Code 说明文档

Part 1.

数据清洗部分：./ raw\_data\_clean/

1. data\_split.py

作用:将从数据库中导出的.csv 文件转化为.h5文件

使用方法:将data\_split.py 中的if \_\_name\_\_ == "\_\_main\_\_":部分下的

data\_file\_path = "F:/qtdata/" csv数据的存放位置

data\_type = 'tick' 要split的数据类型（tick, transaction, order, index）

save\_path = 'F:/{}\_raw'.format(data\_type) 对应split后数据的输出路径

bgd = "20180824" 开始日期

edd = "20180904" 结束日期

multi\_proc = 4 使用几个进程,这里默认4个,可以在内存可以容忍时使用更多

程序的运行结果可以参考F:\tick\_raw, F:\transaction\_raw, F:\order\_raw

1. tick\_bar\_generator\_class.py

作用:将获取的tick\_raw数据转换成对应的tick\_bar数据并生产对应的target\_bar数据

1. 使用方法:将tick\_bar\_generator\_class.py中的if \_\_name\_\_ == "\_\_main\_\_":部分下的

h5\_path = 'F:/tick\_raw' 对应的tick\_raw数据存放地址

tick\_bar = 'F:/tick\_bar' 输出的tick\_bar数据存放地址

targets\_bar = 'F:/targets\_bar\_new\_add' 输出的target\_bar数据存放地址

bgd = "20180830" 开始日期

edd = "20180831" 结束日期

freq = "3S" 转换成几秒的bar

multi\_proc = 0 使用几个进程

(其他参数无用可以忽略)

另外在该脚本的line 162中

TargetSeriesGenerate\_NEW(data, rt\_series=[15, 60, 120, 180, 300])

rt\_series = [15, 60, 120, 180, 300] 是用于定义target的未来秒数, 这里将计算未来15s, 60s, 120s, 180s, 300s 以及15~60s, 60~120s, 120~180s, 18~300s的mid return. 可以根据需要修改这里的rt\_series的数值

1. h5concat.py

作用:将成的tick\_bar, target\_bar, transaction\_raw, order\_raw 数据concat成比较长的flexible h5 file.

1. 使用方法:将h5concat.py中的if \_\_name\_\_ == "\_\_main\_\_":部分下的

h5\_path = 'F:/tick\_bar' 对应的tick\_bar数据地址

data\_type = 'tick' 对应想concat的数据类型（tick, transaction, order, index）

save\_path = 'F:/{}\_concated\_data\_check\_08\_09'.format(data\_type) 输出结果存放路径

save\_name = '{}\_concated.h5'.format(data\_type) 输出结果名

concat\_mode = 'from\_begin' 两种concat方式(从给定的开始结束日期直接concat成一个独立的h5 file, 如果使用’other’则会往现有的h5 file中继续concat文件

begin\_date = 20180802 concat开始日

end\_date = 20180905 concat 结束日 (由于文件过大可能会导致h5 file出现问题 不建议concat长度超过2个月)

依次运行完毕以上的三种文件后, 正常情况可以获得: transaction\_concated.h5, tick\_concated.h5, order\_concated.h5, targets\_concated.h5 这四个文件为在进行feature 构建时的必要元素. 实例路径为F:\targets\_concated\_data\_check, F:\order\_concated\_data\_check, F:\tick\_concated\_data\_check, F:\transaction\_concated\_data\_check

Part2.

feature 构建部分./TANG\_FINISHED\_PNL/

feature测试框架的主要组件在./ TANG\_FINISHED\_PNL/backtest\_main\_files/下

1. baseSignal\_Modified 定义了basefeature的抽象类, 所有feature函数均继承自该base

主要feature函数均存放在./ TANG\_FINISHED\_PNL/backtest\_main\_files/feature\_files/下

以feature\_selected.py中的Transaction\_ACD为例: 定义一个新feature时, 需要新建一个class, class 名即为feature名, 另外需要注意:

from baseSignal\_Modified import base\_feature

class xxxx(base\_feature) 新的feature是继承自base\_Signal\_Modified.py /base\_feature class的

1. 在选择合适的feature name之后, 需要在param\_list = ['nperiod']中定义feature的参数名列表, 大多数情况下只选择一个参数,即为nperiod (每个时刻往前取的tick数量)

在确定好param\_list后, 需要实际构建feature function

def feature(self, data\_fed, p1, p2, p3, …) 这里p1, p2, p3均应该是在param\_list 中按顺序定义的参数名

1. 在feature function的定义过程中首先会从data\_fed中获取数据(data\_fed为一个dictionary, 具体包含哪些数据由feature定义者给定的configfile中的参数确定. 在Transaction\_ACD这个feature中, 只用到了transaction\_data. 在经过feature function的计算后, 每个featurefile应当return一个pandas series 序列作为该feature的计算结果(order, transaction数据算出来的结果index和tick不同这个测试框架会自动处理对齐补全)

可以在一个feature中选择多个数据源进行计算如feature\_selected.py line 641 Transaction\_Order\_Percent\_Diff 这个feature同时使用了tick, transaction, 和order数据但是需要注意时间戳的处理问题

1. Configfile的构建: 在构建一个feature的时候需要考虑测试什么universe的数据, 测试的时间长度, 使用的数据种类, feature参数的参数空间定义, feature的输出地址等, 本框架使用.json完成. 以./TANG\_FINISHED\_PNL/config\_feature\_selected/Transaction\_ACD.json为例:

Import uniserse:1 代表使用外部csv/xls文件import universe, 设置为0表示不从外部import, 会在self\_specified\_universe中自定义一个universe list

如果确定自定义则在universe\_file\_name 中写出universe csv的地址(现统一使用一列Series的universe file, 示例可查看./ TANG\_FINISHED\_PNL/universe\_ZZ500.csv.

test\_result\_save\_path 为feature计算结果的输出地址

feature\_file\_name 为feature file名称, 必须保持和feature的脚本名一致,程序用该种方法定位到具体的feature file

feature\_name为feature名称, 必须保持和feature 的class名一致,程序用该种方法定位到具体的feature function

test\_time\_start 测试的起始日期

test\_time\_end 测试的结束日期

not\_plot\_pnl 是否对该feature进行交易信号转换并画出单个feature转换成交易信号之后的pnl曲线

not\_plot\_pnl = 0 则计算corr, 画出feature单调性曲线柱状图, 并且计算pnl曲线

not\_plot\_pnl = 1 则计算corr, 画出feature单调性曲线柱状图, 不计算pnl曲线

not\_plot\_pnl = 2则只计算pnl曲线

multi\_process\_feature 计算feature过程多进程控制默认0为单进程,一般选择10

multi\_process\_result feature计算完后backtest过程多进程控制. 一般根据feature的参数量确定, 如果5个参数则选择5 (注意次过程内存消耗有些大, 不能选择太多)

feature\_params 根据feature class 中定义的params\_list确定 每个参数要测多个数值 用dictionary of list 表示

feature\_type 计算该feature使用的数据类型可以同时写出多个, 这样在feature 的data\_fed中就能够提取到具体的数据

each\_type\_data\_need 因为计算feature只需要每种data的部分column, 使用该选项可以定义从每种data中选择哪些column放入data\_fed.

generated\_feature\_h5\_path, generated\_target\_h5\_path 这两个变量在使用已经生成的feature file进行回测时使用, 前者为计算好的feature file名,后者为计算好的target file名.

new\_result\_subfolder\_name 为feature backtest输出结果创建一个subfolder, 在这里定义其名称.

operation\_type 定义这个config的计算目的:

operation\_type = full\_backtest 表示先计算feature 然后使用计算的feature进行 backtest

operation\_type = feature\_calculation表示 仅仅计算feature

operation\_type = others/anythingother than full\_backtest or feature\_calculation表示仅仅根据现有feature file进行backtest 此时上两个变量就会发挥作用

1. 每个feature的每次test都要定义一个configfile来运算,可以选择直接修改老的configfile或者新建configfile. configfile的名称应当与feature name相同
2. 运行测试框架需要使用

./TANG\_FINISHED\_PNL/config\_executor.py

使用该文件时需要修改line 24行的configfile储存地址为对应地址在命令行中使用

python config\_executor.py config\_executor.py -c xxxx 其中xxxx即为feature name

或者定义类似的batch test脚本 批量测试多个feature

./TANG\_FINISHED\_PNL/batch\_configs\_executor.py

./TANG\_FINISHED\_PNL/batch\_config\_executor.py

1. 其他注意事项:

数据来源的地址确定:

./TANG\_FINISHED\_PNL/backtest\_main\_files/backtest\_engine\_final.py

Line 40 ~ line 48 直接修改self. tick\_path, self. target\_path, self. order\_path

self.transaction\_path 实例code中可以看到40~43表示的为6~8月的数据 45~48表示的为8~9月的数据

同时在改数据来源的时候也得修改

./TANG\_FINISHED\_PNL/backtest\_main\_files/get\_universe.py中的line 94, 96, 103, 105将对应的数据和上面的数据地址对应一致

Backtest 信号转换选择的分位点 比如达到整体分为点的98进,-90出 应该在

./TANG\_FINISHED\_PNL/backtest\_main\_files/backtest\_engine\_final.py 中line 26的 upper, lower的数值修改成想要的分位点

同时需要在

./TANG\_FINISHED\_PNL/backtest\_main\_files/auxiliaryFunctionsNew1009.py的line 317将0.0015 和0.0005修改为 upper, 和lower (默认情况使用的是绝对数值0.0015 和0.0005作为信号转换数值点) 调整数值也可以在这里直接调整

Part 3

Machine Learning Part

1. Data combination

在通过feature测试框架完成feature的测试以及筛选后, 可以使用各类machine learning 开源框架进行machine learning model的调试与构建.

在构建模型之前, 需要先构建设计阵和清洗数据以保证每个input的sample都是有效的sample.

./ Machine\_Learning\_structure/data\_construct\_sample\_p1.py

在确定好要include的feature的feature路径,名称以及对应的参数组合后, 应当写一个如以上文件格式的py file然后运行以观察每个入选的feature向量的总长度, 有效点长度. (如果有效点长度过短,则应当删除该feature) 这里的sample file之所有分为dict1, dict2两块是因为dict1没有使用到order数据, 所以为500整个universe的数据, 而dict2用到了order数据所以为500支股票中SZ市场的数据. dict1, dict2分别对应的所有feature的长度应当一样长 且dict2 feature的长度应当为dict1 feature长度的约一半. 进一步确认完毕后,使用

./ Machine\_Learning\_structure/data\_construct\_sample\_p2.py

(注意在该文件的line 115 和line134 将交集部分的长度填上 该长度可由p1.py的输出结果得到

对feature file进行combine (由于pandas df/series 的concat速度非常慢(每次concat需要对齐index, 在index很长的情况下效率极低), 故使用numpy.concatenate() 先将所有的feature values提取出来再进一步拼接. 注意 dict的feature部分恰巧对应dict1 feature的前半段, 故使用dict1\_feature.iloc[:len(dict2\_feature)] 即可提取出dict1 feature的对应时间区间(可以找部分feature进行确认)

由该文件得到了对应的feature matrix 之后即可放入提交的machine learning 框架进行model training.

1. Model training

使用./ Machine\_Learning\_structure/lightgbm

或者./ Machine\_Learning\_structure/random\_forest 配上生成的feature\_matrix.h5 均可进行training

1. Modeling test set validation and backtest

在完成model的训练之后, 需要将样本外的数据拿出对比. 在使用同样的流程构建好样本外的feature\_matrix之后 使用model.predict() 即可得到样本外的model prediction y\_hat.

由于同时还需要对应区间的stock spread rate的信息来对prediction数值进行进一步操作(y\_hat – sread\_rate) 所以需要调用

./TANG\_FINISHED\_PNL/config\_ML/Ask\_Bid\_Spread\_rate.json

对feature Ask\_Bid\_Spread\_rate 得到对应所有股票的spread\_rate 序列

计算完毕后, 修改并使用feature\_spread\_operation.py将结果构建为feature形式(这里需要构建成feature形式的具体原因就是可以直接使用feature的测试框架直接bt, 所以需要将计算出来的信号”伪装成一个feature”. 完成后可以修改部分

./ TANG\_FINISHED\_PNL/config\_ML/SZ\_feature.json的参数(将路径改为新生成的信号路径)

Part 4

Feature files

使用的feature files放在./TANG\_FINISHED\_PNL/feature\_files/feature\_selected\_1220.py中

新的feature formula为./Feature\_Formulas\_NEW.txt

关于F:/ 文档说明

TOPT\_1030 存放backtest的结果

TOPT\_1101 , TOPT\_08\_09\_Validate, TOPT\_08\_09\_Validate\_add 存放的为feature在500 universe上的计算结果

其中TOPT\_1101为6~8月，后两者为8~9月 TOPT\_08\_09\_Validate\_add为TOPT\_08\_09\_Validate的补充

TOPT\_1203\_Train， TOPT\_1203\_Test 存放的为feature在1800 universe上的计算结果，其中前者为6~8月，后者为8~9月

Backtest示例说明：

F:/TOPT\_1030下

Test\_SZ\_feature\_On\_TrainSet\_Train\_ADD\_features\_1218\_minus\_spread\_add\_From\_20180601\_To\_20180801

Test\_SZ\_feature\_On\_TestSet\_Train\_ADD\_features\_1218\_minus\_spread\_add\_From\_20180802\_To\_20180903

这两个文件夹中分别存放的是500 universe 中 SZ股票数据在lightgbm model训练后得到的回测结果，前者为训练集部分，后者为测试集部分。

对应的参数组合和训练文件可以参考：./model\_1218/new\_trained\_model\_1218.xlsx,

./model\_1218/lgb\_model\_1218.ipynb

在该model test result 的nperiod 1, 2, 3 分别对应的是该model的三个参数组合

Test\_SZ\_feature\_On\_Test\_predict\_ret\_minus\_spread\_1800\_From\_20180601\_To\_20180801和

Test\_SZ\_feature\_On\_TESTSET\_prediction\_minus\_spread\_1800\_From\_20180802\_To\_20180903

这两个文件夹中分别存放的是1800 universe 中 SZ股票数据在lightgbm model训练后得到的回测结果，前者为训练集部分，后者为测试集部分。

对应的参数组合和训练文件可以参考：./model\_1218/model\_1203\_parameters.xlsx,

./model\_1218/1800\_universe\_train.ipynb 只测试了一个model, nerpiod 10 即为该model的测试结果

以上的测试结果为(predicted 60s return – spread rate) 然后以0.0015 和 0.0005作为分割点转化为交易信号进行交易