloc[7, j] = regeq.tvalues[0]
724     resultvw.iloc[6, j] = regvw.params[0]
725     resultvw.iloc[7, j] = regvw.tvalues[0]
726     ff30 = pd.read_csv('..\DataBase\\ff30.csv')
727     x1 = sm.add_constant(ff30.iloc[:, 1:])
728     X1 = pd.concat([exeq, x1], axis=1)
729     X1 = X1.dropna()
730     X2 = pd.concat([exvw, x1], axis=1)
731     X2 = X2.dropna()
732     regeqq = sm.OLS(np.asarray(X1.iloc[:, 0]), np.asarray(X1.iloc[:, 1:])).fit()
733     regvwq = sm.OLS(np.asarray(X2.iloc[:, 0]), np.asarray(X2.iloc[:, 1:])).fit()
734     resultew.iloc[8, j] = regeqq.params[0]
735     resultew.iloc[9, j] = regeqq.tvalues[0]
736     resultvw.iloc[8, j] = regvwq.params[0]
737     resultvw.iloc[9, j] = regvwq.tvalues[0]
738     resultew.to_csv('EW55BM' + reteq.columns[u] + '.csv')
739     resultvw.to_csv('VW55BM' + reteq.columns[u] + '.csv')
740
741
742     #####6.96 因子 fama macbeth 回归检验结果#####
743     #####
744     import pandas as pd
745     import numpy as np
746     import glob,os

```

```

747 import statsmodels.api as sm
748 from sklearn.preprocessing import scale
749 from factorNWTtest import nwttest_lsamp
750 ret=pd.read_csv('final_return.csv')
751 size=pd.read_csv('..\DataBase\factor\01size.csv')
752 path = r'..\DataBase\factor'
753 file = glob.glob(os.path.join(path, "*.csv"))
754 rf=pd.read_csv('..\DataBase\RF.csv')
755 k = [] # 每个因子一个表
756 for i in range(96):
757     k.append(pd.read_csv(file[i]))
758 factor = [] #因子名称
759 for i in range(len(file)):
760     factor.append(file[i][29:-4])
761 factor.insert(0, 'constant')
762 xishu=pd.DataFrame(index=ret.columns[1:], columns=factor)
763 for i in range(ret.shape[1]-1):
764     final=pd.concat([j.iloc[:, i+1] for j in k], axis=1)
765     final.columns=factor[1:]
766     final = sm.add_constant(final)
767     final['ret']=ret.iloc[:, i+1]
768     X1=final.dropna()
769     if len(X1)==0:
770         xishu.iloc[i, :]=np.nan
771     else:
772         X1.iloc[:, 1:-1] = scale(X1.iloc[:, 1:-1], axis=0)
773         reg = sm.OLS(np.asarray(X1.iloc[:, 0]), np.asarray(X1.iloc[:, 1:])).fit()
774         xishu.iloc[i, :]=reg.params
775 FM=pd.DataFrame(columns=xishu.columns, index=['mean', 't-stats'])
776 gen=xishu.dropna()
777 for i in range(xishu.shape[1]):
778     fa = gen.iloc[:, i]
779     FM.iloc[0, i] = fa.mean() #截面系数时序均值
780     ttest = nwttest_lsamp(fa, 0) #t 检验 因子系数是否异于0
781     FM.iloc[1, i] = ttest.statistic
782
783 FM.to_csv('FM.csv')

```

## g) DFN.py

```
1  # -*- coding:utf-8 -*-
2
3  @description:
4  深度前馈网络库函数, 依赖 mxnet 和 gpu, 在有 GPU 的环境下运行。
5
6  from mxnet import gluon, nd, gpu, autograd
7  from mxnet.gluon import loss as gloss, nn, data as gdata
8  import mxnet as mx
9  import numpy as np
10
11 class DFN(nn.Block):
12     def __init__(self, outputdim, batch_size=10, epoch=10, depth=1, neuralset=[],
13                 activatemethod='relu', ctx=gpu(), trainmethod='sgd', lr=0.5, **kwargs):
14         """
15         主函数, 用于深度前馈网络训练
16         :param outputdim: 输出维度
17         :param batch_size: 小批量大小
18         :param epoch: 训练轮数
19         :param depth: 神经网络深度
20         :param neuralset: 各层神经元数, list 形式 eg. [96, 50, 10]
21         :param activatemethod: 激活函数
22         :param ctx: 训练用设备 默认 gpu
23         :param trainmethod: 训练器
24         :param lr: 学习率
25         :param kwargs: 其他参数
26         """
27         super(DFN, self).__init__(**kwargs)
28         mx.random.seed(521, ctx=gpu())
29         mx.random.seed(521)
30         self.ctx = ctx
31         self.outputdim = outputdim
32         self.net = nn.Sequential()
33         self.neuralset = neuralset
34         self.activatemethod = activatemethod
35         self.depth = depth
36         self.trainmethod = trainmethod
37         self.lr = lr
38         if neuralset:
39             pass
40         else:
41             neuralset = [256]*depth
42         for i in range(depth):
```

```

43         self.net.add(nn.Dense(neuralset[i], activation=activatemethod))
44     self.net.add(nn.Dense(outputdim))
45     self.net.initialize(ctx=self.ctx)
46     self.Trainer = gluon.Trainer(self.net.collect_params(), trainmethod,
47 {'learning_rate': lr})
48     self.loss = gloss.L2Loss()
49     self.batch_size = batch_size
50     self.epoch = epoch
51     print('network initializing finished')
52
53     def retrain(self, Xnd, Ynd, data_iter, k):
54         knew = 0
55         self.net.initialize(ctx=self.ctx, force_reinit=True)
56         self.Trainer = gluon.Trainer(self.net.collect_params(), self.trainmethod,
57 {'learning_rate': self.lr/2})
58         print('reinitialize finished and retrain begin')
59         for i in range(k - 1):
60             # with mx.autograd.record():
61             #     l = self.loss(self.net(Xnd), Ynd).mean()
62             # l.backward()
63             # self.Trainer.step(4400)
64             for xtrain, ytrain in data_iter:
65                 ytrain = nd.array(ytrain, ctx=self.ctx)
66                 with mx.autograd.record():
67                     l = self.loss(self.net(xtrain), ytrain).mean()
68                     l.backward()
69                     self.Trainer.step(1)
70                     print('#')
71                 knew += 1
72             lossout = self.loss(self.net(Xnd), Ynd)
73             print('epoch ' + str(i) + ': loss is %f' % lossout.mean().asnumpy())
74             if lossout.mean().asnumpy() > 1:
75                 self.retrain(Xnd, Ynd, data_iter, knew)
76                 break
77             elif np.isnan(lossout.mean().asnumpy()):
78                 self.retrain(Xnd, Ynd, data_iter, knew)
79                 break
80
81
82         # print(lossout)
83
84
85     def fit(self, X, Y):
86         Xnd = nd.array(X, ctx = self.ctx)

```

```

87     Ynd = nd.array(Y, ctx = self.ctx)
88     dataset = gdata.ArrayDataset(Xnd, Ynd)
89     # 随机读取小批量
90     data_iter = gdata.DataLoader(dataset, self.batch_size, shuffle=False)
91     k = 1
92     for i in range(self.epoch):
93         # with mx.autograd.record():
94         #     l = self.loss(self.net(Xnd), Ynd).mean()
95         # l.backward()
96         # self.Trainer.step(4400)
97         for xtrain, ytrain in data_iter:
98             ytrain = nd.array(ytrain, ctx=self.ctx)
99             with mx.autograd.record():
100                 l = self.loss(self.net(xtrain), ytrain).mean()
101                 l.backward()
102                 self.Trainer.step(1)
103                 print('#', )
104             k += 1
105             lossout = self.loss(self.net(Xnd), Ynd)
106             print('epoch ' + str(i) + ': loss is %f' % lossout.mean().asnumpy())
107             if lossout.mean().asnumpy() > 1:
108                 retrainNN = True
109                 break
110             elif np.isnan(lossout.mean().asnumpy()):
111                 retrainNN = True
112                 break
113             else:
114                 retrainNN = False
115             if retrainNN:
116                 self.retrain(Xnd, Ynd, data_iter, k)
117             print('all finished')
118
119     def predict(self, Xtest):
120         Xtestnd = nd.array(Xtest, ctx = self.ctx)
121         return list(map(lambda x: x[0], nd.array(self.net(Xtestnd), ctx=mx.cpu()).asnumpy()))

```



## h) RNNMODEL.py

```
1  # -*- coding:utf-8 -*-
2  '''
3  @description:
4      用于构建循环神经网络
5  '''
6
7  import mxnet as mx
8  from mxnet import nd, gluon, gpu
9  from mxnet.gluon import rnn, nn, loss as gloss, data as gdata
10 import numpy as np
11
12 class BaseRnn(nn.Block):
13     def __init__(self, num_feature, num_hidden, num_layers, ntype='RNN', bidirectional =
14 False, dropout=0, **kwargs):
15         '''
16         基本循环神经网络单元
17         :param num_feature: 特征个数
18         :param num_hidden: 隐层神经元个数
19         :param num_layers: 隐层数目
20         :param ntype: 神经网络类型, 默认 RNN
21         :param bidirectional: 是否双向, 默认否
22         :param dropout: 丢弃率, 即神经元多大概率不激活
23         :param kwargs: 其他参数
24         '''
25         super(BaseRnn, self).__init__(**kwargs)
26         self.num_hidden = num_hidden
27         if ntype == 'RNN':
28             with self.name_scope():
29                 self.rnn =
30 rnn.RNN(num_hidden, num_layers, input_size=num_feature, bidirectional=bidirectional,
31 layout='TNC', dropout=dropout)
32                 self.decoder = nn.Dense(1, in_units=num_hidden)
33         elif ntype == 'LSTM':
34             with self.name_scope():
35                 self.rnn =
36 rnn.LSTM(num_hidden, num_layers, input_size=num_feature, bidirectional=bidirectional,
37 layout='TNC', dropout=dropout)
38                 self.decoder = nn.Dense(1, in_units=num_hidden)
39         elif ntype == 'GRU':
40             with self.name_scope():
41                 self.rnn =
42 rnn.GRU(num_hidden, num_layers, input_size=num_feature, bidirectional=bidirectional,
```

```

43 layout='TNC', dropout=dropout)
44         self.decoder = nn.Dense(1, in_units=num_hidden)
45
46     def forward(self, inputs, hidden):
47         output, hidden = self.rnn(inputs, hidden)
48         decoded = self.decoder(output.reshape((-1, self.num_hidden)))
49         return decoded, hidden
50
51     def begin_state(self, *args, **kwargs):
52         return self.rnn.begin_state(*args, **kwargs)
53
54
55 class lstmmodule(object):
56     def
57     __init__(self, num_feature, num_hidden, num_layers, epoch, batch_size, ntype='LSTM', bidirectional
58 = False, dropout=0, trainmethod = 'adam', \
59         lr=0.01, loss = gloss.L1Loss(), ctx =
60 gpu(0), datashuffle=False, initialfunc=mx.init.Xavier(), prin=False, **kwargs):
61         '''
62         用于循环神经网络的训练和预测， 主函数
63         :param num_feature: 特征个数
64         :param num_hidden: 隐层神经元数目
65         :param num_layers: 隐层数
66         :param epoch: 训练轮数
67         :param batch_size: 小批量大小， 在这里是股票实体数目多少
68         :param ntype: 神经网络类型， 默认 LSTM
69         :param bidirectional: 是否双向神经网络， 默认否
70         :param dropout: 丢弃率， 默认不丢弃
71         :param trainmethod: 训练器， 默认 adam
72         :param lr: 学习率
73         :param loss: 损失函数
74         :param ctx: 训练设备， 默认 gpu
75         :param datashuffle: 是否随机打乱数据， 默认否
76         :param initialfunc: 初始化方法， 默认 Xavier
77         :param prin: 是否输出训练指示， 默认否
78         :param kwargs: 其他参数
79         '''
80         super(lstmmodule, self).__init__(**kwargs)
81         mx.random.seed(521, ctx=gpu())
82         mx.random.seed(521)
83         self.ctx = ctx
84         self.net = BaseRnn(num_feature, num_hidden, num_layers, ntype, bidirectional, dropout)
85         self.trainmethod = trainmethod
86         self.lr = lr

```

```

87     self.intialfunc = initialfunc
88     self.net.collect_params().initialize(self.intialfunc, ctx=self.ctx)
89     self.Trainer = gluon.Trainer(self.net.collect_params(), trainmethod,
90 {'learning_rate': lr})
91     self.loss = loss
92     self.datashuffle = datashuffle
93     self.epoch = epoch
94     self.batch_size = batch_size
95     self.prin = prin
96
97     def fit(self, X, Y):
98         Xnd = nd.array(X, ctx = self.ctx)
99         Ynd = nd.array(Y, ctx = self.ctx)
100        for i in range(self.epoch):
101            state = self.net.begin_state(batch_size=self.batch_size, ctx=self.ctx)
102            for s in state:
103                s.detach()
104            with mx.autograd.record():
105                (output, state) = self.net(Xnd, state)
106                l = self.loss(output.reshape(len(output)//self.batch_size, self.batch_size),
107 Ynd).mean()
108                l.backward()
109                self.Trainer.step(l)
110                if self.prin:
111                    print('#')
112                if i == self.epoch -1:
113                    print('the last epoch loss is ', l.asnumpy())
114                if self.prin:
115                    print('all finished')
116
117        def predict(self, Xtest):
118            state = self.net.begin_state(batch_size=self.batch_size, ctx=self.ctx)
119            Xtestnd = nd.array(Xtest, ctx = self.ctx)
120            (Y, statenew) = self.net(Xtestnd, state)
121            return nd.array(Y, ctx=mx.cpu()).asnumpy()

```

## i) Ensembleall.py

```

1  #-*- coding:utf-8 -*-
2  """
3  @description:
4      集成所有函数模型模块
5  """
6  import numpy as np
7  class Ensemblelr(object):
8      def __init__(self, classifymodellist, newmodellist, modelname):
9          """
10
11          :param classifymodellist: 非 RNN/LSTM 模型
12          :param newmodellist: RNN/LSTM 模型
13          :param modelname: 所有模型名称,
14          注: PLS 放到传统模型倒数第三位
15          """
16          self.cmodellist = classifymodellist
17          self.nml = newmodellist
18          self.modelname = modelname
19          self.cmodelnum = len(classifymodellist)
20          self.nmodelnum = len(newmodellist)
21          print('Ensembleall model initializing')
22
23      def fit(self, TX, TY, NX, NY):
24          for i in range(self.cmodelnum):
25              self.cmodellist[i].fit(TX, TY)
26              print(self.modelname[i]+' finished')
27          for i in range(self.nmodelnum):
28              self.nml[i].fit(NX, NY)
29              print(self.modelname[self.cmodelnum + i])
30          print('fit finished')
31
32      def predict(self, TXtest, NXtest, indlist, length):
33          predictlist = []
34          for i in range(self.cmodelnum):
35              predictlist.append(self.cmodellist[i].predict(TXtest))
36          predictlist[-3] = list(map(lambda x: x[0], predictlist[-3])) # 为了 PLS 的输出结果变
37  成数值
38          for i in range(self.nmodelnum):
39              KP = self.nml[i].predict(NXtest)
40              predictlist.append(np.array([KP[m][0] for m in range(3571 * (length - 1), 3571 *
41  length)]))
42          return list(np.mean(predictlist, axis=0))

```

## j) transecfee.py

```

1  #-*- coding:utf-8 -*-
2  '''
3  @description
4      用于输出交易费用之后的结果
5  '''
6  import numpy as np
7  import pandas as pd
8  import os
9  import gc
10 from NWttest import nwttest_lsamp
11 import statsmodels.formula.api as smf
12
13 factorlist = os.listdir('../Database/returnseries/12')
14 rf=pd.read_csv('../Database/RF.csv') #无风险 rf
15 rf12=rf.iloc[12:-1,: ]
16 def trasecfee(feerate):
17     '''
18     :param feerate: 交易费率
19     :return: 不输出只打印结果
20     '''
21     for i in range(len(factorlist)):
22         temp = pd.read_csv('../Database/returnseries/12/' + factorlist[i])
23         long_short = np.array(temp['long-short'])
24         afterLS = list(long_short - feerate*2)
25         length = 12
26         name = factorlist[i]
27         T_value = []
28         Mean = []
29         p_value = []
30         sharpratio = []
31         Std = []
32         T0 = [afterLS]
33         for l in T0:
34             t_test = nwttest_lsamp(l, 0, L=1)
35             mean = np.average(l) * 12- rf12.mean().tolist()[0] * 12 / 100
36             STD = np.std(l) * np.sqrt(12)
37             sharp = (mean) / STD
38             T_value.append(t_test.statistic)
39             p_value.append(t_test.pvalue)
40             Mean.append(mean)
41             Std.append(STD)
42             sharpratio.append(sharp)

```

```

43     print(name, 'long-short')
44     print('mean', Mean[0] / 12)
45     print('t-statistic', '(' + str(round(T_value[0], 4)) + ')')
46     ff3 = pd.read_csv('../Database/ff3.csv')
47     ff5 = pd.read_csv('../Database/ff5.csv')
48     A = pd.DataFrame(afterLS, columns=['long-short'])
49     M = pd.concat([A], axis=1)
50     alpha3 = []
51     t3 = []
52     t5 = []
53     alpha5 = []
54     for i in range(1):
55         X1 = ff3.iloc[length:, 1:]
56         X2 = ff5.iloc[length:, 1:]
57         Y = M.iloc[: -2, i]
58         Y.index = X1.index
59         Y = Y - rf12.RF[: -1] / 100
60         used1 = {'X': X1, 'Y': Y}
61         reg = smf.ols(formula='Y~1+X', data=used1).fit(cov_type='HAC',
62 cov_kws={'maxlags': 1})
63         t3.append(reg.tvalues[0])
64         alpha3.append(reg.params[0] * 12)
65         used2 = {'X': X2, 'Y': Y}
66         reg = smf.ols(formula='Y~1+X', data=used2).fit(cov_type='HAC',
67 cov_kws={'maxlags': 1})
68         t5.append(reg.tvalues[0])
69         alpha5.append(reg.params[0] * 12)
70     print('alpha-FF3', alpha3[0]/12)
71     print('t-statistic', '(' + str(round(t3[0], 4)) + ')')
72     print('alpha-FF5', alpha5[0]/12,)
73     print('t-statistic', '(' + str(round(t5[0], 4)) + ')')
74     print('sharpe', sharpratio[0])
75     gc.collect()
76     print('* *30) #分隔开不同收益序列
77
78
79 def showtrasecfee(transectionfee):
80     '''
81     :param transectionfee: 交易费用比例
82     :return: 无返回直接打印交易费用调整后的结果
83     '''
84     trasecfee(transectionfee) # 方便调用

```

### k) selectFactor.py

```
1  #-*- coding:utf-8 -*-
2  '''
3  @description
4  用于筛选特征
5  '''
6  import pandas as pd
7  import numpy as np
8  from sklearn.preprocessing import scale
9  from sklearn.linear_model import LinearRegression
10 import gc
11
12
13 def dropimportant(length, CLF, name, factor, timeseries, base = 0.028813732000000005, inpath =
14 'output'):
15     '''
16     主函数 1, 用于非循环神经网络特征筛选
17     :param length: 滑动窗口长度
18     :param CLF: 模型
19     :param name: 筛选用模型名称
20     :param timeseries: 数据
21     :param factor: 因子名称
22     :param base: 多空对应月度收益
23     :param inpath: 输出路径
24     :return: 月度收益差
25     '''
26     meanlist = []
27     for j in range(96):
28         print(j)
29         print(factor[j])
30         Long_Short = []
31         Long = []
32         Short = []
33         for i in range(len(timeseries) - (length+1)):
34
35             FINALm = pd.concat(timeseries[i:i + (length+1)], axis=0)
36             FINALm = FINALm.fillna(0)
37             FINAL_X= FINALm.iloc[:, :-2].copy()
38             FINAL_X.drop(columns=factor[j], inplace=True)
39             FINAL_x = scale(FINAL_X)
40             final = pd.concat(timeseries[i:i + length], axis=0)
41             x_train = FINAL_x[:len(final)]
42             x_test = FINAL_x[len(final):]
```

```

43         y_train = final.iloc[:, -1]
44         test = timeseries[i + length]
45         y_test = test.iloc[:, -1]
46         # 基准-linear
47         clf = CLF
48         clf.fit(x_train, y_train)
49         PREDICTION = clf.predict(x_test)
50         # 构建投资组合
51         prediction = pd.DataFrame(PREDICTION)
52         r_predict = pd.DataFrame(PREDICTION, columns=['predict'])
53         r_ture = pd.DataFrame(y_test)
54         r_ture.columns = ['ture']
55         r_ture.index = r_predict.index
56         FINAL = pd.concat([r_predict, r_ture], axis=1)
57         FINAL_sort = FINAL.sort_values(by='predict', axis=0)
58         r_final = np.array(FINAL_sort['ture'])
59         m = int(len(r_final) * 0.1) + 1
60         r_final = r_final.tolist()
61         long = r_final[-m:]
62         short = r_final[:m]
63         r_end = (np.sum(long) - np.sum(short)) / m
64         Long_Short.append(r_end)
65         Long.append(np.average(long))
66         Short.append(np.average(short))
67         gc.collect()
68         A = pd.DataFrame(Long_Short, columns=['long-short'])
69         B = pd.DataFrame(Long, columns=['long'])
70         C = pd.DataFrame(Short, columns=['short'])
71         M = pd.concat([A, B, C], axis=1)
72         meanlist.append(np.average(Long_Short))
73         if (i%10) == 0:
74             print(i)
75         gc.collect()
76         u = base - np.array(meanlist)
77         u = pd.DataFrame(u)
78         u.to_csv('../' + inpath + '/' + name + 'returngap.csv')
79         return u
80
81 def dropimportant2(length, CLF, name, factor, timeseries, base = 0.028813732000000005, inpath =
82 'output', a=0, b=96):
83     """
84     主函数2 用于循环神经网络特征筛选
85     :param length: 滑动窗口大小
86     :param CLF: 筛选用模型

```



```

87 :param name: 模型名称
88 :param factor: 因子名称
89 :param timeseries: 数据
90 :param base: 月度收益
91 :param inpath: 输出位置
92 :param a: 起始因子
93 :param b: 终止因子
94 :return: 返回收益差
95 '''
96 meanlist = []
97 for j in range(a,b):
98     print(j)
99     print(factor[j])
100     Long_Short = []
101     Long = []
102     Short = []
103     for i in range(len(timeseries) - length):
104         FINALm = pd.concat(timeseries[i:(i + length + 1)], axis=0)
105         FINALm[~FINALm['ret'].isin(['null'])] =
106         FINALm[~FINALm['ret'].isin(['null'])].fillna(0)
107         FINAL_X = FINALm.iloc[:, :-2]
108         FINAL_X.drop(columns=factor[j], inplace=True)
109         FINAL_x = FINAL_X
110         FINAL_x[~FINALm['ret'].isin(['null'])] =
111         scale(FINAL_X[~FINALm['ret'].isin(['null'])])
112         FINAL_x[FINALm['ret'].isin(['null'])] = 0
113         FINALm[FINALm['ret'].isin(['null'])] = 0
114         x_train = [FINAL_x.iloc[j * 3571:(j + 1) * 3571, :].values for j in
115         range(length)]
116         y_train = [FINALm.iloc[j * 3571:(j + 1) * 3571, -1].values for j in
117         range(length)]
118         x_test = [FINAL_x.iloc[j * 3571:(j + 1) * 3571, :].values for j in range(1,
119         length + 1)]
120         y_test = list(FINALm.iloc[(length) * 3571:(length + 1) * 3571, -1].values)
121         # 基准-linear
122         clf = CLF
123         clf.fit(x_train, y_train)
124         PREDICTION = clf.predict(x_test)
125         PREDICTION = [PREDICTION[m][0] for m in range(3571 * (length - 1), 3571 *
126         length)]
127         # 构建投资组合
128         r_predict = pd.DataFrame(PREDICTION, columns=['predict'])
129         r_ture = pd.DataFrame(y_test)
130         r_ture.columns = ['ture']

```

```

131         r_ture.index = r_predict.index
132         FINAL = pd.concat([r_predict, r_ture], axis=1)
133         FINAL = FINAL[~timeseries[i + length]['ret'].isin(['null'])]
134         FINAL_sort = FINAL.sort_values(by='predict', axis=0)
135         r_final = np.array(FINAL_sort['ture'])
136         m = int(len(r_final) * 0.1) + 1
137         r_final = r_final.tolist()
138         long = r_final[-m:]
139         short = r_final[:m]
140         r_end = (np.sum(long) - np.sum(short)) / m
141         Long_Short.append(r_end)
142         Long.append(np.average(long))
143         Short.append(np.average(short))
144         gc.collect()
145         A = pd.DataFrame(Long_Short, columns=['long-short'])
146         B = pd.DataFrame(Long, columns=['long'])
147         C = pd.DataFrame(Short, columns=['short'])
148         M = pd.concat([A, B, C], axis=1)
149         meanlist.append(np.average(Long_Short))
150         if (i%10) == 0:
151             print(i)
152         gc.collect()
153         u = base - np.array(meanlist)
154         u = pd.DataFrame(u)
155         u.to_csv('../' + inpath + '/' + name + 'returngap.csv')
156         return u
157
158 def FCselect(factor, timeseries):
159     """
160     用于FC 筛选因子
161     :param factor: 因子名称
162     :param timeseries: 数据
163     :return:
164     """
165     length = 12
166     base = 0.0228
167     name = 'FC'
168     inpath = 'output'
169     meanlist = []
170     for j in range(96):
171         print(j)
172         print(factor[j])
173         Long_Short = []
174         Long = []

```

```

175     Short = []
176     for i in range(len(timeseries) - (length + 1)):
177         FINALm = pd.concat(timeseries[i:i + (length + 1)], axis=0)
178         FINALm = FINALm.fillna(0)
179         FINAL_X = FINALm.iloc[:, :-2].copy()
180         FINAL_X.drop(columns=factor[j], inplace=True)
181         FINAL_x = scale(FINAL_X)
182         final = pd.concat(timeseries[i:i + length], axis=0)
183         x_train = FINAL_x[:len(final)]
184         x_test = FINAL_x[len(final):]
185         y_train = final.iloc[:, -1]
186         test = timeseries[i + length]
187         y_test = test.iloc[:, -1]
188         clf = LinearRegression()
189         k = []
190         for ax in range(95):
191             x = x_train[:, ax].reshape(-1, 1)
192             clf.fit(x, y_train)
193             k.append(clf.coef_[0])
194         PREDICTION = []
195         for ap in range(len(x_test)):
196             test0 = np.array(x_test[ap])
197             y = 0
198             for ass in range(95):
199                 y = y + test0[ass] * k[ass]
200             PREDICTION.append(y)
201         y_test = test.iloc[:, -1]
202         prediction = pd.DataFrame(PREDICTION)
203         r_predict = pd.DataFrame(PREDICTION, columns=['predict'])
204         r_ture = pd.DataFrame(y_test)
205         r_ture.columns = ['ture']
206         r_ture.index = r_predict.index
207         FINAL = pd.concat([r_predict, r_ture], axis=1)
208         FINAL_sort = FINAL.sort_values(by='predict', axis=0)
209         r_final = np.array(FINAL_sort['ture'])
210         m = int(len(r_final) * 0.1) + 1
211         r_final = r_final.tolist()
212         long = r_final[-m:]
213         short = r_final[:m]
214         r_end = (np.sum(long) - np.sum(short)) / m
215         Long_Short.append(r_end)
216         Long.append(np.average(long))
217         Short.append(np.average(short))
218     gc.collect()

```

```
219     meanlist.append(np.average(Long_Short))
220     if (i % 10) == 0:
221         print(i)
222     gc.collect()
223
224     u = base - np.array(meanlist)
225     u = pd.DataFrame(u)
226     u.to_csv('../' + inpath + '/' + name + 'returngap.csv')
```

