# 项目简介

## 项目介绍

1. 基于abu开发平台实现不同风险人群对应不同策略的目的
2. 策略中分为选股策略、买卖策略
3. 我们以风险厌恶程度为度量单位将人群分为风险偏好人群、风险中性人群、风险厌恶人群
4. 选股策略通过用户偏好来提供不同策略
5. 买卖策略通过用户偏好提供对应策略，并通过网格搜索交叉调优方法确定最优参数

## 小组信息

* 小组成员：李嘉伟 赵子宸 孙欣然 王雪燕 刘晴 任嘉怡
* 学院：工商管理学院
* 班级： 金融1802班
* 课程名称：金融科技项目实践2（产品设计与开发）
* 上课学期： 2020-2021春季学期

ps:具体代码运行可看low risk；mid risk;high risk 文件

# 实验内容

## 目标人群偏好

1. 风险偏好人群：对于股价波动幅度大小不是特别在意，追求短期超额收益,回撤幅度大，风险承受能力较大
2. 风险中性人群：介于风险偏好与风险厌恶两种人群之间，没有明确的风险偏好。
3. 风险厌恶人群：对于股价波动幅度大小十分在意，追求长期平稳收益，回撤幅度小，风险承受能力较小

## 选股策略

### 策略介绍

* 拟合角度选股策略：通过设定股票上升或下降角度阈值从而挑选出符合参数走势的股票
* 位移路程比选股策略：通过设定股价曲线在一段区间内的位移与路程的比值范围来筛选出波动幅度不同的股票
* 涨幅topN选股策略：通过选取一段时间内涨幅排名靠前的股票

### 核心代码展示

位移路程比选股策略

class AbuPickStockShiftDistance(AbuPickStockBase):  
 """位移路程比选股因子示例类"""  
  
 def \_init\_self(self, \*\*kwargs):  
 """通过kwargs设置位移路程比选股条件，配置因子参数"""  
 self.threshold\_sd = kwargs.pop('threshold\_sd', 2.0)  
 self.threshold\_max\_cnt = kwargs.pop('threshold\_max\_cnt', 4)  
 self.threshold\_min\_cnt = kwargs.pop('threshold\_min\_cnt', 1)  
  
 @ps.reversed\_result  
 def fit\_pick(self, kl\_pd, target\_symbol):  
 """开始根据位移路程比边际参数进行选股"""  
  
 pick\_line = tl.AbuTLine(kl\_pd.close, 'shift distance')  
 shift\_distance = pick\_line.show\_shift\_distance(step\_x=1.2, show\_log=False, show=False)  
 shift\_distance = np.array(shift\_distance)  
 # show\_shift\_distance返回的参数为四组数据，最后一组是每个时间段的位移路程比值  
 sd\_arr = shift\_distance[:, -1]  
 # 大于阀值的进行累加和计算  
 # noinspection PyUnresolvedReferences  
 threshold\_cnt = (sd\_arr >= self.threshold\_sd).sum()  
 # 边际条件参数开始生效  
 if self.threshold\_max\_cnt > threshold\_cnt >= self.threshold\_min\_cnt:  
 return True  
 return False  
  
 def fit\_first\_choice(self, pick\_worker, choice\_symbols, \*args, \*\*kwargs):  
 raise NotImplementedError('AbuPickStockShiftDistance fit\_first\_choice unsupported now!')

拟合角度选股策略

class AbuPickRegressAngMinMax(AbuPickStockBase):  
 """拟合角度选股因子示例类"""  
 def \_init\_self(self, \*\*kwargs):  
 """通过kwargs设置拟合角度边际条件，配置因子参数"""  
  
 # 暂时与base保持一致不使用kwargs.pop('a', default)方式  
 # fit\_pick中 ang > threshold\_ang\_min, 默认负无穷，即默认所有都符合  
 self.threshold\_ang\_min = -np.inf  
 if 'threshold\_ang\_min' in kwargs:  
 # 设置最小角度阀值  
 self.threshold\_ang\_min = kwargs['threshold\_ang\_min']  
  
 # fit\_pick中 ang < threshold\_ang\_max, 默认正无穷，即默认所有都符合  
 self.threshold\_ang\_max = np.inf  
 if 'threshold\_ang\_max' in kwargs:  
 # 设置最大角度阀值  
 self.threshold\_ang\_max = kwargs['threshold\_ang\_max']  
  
 @ps.reversed\_result  
 def fit\_pick(self, kl\_pd, target\_symbol):  
 """开始根据自定义拟合角度边际参数进行选股"""  
 # 计算走势角度  
 ang = ABuRegUtil.calc\_regress\_deg(kl\_pd.close, show=False)  
 # 根据参数进行角度条件判断  
 if self.threshold\_ang\_min < ang < self.threshold\_ang\_max:  
 return True  
 return False  
  
 def fit\_first\_choice(self, pick\_worker, choice\_symbols, \*args, \*\*kwargs):  
 raise NotImplementedError('AbuPickRegressAng fit\_first\_choice unsupported now!')

涨幅topN选股策略

class AbuPickStockNTop(AbuPickStockBase):  
 """根据一段时间内的涨幅选取top N个"""  
  
 def \_init\_self(self, \*\*kwargs):  
 """通过kwargs设置选股条件，配置因子参数"""  
 # 选股参数symbol\_pool：进行涨幅比较的top n个symbol  
 self.symbol\_pool = kwargs.pop('symbol\_pool', [])  
 # 选股参数n\_top：选取前n\_top个symbol, 默认3  
 self.n\_top = kwargs.pop('n\_top', 3)  
 # 选股参数direction\_top：选取前n\_top个的方向，即选择涨的多的，还是选择跌的多的  
 self.direction\_top = kwargs.pop('direction\_top', 1)  
  
 @ps.reversed\_result  
 def fit\_pick(self, kl\_pd, target\_symbol):  
 """开始根据参数进行选股"""  
 if len(self.symbol\_pool) == 0:  
 # 如果没有传递任何参照序列symbol，择默认为选中  
 return True  
 # 定义lambda函数计算周期内change  
 kl\_change = lambda p\_kl: \  
 p\_kl.iloc[-1].close / p\_kl.iloc[0].close if p\_kl.iloc[0].close != 0 else 0  
  
 cmp\_top\_array = []  
 kl\_pd.name = target\_symbol  
 # AbuBenchmark直接传递一个kl  
 benchmark = AbuBenchmark(benchmark\_kl\_pd=kl\_pd)  
 for symbol in self.symbol\_pool:  
 if symbol != target\_symbol:  
 # 使用benchmark模式进行获取  
 kl = ABuSymbolPd.make\_kl\_df(symbol, data\_mode=EMarketDataSplitMode.E\_DATA\_SPLIT\_UNDO,  
 benchmark=benchmark)  
 # kl = ABuSymbolPd.make\_kl\_df(symbol, start=start, end=end)  
 if kl is not None and kl.shape[0] > kl\_pd.shape[0] \* 0.75:  
 # 需要获取实际交易日数量，避免停盘等错误信号  
 cmp\_top\_array.append(kl\_change(kl))  
  
 if self.n\_top > len(cmp\_top\_array):  
 # 如果结果序列不足n\_top个，直接认为选中  
 return True  
  
 # 与选股方向相乘，即结果只去top  
 cmp\_top\_array = np.array(cmp\_top\_array) \* self.direction\_top  
 # 计算本源的周期内涨跌幅度  
 target\_change = kl\_change(kl\_pd) \* self.direction\_top  
 # sort排序小－》大, 非inplace  
 cmp\_top\_array.sort()  
 # [::-1]大－》小  
 # noinspection PyTypeChecker  
 if target\_change > cmp\_top\_array[::-1][self.n\_top - 1]:  
 # 如果比排序后的第self.n\_top位置上的大就认为选中  
 return True  
 return False  
  
 def fit\_first\_choice(self, pick\_worker, choice\_symbols, \*args, \*\*kwargs):  
 raise NotImplementedError('AbuPickStockNTop fit\_first\_choice unsupported now!')

风险厌恶人群选股策略设定如下：拟合角度为0到10度；位移路程比阈值为3，范围为2到4。

stock\_pickers = [{'class': AbuPickRegressAngMinMax,  
 'threshold\_ang\_min': 0.0,'threshold\_ang\_max': 10, 'reversed': False},  
 {'class': AbuPickStockShiftDistance,'threshold\_sd':3.0,'threshold\_max\_cnt':4,  
 'threshold\_min\_cnt':2,  
 'reversed': False}]

风险中性人群选股策略设定如下：拟合角度选股为10到20度

stock\_pickers = [{'class': AbuPickRegressAngMinMax,  
 'threshold\_ang\_min': 10.0,'threshold\_ang\_max': 20, 'reversed': False}]

风险偏好人群选股策略设定如下：拟合角度选股为20度以上；位移路程比阈值为3，范围为2-4；topN选股涨幅前3的股票

stock\_pickers = [{'class': AbuPickRegressAngMinMax,  
 'threshold\_ang\_min': 20.0,'reversed': False},  
 {'class': AbuPickStockShiftDistance,'threshold\_sd':3.0,'threshold\_max\_cnt':4,  
 'threshold\_min\_cnt':2,  
 'reversed': False},  
 {'class':AbuPickStockNTop,'symbol\_pool':choice\_symbols,'n\_top':3,'direction\_top':1,  
 'reversed': False}]

股池设定如下

choice\_symbols = ['usNOAH', 'usSFUN', 'usBIDU', 'usAAPL', 'usGOOG',  
 'usTSLA', 'usWUBA', 'usVIPS','002230', '300104', '300059',   
 '601766', '600085', '600036', '600809', '000002', '002594', '002739','hk03333',   
 'hk00700', 'hk02333', 'hk01359', 'hk00656', 'hk03888', 'hk02318']

选股函数如下

benchmark = AbuBenchmark()  
capital = AbuCapital(1000000, benchmark)  
kl\_pd\_manger = AbuKLManager(benchmark, capital)  
stock\_pick = AbuPickStockWorker(capital, benchmark, kl\_pd\_manger,  
 choice\_symbols=choice\_symbols,  
 stock\_pickers=stock\_pickers)

选股结果展示

风险厌恶人群：

['usBIDU', '601766', 'hk03333']

风险中性人群：

['usTSLA', '300104', '002594', 'hk03888']

风险偏好人群：

['hk00700']

## 买卖策略选择及仓位配置

### 策略介绍

1. 向上突破买策：通过设定突破一段时间内的最高点来实现突破买入的策略【常见日期为42天、60天】
2. 均值回复买策：当价格跌破均线一段距离时进行买入操作，等待价格回复到均值处。该类操作对于稳定增长的股票而言十分管用。
3. 止跌止损卖策： 当盈利超过一定范围或亏损超过一定范围进行卖出操作
4. pt仓位管理策略：搭配均值回复策略最有效。即偏离均值的程度越大，仓位越大；偏离均值的程度越小，仓位越小。

### 核心代码展示

向上突破买入策略

class AbuFactorBuyBreak(AbuFactorBuyBase, BuyCallMixin):  
 """示例正向突破买入择时类，混入BuyCallMixin，即向上突破触发买入event"""  
  
 def \_init\_self(self, \*\*kwargs):  
 """kwargs中必须包含: 突破参数xd 比如20，30，40天...突破"""  
 # 突破参数 xd， 比如20，30，40天...突破, 不要使用kwargs.pop('xd', 20), 明确需要参数xq  
 self.xd = kwargs['xd']  
 # 在输出生成的orders\_pd中显示的名字  
 self.factor\_name = '{}:{}'.format(self.\_\_class\_\_.\_\_name\_\_, self.xd)  
  
 def fit\_day(self, today):  
 """  
 针对每一个交易日拟合买入交易策略，寻找向上突破买入机会  
 :param today: 当前驱动的交易日金融时间序列数据  
 :return:  
 """  
 # 忽略不符合买入的天（统计周期内前xd天）  
 if self.today\_ind < self.xd - 1:  
 return None  
  
 # 今天的收盘价格达到xd天内最高价格则符合买入条件  
 if today.close == self.kl\_pd.close[self.today\_ind - self.xd + 1:self.today\_ind + 1].max():  
 # 把突破新高参数赋值skip\_days，这里也可以考虑make\_buy\_order确定是否买单成立，但是如果停盘太长时间等也不好  
 self.skip\_days = self.xd  
 # 生成买入订单, 由于使用了今天的收盘价格做为策略信号判断，所以信号发出后，只能明天买  
 return self.buy\_tomorrow()  
 return None

均值回复策略

class AbuUpDownTrend(AbuFactorBuyXD, BuyCallMixin):  
 """示例长线上涨中寻找短线下跌买入择时因子，混入BuyCallMixin"""  
  
 def \_init\_self(self, \*\*kwargs):  
 """  
 kwargs中可以包含xd: 比如20，30，40天...突破，默认20  
 kwargs中可以包含past\_factor: 代表长线的趋势判断长度，默认4，long = xd \* past\_factor->eg: long = 20 \* 4  
 kwargs中可以包含up\_deg\_threshold: 代表判断上涨趋势拟合角度阀值，即长线拟合角度值多少决策为上涨，默认3  
 """  
 if 'xd' not in kwargs:  
 # 如果外部没有设置xd值，默认给一个30  
 kwargs['xd'] = 20  
 super(AbuUpDownTrend, self).\_init\_self(\*\*kwargs)  
 # 代表长线的趋势判断长度，默认4，long = xd \* past\_factor->eg: long = 30 \* 4  
 self.past\_factor = kwargs.pop('past\_factor', 4)  
 # 代表判断上涨趋势拟合角度阀值，即长线拟合角度值多少决策为上涨，默认4  
 self.up\_deg\_threshold = kwargs.pop('up\_deg\_threshold', 3)  
  
 def fit\_day(self, today):  
 """  
 长线周期选择目标为上升趋势的目标，短线寻找近期走势为向下趋势的目标进行买入，期望是持续之前长相的趋势  
 1. 通过past\_today\_kl获取长周期的金融时间序列，通过AbuTLine中的is\_up\_trend判断  
 长周期是否属于上涨趋势，  
 2. 今天收盘价为最近xd天内最低价格，且短线xd天的价格走势为下跌趋势  
 3. 满足1，2发出买入信号  
 :param today: 当前驱动的交易日金融时间序列数据  
 """  
 long\_kl = self.past\_today\_kl(today, self.past\_factor \* self.xd)  
 tl\_long = AbuTLine(long\_kl.close, 'long')  
 # 判断长周期是否属于上涨趋势  
 if tl\_long.is\_up\_trend(up\_deg\_threshold=self.up\_deg\_threshold, show=False):  
 if today.close == self.xd\_kl.close.min() and AbuTLine(  
 self.xd\_kl.close, 'short').is\_down\_trend(down\_deg\_threshold=-self.up\_deg\_threshold, show=False):  
 # 今天收盘价为最近xd天内最低价格，且短线xd天的价格走势为下跌趋势  
 return self.buy\_tomorrow()

风险厌恶人群买策设定为均值回复策略

buy\_factors = {'class': [AbuDownUpTrend]}

风险中性人群买策设定为均值回复策略和向上突破策略

buy\_factors = {'class': [AbuDownUpTrend]}  
  
buy\_bk\_factor\_grid1 = {  
 'class': [AbuFactorBuyBreak],  
 'xd': [42]  
}  
  
buy\_bk\_factor\_grid2 = {  
 'class': [AbuFactorBuyBreak],  
 'xd': [60]  
}

风险偏好人群买策设定为向上突破型策略

buy\_bk\_factor\_grid1 = {  
 'class': [AbuFactorBuyBreak],  
 'xd': [42]  
}  
  
buy\_bk\_factor\_grid2 = {  
 'class': [AbuFactorBuyBreak],  
 'xd': [60]  
}

统一卖策为止跌止损型策略

参数区间的设定影响了策略的总数，分的区间越细致所组合成的策略集合更多，数据量更大，在接下来的网格调优阶段更难运行，需要的算力更大

stop\_win\_range = np.arange(2.0, 4.5, 0.5) #该位置用来设定止盈范围  
stop\_loss\_range = np.arange(0.5, 2, 0.5) #该位置用来设定止损范围  
  
sell\_atr\_nstop\_factor\_grid = {  
 'class': [AbuFactorAtrNStop],  
 'stop\_loss\_n' : stop\_loss\_range,  
 'stop\_win\_n' : stop\_win\_range  
 }  
   
 close\_atr\_range = np.arange(1.0, 4.0, 0.5) #该位置用来设定盈利保护止盈参数  
pre\_atr\_range = np.arange(1.0, 3.5, 0.5) #该位置用来设定暴跌保护止损参数  
   
   
sell\_atr\_pre\_factor\_grid = {  
 'class': [AbuFactorPreAtrNStop],  
 'pre\_atr\_n' : pre\_atr\_range  
 }  
  
sell\_atr\_close\_factor\_grid = {  
 'class': [AbuFactorCloseAtrNStop],  
 'close\_atr\_n' : close\_atr\_range  
 }

pt仓位管理策略

class AbuPtPosition(AbuPositionBase):  
 """  
 示例价格位置仓位管理类：  
  
 根据买入价格在之前一段时间的价格位置来决策仓位大小  
  
 假设过去一段时间的价格为[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]  
 如果当前买入价格为2元：则买入仓位配比很高(认为均值回复有很大向上空间)  
 如果当前买入价格为9元：则买入仓位配比很低(认为均值回复向上空间比较小)  
 """  
  
 def fit\_position(self, factor\_object):  
 """  
 针对均值回复类型策略的仓位管理：  
 根据当前买入价格在过去一段金融序列中的价格rank位置来决定仓位  
 fit\_position计算的结果是买入多少个单位（股，手，顿，合约）  
 :param factor\_object: ABuFactorBuyBases子类实例对象  
 :return:买入多少个单位（股，手，顿，合约）  
 """  
  
 # self.kl\_pd\_buy为买入当天的数据，获取之前的past\_day\_cnt天数据  
 last\_kl = factor\_object.past\_today\_kl(self.kl\_pd\_buy, self.past\_day\_cnt)  
 if last\_kl is None or last\_kl.empty:  
 precent\_pos = self.pos\_base  
 else:  
 # 使用percentileofscore计算买入价格在过去的past\_day\_cnt天的价格位置  
 precent\_pos = stats.percentileofscore(last\_kl.close, self.bp)  
 precent\_pos = (1 + (self.mid\_precent - precent\_pos) / 100) \* self.pos\_base  
 # 最大仓位限制，依然受上层最大仓位控制限制，eg：如果算出全仓，依然会减少到75%，如修改需要修改最大仓位值  
 precent\_pos = self.pos\_max if precent\_pos > self.pos\_max else precent\_pos  
 # 结果是买入多少个单位（股，手，顿，合约）  
 return self.read\_cash \* precent\_pos / self.bp \* self.deposit\_rate  
  
 def \_init\_self(self, \*\*kwargs):  
 """价格位置仓位控制管理类初始化设置"""  
 # 默认平均仓位比例0.10，即10%  
 self.pos\_base = kwargs.pop('pos\_base', 0.10)  
 # 默认获取之前金融时间序列的长短数量  
 self.past\_day\_cnt = kwargs.pop('past\_day\_cnt', 20)  
 # 默认的比例中值，一般不需要设置  
 self.mid\_precent = kwargs.pop('mid\_precent', 50.0)

pt仓位管理策略搭配均值回复买策适用于风险厌恶人群

buy\_factors = [{'class': AbuDownUpTrend, 'position': {'class': AbuPtPosition, 'past\_day\_cnt': 80}}]

## 买卖策略配对及网格搜索参数调优

风险厌恶人群相应买策和卖策搭配

buy\_factors\_product = ABuGridHelper.gen\_factor\_grid(  
 ABuGridHelper.K\_GEN\_FACTOR\_PARAMS\_BUY, [buy\_factors])  
sell\_factors\_product = ABuGridHelper.gen\_factor\_grid(  
 ABuGridHelper.K\_GEN\_FACTOR\_PARAMS\_SELL,  
 [sell\_atr\_nstop\_factor\_grid, sell\_atr\_pre\_factor\_grid, sell\_atr\_close\_factor\_grid], need\_empty\_sell=True)

风险中性人群相应买策和卖策搭配

buy\_factors\_product = ABuGridHelper.gen\_factor\_grid(  
 ABuGridHelper.K\_GEN\_FACTOR\_PARAMS\_BUY, [buy\_bk\_factor\_grid1, buy\_bk\_factor\_grid2,buy\_factors])  
sell\_factors\_product = ABuGridHelper.gen\_factor\_grid(  
 ABuGridHelper.K\_GEN\_FACTOR\_PARAMS\_SELL,  
 [sell\_atr\_nstop\_factor\_grid, sell\_atr\_pre\_factor\_grid, sell\_atr\_close\_factor\_grid], need\_empty\_sell=True)

风险偏好人群相应买策和卖策搭配

buy\_factors\_product = ABuGridHelper.gen\_factor\_grid(  
 ABuGridHelper.K\_GEN\_FACTOR\_PARAMS\_BUY, [buy\_bk\_factor\_grid1, buy\_bk\_factor\_grid2])  
sell\_factors\_product = ABuGridHelper.gen\_factor\_grid(  
 ABuGridHelper.K\_GEN\_FACTOR\_PARAMS\_SELL,  
 [sell\_atr\_nstop\_factor\_grid, sell\_atr\_pre\_factor\_grid, sell\_atr\_close\_factor\_grid], need\_empty\_sell=True)

初始化网格搜索模型

read\_cash=1000000  
grid\_search = GridSearch(read\_cash, choice\_symbols=stock\_pick.choice\_symbols,  
 buy\_factors\_product=buy\_factors\_product,  
 sell\_factors\_product=sell\_factors\_product)

调优运行界面

scores = None  
score\_tuple\_array = None  
  
def run\_grid\_search():  
 global scores, score\_tuple\_array  
 # 运行GridSearch n\_jobs=-1启动cpu个数的进程数  
 scores, score\_tuple\_array = grid\_search.fit(n\_jobs=-1)  
 # 运行完成输出的score\_tuple\_array可以使用dump\_pickle保存在本地，以方便之后使用  
 ABuFileUtil.dump\_pickle(score\_tuple\_array, '../gen/score\_tuple\_array')  
  
def select(select):  
 if select == 'run gird search':  
 run\_grid\_search()  
 else: # load score cache  
 load\_score\_cache()  
  
\_ = ipywidgets.interact\_manual(select, select=['run gird search', 'load score cache'])

返回最优调参结果

best\_score\_tuple\_grid = grid\_search.best\_score\_tuple\_grid  
AbuMetricsBase.show\_general(best\_score\_tuple\_grid.orders\_pd, best\_score\_tuple\_grid.action\_pd,  
 best\_score\_tuple\_grid.capital, best\_score\_tuple\_grid.benchmark)

# 不足及展望

1. 设定参数区间决定了调优的快慢，在当前电脑下很难进行更细致的参数区间细分，导致区间内的参数过于单一。
2. 该项目初步实现了不同风险承受能力的人群对应不同的策略集合，而这些集合都是通过调优模型得出来的，具有一定参考价值。之后还差系统界面的封装，以实现输入一个参数，返回对应策略的结果，之后会加以完善的。
3. 买策卖策可以进一步的增加，再算力许可的范围内也可以进行多组合之间的叠加，但要调优回测时的数据量就更大了。
4. 回测股票过少，可以增加为全市场的股票，但相应的计算量增大。
5. 还可以结合机器学习的方法实现一定的预测功能，但是就需要很大的数据去训练，且预测结果也很难保证
6. 调优模型只用了网格搜索模型，还可以增加别的模型来进行参数调优
7. 另外还可以进行仓位管理的设定，在一定的策略下实现仓位的合理控制。
8. 可以用UMP裁决的方法把一些高风险，无效的交易舍去。