# 项目

**一、项目题目：Adaboost 与多因子模型在 A 股市场的应用与分析**

**二、数据集说明：**

本项目所用的股票以及因子数据均来源于 AutoTrader 平台

三、理论算法介绍：

项目简介

本项目主要研究的是 Adaboost 算法与多因子选股模型在 A 股市场的应用研究，通过机器学习算法，提高模型选股的性能，更好地解释我国的股票市场。首先通过 Atrader 平台获取上证 50 成份股 2016 年 1 月至 2019 年 5 月每月最后一个交易日的股票数据作为数据样本，这个时间段中包含了一轮牛市与熊市的交替，以此进行回测所得的结果更具说服力。选择上证 50 成份股作为备选股票池，主要是因为上证 50 成份股多为规模较大、流动性较好的股票，具有良好的 A 股市场代表性。通过查阅相关文献，本文首先选取了 22 个因子作为候选因子，总共分为八个大类，其中包括质量类因子、成才类因子、估值类因子、规模类因子、交易类因子、情绪类因子、技术指标类因子和 WorldOuant Alpha101 部分因子。由于候选因子数较多，通过 IC 值筛选从 22 个候选因子中选出 10 个最有效因子；然后引进 AdaBoost 算法对传统的多因子模型进行优化，逐步改进模型改并且使用 Atrader 平台进行回测；将 AdaBoost 算法与多因子模型结合并和传统多因子模型的选股效果对比，我们发现机器学习 AdaBoost 算法能增强传统多因子模型的选股效果，收益率显著增加。

**项目算法简介**

Adaboost:

Boosting，也称为增强学习或提升法， 能够将预测精度仅比随机猜度略高的弱学习器增强为预测精度高的强学习器。AdaBoost(Adaptive Boosting)(自适应增强）的自适应在于：前一个基本分类器被错误分类的样本的权值会增大，而正确分类的样本的权值会减小，并再次用来训练下一个基本分类器。同时，在每一轮迭代中，加入一个新的弱分类器，直到达到某个预定的足够小的错误率或达到预先指定的最大迭代次数才确定最终的强分类器。目的是从训练数据中学习一系列弱分类器或基本分类器，然后将这些弱分类器组合成一个强分类器。

本项目核心步骤是每次训练上一个交易周期的训练数据，然后将当前的因子

数据输入模型进行预测分类，来获取收益率为正的股票代码，进行委托下单。每个交易周期训练一次模型，让模型能够适应市场的变化

**四、 程序的结构和算法描述：**

**1. 有效因子的选取**

**1.1.候选因子的选择**

这一步是多因子选股模型比较重要的步骤，选取了什么样的因子会直接影响模型的有效性。在选取因子的时候，我们尽可能多地考虑到了不同的角度，借鉴了之前的研究和学者的经验加入部分因子（见参考文献），并且根据市场的具体情况选择市场敏感度高的因子。

**表 1-1：候选因子表**

| 大类因子 | 简称 | 因子名称 |
|---|---|---|
| 质量类 | CurrentRatio | 流动比率 |
| | NetProfitRatio | 销售净利率 |
| | ROE | 权益回报率 |
| 情绪类 | VSTD10 | 10 日成交量标准差 |
| | TVMA20 | 10 日平均换手率 |
| | VR | 成交量比率 |
| 成长类 | NetProfitGrowRate | 净利润增长率 |
| 常用技术指标类 | BOLLDOWN | 下轨线（布林线）指标 |
| | MTM10 | 动量指标 |
| | KDJ_D | 随机指标 D |
| 动量类 | BBI | 多空指数 |
| | BIAS20 | 20 日乖离率 |
| | DDI | 方向标准离差指数 |
| 价值类 | NegMktValue | 流通市值 |
| | PE | 市盈率 |
| | PS | 市销率 |
| | PB | 市净率 |
| 每股指标类 | MktValue | 总市值 |
| | BasicEPS | 基本每股收益 |
| | EPS | 每股收益 TTM 值 |
| 特色技术指标类 | NATR | 归一化平均真实范围 |
| | STDDEV | 标准差 |

数据来源：点宽网

## 1.2.数据预处理

在进行模型构建与分析之前，为了提高数据的质量，需要对数据进行预处理。数据预处理有多种方法，包括数据清理，数据集成，数据变换，数据归约等，本项目主要对数据进行以下预处理：

**缺失值处理**：在模型训练的时候，若出现某一个股票特征值缺失的情况，则使用后一个股票的数据进行填补，若填补完后仍存在缺失情况，说明本次获取的该特征值均为 NaN 值，直接删除该特征值。

| | ROE | VSTD10 | TVMA20 | NetProfitGrowRate | MTM10 | NegMktValue | PE | PB | MktValue | NATR | Sharperatio20 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 | 0.16713 | 2.0924e+07 | 4365.5 | -0.2796 | -9.41 | 5.9818e+11 | 9.7824 | 2.0112 | 1.0094e+12 | 3.166 | -3.1946 |
| 32 | 0.10692 | 5.2254e+07 | 367.05 | -0.46515 | -0.41 | 2.198e+11 | 5.8829 | 0.6269 | 4.1587e+11 | 1.5365 | -6.4715 |
| 33 | 0.093999 | 9.3219e+06 | 883.68 | -0.30612 | -6.28 | 8.5399e+10 | 20.748 | 1.9506 | 1.2775e+11 | 3.6843 | -1.1638 |
| 34 | 0.084959 | 2.3238 | 0 | -0.027665 | 0 | 1.3691e+11 | 12.014 | 1.0652 | 1.7065e+11 | 94.921 | NaN |
| 35 | 0.13295 | 9.756e+07 | 988.67 | -0.4833 | -0.62 | 1.3696e+12 | 6.2986 | 0.8338 | 1.8105e+12 | 2.4412 | -5.1628 |
| 36 | 0.11539 | 1.1378e+07 | 882.47 | 0.68138 | -6.37 | 1.8716e+11 | 17.996 | 1.9053 | 2.6978e+11 | 4.3523 | -2.8984 |
| 37 | 0.11808 | 8.6771e+06 | 265.73 | -1.0473 | -2.7562 | 4.4812e+11 | 15.158 | 1.8139 | 6.0826e+11 | 2.9855 | -5.0947 |
| 38 | 0.13237 | 9.4269e+07 | 614.27 | 0.81512 | -0.57514 | 2.152e+11 | 4.6531 | 0.9517 | 2.1706e+11 | 2.44 | -4.6262 |
| 39 | 0.068333 | NaN | 78.822 | -0.18363 | NaN | 5.9734e+10 | 9.7558 | 1.0205 | 8.2003e+10 | NaN | -8.3791 |
| 40 | 0.11039 | 1.4879e+07 | 598.2 | -0.48402 | -2.19 | 7.8063e+10 | 10.317 | 1.155 | 1.0271e+11 | 4.0004 | -4.8067 |
| 41 | 0.091044 | 1.6582e+07 | 277.4 | 2.8633 | -1.19 | 1.712e+11 | 16.43 | 1.7466 | 2.1438e+11 | 2.7186 | -6.6274 |
| 42 | 0.10222 | 4.3963e+06 | 88.456 | -0.47998 | -0.76 | 1.2981e+11 | 8.3837 | 0.9737 | 1.7873e+11 | 2.8578 | -5.8223 |
| 43 | 0.10465 | 4.6199e+07 | 298.04 | -0.026247 | -0.38 | 1.3973e+11 | 5.8283 | 0.6023 | 1.8424e+11 | 1.897 | -8.7395 |
| 44 | 0.029729 | 1.1291e+07 | 231.54 | 0.30549 | -0.17733 | 1.216e+12 | 37.362 | 1.1444 | 1.3745e+12 | 2.4095 | -2.8005 |
| 45 | 0.03492 | 2.1897e+07 | 164.15 | 0.25222 | 0.76 | 5.499e+10 | 43.803 | 1.4553 | 5.499e+10 | 3.2557 | 4.6731 |
| 46 | 0.10007 | 6.2741e+06 | 131.68 | 0.27852 | -0.44 | 8.6544e+10 | 10.769 | 1.9455 | 8.6544e+10 | 2.5279 | -3.6407 |
| 47 | 0.11798 | 7.3125e+07 | 398.48 | 0.45942 | -0.28 | 7.3136e+11 | 5.5222 | 0.6791 | 1.0215e+12 | 1.8078 | -5.7585 |
| 48 | -0.0023693 | 2.9245e+07 | 257.28 | 0.20617 | -0.25 | 7.2345e+10 | -454.93 | 1.0741 | 9.0146e+10 | 2.7737 | -5.4614 |
| 49 | 0.10467 | 1.6605e+07 | 136.3 | 0.43727 | -0.09 | 1.9717e+11 | 6.8864 | 0.7435 | 3.0242e+11 | 1.9163 | -2.377 |

图 1.21 存在缺失值的数据

**标准化处理**：由于不同因子所描述的对象单位不一样，那么可能导致不同因子数值差异很大，因此在进行因子测试与数据建模之前，需要对其进行标准化，常用的标准化方法有均值标准差法、最大最小标准化等，本文为了使数据特征更加鲜明，采用了规范化方法，即

对 $x_1, x_2, ..., x_n$ 进行变换

$$y_i = \frac{x_i - \min_{1\le j\le 1}\{x_j\}}{\max_{1\le i\le n}\{x_j\} - \min_{1\le i\le n}\{x_j\}}$$

得到 $y_1, y_2, ..., y_n$ 属于 $[0, 1]$ 且无量纲

| | ROE | VSTD10 | TVMA20 | NetProfitGrowRate | MTM10 | NegMktValue | PE | PB | MktValue | NATR |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0.16359 | 1.46118e+07 | 1314.48019 | 0.85589 | -1.01 | 3.32032e+11 | 6.68502 | 1.0535 | 3.32032e+11 | 4.61273 |
| 1 | -0.03887 | 3.10710e+07 | 279.86534 | -1.38053 | -0.45 | 5.13190e+10 | -54.99040 | 2.2538 | 1.06148e+11 | 3.93135 |
| 2 | 0.16646 | 2.96874e+07 | 636.47676 | -0.96235 | -0.05 | 8.89851e+10 | 6.62020 | 1.0379 | 1.22136e+11 | 3.78071 |
| 3 | 0.15695 | 2.16775e+08 | 1809.20184 | -35.96685 | -0.04 | 2.66852e+11 | 7.13046 | 1.0938 | 3.29463e+11 | 3.73315 |
| 4 | 0.12197 | 9.23385e+06 | 174.00090 | -0.15078 | -0.85 | 1.36076e+11 | 17.59093 | 2.3248 | 1.38579e+11 | 3.12300 |
| 5 | 0.03801 | 3.83566e+07 | 654.56613 | 0.28001 | -0.22 | 4.56766e+11 | 19.38372 | 0.8569 | 5.78720e+11 | 2.25308 |
| 6 | 0.15776 | 1.67199e+08 | 4788.22768 | -19.49041 | -1.41 | 1.71266e+11 | 9.82900 | 1.5196 | 2.11440e+11 | 5.73763 |
| 7 | 0.16835 | 3.61254e+07 | 1215.65266 | -1.24846 | -0.24 | 3.56468e+11 | 7.39695 | 1.2078 | 4.35799e+11 | 2.84204 |
| 8 | 0.17798 | 5.65970e+07 | 1754.52900 | -0.14544 | -1.72 | 1.03573e+11 | 6.62254 | 1.4481 | 1.03573e+11 | 6.26607 |
| 9 | 0.04096 | 5.83223e+07 | 592.71517 | -0.08978 | -0.48 | 1.22940e+11 | 12.86810 | 1.5625 | 1.22940e+11 | 3.55813 |
| 10 | 0.19363 | 1.06867e+07 | 376.82454 | 0.13041 | -0.97 | 2.17534e+11 | 5.57154 | 1.2421 | 2.17534e+11 | 6.57169 |
| 11 | 0.12272 | 2.83494e+07 | 1048.82435 | 0.57965 | -1.79 | 4.12196e+10 | 22.32553 | 2.6667 | 4.39439e+10 | 5.10795 |
| 12 | 0.02759 | 2.16649e+07 | 602.73856 | -0.61358 | -1.48 | 2.80951e+10 | 166.05442 | 5.5310 | 4.59946e+10 | 4.62895 |
| 13 | -0.00118 | 4.67957e+06 | 474.02238 | -0.16596 | -3.35 | 4.43892e+10 | 321.38880 | 2.5294 | 4.43892e+10 | 4.19126 |
| 14 | 0.15982 | 2.04353e+07 | 698.41330 | -0.01676 | -1.55 | 6.76325e+10 | 23.35449 | 3.6042 | 6.76325e+10 | 22.80420 |
| 15 | 0.27488 | 1.28334e+06 | 461.99783 | -0.31121 | -6.86 | 2.63827e+11 | 15.50055 | 4.1271 | 2.63827e+11 | 2.09879 |
| 16 | 0.12623 | 2.17102e+07 | 469.88870 | 1.54753 | -0.80 | 6.42752e+10 | 9.40824 | 1.2081 | 8.51598e+10 | 3.15316 |
| 17 | 0.04892 | 1.27208e+07 | 971.86666 | 1.25541 | -2.60 | 5.45233e+10 | 124.30440 | 3.6243 | 9.06156e+10 | 4.17593 |
| 18 | 0.12561 | 4.98910e+07 | 398.21545 | -0.37062 | -0.40 | 6.43141e+10 | 7.54236 | 1.4080 | 7.09379e+10 | 2.41302 |
| 19 | 0.13074 | 7.03099e+07 | 1695.46109 | 0.19464 | -1.50 | 1.15556e+11 | 11.23194 | 1.5251 | 1.64244e+11 | 6.72575 |
| 20 | 0.21806 | 1.32755e+08 | 1897.37365 | -0.18440 | -0.95 | 9.02874e+10 | 21.63418 | 4.6037 | 9.20030e+10 | 4.34044 |

图 1.22 标准化处理前的因子值

| | ROE | VSTD10 | TVMA20 | NetProfitGrowRate | MTM10 | NegMktValue | PE | PB | MktValue | NATR |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0.64528 | 0.06185 | 0.24717 | 0.96333 | 0.84783 | 0.24011 | 0.16387 | 0.04604 | 0.18682 | 0.13484 |
| 1 | 0.00000 | 0.13823 | 0.02294 | 0.90482 | 0.92899 | 0.02342 | 0.00000 | 0.30177 | 0.04034 | 0.10243 |
| 2 | 0.65442 | 0.13181 | 0.10023 | 0.91577 | 0.98696 | 0.05250 | 0.16369 | 0.04272 | 0.05071 | 0.09527 |
| 3 | 0.62412 | 1.00000 | 0.35438 | 0.00000 | 0.98841 | 0.18980 | 0.16505 | 0.05463 | 0.18515 | 0.09300 |
| 4 | 0.51263 | 0.03689 | 0.00000 | 0.93700 | 0.87101 | 0.08885 | 0.19284 | 0.31690 | 0.06137 | 0.06399 |
| 5 | 0.24504 | 0.17204 | 0.10415 | 0.94827 | 0.96232 | 0.33640 | 0.19760 | 0.00415 | 0.34679 | 0.02261 |
| 6 | 0.62672 | 0.76994 | 1.00000 | 0.43105 | 0.78986 | 0.11601 | 0.17222 | 0.14535 | 0.10862 | 0.18833 |
| 7 | 0.66045 | 0.16169 | 0.22575 | 0.90828 | 0.95942 | 0.25897 | 0.16576 | 0.07892 | 0.25411 | 0.05062 |
| 8 | 0.69115 | 0.25669 | 0.34253 | 0.93714 | 0.74493 | 0.06376 | 0.16370 | 0.13011 | 0.03867 | 0.21347 |
| 9 | 0.25444 | 0.26469 | 0.09074 | 0.93859 | 0.92464 | 0.07871 | 0.18029 | 0.15449 | 0.05123 | 0.08468 |
| 10 | 0.74102 | 0.04364 | 0.04396 | 0.94435 | 0.85362 | 0.15173 | 0.16091 | 0.08622 | 0.11257 | 0.22800 |
| 11 | 0.51503 | 0.12560 | 0.18959 | 0.95611 | 0.73478 | 0.01562 | 0.20542 | 0.38974 | 0.00000 | 0.15839 |
| 12 | 0.21183 | 0.09458 | 0.09292 | 0.92489 | 0.77971 | 0.00549 | 0.58729 | 1.00000 | 0.00133 | 0.13561 |
| 13 | 0.12015 | 0.01576 | 0.06502 | 0.93660 | 0.50870 | 0.01807 | 1.00000 | 0.36049 | 0.00000 | 0.11479 |
| 14 | 0.63328 | 0.08888 | 0.11365 | 0.94050 | 0.76957 | 0.03601 | 0.20815 | 0.58948 | 0.01536 | 1.00000 |
| 15 | 1.00000 | 0.00000 | 0.06241 | 0.93280 | 0.00000 | 0.18746 | 0.18729 | 0.70089 | 0.14259 | 0.01528 |
| 16 | 0.52622 | 0.09479 | 0.06413 | 0.98143 | 0.87826 | 0.03342 | 0.17110 | 0.07898 | 0.02673 | 0.06542 |
| 17 | 0.27981 | 0.05308 | 0.17291 | 0.97378 | 0.61739 | 0.02589 | 0.47637 | 0.59377 | 0.03027 | 0.11406 |
| 18 | 0.52423 | 0.22557 | 0.04859 | 0.93125 | 0.93623 | 0.03345 | 0.16614 | 0.12157 | 0.01751 | 0.03022 |
| 19 | 0.54060 | 0.32032 | 0.32973 | 0.94603 | 0.77681 | 0.07301 | 0.17595 | 0.14652 | 0.07801 | 0.23533 |
| 20 | 0.81889 | 0.61010 | 0.37349 | 0.93612 | 0.85652 | 0.05350 | 0.20358 | 0.80243 | 0.03117 | 0.12189 |

图 1.23 标准化处理后的因子值

**定义预测标签：**将每个股票根据夏普率因子的正负进行标记，夏普率是一个可以同时对收益与风险加以综合考虑的指标，广泛运用于基金市场，目前已成为国际上用以衡量基金绩效表现的最为常用的一个标准化指标。

夏普比率可以帮助投资者在固定所能承受的风险下，追求最大的报酬；或在固定的预期报酬下，求最低的风险。这就是夏普比率的核心思想。夏普比率的计算非常简单，用净值增长率的平均值减无风险利率再除以基金净值增长率的标准差就可以得到基金的夏普比率

$$Sharpe\ Ratio = \frac{R_p - R_f}{\sigma_p}$$

$Rp$ 是策略的回报率，$Rf$ 为无风险利率，$\sigma p$ 为收益率的标准差

若当前股票的上个交易周期内夏普比率大于零，则将其标记为 1，其他标记为 0。

### 1.3.因子有效性检验及筛选

本文采用因子 IC 值来检验因子的有效性，选取 2016 年 1 月至 2018 年 1 月的数据作为检验因子有效性的样本。

因子 IC（信息系数）：即计算各个股票的因子暴露与各个股票下期收益率的之间的秩相关系数，一般而言，一个因子的 IC 值的绝对值高于 0.02，便可以认为该因子有效性较好.

利用 Atrader 的因子分析平台 API，我们可以列出各候选因子的 IC 值原始数据，如表 1-2

表 1-2：因子 IC 表

| 大类因子 | 因子代码 | 因子名称 | IC 值 | IR 值 |
|---|---|---|---|---|
| 质量类 | CurrentRatio | 流动比率 | 0.0138 | 2.14 |
| | NetProfitRatio | 销售净利率 | 0.0010 | 0.79 |
| | ROE | 权益回报率 | 0.0385 | 20.65 |
| 情绪类 | VSTD10 | 10 日成交量标准差 | -0.0394 | -23.46 |
| | TVMA20 | 10 日平均换手率 | -0.0332 | -21.37 |
| | VR | 成交量比率 | -0.0097 | -6.89 |
| 成长类 | NetProfitGrowRate | 净利润增长率 | -0.0024 | -4.24 |
| 常用技术指标类 | BOLLDOWN | 布林线指标 | 0.0162 | 8.81 |
| | MTM10 | 动量指标 | -0.0303 | -16.02 |
| | KDJ_D | 随机指标 D | -0.0128 | -7.57 |
| | BBI | 多空指数 | 0.0150 | 7.58 |
| 动量类 | BIAS20 | 20 日乖离率 | -0.0189 | -9.96 |
| | DDI | 标准离差指数 | -0.0058 | -3.38 |
| 价值类 | NegMktValue | 流通市值 | 0.0476 | 24.76% |

| | | | | |
|---|---|---|---|---|
| | PE | 市盈率 | -0.0307 | -11.98% |
| | PS | 市销率 | -0.0160 | -23.65% |
| | PB | 市净率 | -0.0220 | -7.61% |
| | MktValue | 总市值 | 0.0473 | 22.26% |
| 每股指标类 | BasicEPS | 基本每股收益 | 0.0127 | 9.55% |
| | EPS | 每股收益 TTM | 0.0109 | 7.99% |
| 特色技术指标类 | NATR | 归一化范围值 | -0.0289 | -20.53% |
| | STDDEV | 标准差 | -0.0148 | -9.95% |

以一个因子的IC值的绝对值高于0.02,便可以认为该因子有效性较好的原则，我们筛选出了以下 10 个因子，表 1-3 所示

表 1-3 因子表

| 大类因子 | 简称 | 因子名称 |
|---|---|---|
| 质量类 | ROE | 权益回报率 |
| 情绪类 | VSTD10 | 10 日成交量标准差 |
| | TVMA20 | 10 日平均换手率 |
| 成长类 | NetProfitGrowRate | 净利润增长率 |
| 常用技术指标类 | MTM10 | 动量指标 |
| 价值类 | NegMktValue | 流通市值 |
| | PE | 市盈率 |
| | PB | 市净率 |
| | MktValue | 总市值 |
| 特色技术指标类 | NATR | 归一化平均真实范围 |

## 2. 机器学习改进传统多因子选股模型

### 2.1 机器学习对模型的改进

为了对机器学习算法进行训练，将股票池中的股票，根据夏普率的正负分为两组，夏普率为正的作为强势股，负的作为弱势股，对于强势股标记为 1，弱势股标记为 0，对 Adaboost 模型进行训练。

本项目核心步骤是每次训练上一个交易周期的训练数据，然后将当前的因子数据输入模型进行预测分类，来获取收益率为正的股票代码，进行委托下单。每个交易周期训练一次模型，让模型能够适应市场的变化

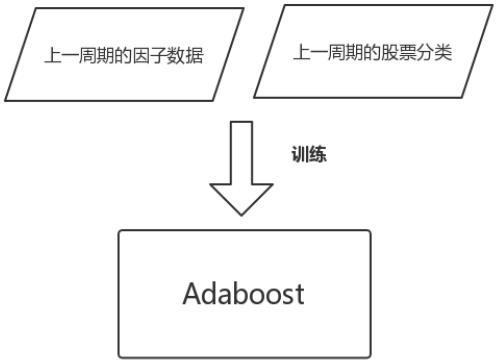



图 2.1 每次先训练模型


图 2.2 再对本周期的股票进行预测

## 五、 实现的源代码：

备注：本项目代码需要在 python3 环境下运行，并且安装 Atrader 库和 Autotrader 软件，运行时需要保证 Autotrader 软件处于开启状态

**Adaboost 策略代码**

```
# -*- coding: utf-8 -*-
from atrader import *
import numpy as np
import pandas as pd
import datetime as dt
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
```

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.pipeline import Pipeline
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.externals import joblib
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import Imputer
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC

'''
策略思路：
1. 回测标的：上证 40 成分股
2. 回测时间段：2016-01-01 至 2019-05-30
3. 特征选择：
质量类
    CurrentRatio     流动比率
    NetProfitRatio   销售净利率
    ROE 权益回报率
情绪类
    VSTD10   10 日成交量标准差
    TVMA20   10 日平均换手率
    VR   成交量比率
成长类
    NetProfitGrowRate    净利润增长率
常用技术指标类
    BOLLDOWN     下轨线（布林线）指标
    MTM10    动量指标
    KDJ_D    随机指标 D
```

```
    BBI 多空指数
动量类
    BIAS20  20 日乖离率
    DDI 方向标准离差指数
价值类
    NegMktValue 流通市值
    PE  市盈率
    PS  市销率
    PB  市净率
    MktValue    总市值
每股指标类
    BasicEPS    基本每股收益
    EPS 每股收益 TTM 值
特色技术指标类
    NATR    归一化平均真实范围
    STDDEV  标准差
'''
#选取的因子列表
factor_list =
['ROE','VSTD10','TVMA20','NetProfitGrowRate','MTM10','NegMktValue','PE','PB','MktValue','NATR','Sharperatio20']

#初始化函数
def init(context):
    # 设置回测初始金额
    set_backtest(initial_cash=10000000)
    # 注册 K 线数据
    reg_kdata('day', 1)
    # 注册因子数据
    reg_factor(factor_list)
    # 设置交易日期
```

```
    days = get_trading_days('SSE', '2016-01-01', '2019-05-30')
    months = np.vectorize(lambda x: x.month)(days)
    month_begin = days[pd.Series(months) != pd.Series(months).shift(1)]
    context.month_begin = pd.Series(month_begin).dt.strftime('%Y-%m-%d').tolist()
    # 模型训练次数
    context.num = 0
    # 建立 Adaboost 模型
    context.pipeline = make_pipeline(preprocessing.StandardScaler(), RandomForestRegressor(n_estimators=100))
    context.hyperparameters = { 'randomforestregressor__max_features' : ['auto', 'sqrt', 'log2'],'randomforestregressor__max_depth': [None, 5, 3, 1]}
    # 读取保存的模型
    #context.clf = joblib.load("model.joblib")
    context.clf = AdaBoostClassifier(DecisionTreeClassifier(max_depth=1),n_estimators=200,algorithm="SAMME.R",learning_rate=0.1)

# 标准化函数
def Get_BZH(list):
    L = (list - min(list))/(max(list) - min(list))
    return L

# 数据处理函数，每次交易都会运行一遍
def on_data(context):
    if dt.datetime.strftime(context.now, '%Y-%m-%d') not in context.month_begin:  # 调仓频率为月
        return

    # 读取因子值
    factor = get_reg_factor(context.reg_factor[0], target_indices=[x for x in range(50)], length=1, df=True).fillna(method = 'backfill', axis = 1)
```

```
    #print(factor)

    # 存储训练特征及标签样本
    FactorData = pd.DataFrame(columns=(['target_idx'] + factor_list)).set_index('target_idx')
    for i in factor_list:
        FactorData[i]=factor[factor['factor'] == i].value.values

    # 标准化处理
    for i in factor_list:
        if i != 'Sharperatio20':
            FactorData[i]=Get_BZH(FactorData[i])


    # 收益率分类：夏普比率为正的分类为 1，其他为 0
    for i,v in enumerate(FactorData['Sharperatio20']):

        if(v>0):
            FactorData['Sharperatio20'][i] = 1
        else:
            FactorData['Sharperatio20'][i] = 0

    # 缺失值处理
    # 先向后取值，若仍存在 Nan 值，则删除该特征因子
    X_test = FactorData.drop(columns=['Sharperatio20']).fillna(method = 'ffill')
    context.y_train = FactorData['Sharperatio20'].fillna(method = 'ffill')
    X_test = X_test.dropna(axis = 1)

    # 建立标的列表，存放目标股票的序号
    buy_list = []
```

```
# 机器学习优化
# 第一次就取平均权重
if context.num ==0:
    context.num = context.num + 1
    context.X_train=X_test
    # 所有特征值相加，取最大的五个股票
    temp = context.X_train.sum(axis=1).sort_values(ascending=False)
    buy_list = temp.index.values[0:5]

# 从第二次开始，每次用新的训练集进行训练
else:
    # Fit and tune model
    #print('X_train',context.X_train)
    #print('y_train',context.y_train)

    #开始训练本次的模型
    context.clf.fit(context.X_train, context.y_train.astype('int'))
    print("第",context.num,"次模型训练完毕")
    # clf = joblib.load('rf_regressor.pkl')
    # 储存下一次的训练集
    # joblib.dump(context.clf, "m/model.joblib")

    # 开始预测
    y_pred = context.clf.predict(X_test).tolist()

    # 将本次预测的特征作为下次训练的特征集
    context.X_train=X_test

    # 选取预测收益率为正的股票，放入 buy_list 中
    for i in range(len(y_pred)):
        if y_pred[i] == 1:
```

```
                    buy_list.append(i)
            print("buy_list",buy_list)

        # 模型训练次数+1
        context.num = context.num + 1



        # 交易设置：
        #positions_long = context.account().positions['volume_long']  # 多头持仓数量
        #valid_cash = context.account(account_idx=0).cash['valid_cash'][0]  # 可用资金
        # 获取股票的权重
        #buy_value = valid_cash / (len(buy_list) + 1)  # 设置每只标的可用资金比例 + 1 防止分母为 0
        #print('buyvalue',buy_value)

        # 开始交易
        positions = context.account().positions
        # 平不在标的池的股票
        for target_idx in positions.target_idx.astype(int):
            if target_idx not in buy_list:
                if positions['volume_long'].iloc[target_idx] > 0:
                    order_volume(account_idx=0, target_idx=target_idx,

volume=int(positions['volume_long'].iloc[target_idx]),
                                 side=2, position_effect=2, order_type=2, price=0)
                    print('卖出', context.target_list[target_idx])

        # 获取股票的权重
        percent = 0.2
        # 买在标的池中的股票
        for target_idx in buy_list:
```

```
            context.order_id=order_target_percent(account_idx=0,
target_idx=int(target_idx),
                    target_percent=percent, side=1, order_type=2,
                    price=0)
            print('买入', context.target_list[target_idx])



if __name__ == '__main__':
    # 设置开始与结束时间
    begin = '2016-01-01'
    end = '2019-05-30'
    cons_date = dt.datetime.strptime(begin, '%Y-%m-%d') - dt.timedelta(days=1)
    # 设置目标股票池，此处为上证 50
    sz50 = get_code_list('sz50', cons_date)[['code', 'weight']]
    targetlist = list(sz50['code'])
    # 开始回测
    run_backtest(strategy_name='Adaboost',
                         file_path='adaboost.py',
                         target_list=targetlist,
                         frequency='day',
                         fre_num=1,
                         begin_date=begin,
                         end_date=end,
                         fq=1)
```

**六、运行的结果截图：**

程序每一个交易周期都会根据股票的预测值进行判断买入还是买入

```
第 41 次模型训练完毕
buy_list [0, 8, 15, 20, 31, 35]
卖出 SSE.600010
卖出 SSE.600018
卖出 SSE.600795
卖出 SSE.600837
卖出 SSE.601006
买入 SSE.600000
买入 SSE.600048
买入 SSE.600519
买入 SSE.600887
买入 SSE.601318
买入 SSE.601398
回测完毕，正在处理绩效报告，请稍等...
回测总耗时 24.112298 秒
```

图 6.1 运行过程图

**Adaboost 模型回测结果：**

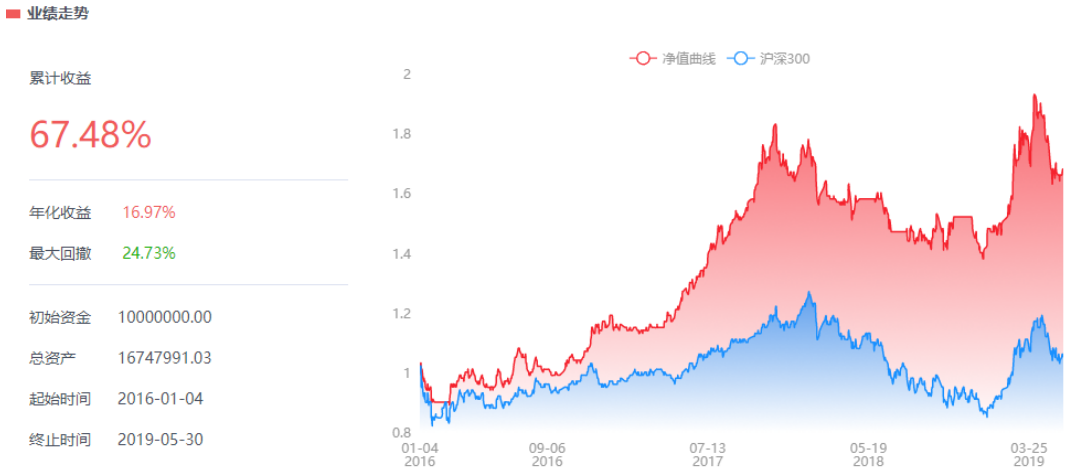
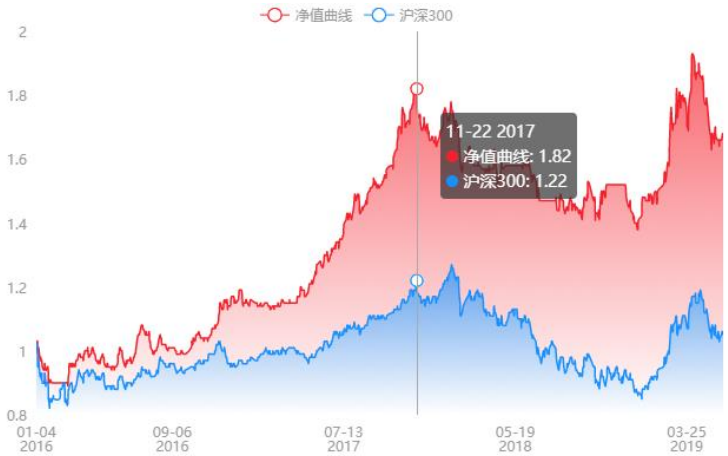



图 6.2 回测结果

从结果可以看出，Adaboost 改进的多因子模型效果很好，年化收益率高达 17%，2016 年投入 1000 万元，到了 2019 年已经盈利成为 1674 万元。

从牛市来看，该选股模型的盈利能力是市场平均的 3-4 倍



从熊市来看，该模型的最大回撤（最大回撤指的是最大亏损额）只有市场的一半左右，从收益曲线上观察更为直观，牛市时比大盘上升的更陡，熊市是下降的更缓，可见，该模型能够反映出一定的市场规律

我们可以从具体股票的买入卖出来观察模型效果,此处以宝钢股份和中国联通为例，图 6.4 6.5 中红色代表买入，绿色代表卖出，能够观察到模型对股票的买入卖出时机把握的较为精准，这也能够反映出 Adaboost 模型的实用性与准确性。

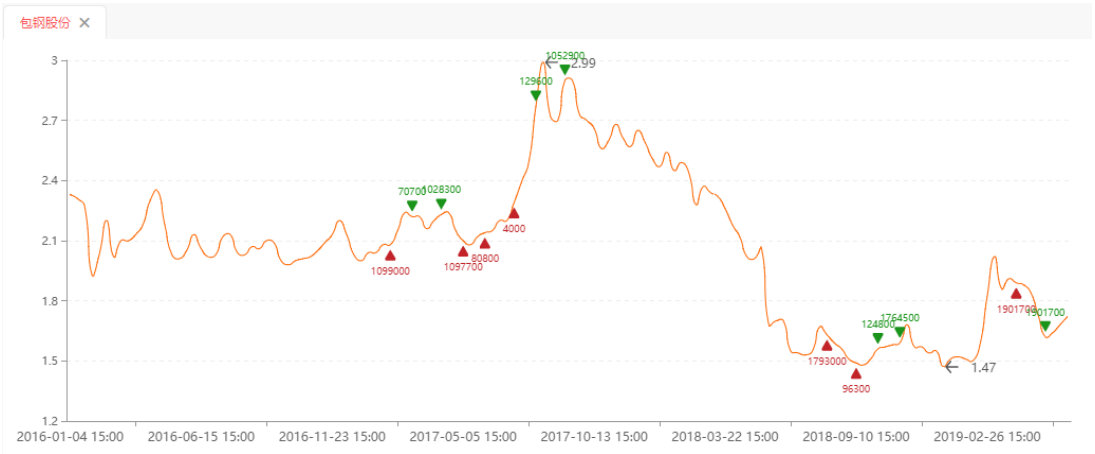



图 6.4 宝钢股份


图 6.5 中国联通

备注：更多关于策略的回测信息可以查看附件中的回测报告

**结论**

本项目实证的过程和思路简要概括为：候选因子的选取、有效因子的检验、冗余因子的剔除、选股模型的构建、基于 Adaboost 算法的多因子模型的构建。

结果也很显著，基于机器学习 AdaBoost 算法构建的多因子选股模型，回测的结果还是比较优秀，相比传统的多因子选股模型在选股效果和绩效上都有所提高。当然，模型也存在着一些不足，回撤率较高，存在一定风险，需要继续引入一些止损止盈的算法和机制来降低回撤率，这方面还需要继续花时间与精力进行研究，希望随着学习的进一步深入，能够优化出更完善的模型。

# 参考文献


[1]丁鹏.《量化投资一策略与技术》. 电子工业出版社. 2012. ‘

[2]魏妹金. 支持向量机多因子选股模型[D]. 华侨大学，2015.

[3]司晓彤. 基于回归法的多因子选股模型的投资组合分析[D]. 青岛大学，2017

[4]龚利琴. 基于 AdaBoost 算法的 Alpha 组合研究[D].郑州大学,2018.

[5]肖晞晖. 基于大数据和机器学习的量化选股模型研究[D].华中师范大学,2018.

[6]车洋. 基于机器学习方法的多因子选股策略研究[D].天津大学,2018.

[7]周渐. 基于 SVM 算法的多因子选股模型实证研究[D].浙江工商大学,2017.

[8]点宽网 Python API https://www.digquant.com.cn/documents/17