介绍一套比较简单且完备的量化框架，分

**数据采集，数据预处理，利用SVM算法进行建模**。

>> 数据采集 <<

本系列的量化框架，全部采用本地化计算。为什么要本地化计算呢，因为相比在线获取数据进行分析计算，本地化计算有如下优势：

* 1. **稳定**——不会因网络不稳定而导致分析过程中断。
* 2. **快速**——本地化运算对于数据的访问速度比在线获取数据快，当机器学习的算法涉及到海量数据做训练集或迭代训练的时候，这一点尤其重要。
* 3. **可复用**——无论基础的行情数据还是加工处理后的数据，保存在本地后，对于后续进行结果分析或策略优化时更为方便。

我们进行本地化计算，首先要做的，就是将所需的基础数据采集到本地数据库里，本篇的示例源码采用的数据库是**MySQL5.5**，数据源是**tushare pro**接口。

我们现在要取一批特定股票的日线行情，部分代码如下：

# 设置tushare pro的token并获取连接

ts.set\_token('xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx')

pro = ts.pro\_api()

# 设定获取日线行情的初始日期和终止日期，其中终止日期设定为昨天。

start\_dt = '20100101'

time\_temp = datetime.datetime.now() - datetime.timedelta(days=1)

end\_dt = time\_temp.strftime('%Y%m%d')

# 建立数据库连接,剔除已入库的部分

db = pymysql.connect(host='127.0.0.1', user='root', passwd='admin', db='stock', charset='utf8')

cursor = db.cursor()

# 设定需要获取数据的股票池

stock\_pool = ['603912.SH','300666.SZ','300618.SZ','002049.SZ','300672.SZ']

total = len(stock\_pool)

# 循环获取单个股票的日线行情

for i in range(len(stock\_pool)):

try:

df = pro.daily(ts\_code=stock\_pool[i], start\_date=start\_dt, end\_date=end\_dt)

# 打印进度

print('Seq: ' + str(i+1) + ' of ' + str(total) + ' Code: ' + str(stock\_pool[i]))

上述代码的注释部分已将每行代码的功能解释清楚了，实际上数据采集的程序主要设置三个参数：获取行情的初始日期，终止日期，以及股票代码池。

当我们获取数据后，就要往本地数据库进行写入（存储）操作了，本篇代码用的是SQL语言，需提前在数据库内建好相应的表，表配置和表结构如下：

**库名：stock       表名：stock\_all**  


其中 state\_dt 和 stock\_code 是**主键和索引**。state\_dt 的格式是 ‘yyyy-mm-dd’（例：'2018-06-11'）。这样的日期格式便于查询，且在MySQL内部能够进行大小比较。  
（完整的数据采集代码详见 [Init\_StockAll\_Sp.py](http://tushare.org/pro/u1/qt_1.rar) 文件）

>> 数据预处理 <<

无论是量化策略还是单纯的机器学习项目，数据预处理都是非常重要的一环。以机器学习的视角来看，数据预处理主要包括数据清洗，排序，缺失值或异常值处理，统计量分析，相关性分析，主成分分析（PCA），归一化等。本篇所要介绍的数据预处理比较简单，只是将存在本地数据库的日线行情数据整合成一份**训练集数据**，以用于后续的机器学习**建模和训练**。

在介绍具体的示例代码之前，我们需要先思考一个问题，应用有监督学习的算法对个股进行建模，我们的输入数据有哪些，我们期望得到的输出数据又是什么？

这个问题的答案因人而异，因策略而异。这个问题本身是**将市场问题转化为数学问题**的一个过程。依赖的是量化宽客自己的知识体系和对市场的理解。

回到正题，本篇示例我们将以最简单的数据进行分析，我们**输入端的数据是个股每日基础行情，输出端数据是股价相较前一交易日的涨跌状态。**简单点说就是，我们向模型输入今天的基础行情，让模型预测明天股价是涨还是跌。

在代码实现方式上，我们采用面向对象的思想，将整个数据预处理过程和结果，**封装成一个类**，每次创建一个类实例，就得到了特定条件下的一份**训练集**。示例代码如下：

class data\_collect(object):

def \_\_init\_\_(self, in\_code,start\_dt,end\_dt):

ans = self.collectDATA(in\_code,start\_dt,end\_dt)

def collectDATA(self,in\_code,start\_dt,end\_dt):

# 建立数据库连接，获取日线基础行情(开盘价，收盘价，最高价，最低价，成交量，成交额)

db = pymysql.connect(host='127.0.0.1', user='root', passwd='admin', db='stock', charset='utf8')

cursor = db.cursor()

sql\_done\_set = "SELECT \* FROM stock\_all a where stock\_code = '%s' and state\_dt >= '%s' and state\_dt <= '%s' order by state\_dt asc" % (in\_code, start\_dt, end\_dt)

cursor.execute(sql\_done\_set)

done\_set = cursor.fetchall()

if len(done\_set) == 0:

raise Exception

self.date\_seq = []

self.open\_list = []

self.close\_list = []

self.high\_list = []

self.low\_list = []

self.vol\_list = []

self.amount\_list = []

for i in range(len(done\_set)):

self.date\_seq.append(done\_set[i][0])

self.open\_list.append(float(done\_set[i][2]))

self.close\_list.append(float(done\_set[i][3]))

self.high\_list.append(float(done\_set[i][4]))

self.low\_list.append(float(done\_set[i][5]))

self.vol\_list.append(float(done\_set[i][6]))

self.amount\_list.append(float(done\_set[i][7]))

cursor.close()

db.close()

# 将日线行情整合为训练集(其中self.train是输入集，self.target是输出集，self.test\_case是end\_dt那天的单条测试输入)

self.data\_train = []

self.data\_target = []

最终这个类实例化后是要整合出三个数据：

* 1. **self.train** ：训练集中的输入端数据，本例中是每日基础行情。
* 2. **self.target** ：训练集中的输出数据，本例中相较于前一天股价的涨跌，涨为1，不涨为0。并且在排序上，每条 t 交易日的self.train里的数据对应的是 t+1 天股价的涨跌状态。
* 3. **self.test\_case** ：在 t 末交易日的基础行情数据，作为输入端，用于模型训练完成后，对第二天的涨跌进行预测。

（完整的数据预处理代码详见 [DC.py](http://tushare.org/pro/u1/qt_1.rar) 文件）

>> SVM建模 <<

机器学习中有诸多有监督学习算法，SVM是比较常见的一种，本例采用SVM算法进行建模。关于SVM的理论原理本篇不做详述，以下仅从实践角度进行建模介绍。

先贴一段建模、训练并进行预测的代码大家感受一下：）

model = svm.SVC() # 建模

model.fit(train, target) # 训练

ans2 = model.predict(test\_case) # 预测

三行代码，让人想起了把大象装冰箱分几步的冷笑话……

不过这侧面也说明Python在数据挖掘方面的强大之处：简单，方便，好用。

本例用的机器学习框架是**scikit-learn**。是个非常强大的算法库，熟悉算法原理的朋友可以查阅官方API文档，可修改模型参数，进一步调优模型；亦可尝试其他算法比如**决策树，逻辑回归，朴素贝叶斯**等。

（完整的SVM建模代码详见 [SVM.py](http://tushare.org/pro/u1/qt_1.rar) 文件）

最后，我们虽然顺利建模并作出预测，但仍面对两个主要问题：1.模型预测能力如何？或者说该如何评估一个模型的质量？2.该如何结合模型进行仓位管理？风险如何？如何量化？

在上一篇中，我们介绍了简单的数据收集，数据预处理与建模案例，本篇承接上篇内容，主要介绍两部分：**模型评估，仓位管理**。

### >> 模型评估 <<

在机器学习领域，有诸多评估模型的方法和指标，本篇介绍一个最简单常用的指标——**F1分值**。

要计算模型的F1分值，就要了解**混淆矩阵**。但在介绍混淆矩阵之前，我们先从实际场景出发，去理解模型评估的思路。

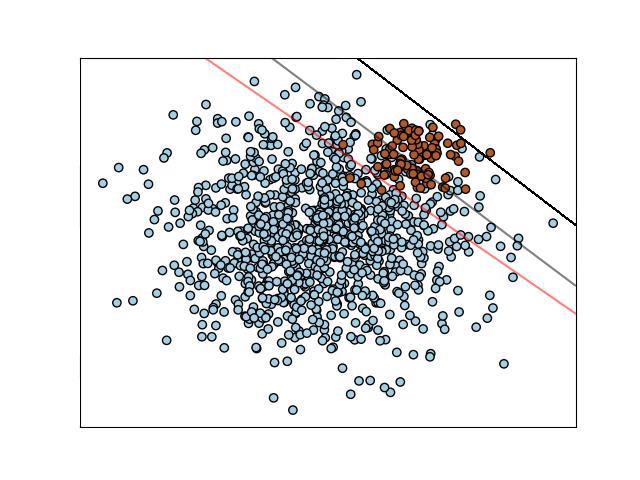
当我们想设计一个模型评估的指标时，一个朴素的想法就是准确率，即模型正确预测的数量，**Acc = T（正确预测的次数）/ N（总预测次数）**。但在实际的场景中，我们关注的往往是单边的预测准确率，比如在上篇的数据建模里，想要盈利，我们最关心的是预测股价上涨的概率，而预测不上涨的概率则不那么重要；那么公式就需要稍微修改一下：**Acc = Tp（预测为上涨且预测正确的次数）/ N（总预测次数）**。这个公式虽然能描述正样本的预测准确率，但却有个**致命缺陷——当模型把所有样本都预测为正样本时，该公式的准确率是100%**！反映在实际场景中，就是无脑买多，这显然是不合理的。究其根本，问题出在公式的分母中，由于无论如何预测，分母都是不变的，那么显然把所有样本点都预测为正得到的准确率最大。为此，我们还需要修改公式，而且是从分母入手，可以有两种方案：

* **1. Acc（Precision） = Tp（预测为上涨且正确的次数）/ Tp + Fp(预测为上涨但实际不上涨的次数)。 该公式的分母，是所有预测为上涨（无论实际上涨与否）的次数。**
* **2. Acc（Recall） = Tp（预测为上涨且正确的次数）/ Tp + Fn(预测不上涨但实际上涨的次数)。 该公式的分母，是所有实际上涨（无论预测上涨与否）的次数。**

这两个公式没有好坏优劣之分，而是通过不同的角度来对模型进行评价。

现在，我们再来看混淆矩阵就非常简单了：  


上面的两个公式，实际上是有区分的，第一个公式计算得到的值叫**精度（也叫查准率）**，反映的是**预测能力**。第二个公式计算得到的值叫**召回率（也叫查全率）**，反映的是对正样本的**拟合能力**。  
下面我们结合一个实际的场景来解释这个混淆矩阵（图片来自sklearn官档，略作修改，侵删）：



**样本背景**：在过去n个交易日中（即图中圆点个数），个股股价相较于前一交易日，**涨的话是红色点，不涨是蓝色点**。横纵坐标分别是个股的两个自定义的**状态**（比如上篇中的收盘价和成交量）。

现在我们有一个最简单粗暴的分类预测器——**画线，而且是直线**。

用第一个公式计算的话，**Precision = 图中直线右上的所有红点个数 / 图中直线右上所有（红点 + 蓝点）的个数**。显然图中红灰黑三条直线中，**黑线的Precision最高，高达100%**，但在所有红点中占比很小，反映在实际场景中，即遇到机会非常少，实用性下降。

再看第二个公式，**Recall= 图中直线右上的所有红点个数 / 图中所有红点个数**。显然图中红灰黑三条直线中，**红线的Recall最大，接近100%**，但同时也会有许多误判（红线右上的蓝点都被预测成了红点），反映在实际场景中，即遇到的机会增大，但误判的概率也变大了。

回到这个场景和图片本身，一个真正好的分类预测模型，应该是类似图中灰色直线的效果，**在Precision和Recall之间取一个平衡，在两者间取加权平均值的话就是F1分值**：

**F1 = ( 2 \* Precision \* Recall ) / ( Precision + Recall )**

图中红灰黑三条直线的F1分值，灰色线的F1最高，而红线由于Precision较低，黑线由于Recall较低，F1分值不会太高。

接下来，我们通过代码来实现计算F1分值并评估模型，在实践中，我们不仅记录F1分值，同时还记录Precision和Recall以及负样本的Precision，综合这几个指标**可以粗略判断出模型的状态，过拟合或欠拟合，从而为优化指出方向**。

代码的实现需要用到数据库操作，主要用到两张表，一张是**结果表**，用于记录模型的F1分值等。另一张是**中间表**，用于存储F1计算过程的一些变量，功能上与内存相似。

**结果表——库名：stock       表名：model\_ev\_resu**

| **字段名** | **字段类型** | **字段说明** |
| --- | --- | --- |
| state\_dt | varchar2(45) | 评估日期 |
| stock\_code | varchar2(45) | 股票代码 |
| acc | decimal(20, 4) | 查准率 |
| recall | decimal(20, 4) | 查全率 |
| f1 | decimal(20, 4) | f1分值 |
| acc\_neg | decimal(20, 4) | 查准率（负样本） |
| bz | varchar2(45) | 用于标注模型类别，比如svm、决策树等 |
| predict | varchar2(45) | 对评估日后一个交易日的预测值 |

**中间表——库名：stock       表名：model\_ev\_mid**

| **字段名** | **字段类型** | **字段说明** |
| --- | --- | --- |
| state\_dt | varchar2(45) | 回测日期 |
| stock\_code | varchar2(45) | 股票代码 |
| resu\_predict | decimal(20, 2) | 预测值 |
| resu\_real | decimal(20, 2) | 真实值 |

在数据库内建好两张表，就可以对模型进行评估了，本篇代码用的是推进式建模（即每天获得最新的股票数据后添加到训练集中，重新建模并对第二天进行预测），部分代码如下：

# 计算查全率

sql\_resu\_recall\_son = "select count(\*) from model\_ev\_mid a where a.resu\_real is not null and a.resu\_predict = 1 and a.resu\_real = 1"

cursor.execute(sql\_resu\_recall\_son)

recall\_son = cursor.fetchall()[0][0]

sql\_resu\_recall\_mon = "select count(\*) from model\_ev\_mid a where a.resu\_real is not null and a.resu\_real = 1"

cursor.execute(sql\_resu\_recall\_mon)

recall\_mon = cursor.fetchall()[0][0]

recall = recall\_son / recall\_mon

# 计算查准率

sql\_resu\_acc\_son = "select count(\*) from model\_ev\_mid a where a.resu\_real is not null and a.resu\_predict = 1 and a.resu\_real = 1"

cursor.execute(sql\_resu\_acc\_son)

acc\_son = cursor.fetchall()[0][0]

sql\_resu\_acc\_mon = "select count(\*) from model\_ev\_mid a where a.resu\_real is not null and a.resu\_predict = 1"

cursor.execute(sql\_resu\_acc\_mon)

acc\_mon = cursor.fetchall()[0][0]

if acc\_mon == 0:

acc = recall = acc\_neg = f1 = 0

else:

acc = acc\_son / acc\_mon

基本的实现思路是：

* **1. 建回测时间序列。**
* **2. 进行第一次时间序列的遍历，推进式建模，向中间表存入相关过程变量（包含每次的预测值）。**
* **3. 进行第二次时间序列的遍历，向中间表中更新每次迭代的真实值。**
* **4. 对中间表进行统计，计算Precision，Recall，F1分值等，并存入结果表。**

（完整的模型评估代码详见 [Model\_Evaluate.py](http://tushare.org/pro/u1/k2.rar) 文件）

### >> 仓位管理 <<

在投资领域，交易择时和风险控制是同等重要的两大模块。前述机器学习的模型解决了交易择时的问题，而马科维茨投资组合理论，则是在投资组合确定的条件下，通过仓位配比来实现风险控制的强大工具。

我们取下面5只股票作为一套投资组合

| **股票代码** | **名称** | **行业** |
| --- | --- | --- |
| 603912 | 佳力图 | 通用设备/5G/次新股 |
| 300666 | 江丰电子 | 半导体/芯片/次新股 |
| 300618 | 寒锐钴业 | 有色金属/锂电池 |
| 002049 | 紫光国芯 | 半导体/5G/两融股 |
| 300672 | 国科微 | 半导体/芯片/次新股 |

现在我们计算一下2018年1月1日的**风险系数**和其对应的**头寸比例**，采样长度90天。

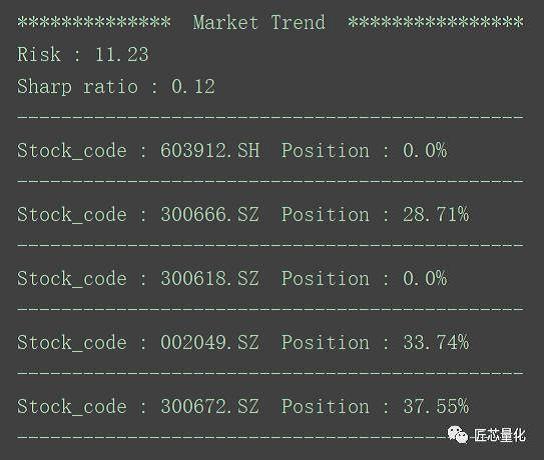
计算的过程非常简单，步骤如下：

* 1. 计算投资组合在采样区间内的**每日收益率**，组成一个 m\*n 的矩阵A（其中m为交易日天数，n为投资组合股票或基金数，m必须大于n）。
* 2. 计算该矩阵A的**协方差矩阵**。
* 3. 求解该协方差矩阵的**特征值和特征向量**。
* 4. （可选） 计算**夏普率**。

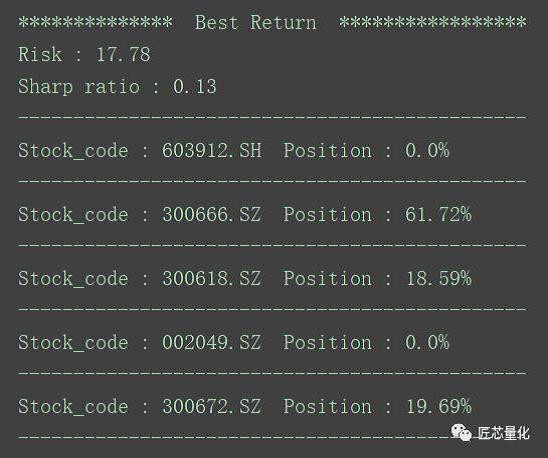
在求解特征值和特征向量后，我们能得到若干（不大于n）个特征值和其对应的特征向量。这些特征值就是马科维茨理论中的投资组合的**风险**，其对应的特征向量做**归一化处理**后就是投资组合中各个股票或基金的**头寸比例**。

注意，特征向量中会有负值，一起归一化后是作为卖空账户的保证金头寸比例的，鉴于A股融券市场的高门槛和扭曲，不建议做空操作。号主是将特征向量中的负值剔除，将剩余的做归一化处理。

由于特征值和特征向量有多组，显然我们倾向于选风险小的，也即特征值最小的，这代表着市场的方向。本篇的这套portfolio运算后的风险和头寸如下：



市场的方向往往也是大盘的方向，投资组合的收益围绕大盘小范围波动。对于激进型投资者来说，需要一个风险稍稍提高但收益带来明显提升的方案。对于这种需求，则是取**次最小的特征值和特征向量**，剔除负值并归一化，如下图：



（完整的模型评估代码详见 [Portfolio.py](http://tushare.org/pro/u1/k2.rar) 文件）

可见，两套头寸配比差异较大，那么这两套方案的实际表现如何呢，请关注下期内容：

当我们的经验和策略通过代码的方式实现时，除了一些机器学习的评估方法，还需要通过模拟交易的方式来回测整个策略。策略整体在市场中的表现效果如何，该如何用量化的手段来评估，则是本篇要向大家介绍的内容——**模拟交易与回测**。

话不多说马上进入正题，我们现在要做的，就是构建一套自己的模拟交易系统，并用这套系统来回测各种策略。为了让本文更接地气，作者不打算画各种程序流程图或拓扑图等，这样的 “**PPT Style**” 太不接地气了，我们换成以一个**交易员**的视角来思考问题。

**>> 模拟交易 <<**

股市的交易规则是实时的撮合交易，我们没办法也没必要做到实时的撮合交易，所以在模拟交易系统里，交易规则要简化为：**以下单当日的收盘价作为成交价**。实际上，各大量化平台也是这么做的，做得更细致一点的，可以设定一个参数：“**滑点**”——来控制模拟交易和实盘交易的误差。

有了这样一种简化，模拟交易就变得十分简单了——复杂的撮合交易机制简化成了以股票行情的收盘价作为成交价，剩下的只是简单的“**买**”和“**卖**”的交易动作。作为交易的基本动作，号主专门用一个程序来封装，代码如下：Operator.py

import pymysql.cursors

import Deal

def buy(stock\_code, opdate, buy\_money):...

def sell(stock\_code, opdate, predict):...

可见只是封装了 **buy** 和 **sell** 两个函数而已。

在buy函数里我定义了几个参数，分别是**股票代码、交易日期、交易金额**；在sell函数里定义的几个参数分别是 **股票代码、交易日期、交易量、交易类型（主动卖或止损卖）**。这几个参数见名知意，这里就不多解释了。

现在我们有了模拟交易的两个基本操作函数，但交易是双向的，我们买和卖的结果体现在哪里呢，这就需要一个**资产账户**来记录。细心的读者肯定发现了，上面的截图里引入了两个包，一个是数据库框架pymysql，另一个则是Deal包，这其实是号主自定义的一个python程序，也就是模拟交易中的**资产账户**。先来看一下这个Deal文件到底是什么：

import pymysql.cursors

class Deal(object):

cur\_capital = 0.00

cur\_money\_lock = 0.00

cur\_money\_rest = 0.00

stock\_pool = []

stock\_map1 = {}

stock\_map2 = {}

stock\_map3 = {}

stock\_all = []

ban\_list = []

...

可见，**Deal**类封装了一些参数，初始化函数就是为了更新这些参数。实际上，这些参数分别是**账户总资产，股票资产，现金资产，股票池，股票资产详情**等，整个Deal类就是一份**资产账户详单**。

关于资产账户的数据架构，底层的实现是mysql数据库，分成两张sql表来实现，一张是**账本表**（记录每一次的买和卖操作），表结构如下：

库名：**stock**               表名：**my\_capital**

| **字段名** | **字段类型** | **字段说明** |
| --- | --- | --- |
| capital | DECIMAL(20, 4) | 总资产 |
| money\_lock | DECIMAL(20, 4) | 股票资产 |
| money\_rest | DECIMAL(20, 4) | 现金资产 |
| deal\_action | VARCHAR2(45) | 交易动作 |
| stock\_code | VARCHAR2(45) | 股票代码 |
| deal\_price | DECIMAL(20, 4) | 成交价 |
| stock\_vol | INT(11) | 成交量 |
| profit | DECIMAL(20, 4) | 收益额 |
| profit\_rate | DECIMAL(20, 4) | 收益率 |
| bz | VARCHAR2(45) | 备注 |
| state\_dt | VARCHAR2(45 | 交易日期 |
| seq | INT(11) | 序号（用作表主键） |

另一张则是持仓表，表结构如下：

库名：**stock**               表名：**my\_stock\_pool**

| **字段名** | **字段类型** | **字段说明** |
| --- | --- | --- |
| stock\_code | VARCHAR2(45) | 股票代码 |
| buy\_price | DECIMAL(20, 2) | 买入价 |
| hold\_vol | INT(11) | 持仓量（单位：股） |
| hold\_days | INT(11) | 持仓天数（只计算交易日） |

对于交易来说，只需要持仓股票代码和持仓量即可，买入价是为了测算收益，持仓天数则是为了某些策略用的（比如策略对持仓天数有限制时）。

至此，一个最简单的模拟交易过程就完成了，从交易的角度来看，就是通过buy和sell函数对**Deal类（资产账户）**里的数据做**写操作**，比如，买入股票就是扣除现金资产，增加股票资产，同时在持仓表中增加相应记录；卖出股票则是反向操作。

**>> 策略回测 <<**

有了上述一套模拟交易过程，接下来我们要考虑的就是策略层面的问题了。从交易的角度来看，策略是整个交易过程的入口，是逻辑和决策层。笔者直接用main函数来写策略了：

if \_\_name\_\_ == '\_\_main\_\_':

# 先清空之前的测试记录

# 建回测时间序列

# 开始模拟交易

for i in range(1, len(date\_seq)):

# 选股初始化模块

# 交易预警模块

# 模型训练模块

# 买卖点判断模块（包括但不限于模型的预测结果）

# 仓位管理

# 交易执行模块

# 结果数据可视化模块

代码中简明清晰地展示了号主策略的回测框架：首先清空之前的测试记录，然后**取回测时间段内的交易日序列**，通过for循环来遍历这个序列，每一次迭代，都是一个交易日，都包含了策略的多个功能模块，上一篇的策略并非全部用到这些模块，未用到的下面以“可选”标记：

**选股初始化模块（可选）：**这个模块的功能主要是选股，由于涉及的逻辑和计算量可能非常庞大，并非每日执行，可以每隔x个交易日执行一次。

**交易预警模块（可选）：**当模型的预测存在结构性误差时，往往需要该预警模块来作为买卖点判断的补充，比如大趋势转变，基本面变化，政策变化等。

**模型训练模块：**在策略中，建模方式分单次建模和推进建模，区别是推进建模每日收盘后会根据最新交易日的数据进行重训练，对于推进建模，该模块是必须的（在上篇的策略中，就是应用的推进建模）。

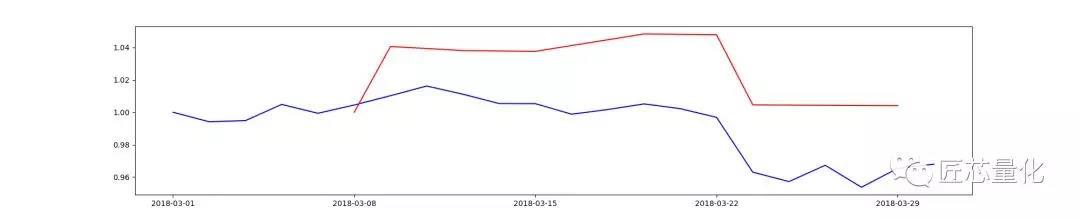
**买卖点判断：**包含但不限于模型的预测结果，往往结合其他的逻辑或信号进行判断（比如预警模块给出的信号），最终确定是否买卖。

**仓位管理：**确定交易股票的配仓（买入金额或卖出股数）。

**交易执行模块：**即上文详述的模拟交易过程，Operator.py里的buy和sell函数。

**结果数据可视化模块：**当跑完回测后，给出一个直观的结果（折线图，柱图等）。

接下来，让我们看一下上一篇中的那套portfolio的回测结果。为了跟测试集的时间序列保持连续，现在**取2018.03.01~2018.04.01区间的交易日序列作为回测区间**。首先来看一下投资组合的**市场方向（特征值最小）**的收益情况：



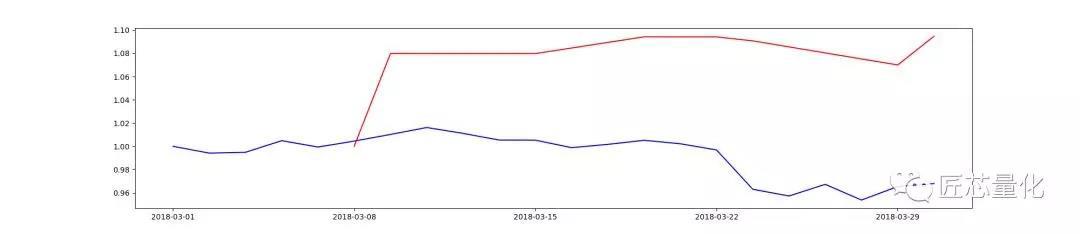
图中**蓝线**代表**大盘的收益曲线**（收益 = 当日收盘价 / 首日收盘价），在这一个月的回测周期中，**大盘指数**由 **3273**点震荡到 **3168** 点，**投资组合**的收益曲线跟大盘趋势基本**保持一致**，在期末的收益率为 **0.39%** ，略微跑赢大盘。

接下来再看这套投资组合的账单表：



总计 **7** 次卖出操作，其中 **3** 次止盈，**3** 次超时平仓，**1** 次止损。从收益情况来看，**7** 次操作中 **5** 次盈利，**2** 次亏损。

作为对比，我们再来看一下投资组合的**最大收益方向（次最小特征值）**：



可见，投资组合的收益曲线**背离大盘**。在期末的收益率达到 **9.45%**，明显跑赢市场。接下来再看这套投资组合的账单详情：



总计 **9** 次卖出操作，其中 **4** 次止盈，**2** 次超时平仓，**2** 次止损，**1** 次预测卖。从收益情况来看，**7** 次盈利，**2** 次亏损。

**与市场方向相比，操作数变多，止盈和止损次数均多于前次。从操作和收益来看也印证了“高风险高收益”的道理。**

至此，**构建投资组合==>回测验证策略** 的流程已经结束。详细代码清单与功能如下（[点击这里下载全部代码](http://tushare.org/pro/code_3.rar)）：

| **文件名** | **功能** |
| --- | --- |
| DC.py | 【数据预处理】将本地存储的日基础行情整合成一份训练集。 |
| Model\_Evaluate.py | 【模型评估】通过回测+推进式建模的方式对模型进行评估，主要计算查准率Precision，查全率Recall，F1分值，并存入结果表。 |
| Portfolio.py | 【仓位管理】基于马科维茨投资组合理论，计算一段时间序列内投资组合的风险、仓位配比和夏普率，有市场方向和最佳收益方向两种结果。 |
| Deal.py.py | 【模拟交易】封装类，用于模拟交易过程中获取最新的资产账户相关数据。 |
| Operator.py | 【模拟交易】封装函数，用于模拟交易过程中执行买和卖操作。 |
| Cap\_Update\_daily.py | 【模拟交易】封装函数，用于在回测过程中，每日更新资产表中相关数据。 |
| Filter.py | 【策略回测】封装函数，用于在回测过程中，处理一些简单的逻辑（更新持仓天数，买卖顺序等）。 |
| main.py | 【策略回测】策略的框架，回测的主函数。 |