项目名称：基于遗传规划的选股因子挖掘

指导老师：元向辉

组员姓名：周一飞 2174214110

刘晓 2176122500

马毓婕 2176223611

姜泓任 2173711663

学院：经济与金融学院

日期：2020 年 8 月 3 日

目录

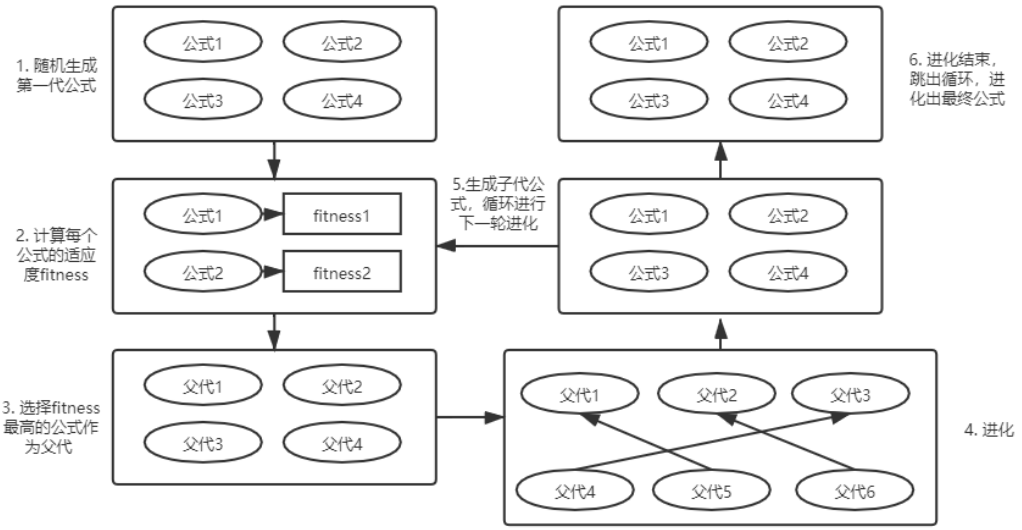
1.遗传规划基本原理介绍

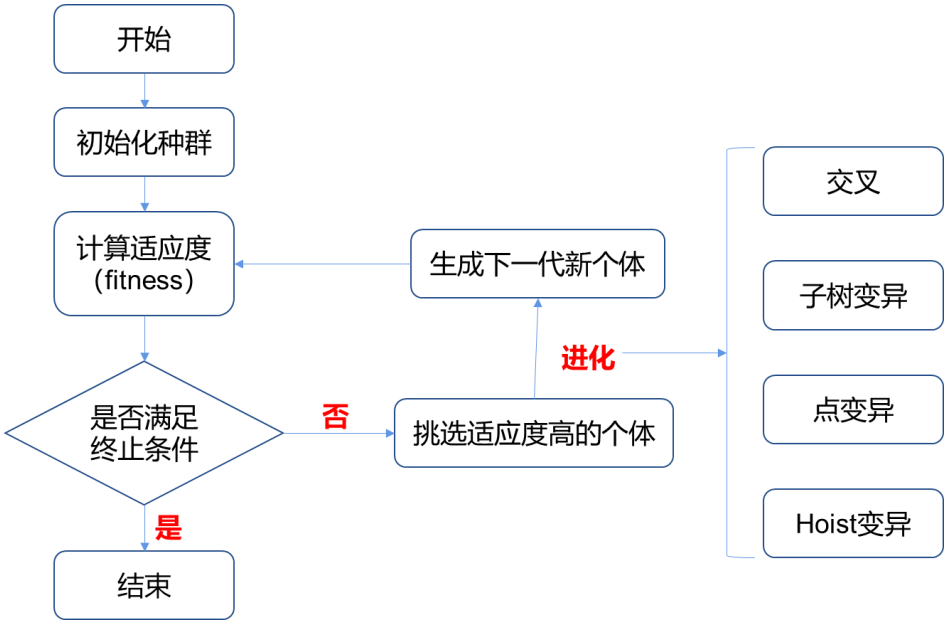
遗传规划(genetic programming)是演化算法(evolutionary algorithm)的分支，是一种启发式的公式演化技术。遗传规划从随机生成的公式群体开始，通过模拟自然界中遗传进化的过程，来逐渐生成契合特定目标的公式群体。作为一种监督学习方法，遗传规划可以根据特定目标，发现某些隐藏的、难以通过人脑构建出的数学公式。传统的监督学习算法主要运用于特征与标签之间关系的拟合，而遗传规划则更多运用于特征挖掘(特征工程)。

在以往的因子研究中，常见的是“先有逻辑、后有公式”的方法（演绎法），如常见因子有估值、成长、波动率等。而遗传规划的方法可以在海量的数据中探索并进一步筛选有效的因子，在试图取解释因子的内涵，即“先有公式、后有逻辑”的方法（归纳法）。本篇报告将从第二个角度出发，利用启发式搜索试图挖掘出隐藏的有效因子。

1.1 遗传规划的流程

1. 随机生成一组未经选择和进化的原始公式会被随机生成(第一代公式)
2. 通过某种规则计算每个公式的适应度，从中选出适合的个体作为下一代进化的父代。
3. 将这些被选择出来的父代通过多种方法进化，形成不同的后代公式。
4. 然后循环进行下一轮进化。随着迭代次数的增长，公式不断繁殖、变异、进化，从而不断逼近数据分布的真相。





图表 1 遗传规划图示及流程图

1.2 遗传规划的公式表示方法

为了方便进行公式的进化，遗传规划中的公式一般会被表示成二叉树的形式。假设有特征X_0和 X_1，需要预测目标 y。一个可能的公式是：

$$y = X_0^2 - 3 * X_1 + 0.5$$

在遗传规划中上式用 S-表达式(S-expression)表示为：

$$y = \left(+(-*X_0X_0)(*3X_1)\right)0.5)$$

公式里包括了变量(X_0和 X_1)、函数(加、减、乘)和常数(3 和 0.5)。即可把公式表示为一个二叉树。

下图为利用股票价格数据生成的因子公式图，表达式为 square(delta(div(5, high)))：

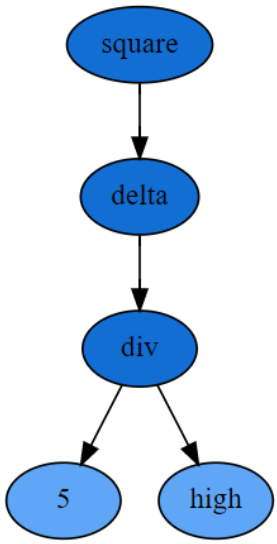


图 1 公式树图示

1.3 遗传规划中的适应度

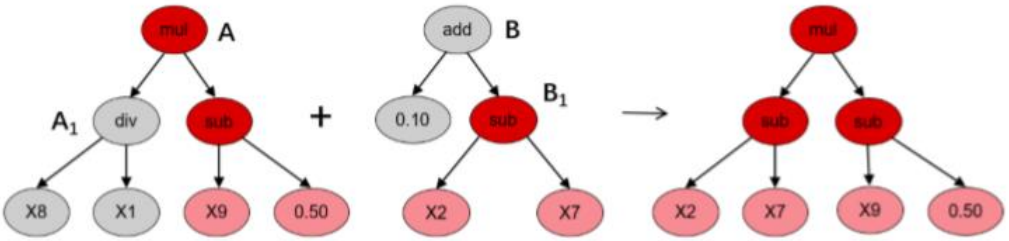
类比于自然界中个体对其生存环境的适应程度，在遗传规划中，每个公式也有自己的适应度，适应度衡量了公式运算结果与给定目标的相符程度，是公式进化的重要参考指标。在不同的应用中，可以定义不同的适应度，例如对于回归问题，可以使用公式结果和目标值之间的均方误差为适应度，对于分类问题，可以使用公式结果和目标值之间的交叉熵为适应度。对于使用遗传规划生成的选股因子来说，可以使用因子在回测区间内的平均 Rank IC 或因子收益率来作为适应度。

1.4 遗传规划中的进化方法

遗传规划的核心步骤是公式的进化，算法会参照生物进化的原理，使用多种方式对公式群体进行进化，来生成多样性的、更具适应性的下一代公式群体。本节将依次介绍这些进化方法。

a) 交叉

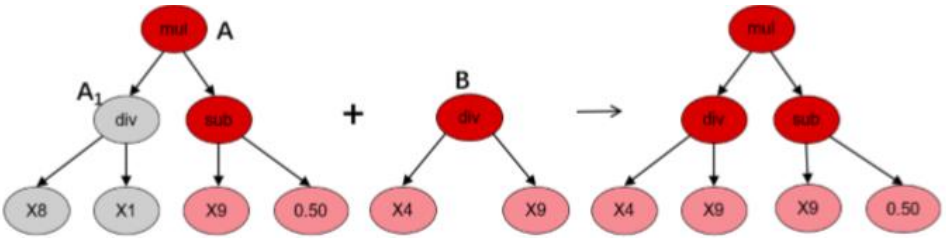
交叉是在两个已有公式树之间生成子树的方法，是最常用也最有效的进化方式。交叉需要通过两次选取找到父代和捐赠者，如图表 3 所示，首先选取适应度最高的公式树 A 作为父代，从中随机选择子树 A1 进行替换；然后在剩余公式树中找到适应度最高的公式树 B 作为捐赠者，从中随机选择子树 B1，并将其插入到公式树 A 中替换 A1 以形成后代。



图表 2 交叉

b) 子树变异

子树变异是一种激进的变异操作，父代公式树的子树可以被完全随机生成的子树所取代。这可以将已被淘汰的公式重新引入公式种群，以维持公式多样性。如图表 4 所示，子树变异选择适应度最高的公式树 A 作为父代，从中随机选择子树 A1 进行替换，然后随机生成用以替代的子树 B，并将其插入到公式树 A 中以形成后代。

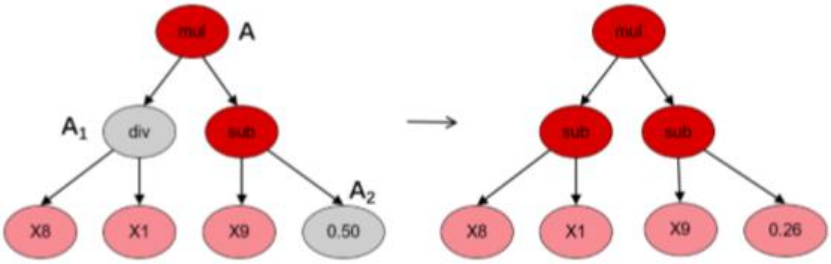


图

图表 3 子树变异

c) 点变异

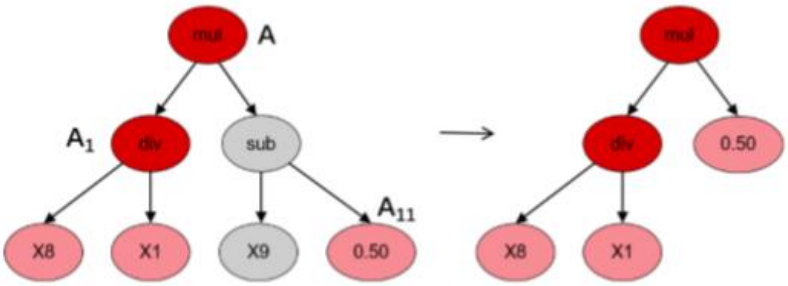
点变异是另一种常见的变异形式。与子树变异一样，它也可以将已淘汰的公式重新引入种群中以维持公式多样性。如图表 5 所示，点变异选取适应度高的父代公式树 A，并从中随机选择节点和叶子进行替换。叶子 A2 被其他叶子替换，并且某一节点 A1 上的公式被与其含有相同参数个数的公式所替换，以此形成后代。



图表 4 点变异

d) Hoist 变异

Hoist(提升)变异是一种对抗公式树过于复杂的方法。这种变异的目的是从公式树中移除部分叶子或者节点，以精简公式树。如图表 6 所示，Hoist 变异选取适应度高的父代公式树 A 并从中随机选择子树 A1。然后从该子树中随机选取子树 A11，并将其“提升”到原来子树 A1 的位置，以此形成后代。



图表 5 Hoist 变异

2.遗传规划选股因子挖掘测试流程

2.1 数据采集与处理

训练集数据范围： 2017.6.1_2018.6.1(包含牛熊)， 2017.1.1_2018.1.1(只有牛市)
股票池：沪深 300 成分股
数据字段：['open', 'high', 'low', 'avg', 'pre_close', 'close','volume']
预测目标： 个股未来 3 天的收益率，未来 20 天的收益率
回测区间：2020.04-2020.07

名称	定义
open, close, high, low, volume	个股日频开盘价、收盘价、最高价、最低价、成交量。
avg, pre_close,	个股单日价格平均、前一个交易日收盘价

图表 6 原始因子列表

2.2 gplearn 的处理

gplearn(https：//gplearn.readthedocs.io)是目前最成熟的 Python 遗传规划项目之一。gplearn 提供类似于 scikit-learn 的调用方式，并通过设置多个参数来完成特定功能。展示了 gplearn 的主要参数。

参数名称	定义和参数设置说明	参数设置
generations	公式进化的世代数量。世代数量越多,消耗算力越多，公式的进化次数越多	3
population_size	每一代公式群体中的公式数量。公式数量越大,消耗算力越多,公式之间组合的空间越大	1000
function_set	用于构建和进化公式时使用的函数集，可自定义更多函数	使用图表中的函数集
Init_depth	公式树的初始化深度。Init_deplh 是一个二元(min_depth,max_deph)，树的初始深度将处在[min_depth.max dept]区间内。设置树深度最小 1 层，最大 4 层，最大深度越深,可能得出越复杂的因子，但是因子的意义更难解释。	(1，4)
Toumament_size	进化到下一代的个体数目(从每一代的所有公式中，tournament_size 个公式会被随机选中，其中适应度最高的公式将被认定为生存竞争的胜利者，进入下一代。tournament_size 的大小与进化论中的选择压力息息相关：tournament_size 越小，选择压力越大，算法收敛的速度可能更快，但也有可能	20

	错过一些隐藏的优秀公式)。	
metric	适应度指标，可自定义更多指标	自定义的 RankIC 指标
parsimony_coefficient	惩罚系数，越大,约束越强	0.001
p_crossover	对胜者进行交叉的概率，用于合成新的树	0.4
p_subtree_mutation	控制胜者中进行子树变异的比例(优胜者的一棵子树将被另一棵完全随机的全新子树代替)所选值表示进行子树突变的部分。	0.01
p_hoist_mutation	控制进行 hoist 变异的比例，hoist 变异是一种对抗公式树膨胀（bloating，即过于复杂）的方法：从优胜者公式树内随机选择一个子树 A，再从 A 里随机选择一个子树 B，然后把 B 提升到 A 原来的位置，用 B 替代 A。hoist 的含义即「升高、提起」。	0
p_point_mutation	控制点进行突变的比例	0.01
p_point_replace	对于点突变时控制某些点突变的概率。	0.4

图表 7 gplearn 的主要参数

gplearn 提供了一套简洁、规范的遗传规划实现代码，但是不能直接运用于选股因子的挖掘。我们从源代码的层面，扩充了 gplearn 的函数集(function_set)，提供了更多特征计算方法，以提升其因子挖掘能力。除了 gplearn 提供的基础函数集(加、减、乘、除、开方、取对数、绝对值等)，我们还自定义了一些函数(包括多种时间序列运算函数，这是 gplearn 不支持的)，函数列表详细展示在下表中。

类型	名称	定义
	X：以下函数中自变量	X 一般可以理解为向量{X},代表 N 只个股在某指定截面日的因子值,例如：X=CLOSE+OPEN;若 X 为矩阵,则以下函数可以理解为对每个列向量分别进行运算,按列合并。
基础函数	add(X,Y)	返回值为向量,其中第 i 个元素为 X_i+Y_i
基础函数	sub(X,Y)	返回值为向量,其中第 i 个元素为 X_i-Y_i
基础函数	mul(X,Y)	返回值为向量,其中第 i 个元素为 X_i*Y_i(对应 matlab 中的点乘)
基础函数	div(X,Y)	返回值为向量,其中第 i 个元素为 X_i/Y_i(对应 matlab 中的点除)
基础函数	abs(X)	返回值为向量,其中第 i 个元素为 X_i 的绝对值
基础函数	sqrt(X)	返回值为向量,其中第 i 个元素为 abs(X_i)的开方
基础函数	log(X)	返回值为向量,其中第 i 个元素为 abs(X_i)的对数
基础函数	inv(X)	返回值为向量,其中第 i 个元素为 X_i 的倒数
自定义函数	_rolling_rank (X)	返回值为向量,其中第 i 个元素为 X_i 在向量 X_i 中的分位数。
自定义函数	_rolling_prod(X)	返回值为向量, X_i 的连乘。
自定义函数	_corr (X,Y,n)	返回值为向量,其中第 i 个元素为过去 n 天 X_i 值构成的时序数列和 Y_i 值构成的时序数列的相关系数。

自定义函数	_square (X)	返回值为向量，X_i 的平方。
自定义函数	_cube (X)	返回值为向量，X_i 的立方。
自定义函数	_delta (X)	返回值为向量，X_i 的一阶差分。
自定义函数	_delay (X)	返回值为向量，X_i 的想下平移一个单位。
自定义函数	ts_min(X,d)	返回值为向量,其中第 i 个元素为过去 d 天 X_i 值构成的时序数列中最小值。
自定义函数	ts_max(X,d)	返回值为向量,其中第 i 个元素为过去 d 天 X_i 值构成的时序数列中最大值。
自定义函数	ts_argmin(X,d)	返回值为向量,其中第 i 个元素为过去 d 天 X_i 值构成的时序数列中最小值出现的位置。
自定义函数	ts_argmax(X,d)	返回值为向量,其中第 i 个元素为过去 d 天 X_i 值构成的时序数列中最大值出现的位置。
自定义函数	ts_rank(X,d)	返回值为向量,其中第 i 个元素为过去 d 天 X_i 值构成的时序数列中本截面日 X_i 值所处分位数。
自定义函数	ts_sum(X,d)	返回值为向量,其中第 i 个元素为过去 d 天 X_i 值构成的时序数列之和。
自定义函数	ts_product(X,d)	返回值为向量,其中第 i 个元素为过去 d 天 X_i 值构成的时序数列的连乘乘积。
自定义函数	ts_stddev(X,d)	返回值为向量,其中第 i 个元素为过去 d 天 X_i 值构成的时序数列的标准差。

图表 8 函数列表

2.3 基于遗传规划的因子挖掘

1)使用图表 7 中的因子和图表 8 中的函数集，生成大量公式，并按照图表 1 的流程进行公式的进化和筛选。

2)公式适应度的计算：得出该公式在截面 t 上对所有个股因子向量Ft后，我们会对因子进行标准化：将经过以上处理后的因子暴露度序列减去其现在的均值、除以其标准差，得到一个新的近似服从 N(0,1)分布的序列。经过以上处理后，计算处理后因子在每个截面上与 20 个交易日后收益率的 Rank IC，取 Rank IC 均值为公式 F 的适应度。

2.4 遗传规划的测试结果分析

- 测试集数据范围： 2020.04-2020.07
- 股票池：沪深 300 成分股
- 数据字段：['open', 'high', 'low', 'avg', 'pre_close', 'close', 'volume']

针对训练集数据区间和未来收益率不同区间，我们得出四个基本模型并对其中适应度和复杂度较好的因子进行分析。

a) 模型 1：牛熊两市+未来 3 日收益率

	fitness	expression	depth	length
alpha_1	0.0314829	log(stddev(sma(low)))	3	4
alpha_3	0.0263626	stddev(stddev(sma(close)))	3	4
alpha_6	0.0238864	log(stddev(pre_close))	2	3
alpha_2	0.0214272	log(stddev(stddev(sma(close))))	4	5
alpha_4	0.0211335	stddev(stddev(stddev(close)))	3	4
alpha_9	0.0164814	sma(low)	1	2
alpha_5	0.0111292	stddev(sma(ts_max(stddev(pre_close))))	4	5
alpha_8	0.00903541	stddev(ts_max(stddev(pre_close)))	3	4
alpha_7	0.00294866	log(stddev(ts_max(stddev(pre_close))))	4	5
alpha_10	-0.0387033	log(sin(ts_min(close)))	3	4

- **alpha_6 因子分析　　log(*stddev(pre_close)*)**

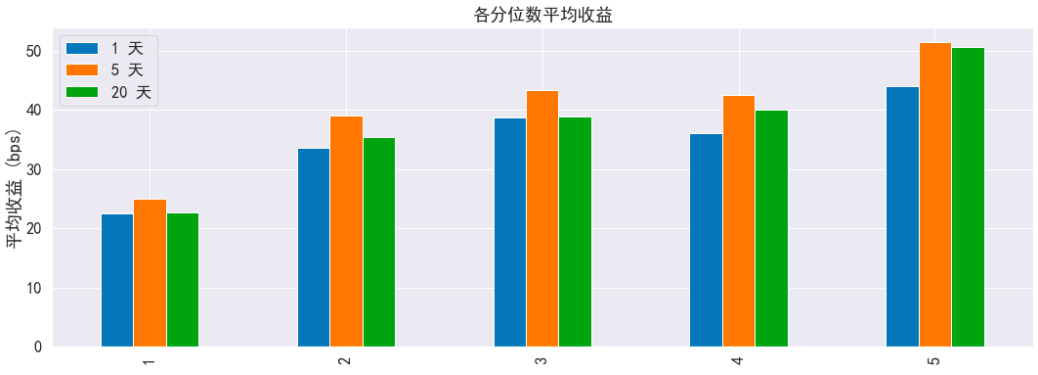
收益分析

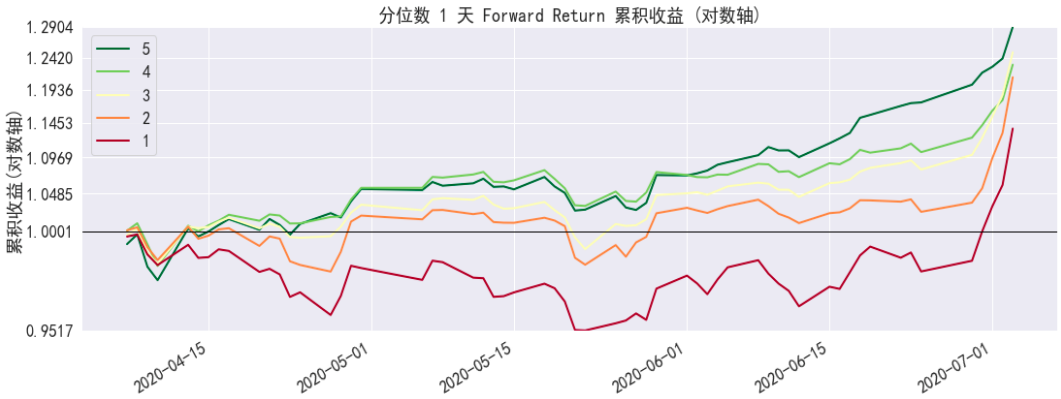
	period_1	period_5	period_20
Ann. alpha	0.265	0.454	0.389
beta	0.212	0.080	0.130
Mean Period Wise Return Top Quantile (bps)	44.019	51.361	50.595
Mean Period Wise Return Bottom Quantile (bps)	22.539	24.944	22.713
Mean Period Wise Spread (bps)	21.480	26.847	27.969

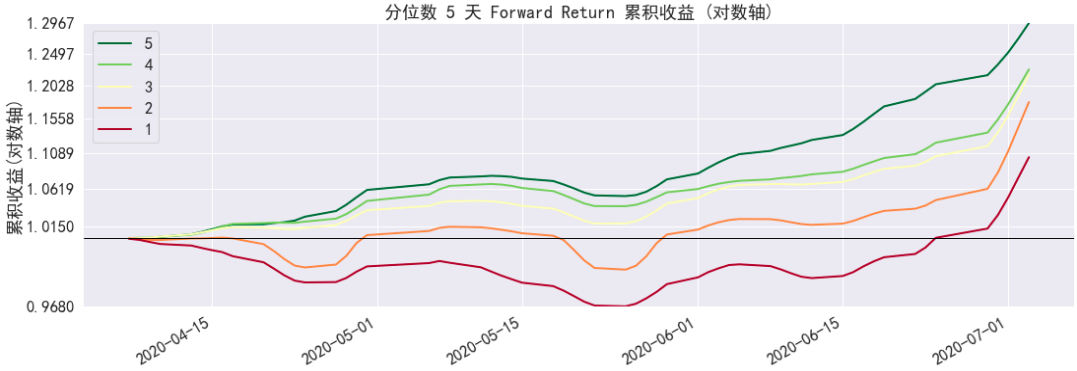
	period_1	period_5	period_20
IC Mean	0.028	0.105	0.217
IC Std.	0.190	0.175	0.128
IR	0.147	0.600	1.705
t-stat(IC)	1.130	4.610	13.099
p-value(IC)	0.263	0.000	0.000
IC Skew	-0.255	-0.960	0.352
IC Kurtosis	-0.482	0.087	-0.882

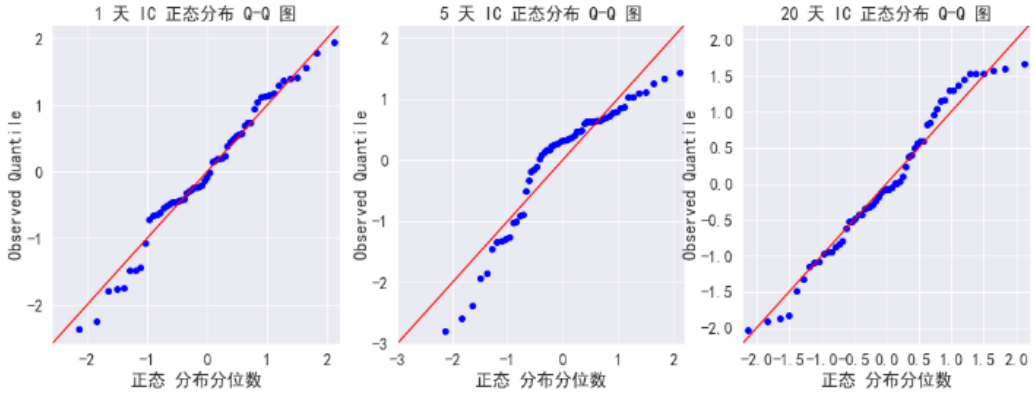
	period_1	period_20	period_5
Quantile 1 Mean Turnover	0.042	0.138	0.121
Quantile 2 Mean Turnover	0.089	0.282	0.262
Quantile 3 Mean Turnover	0.107	0.374	0.323
Quantile 4 Mean Turnover	0.113	0.398	0.329
Quantile 5 Mean Turnover	0.053	0.176	0.151

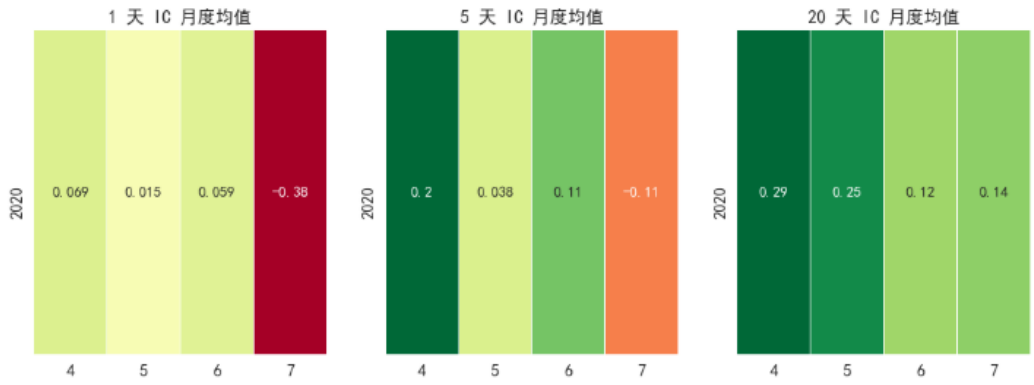
	period_1	period_5	period_20
Mean Factor Rank Autocorrelation	0.996	0.97	0.961


各分位数平均收益
1 天
5 天
20 天
平均收益（bps）
50
40
30
20
10
0
1
2
3
4
5


分位数 1 天 Forward Return 累积收益（对数轴）
5
4
3
2
1
累积收益（对数轴）
1.2904
1.2420
1.1936
1.1453
1.0969
1.0485
1.0001
0.9517
2020-04-15
2020-05-01
2020-05-15
2020-06-01
2020-06-15
2020-07-01


分位数 5 天 Forward Return 累积收益（对数轴）
5
4
3
2
1
累积收益（对数轴）
1.2967
1.2497
1.2028
1.1558
1.1089
1.0619
1.0150
0.9680
2020-04-15
2020-05-01
2020-05-15
2020-06-01
2020-06-15
2020-07-01


1 天 IC 正态分布 Q-Q 图
5 天 IC 正态分布 Q-Q 图
20 天 IC 正态分布 Q-Q 图
Observed Quantile
正态 分布分位数



b) 模型 2：牛熊两市+未来 3 日收益率

	depth	expression	fitness	length
alpha_3	1	stddev(low, 20)	0.0785097	3
alpha_4	1	stddev(close, 20)	0.0770966	3
alpha_5	1	stddev(open, 20)	0.0770124	3
alpha_7	1	stddev(pre_close, 20)	0.0764108	3
alpha_9	1	stddev(high, 20)	0.0737471	3
alpha_10	1	stddev(high, 20)	0.0737471	3
alpha_1	2	stddev(stddev(high, 20), 20)	0.066714	5
alpha_6	2	cube(stddev(avg, 20))	0.0666987	4
alpha_8	2	sqrt(stddev(pre_close, 20))	0.0664108	4
alpha_2	2	stddev(stddev(high, 20), 15)	0.0649213	5

- **alpha_8 因子分析　sqrt(stddev(pre_close，20))**

收益分析

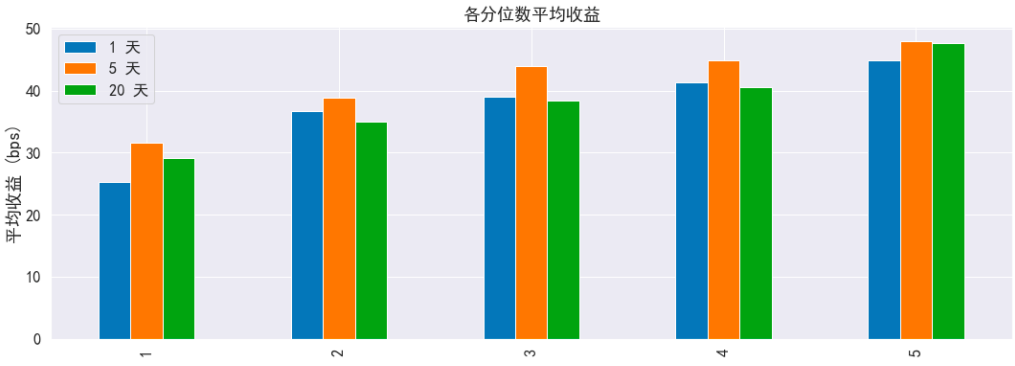
	period_1	period_5	period_20
Ann. alpha	0.370	0.434	0.435
beta	0.813	0.692	0.816
Mean Period Wise Return Top Quantile (bps)	44.942	47.889	47.579
Mean Period Wise Return Bottom Quantile (bps)	25.353	31.672	29.193
Mean Period Wise Spread (bps)	19.588	16.630	18.752

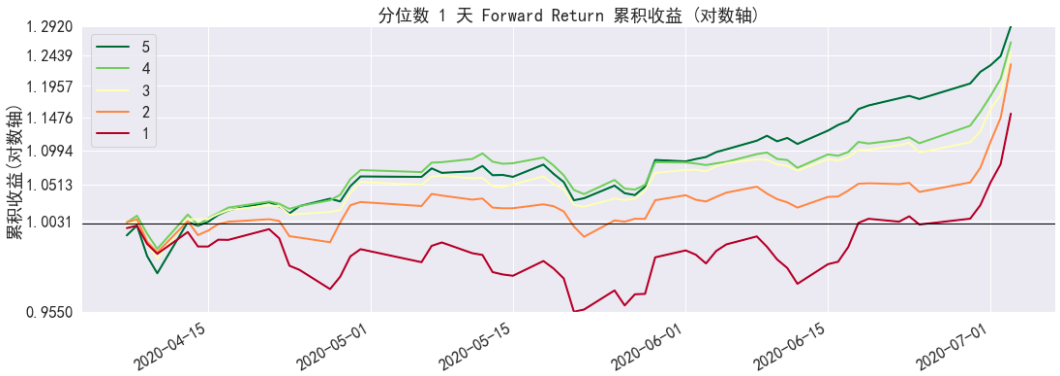
	period_1	period_5	period_20
IC Mean	0.026	0.074	0.165
IC Std.	0.152	0.153	0.143
IR	0.173	0.483	1.158
t-stat(IC)	1.320	3.677	8.816
p-value(IC)	0.192	0.001	0.000
IC Skew	-0.295	-0.743	0.583
IC Kurtosis	-0.539	-0.306	-0.617

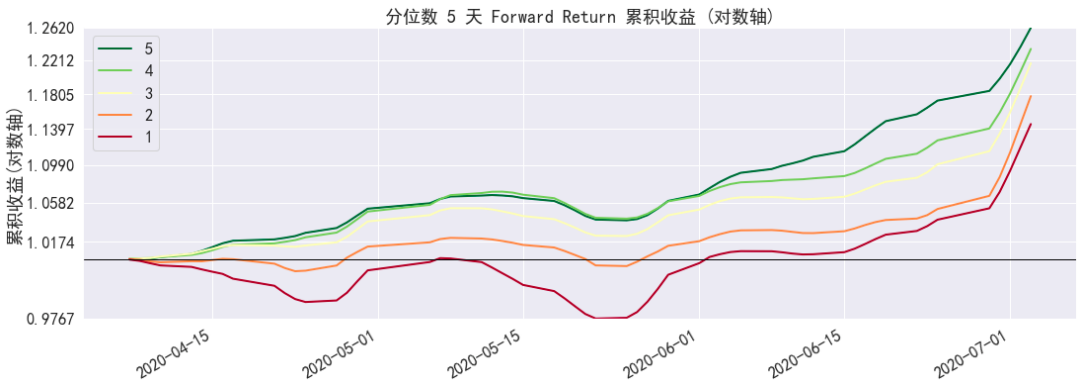
换手率分析

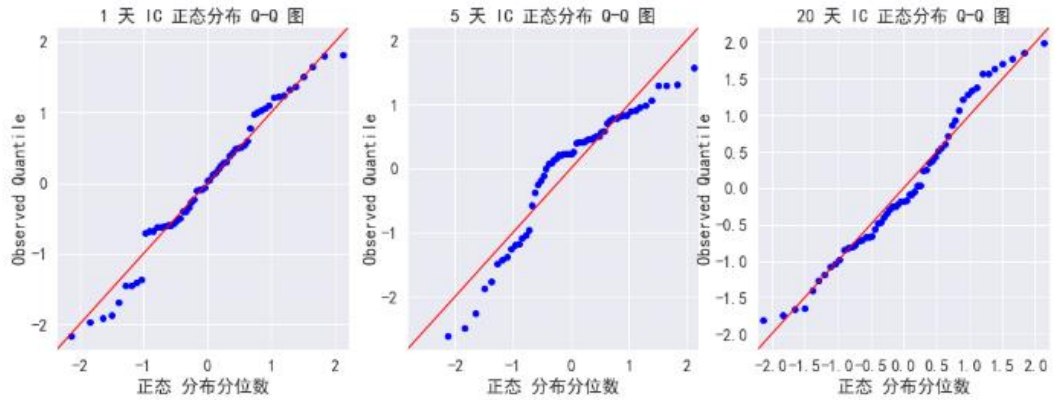
	period_1	period_20	period_5
Quantile 1 Mean Turnover	0.229	0.494	0.497
Quantile 2 Mean Turnover	0.251	0.589	0.550
Quantile 3 Mean Turnover	0.243	0.597	0.553
Quantile 4 Mean Turnover	0.218	0.585	0.537
Quantile 5 Mean Turnover	0.094	0.354	0.354

	period_1	period_5	period_20
Mean Factor Rank Autocorrelation	0.907	0.673	0.696

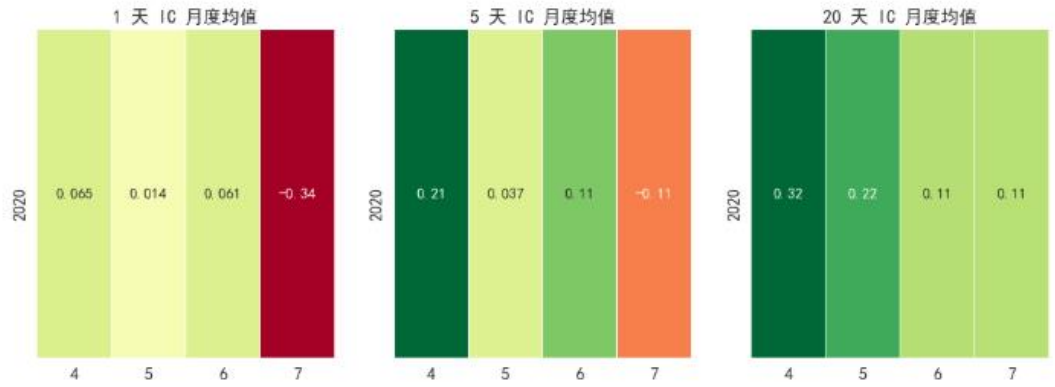








<Figure size 432x288 with 0 Axes>



c) 模型 3：牛市+未来 3 日收益率

	depth	expression	fitness	length
alpha_2	1	ts_rank(pre_close)	0.0545167	2
alpha_1	2	ts_rank(ts_rank(pre_close))	0.0543059	3
alpha_3	2	max(ts_rank(low), ts_rank(pre_close))	0.0469682	5
alpha_4	3	ts_rank(max(ts_rank(low), ts_rank(pre_close)))	0.0444154	6
alpha_5	3	mul(ts_rank(scale(close)), 0.671)	0.0434614	5
alpha_6	2	max(ts_rank(low), ts_rank(low))	0.0422986	5
alpha_7	2	ts_rank(ts_rank(low))	0.0379246	3
alpha_8	1	ts_rank(volume)	0.0166262	2
alpha_9	2	delay(add(-0.807, open))	0.00500159	4
alpha_10	2	sin(ts_min(close))	0.00112382	3

- **alpha_9 因子分析** ***Delay(add(-0.807,open))***

	period_1	period_5	period_20
Ann. alpha	0.440	0.608	0.480
beta	0.842	0.749	0.820
Mean Period Wise Return Top Quantile (bps)	45.035	50.723	47.219
Mean Period Wise Return Bottom Quantile (bps)	21.827	25.220	20.881
Mean Period Wise Spread (bps)	23.208	26.286	26.687

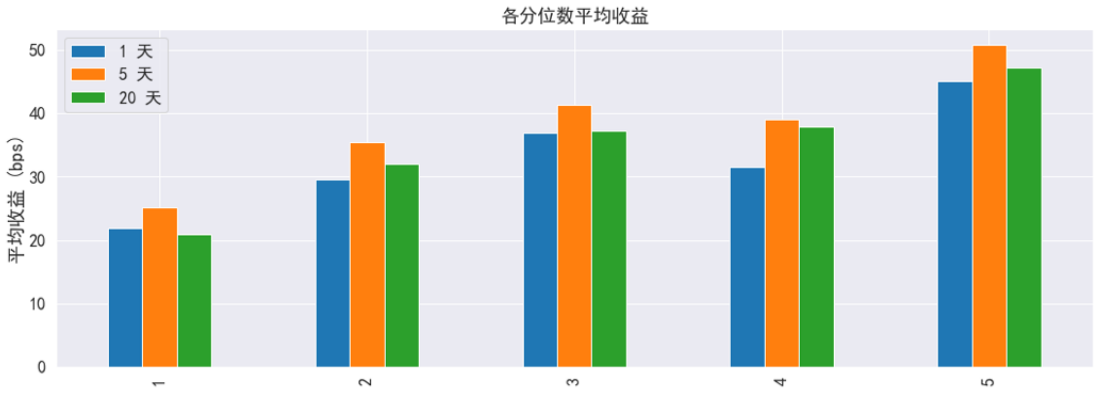
IC 分析

	period_1	period_5	period_20
IC Mean	0.038	0.117	0.222
IC Std.	0.174	0.170	0.130
IR	0.218	0.687	1.708
t-stat(IC)	1.784	5.623	13.983
p-value(IC)	0.079	0.000	0.000
IC Skew	-0.590	-1.061	0.130
IC Kurtosis	0.096	0.737	-1.201

换手率分析

	period_1	period_20	period_5
Quantile 1 Mean Turnover	0.004	0.014	0.009
Quantile 2 Mean Turnover	0.012	0.048	0.029
Quantile 3 Mean Turnover	0.022	0.093	0.048
Quantile 4 Mean Turnover	0.020	0.080	0.042
Quantile 5 Mean Turnover	0.007	0.023	0.014

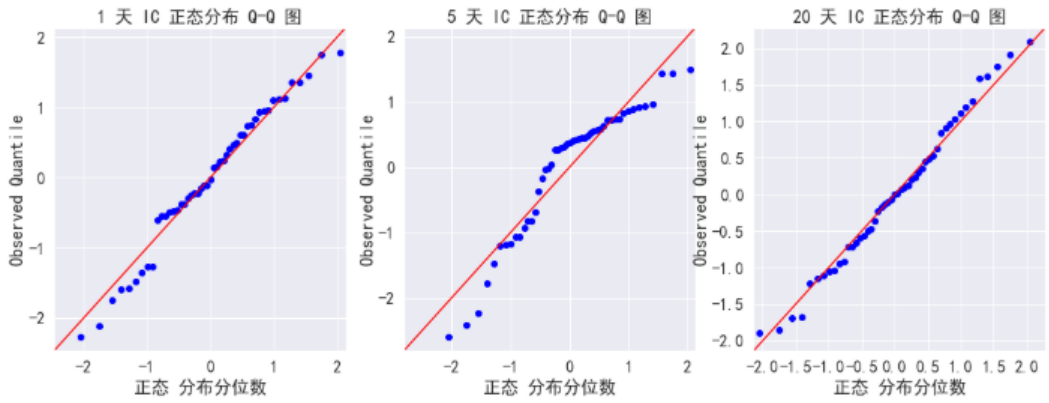
	period_1	period_5	period_20
Mean Factor Rank Autocorrelation	1.0	1.0	0.999

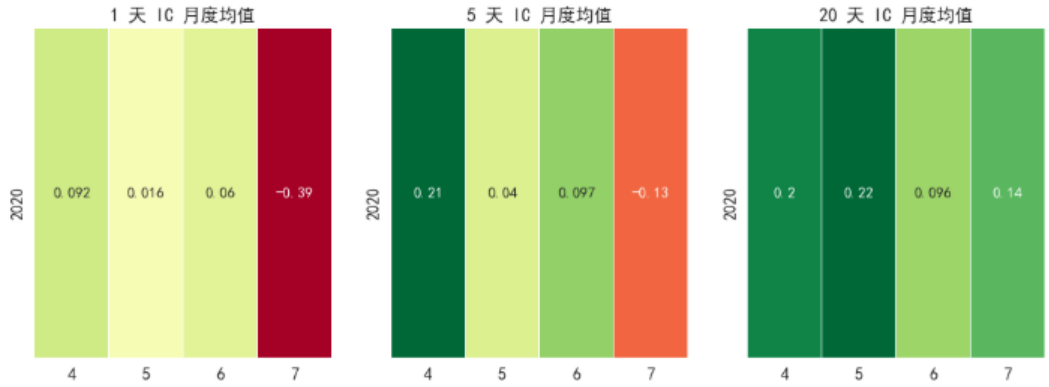

各分位数平均收益
1 天
5 天
20 天
平均收益（bps）
0
10
20
30
40
50
1
2
3
4
5


分位数 1 天 Forward Return 累积收益（对数轴）
累积收益（对数轴）
1.3450
1.2891
1.2331
1.1772
1.1213
1.0653
1.0094
0.9534
5
4
3
2
1
2020-04-01
2020-04-15
2020-05-01
2020-05-15
2020-06-01
2020-06-15
2020-07-01


分位数 5 天 Forward Return 累积收益（对数轴）
累积收益（对数轴）
1.3454
1.2913
1.2371
1.1830
1.1288
1.0747
1.0205
0.9664
5
4
3
2
1
2020-04-01
2020-04-15
2020-05-01
2020-05-15
2020-06-01
2020-06-15
2020-07-01

、


1 天 IC 正态分布 Q-Q 图
5 天 IC 正态分布 Q-Q 图
20 天 IC 正态分布 Q-Q 图
Observed Quantile
正态 分布分位数



d) 模型 4：牛市+未来 20 日收益率

	depth	expression	fitness	length
alpha_1	2	ts_max(max(volume, 0.225))	0.0485949	4
alpha_2	2	ts_sum(scale(volume))	0.0467758	3
alpha_3	1	ts_sum(volume)	0.0506937	2
alpha_4	2	ts_sum(ts_sum(volume))	0.045971	3
alpha_5	1	scale(volume)	0.0471454	2
alpha_6	1	delay(volume)	0.0459509	2
alpha_7	2	delay(add(-0.807, open))	0.00644231	4
alpha_8	1	square(high)	0.0141421	2
alpha_9	1	ts_max(0.225)	-0.00461432	2
alpha_10	2	sin(ts_min(close))	-0.00953748	3

- **alpha_2 因子分析** ***ts_sum(scale (volume))***

收益分析

	period_1	period_5	period_20
Ann. alpha	0.508	0.756	0.466
beta	0.754	0.667	0.868
Mean Period Wise Return Top Quantile (bps)	54.548	59.554	53.762
Mean Period Wise Return Bottom Quantile (bps)	34.728	38.882	35.088
Mean Period Wise Spread (bps)	19.820	21.489	19.198

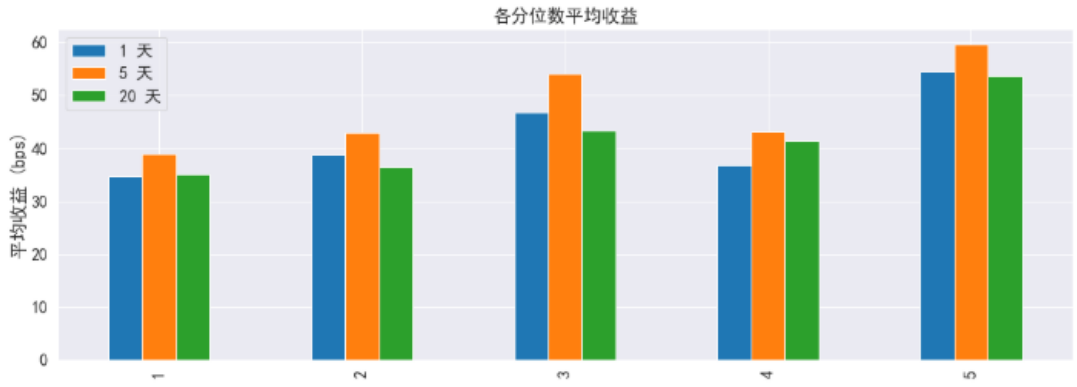
IC 分析

	period_1	period_5	period_20
IC Mean	0.021	0.081	0.162
IC Std.	0.205	0.192	0.110
IR	0.104	0.421	1.473
t-stat(IC)	0.725	2.950	10.310
p-value(IC)	0.472	0.005	0.000
IC Skew	-0.348	-0.915	0.121
IC Kurtosis	-0.527	0.058	-0.635

换手率分析

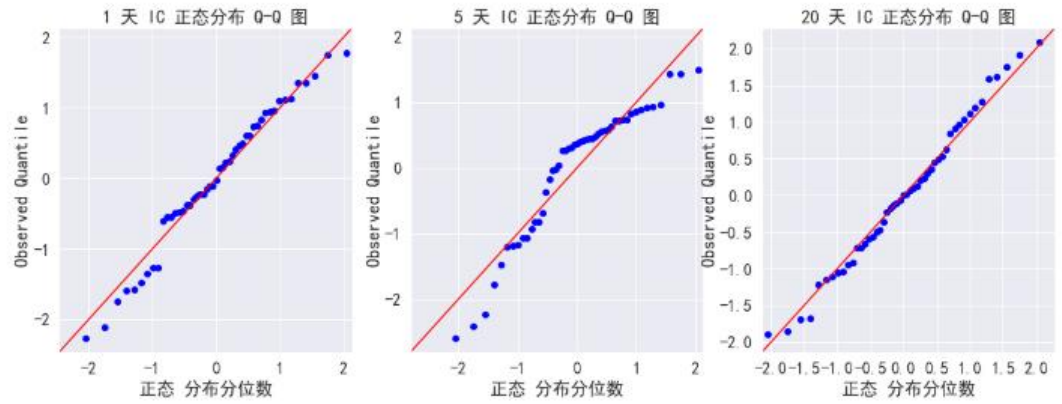
	period_1	period_20	period_5
Quantile 1 Mean Turnover	0.024	0.185	0.088
Quantile 2 Mean Turnover	0.047	0.360	0.175
Quantile 3 Mean Turnover	0.052	0.437	0.199
Quantile 4 Mean Turnover	0.056	0.442	0.219
Quantile 5 Mean Turnover	0.026	0.216	0.109

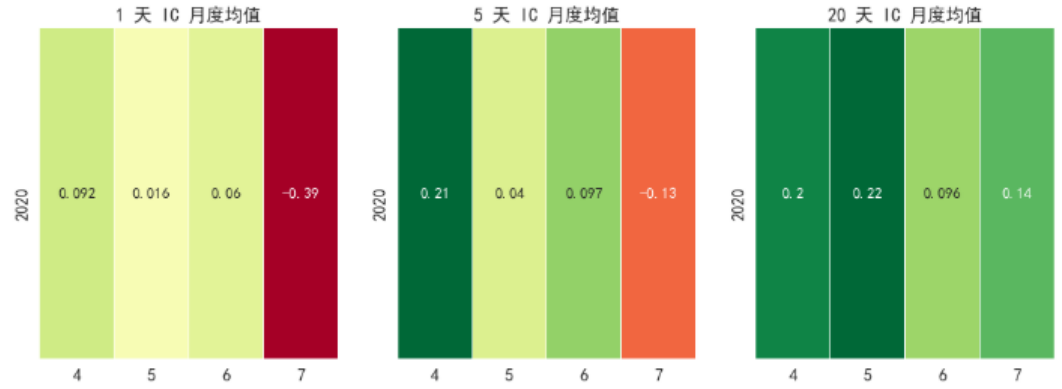
	period_1	period_5	period_20
Mean Factor Rank Autocorrelation	0.999	0.984	0.944




分位数 1 天 Forward Return 累积收益（对数轴）
累积收益(对数轴)
1.3010
1.2541
1.2071
1.1601
1.1131
1.0662
1.0192
0.9722
5
4
3
2
1
2020-05-01
2020-05-15
2020-06-01
2020-06-15
2020-07-01


分位数 5 天 Forward Return 累积收益（对数轴）
累积收益(对数轴)
1.2819
1.2394
1.1969
1.1544
1.1120
1.0695
1.0270
0.9845
5
4
3
2
1
2020-05-01
2020-05-15
2020-06-01
2020-06-15
2020-07-01


1 天 IC 正态分布 Q-Q 图
5 天 IC 正态分布 Q-Q 图
20 天 IC 正态分布 Q-Q 图
Observed Quantile
正态 分布分位数


1 天 IC 月度均值
5 天 IC 月度均值
20 天 IC 月度均值
2020
0.092
0.016
0.06
-0.39
0.21
0.04
0.097
-0.13
0.2
0.22
0.096
0.14
4
5
6
7

3.分工、总结与展望

3.1 项目学习总结

1. 了解了遗传规划的基本原理及操作。
2. 学习了 Python 中 gplearn 库中对遗传规划的使用，各种参数的定义。
3. 实验：选取了沪深 300 成分股，在牛市和牛熊两市时间段下的数据，并依据 3 日，20 日收益率作为标签进行因子挖掘。
4. 单因子检验：回顾单因子分析流程，探索因子公式的表现。

3.2 困难与解决对策

1. 聚宽上无法使用 gplearn —— 跨平台使用本地 jupyter 与聚宽平台
2. 时间线太长，噪声太多 —— 取特定的牛熊时间段训练
3. 函数太少缺少灵活性 —— 改写了部分函数使得参数可变
4. 因子长度太长 —— 加惩罚因子参数，约束公式深度，防止过拟合
5. 无法批量生成因子 —— 研究 gplearn 参数文档，找到对应方法

3.3 仍待提升的部分

1. 因子预处理： 无法在本地用聚宽的中性化函数，因子在剔除行业、市值等方面影响后是否有稳定的 RankIC 仍待验证。
2. 因子分析：较为依赖聚宽平台的因子分析模板，可将单因子测试的过程引入 gplearn，对待挖掘因子进行传统风格因子中性化处理。
3. 数据量及速度：当数据量变大时，可引入并行运算技术，加快因子矩阵的运算速度。

3.4 组内分工

周一飞:统筹规划、基础代码结构实现及优化、因子挖掘代码优化、因子分析代码优化，调整参数训练模型和因子测试、汇报进展；

刘晓: 基础代码结构实现及优化、因子挖掘代码优化、因子结果分析、PPT 制作与展示、汇报进展；

马毓婕：资料搜集、基础代码结构实现、Word 报告；

姜泓任：基础代码结构实现、代码测试；

附录（源代码）：

1. 【数据获取模块】聚宽量化交易平台运行

```
import numpy as np
import pandas as pd
import pickle
# 沪深 300 股价及 1 日收益率数据  总表
train_start_date = '2017-01-26'
train_end_date = '2018-01-26'
fields = ['open', 'high', 'low', 'avg', 'pre_close', 'close','volume']
index = '000300.XSHG'

stock_list = get_index_stocks(index, date=train_end_date)

train_total_data = pd.DataFrame(columns=fields+['code','pct','predict_pct'])

for stock in stock_list :
    stock_price = get_price(stock, start_date=train_start_date, end_date=train_end_date, fq='post', fields=fields, panel=False)
    stock_price['code'] = stock

    #3 日收益率及预测
    period=3
    stock_price['pct'] = stock_price['close'].pct_change(periods=period)
    stock_price['predict_pct']= stock_price['pct'].shift(-1*period)


    stock_price = stock_price.dropna(axis=0,subset = ['predict_pct'])
    train_total_data = pd.concat([train_total_data,stock_price], axis=0, join='outer') #将不同股票的特征与收益率预测值拼接
with open('train_2017.1_2018.1_pre3_niu.pkl', 'wb') as f:
    pickle.dump(train_total_data, f)
train_total_data
```

2. 【遗传算法训练模块】桌面端或云端 Jupyter 运行

```
import numpy as np
import pandas as pd
from scipy.stats import rankdata
import pickle

from gplearn import genetic
from gplearn.functions import make_function
from gplearn.genetic import SymbolicTransformer, SymbolicRegressor
from gplearn.fitness import make_fitness

from sklearn.utils import check_random_state
from sklearn.model_selection import train_test_split

with open('train_2017.6_2018.6_pre3.pkl', 'rb') as train_f:
    train_data = pickle.load(train_f)
# with open('/home/kesci/test_total_data.pkl', 'rb') as test_f:
#     test_data = pickle.load(test_f)

fields = ['open', 'high', 'low', 'avg', 'pre_close', 'close','volume']
train_data['3'] = 3
train_data['5'] = 5
train_data['6']=6
train_data['8'] = 8
train_data['10'] = 10
train_data['12'] = 12
train_data['15'] = 15
train_data['20']=20
fields = fields + ['3','5','6','8','10','12','15','20']
# 训练数据
X_train = train_data[fields].values
y_train = train_data['predict_pct'].values



# import numba
# # 测试数据
# X_test = test_data[fields].values
# y_test = test_data['predict_pct'].values

# 系统自带的函数群

"""
```

```
Available individual functions are:

  ‘add’ : addition, arity=2.
  ‘sub’ : subtraction, arity=2.
  ‘mul’ : multiplication, arity=2.
  ‘div’ : protected division where a denominator near-zero returns 1., arity=2.
  ‘sqrt’ : protected square root where the absolute value of the argument is used, arity=1.
  ‘log’ : protected log where the absolute value of the argument is used and a near-zero argument returns 0., arity=1.
  ‘abs’ : absolute value, arity=1.
  ‘neg’ : negative, arity=1.
  ‘inv’ : protected inverse where a near-zero argument returns 0., arity=1.
  ‘max’ : maximum, arity=2.
  ‘min’ : minimum, arity=2.
  ‘sin’ : sine (radians), arity=1.
  ‘cos’ : cosine (radians), arity=1.
  ‘tan’ : tangent (radians), arity=1.
"""

# init_function = ['add', 'sub', 'mul', 'div', 'sqrt', 'log', 'abs', 'neg', 'inv', 'max', 'min', 'sin', 'cos', 'tan']
init_function = ['add', 'sub', 'mul', 'div','sqrt', 'log','inv','sin','max','min']

# 自定义函数, make_function 函数群

def _rolling_rank(data): #第 i 个元素为Xi在向量 X 中的分位数
    value = rankdata(data)[-1]
    return value    #scipy.rankdata 排序

def _rolling_prod(data):
    return np.prod(data) #所有元素乘积


def _delta(data):
    value = np.diff(data.flatten())
    value = np.append(0, value)
    return value

def _delay(data): # d 天以前的 X 值
    period=1   #当 period 为正时，默认是 axis = 0 轴的设定，向下移动
    value = pd.Series(data.flatten()).shift(period)
    value = np.nan_to_num(value)
    return value
```

```
# @numba.jit
def _ts_sum(data,n):

    with np.errstate(divide='ignore', invalid='ignore'):

        try:
            if n[0] == n[1] and n[1] ==n[2]:
                window    = n[0]

                value = np.array(pd.Series(data.flatten()).rolling(window).sum().tolist())
                value = np.nan_to_num(value)

                return value
            else:
                return np.zeros(data.shape[0])

        except:
            return np.zeros(data.shape[0])


# @numba.jit
def _sma(data,n):
    with np.errstate(divide='ignore', invalid='ignore'):

        try:
            if n[0] == n[1] and n[1] ==n[2]:
                window    = n[0]

                value = np.array(pd.Series(data.flatten()).rolling(window).mean().tolist())
                value = np.nan_to_num(value)

                return value
            else:
                return np.zeros(data.shape[0])

        except:
            return np.zeros(data.shape[0])
# @numba.jit
def _stddev(data,n):
    with np.errstate(divide='ignore', invalid='ignore'):

        try:
            if n[0] == n[1] and n[1] ==n[2]:
                window    = int(np.mean(n))
```

```
                    value = np.array(pd.Series(data.flatten()).rolling(window).std().tolist())
                    value = np.nan_to_num(value)

                    return value
                else:
                    return np.zeros(data.shape[0])

            except:
                return np.zeros(data.shape[0])
# @numba.jit
def _ts_rank(data,n):

    with np.errstate(divide='ignore', invalid='ignore'):

        try:
            if n[0] == n[1] and n[1] ==n[2]:
                value                                                                    =
np.array(pd.Series(data.flatten()).rolling(window).apply(_rolling_rank).tolist())
                value = np.nan_to_num(value)

                return value
            else:
                return np.zeros(data.shape[0])
        except:
            return np.zeros(data.shape[0])


ts_rank = make_function(function=_ts_rank, name='ts_rank', arity=2)

# @numba.jit

def _ts_argmin(data,n):

    try:
        if n[0] == n[1] and n[1] ==n[2]:
            window=n[0]
            value = pd.Series(data.flatten()).rolling(window).apply(np.argmin) + 1
            value = np.nan_to_num(value)
            return value
        else:
            return np.zeros(data.shape[0])
    except:
        return np.zeros(data.shape[0])
```

```
# @numba.jit
def _ts_argmax(data,n):
    with np.errstate(divide='ignore', invalid='ignore'):
        try:
            if n[0] == n[1] and n[1] ==n[2]:
                window=n[0]
                value = pd.Series(data.flatten()).rolling(window).apply(np.argmax) + 1
                value = np.nan_to_num(value)
                return value
            else:
                return np.zeros(data.shape[0])
        except:
            return np.zeros(data.shape[0])
# @numba.jit
def _ts_min(data,n):
    with np.errstate(divide='ignore', invalid='ignore'):

        try:
            if n[0] == n[1] and n[1] ==n[2]:
                window    = n[0]
                #window    = int(np.mean(n))
                value = np.array(pd.Series(data.flatten()).rolling(window).min().tolist())
                value = np.nan_to_num(value)


                return value
            else:
                return np.zeros(data.shape[0])

        except:
            return np.zeros(data.shape[0])
# @numba.jit
def _ts_max(data,n):
    with np.errstate(divide='ignore', invalid='ignore'):

        try:
            if n[0] == n[1] and n[1] ==n[2]:
                window    = n[0]

                value = np.array(pd.Series(data.flatten()).rolling(window).max().tolist())
                value = np.nan_to_num(value)

                return value
            else:
```

```
                return np.zeros(data.shape[0])


        except:
            return np.zeros(data.shape[0])

def _ts_argmaxmin(data,n):
    return _ts_argmax(data,n) - _ts_argmin(data,n)
##########################################
def _cube(data):
    return np.square(data)*data

def _square(data):
    return np.square(data)
# @numba.jit
def _corr(data1,data2,n):

    with np.errstate(divide='ignore', invalid='ignore'):


        try:
            if n[0] == n[1] and n[1] ==n[2]:
                window    = n[0]

                x1 = pd.Series(data1.flatten())
                x2 = pd.Series(data2.flatten())

                df = pd.concat([x1,x2],axis=1)
                temp = pd.Series()
                for i in range(len(df)):
                    if i<=window-2:
                        temp[str(i)] = np.nan
                    else:
                        df2 = df.iloc[i-window+1:i,:]
                        temp[str(i)] = df2.corr('spearman').iloc[1,0]
                return np.nan_to_num(temp)
            else:
                return np.zeros(data1.shape[0])

        except:
            return np.zeros(data1.shape[0])
# @numba.jit
def _stddev(data,n):
    with np.errstate(divide='ignore', invalid='ignore'):
```

```
        try:
            if n[0] == n[1] and n[1] ==n[2]:
                window    = int(np.mean(n))

                value = np.array(pd.Series(data.flatten()).rolling(window).std().tolist())
                value = np.nan_to_num(value)

                return value
            else:
                return np.zeros(data.shape[0])

        except:
            return np.zeros(data.shape[0])

delta = make_function(function=_delta, name='delta', arity=1)
delay = make_function(function=_delay, name='delay', arity=1)
rank = make_function(function=_rank, name='rank', arity=1)
scale = make_function(function=_scale, name='scale', arity=1)
sma = make_function(function=_sma, name='sma', arity=2)
stddev = make_function(function=_stddev, name='stddev', arity=2)
product = make_function(function=_product, name='product', arity=1)
ts_rank = make_function(function=_ts_rank, name='ts_rank', arity=2)
ts_min = make_function(function=_ts_min, name='ts_min', arity=2)
ts_max = make_function(function=_ts_max, name='ts_max', arity=2)
ts_argmax = make_function(function=_ts_argmax, name='ts_argmax', arity=2)
ts_argmin = make_function(function=_ts_argmin, name='ts_argmin', arity=2)
ts_sum = make_function(function=_ts_sum, name='ts_sum', arity=2)

cube = make_function(function=_cube, name='cube', arity=1)
square = make_function(function=_square, name='square', arity=1)


corr = make_function(function=_corr, name='corr', arity=3)#

user_function = [delta, delay, rank, scale, sma,
                 stddev, product, ts_rank, ts_min,
                 ts_max, ts_argmax, ts_argmin, ts_sum,
                 cube, square, stddev, corr]

# import numba
# @numba

generations = 3
```

```
function_set = init_function + user_function
#metric = my_metric
init_depth=(1,2) #最初生成树的深度(min_depth, max_depth)
population_size = 500
random_state=0
tournament_size=100
est_gp = SymbolicTransformer(
                                    feature_names=fields,
                                    function_set=function_set,
                                    generations=generations,
                                    metric='spearman',     #'spearman'秩相关系数
                                    parsimony_coefficient=0.01,#惩罚 节俭系数(越大,约束
越强,默认 0.001)
                                    init_depth=init_depth, # 公式树的初始化深度
                                    population_size=population_size,
                                    tournament_size= tournament_size,
                                    random_state=random_state,
                                    p_crossover = 0.4,
                                    p_subtree_mutation = 0.01,
                                    p_hoist_mutation = 0.01,
                                    p_point_mutation = 0.2,
                                    p_point_replace = 0.5,
                                )

est_gp.fit(X_train, y_train)

# 获取较优的表达式

best_programs = est_gp._best_programs
best_programs_dict = {}
for p in best_programs:
    factor_name = 'alpha_' + str(best_programs.index(p) + 1)
    best_programs_dict[factor_name]     =     {'fitness':p.fitness_,     'expression':str(p),
'depth':p.depth_, 'length':p.length_}

best_programs_dict = pd.DataFrame(best_programs_dict).T

# best_programs_dict = best_programs_dict.sort_values(by='fitness',ascending=False)
best_programs_dict

# # 将模型保存到本地
with open('gp_model_2017.6_2018.6_pre20.pkl', 'wb') as f:
    pickle.dump(est_gp, f)
```

3. 【遗传因子测试模块】 在桌面端或云端 jupyter 运行

```
import numpy as np
import pandas as pd
from scipy.stats import rankdata
import pickle


from sklearn.utils import check_random_state
from sklearn.model_selection import train_test_split

from    jqdatasdk import *
import jqfactor_analyzer as ja


auth('账号','密码')

import warnings
warnings.filterwarnings("ignore")

# 设置数据
fields = ['open', 'high', 'low', 'avg', 'pre_close', 'close','volume']
index = '000300.XSHG'

# 导入模型
with open('gp_model_pre3_niu.pkl', 'rb') as f:
        est_gp = pickle.load(f)

# 设置样本外时间

# 设置起止时间
new_start_date = '2020-03-24'
new_end_date = '2020-07-24'
```

```
# 设置调仓周期
periods=(1, 5, 20)
# 设置分层数量
quantiles=5

# 设置股票池
securities = get_index_stocks('000300.XSHG' , date=new_end_date)




# 获取需要分析的数据,并计算相应的因子暴露

factor_dict = {}
df = get_price('000300.XSHG', start_date=new_start_date, end_date=new_end_date, fq='post',
fields=fields)
index = df.index
columns                                                                                    =
['alpha1','alpha2','alpha3','alpha4','alpha5','alpha6','alpha7','alpha8','alpha9','alpha10']

for security in securities:
    s_price = get_price(security, start_date=new_start_date, end_date=new_end_date,
fq='post', fields=fields)
#         s_price['3'] = 3
#         s_price['5'] = 5
#         s_price['6']=6
#         s_price['8'] = 8
#         s_price['10'] = 10
#         s_price['12'] = 12
#         s_price['15'] = 15
#         s_price['20']=20

    # s_price['pct'] = s_price['close'].pct_change(periods=5)
#         s_price['alpha'] = alpha_4(s_price)
    factor_dict[str(security)]                                                             =
pd.DataFrame((est_gp.transform(s_price)),index=index,columns=columns)
factor_data = pd.Panel(factor_dict)

# 因子分析
for i in range(10):
    print('alpha'+ str(i+1))
    factor = factor_data.iloc[:,:,i]
#         factor = drop_extreme(factor)
#         factor = standardize(factor)
```

```
# 使用获取的因子值进行单因子分析
far = ja.analyze_factor(factor=factor,
                                    weight_method='avg',
                                    industry='jq_l1',
                                    quantiles=quantiles,
                                    periods=periods,
                                    max_loss=0.9)
    # 生成统计图表
far.create_full_tear_sheet(
    demeaned=False, group_adjust=False, by_group=False,
    turnover_periods=None, avgretplot=(5, 15), std_bar=False
    )
print('alpha'+ str(i+1)+'分析完成')
```