1. **json文件**

参数说明：

可以修改的参数：标红

|  |  |  |
| --- | --- | --- |
| 参数名 | 默认值 | 备注 |
| "FactorData" |  | "//192.168.2.39/YangqinStrategies/factordata.csv", |
| "Path" |  | "//192.168.2.39/YangqinStrategies/factor\_momentum/" |
| "start\_date" | "2023-04-30" | 回测开始的时间 |
| "end\_date" | "2023-07-05" | 回测结束的时间 |
| "backdays" | 252 | 因子收益率计算周期 备选：252,126,31 |
| "period" | 5 | 调仓周期 备选：5,7,10,20 |
| "return\_period" | 5 | 计算最近因子收益的周期，用于回归筛选特征 备选：5,10,20 |
| "factor\_period" | 21 | 风格切换的周期 备选：10,21,42 |
| "amount\_num" | 5 | 股票账户数量（分仓数量） 备选：5,7 |
| "nums\_fin" | 10 | 筛选的因子数量 备选：10,20 |
| "factor\_quantile" | 20 | 股票分组数 20表示买卖前5% 备选：5,10 |
| "if\_industry\_market\_cap\_norm" | true | 是否行业、市值中性化 |
| "sample\_weight" | "equal\_weight" | 回归的样本权重 备选：’sample\_weight’ |
| "regression\_period" | 30 |  |
| "method\_return\_cal" | "long\_short" | 因子单日收益率估计方法 备选：'long\_short','long\_only','short\_only’ |
| "method\_factor\_weight" | "return" | 因子权重分配方法 备选：'return','equal' |
| "method\_stock\_weight" | "return" | 股票权重分配方法 备选：'return','equal |
| "method\_factor" | "long" | 股票买卖和因子归一化的计算方法 备选：'long\_short','long\_only','short\_only’ |
| "fee\_rate" | 0.003 | 手续费率 |
| "nums\_xg" | 30 | 训练xgboost得到的因子数量 |
| "if\_xgboost" | false | 是否使用xgboost预先筛选变量，不建议使用 |
| "if\_indicators" | false | 是否使用正负1来回归 备选：True,False |
| "n\_splits" | 10 | 交叉验证折数 |
| "verbose" | 1 | 交叉验证输出信息等级 |
| "index" | "date" | xgboost使用的参数 |
| "y\_index" | "return\_norm" | xgboost训练目标 |
|  |  |  |

**注**：

**1． 默认值是优先值**

**2． method的四种情形：**

①同时允许做多与做空情形下（long\_short）；

②只允许做多股票情形下的策略（long；含义是允许做多与做空因子，做多因子时买入因子值排名靠前的股票，卖空因子时转为买入因子值排名靠后的股票）；

③只允许做多因子情形下的策略（long\_only；含义是只在做多因子时买入因子值排名靠前的股票，而不考虑做空因子）；

④只允许做空因子情形下的策略（short\_only；含义是只在做空因子时买入因子值排名靠后的股票，而不考虑做多因子）

# long\_short: sum (weight<0) = -1/2, sum (weight>0) = 1/2 分方向归一化后，再一起归一化，w有正有负，不作为投资建议，因为无法实现，只能用来判断因子是否有效,正收益说明策略有效

# long: sum abs(weight) = 1, w有正有负 做空的因子变成买底部的，做多的因子变成买顶部的

# long\_only : sum weight = 1, w全正,卖空的因子为0

# short\_only: sum weight = 1, w全正,做多的因子为0

1. **py文件**

流程没有问题，可以优化/修改的方向：

**1.因子权重的计算函数**

Def get\_weight(fac\_r,fac\_mom\_l,fac\_mom\_s,fac\_mom\_63,fac\_mom\_ternary,coef\_df,top\_features,method\_factor\_weight = 'equal',method\_factor = 'long'):  ###做了修改调整

                """

                fac\_mom\_ternary: 每个因子是否买入的矩阵，只包含选出来的特征

                method: 权重分配方法

                return: 每个因子的权重 days \* factors ,shape = len(days-1)\*len(factors)，有正有负

                """

                # 1月3号的持仓权重保存在1月2号（1月2号收盘时的数据作为1月3号的权重）

                # 但是基于1月1号的因子值（因为用了1月2号的收益率来计算1月1号的因子收益率），所以要用shift(1)来调整

                # 在backtest中用第二天的收益率来计算策略收益的。

                # 每个因子买卖的权重

                if method\_factor\_weight == 'equal':

                    weight = fac\_mom\_ternary[top\_features].shift(1).apply(lambda x : trans(x,method\_factor=method\_factor), axis=1)

                elif method\_factor\_weight == 'return':

                    fac\_mom\_l = fac\_mom\_l[top\_features].reindex(index=fac\_mom\_s.index) # 保留和fac\_mom\_s相同的index对应的值

                    fac\_mom\_63 = fac\_mom\_63[top\_features].reindex(index=fac\_mom\_s.index) # 保留和fac\_mom\_s相同的index对应的值

                    # 方法 1：

                    # weight = np.sign(0.8\*(fac\_mom\_l-1)+0.19\*(fac\_mom\_63-1)+0.01\*(fac\_mom\_s-1))\*abs(0.85\*(fac\_mom\_l-1)+0.14\*(fac\_mom\_63-1)+0.01\*(fac\_mom\_s-1))\

                    #     .shift(1).apply(lambda x : trans(x,method\_factor=method\_factor), axis=1)

                    # 方法 2：

                    # 方法 3:

                weight.columns =  ['factor\_quantile\_'+ i for i in weight.columns]

                return weight

注意项：

1使用相关收益矩阵的时候要.shift(1)，避免使用未来数据

2输出格式为dates为行，factors为列的dataframe

3不建议使用等权

**2.筛选因子的函数**

def feature\_collect(factors\_r\_all,fac\_mom\_l, fac\_mom\_s,factor\_name = top\_20\_features, nums\_fin = 10, start\_date = datetime.date(2022,6,15), return\_period = 7,sample\_weight = 'integer\_weight',if\_indicators=False):

                coefficients = []

                coefficients2 = []

                const = []

                const2 = []

                t = []

                t2 = []

                std = []

                std2 = []

                r2 = []

                sign = []

                # df 是 fac\_mom, 取start\_date之前的数据 ,shift 1 day,避免使用未来数据

                # df = (fac\_mom\_l>1)....

                df = (2\*(fac\_mom\_l>1)-1).shift(1+return\_period).loc[:start\_date] # 过去一年的收益率不应包含当前考察时间段内的收益率，所以shift 1+period day

                df2 = fac\_mom\_s.shift(1).loc[:start\_date]

                df3 = factors\_r\_all.shift(1).loc[:start\_date].iloc[-63:]

                df3['x'] = range(1, len(df3)+1)

                for factor in factor\_name:##后续改为factor\_name

                    X = sm.add\_constant(df[factor]) #################################### 不能用数值来计算

                    y = df2[factor]

                    cleaned\_data = pd.concat([X, y], axis=1).dropna().iloc[-factor\_period:]

                    position\_index = ([j // return\_period \* return\_period for j in range(0, len(cleaned\_data))])

                    ##这里可以优化，引入样本点的权重

                    if sample\_weight == 'integer\_weight':

                        position\_index = list(range(0,len(cleaned\_data),return\_period))

                        position\_index = [[i]\*(position\_index.index(i)+1) for i in position\_index]

                        position\_index = sum(position\_index,[])

                    X = cleaned\_data.iloc[position\_index,:-1]

                    if if\_indicators:

                        X = np.sign(X)

                    y = cleaned\_data.iloc[position\_index,-1]

                    model = sm.OLS(y, X).fit()

                    # X2从1到len(df3)

                    x = pd.DataFrame({factor:df3['x']})

                    X2 = sm.add\_constant(x)

                    y2 = (df3[factor]+1).cumprod()

                    cleaned\_data2 = pd.concat([X2, y2], axis=1).iloc[-151:].dropna()

                    X2 = cleaned\_data2.iloc[:,:-1]

                    y2 = cleaned\_data2.iloc[:,-1]

                    model2 = sm.OLS(y2, X2).fit()

                    #提取回归系数

                    coefficients.append(model.params[factor])

                    coefficients2.append(abs(model2.params[factor]))

                    #提取截距

                    const.append(model.params['const'])

                    const2.append(abs(model2.params['const']))

                    #提取t值

                    t.append(model.tvalues[factor])

                    t2.append(abs(model2.tvalues[factor]))

                    #提取残差标准差

                    std.append(np.sqrt(model.scale))

                    std2.append(np.sqrt(model2.scale))

                    #提取正负号

                    sign.append(np.sign(model2.params[factor]))

                    #提取R2

                    r2.append(model2.rsquared)

                coefficients\_df = pd.DataFrame({'coefficients':coefficients,'coefficients2':coefficients2}, index=factor\_name)

                const\_df = pd.DataFrame({'const':const,'const2':const2}, index=factor\_name)

                t\_df = pd.DataFrame({'t':t,'t2':t2}, index=factor\_name)

                std\_df = pd.DataFrame({'std':std,'std2':std2}, index=factor\_name)

                sign\_df = pd.DataFrame({'sign2':sign}, index=factor\_name)

                r2\_df = pd.DataFrame({'r2':r2}, index=factor\_name)

                coef\_df = pd.concat([coefficients\_df,const\_df, t\_df, std\_df, sign\_df,r2\_df], axis=1)

                # coefficients的秩和coefficients2的秩加权求和 rank越小表示值越大

                coef\_df['up'] = coef\_df['coefficients2'] + 1.96\*coef\_df['std2']

                coef\_df['down'] = coef\_df['coefficients2'] - 1.96\*coef\_df['std2']

                # coef\_df['rank'] = coef\_df['up'].rank(ascending=False) + 1.2\* coef\_df['down'].rank(ascending=False)

                # coef\_df['rank'] = coef\_df['up'].rank(ascending=False) + 0.8\* coef\_df['down'].rank(ascending=False)

                coef\_df['rank'] = coef\_df['coefficients2'].rank(ascending=False) + 0.3\*coef\_df['r2'].rank(ascending=False) - 0.3\* coef\_df['std2'].rank(ascending=False)

                # coef\_df['rank'] = coef\_df['coefficients'].rank(ascending=False) - 0.1\* coef\_df['std2'].rank(ascending=False)

                top\_features = coef\_df.sort\_values(by='rank', ascending= True).index[:nums\_fin] #这里的10是参数

                return top\_features , coef\_df

可以修改的地方：

因子的排序方式

**解释**：

1.Coefficients：5日内的因子收益率与过去一年因子收益率的回归系数

Std：对应的回归方程的残差标准差

2.Coefficients2：对筛选出的因子的前3个月内的因子累计收益率进行对时间进行回归所得到的回归系数的绝对值

Std2：对应的回归方程的残差标准差

Sign：对应的回归方程的回归系数的符号

Up：对应的回归方程的未来一天变动量的预测值的上界

Down：对应的回归方程的未来一天变动量的预测值的下界