浙江工业大学

金融数据挖掘课程设计

(2021 级)

姓　　名：温家伟

班　　级：大数据分析 2101 班

学　　号：202103151422

提交日期：2024 年 6 月 3 日

浙江工业大学　数学科学学院

目录

1 课题背景

1.1 金融数据挖掘简介

目前，数据挖掘技术是时下较为有用的量化投资方法，主要有三个步骤。首先是数据预处理，通常数据挖掘会处理很大的数据量，这些数据来源不一导致格式也不统一，也许有的数据还存在一些缺失值或无效值，如果不经过处理直接将这些原始数据放到模型中去跑，非常容易导致模型计算的失败或者可用性非常差，所以数据预处理是所有数据挖掘过程中都不可或缺的一步。完成了数据的预处理，再进行特征的构造然后放到特定的模型中去计算，利用某种标准去评判不同模型或组合模型的表现，最后确定一个最合适的模型用于我们的后处理。后处理的过程相当于我们已经发现了那个我们想要找到的模式，我们会去应用它或者用合适的方式将其表示出来。

量化投资目前具体应用于分析股市趋势、量化选股及选择上。量化分析股市趋势是通过对股市大盘或者个股的趋势判断，进行相应的投资操作。例如，如果判断趋势向上则做多，如果判断趋势向下则做空，如果判断趋势盘整，则进行高抛低吸。量化选股则是利用量化的方法选择股票组合，以尽可能确保该股票组合能够获得超越基准的收益率。量化选股策略总的来说可以分为两类：第一类是基本面选股，第二类是市场行为选股。而量化选择是利用量化的方法，通过对各种宏观微观指标的量化分析，试图找到影响大盘走势的关键信息，并以此预测未来走势。通过数理模型和机器学习的方法实现套利交易、算法交易、资产配置、风险控制、预测模型的建立。以上几类量化投资应用都可以通过应用数据挖掘技术来完成。

金融数据挖掘中采用机器学习和一般的机器学习任务有很大的不同。首先，金融市场交易本身噪音非常大。其次，一个模型不可能长期有效，因此需要不停的构造新的有效测试数据，新的模型。而且不同的调仓周期，不同的仓位管理，不同的止损策略对于同一个模型的评判标准可能是完全不同的。

1.2 量化投资概述

1.2.1 量化投资简介

量化投资是指利用量化金融分析方法进行资产管理的投资策略。量化金融分析综合金融数据、个人经验、数学模型及计算机技术，通过复杂金融

建模及分析手段辅助决策。

1.2.2 量化投资策略的类型

策略分类多样，常见的划分方式包括：

1. **股票策略**直接投资股票市场。
2. **基金策略**包括指数基金、分级基金等。
3. **期货、期权策略**涉及各类衍生品市场。
4. **债券策略**投资于不同类型的债券。
5. **海外资产策略**直接或间接投资海外证券。

1.2.3 量化研究流程

1. **获取数据**包括公司财务、新闻、行情数据等。
2. **数据分析挖掘**应用传统分析与现代数据挖掘技术。
3. **构建信号**数据预处理后形成基础与复杂信号。
4. **构建策略**设计兼容多资产、多周期的策略模板。
5. **回测**模拟历史环境，考虑市场细节如分红、滑点等。
6. **策略分析**包括归因分析、风险监控等。
7. **模拟交易**实时行情接入，模拟真实交易环境。
8. **实盘交易**与真实券商账户对接，实施交易决策。

2 研究目的和研究平台

本次金融数据挖掘课程设计的目的是使用机器学习、数据挖掘的方法进行模拟的自动化交易，研究机器学习的方法是否在量化金融中可行，表现如何，同时更深入的掌握机器学习方法在实际中的应用。

研究平台为优矿（uqer）量化金融平台，平台内有 notebook 环境供编写策略，提供免费 1G 内存。python 环境为 2.7，内置有大部分的 python

数据处理和机器学习库，如 numpy、pandas、scipy、sklearn。优矿平台的优点在于交易数据十分充分，股票基本信息、股票行情等都可以通过研究数据获取标准 API 接口获得。

3 课题内容

我们认为股票的价格序列可以被拆分为线性和非线性趋势，所以本文采用线性的逻辑回归与一个非线性模型结合的方法，使之更好的描述股票的价格趋势。

3.1 逻辑回归

当我们面对一个分类问题时，逻辑回归是一种常用的统计学习方法。它的原理基于线性回归模型，并利用 Sigmoid 函数将线性预测转换为概率输出。

假设我们有一个二分类问题，标签 y 只能取 0 或 1。逻辑回归模型的基本形式可以表示为：

$$P(y=1|X)=\frac{1}{1+e^{-\beta^T X}}$$

其中，X 是输入特征向量，β 是模型参数。在这个公式中，$\beta^T X$ 给出了线性组合的结果，而 Sigmoid 函数 $g(z)=\frac{1}{1+e^{-z}}$ 将其映射到 (0,1) 之间，表示样本属于类别 1 的概率。

逻辑回归的训练过程通常使用极大似然估计来求解模型参数 β，使得观测数据出现的概率最大化。通过最大化似然函数，我们可以得到逻辑回归模型的参数估计值 $\hat{\beta}$。在预测阶段，我们可以使用学习到的参数进行分类预测，根据输入特征计算出样本属于类别 1 的概率，进而根据设定的阈值进行分类决策。

逻辑回归模型简单而有效，经常用于二分类问题的建模和预测。

3.2 SVM

支持向量机（SVM）是一种常用的监督学习算法，可以用于分类和回归问题。在分类问题中，SVM 的目标是找到一个能够将不同类别的样本分隔

开的超平面。当样本不是线性可分时，SVM 通过使用核函数来将输入映射到更高维的特征空间，并在该特征空间中寻找最优的超平面。

对于 SVM 中的高斯核函数，也称为径向基函数（RBF），其形式为：

$$K(x, x') = \exp\left(-\frac{\|x - x'\|^2}{2\sigma^2}\right)$$

其中，x 和 x' 表示输入样本，σ 是高斯核函数的一个超参数。高斯核函数的作用是将输入样本映射到无限维的特征空间，使得非线性问题也能够被解决。

SVM 的优化目标是找到最大间隔超平面，同时使得训练样本尽可能远离超平面，以达到更好的泛化能力。通过引入拉格朗日乘子和 KKT 条件，可以得到 SVM 的对偶优化问题，并通过求解对偶问题来得到最优的超平面参数和支持向量。最终的分类决策函数可以表示为：

$$f(x) = sign\left(\sum_{i=1}^{N} \alpha_i y_i K(x, x_i) + b\right)$$

其中，N 是训练样本数量，α_i 是支持向量对应的拉格朗日乘子，y_i 是对应的类别标签，x_i 是支持向量，b 是偏置项。

SVM 通过最大化分类边界的间隔和引入核函数来处理非线性问题，因此在实际应用中得到了广泛的应用。

3.3 MLP

多层感知机（MLP）是一种常见的人工神经网络模型，由多个神经元组成的多层结构。MLP 通过使用非线性激活函数和反向传播算法来学习复杂的非线性关系。

如果你想使用高斯核函数作为 MLP 的激活函数，一种常见的选择是高斯函数（也称为径向基函数），其形式可以表示为：

$$f(x) = e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

其中，x 表示输入，μ 表示均值，σ 表示标准差。高斯函数可以将输入映射到一个非线性的特征空间，从而增强了 MLP 模型的表达能力。

MLP 通过前向传播和反向传播算法来训练模型。在前向传播过程中，输入样本经过权重和激活函数的作用，逐层传递，最终得到输出结果。在反

向传播过程中，通过计算损失函数的梯度，利用梯度下降法更新模型参数，以减小预测值与真实值之间的误差。

MLP 的前向传播过程可以表示为：

$$a^{(l)} = f(W^{(l)}a^{(l-1)} + b^{(l)})$$

其中，$a^{(l)}$ 表示第 l 层的激活值，$W^{(l)}$ 表示第 l 层的权重矩阵，$b^{(l)}$ 表示第 l 层的偏置项，f 表示激活函数。

MLP 通过多层神经元和非线性激活函数的组合，可以学习并表示复杂的非线性关系，因此在深度学习领域得到了广泛的应用。

3.4 随机森林

随机森林是一种集成学习方法，通过构建多个决策树并将它们组合起来进行预测。每棵决策树都是基于随机抽取的子样本和特征进行训练，然后通过投票或取平均值的方式进行预测。

假设我们有一个包含 m 个样本的数据集，每个样本有 n 个特征。随机森林的训练过程包括以下步骤：

1. 从原始数据集中随机选择包含 m' 个样本的子样本集（通常 m' 小于 m）作为训练数据。

2. 从 n 个特征中随机选择 k 个特征（通常 k 远小于 n）作为候选特征。

3. 基于上述子样本集和候选特征集训练一颗决策树。

在随机森林中，决策树的预测结果可以通过投票或取平均值的方式得到。对于分类问题，预测结果是所有决策树的分类结果中出现次数最多的类别；对于回归问题，预测结果是所有决策树的预测值的平均值。

随机森林的决策树训练过程可以表示为：

1. 随机选择包含 m' 个样本的子样本集：
$D = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ..., (\mathbf{x}_{m'}, y_{m'})\}$，其中 $\mathbf{x}_i$ 是特征向量，y_i 是对应的标签。

2. 随机选择 k 个特征：$F = \{f_1, f_2, ..., f_k\}$。

3. 基于子样本集 D 和候选特征集 F 训练一颗决策树：T_i。

通过组合多棵决策树的预测结果，随机森林能够减少过拟合的风险，并具有较强的泛化能力，因此在实际应用中得到了广泛的应用。

3.5 朴素贝叶斯

朴素贝叶斯（Naive Bayes）是一种基于贝叶斯定理和特征条件独立假设的分类算法。它假设给定类别下特征之间相互独立，即每个特征对于类别的贡献是相互独立的。尽管这个假设在现实中并不总是成立，但朴素贝叶斯仍然在许多实际问题中表现良好。

假设我们有一个包含 n 个特征的数据集以及一个类别标签，朴素贝叶斯的分类过程可以描述如下：

1. 基于训练数据计算每个类别的先验概率：$P(Y = c_k)$，其中 c_k 表示第 k 个类别。2. 对于每个特征 X_i，计算在给定类别下的条件概率：$P(X_i = x|Y = c_k)$，表示在类别 c_k 下特征 X_i 取值为 x 的概率。3. 对于一个新的样本 $\mathbf{x} = (x_1, x_2, ..., x_n)$，通过贝叶斯定理计算其属于每个类别的后验概率：

$$P(Y = c_k|\mathbf{x}) = \frac{P(Y = c_k)\prod_{i=1}^{n} P(X_i = x_i|Y = c_k)}{P(\mathbf{x})}$$

4. 最终将样本分到具有最大后验概率的类别中：$\hat{y} = \arg\max_{c_k} P(Y = c_k|\mathbf{x})$。

朴素贝叶斯算法的关键在于计算先验概率和条件概率。对于连续特征，可以使用概率密度函数（如高斯分布）来估计条件概率；对于离散特征，可以直接统计频次来计算条件概率。

3.6 极端随机树

极端随机树（Extremely Randomized Trees，ERT）是一种集成学习方法，与随机森林类似，但在构建每棵树时更加随机化。在极端随机树中，每棵决策树的节点划分不再是基于最优的特征和阈值，而是随机选择特征和阈值进行划分。

假设我们有一个包含 m 个样本的数据集，每个样本有 n 个特征。极端随机树的训练过程包括以下步骤：

1. 从原始数据集中随机选择包含 m' 个样本的子样本集（通常 m' 小于 m）作为训练数据。

2. 从 n 个特征中随机选择 k 个特征（通常 k 远小于 n）作为候选特征。

3. 随机选择一个特征和阈值，对选定的特征进行随机划分，而不是基于最优的划分条件。

4. 重复步骤 3 直到达到指定的树的大小或无法再划分为止。

极端随机树的决策树训练过程可以表示为：

1. 随机选择包含 m' 个样本的子样本集：
$D = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ..., (\mathbf{x}_{m'}, y_{m'})\}$，其中 $\mathbf{x}_i$ 是特征向量，y_i 是对应的标签。

2. 随机选择 k 个特征：$F = \{f_1, f_2, ..., f_k\}$。

3. 随机选择特征和阈值，进行随机划分：T_i。

极端随机树通过引入更大程度的随机性，能够减少过拟合的风险，并在一些情况下比随机森林表现更好。其预测过程与随机森林类似，通过组合多棵树的预测结果来得到最终的预测结果。

3.7 AdaBoost

策略描述:AdaBoost（Adaptive Boosting）是一种集成学习方法，通过组合多个弱分类器来构建一个强分类器。它的核心思想是在每一轮训练中，调整样本的权重，使得前一轮分类错误的样本在下一轮中得到更多的关注，从而使得模型能够不断地聚焦于难以分类的样本。

假设我们有一个包含 m 个样本的训练集 $D = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ..., (\mathbf{x}_m, y_m)\}$，其中 $\mathbf{x}_i$ 是特征向量，y_i 是对应的标签。AdaBoost 的训练过程可以描述如下：

1. 初始化样本权重：$w_1(i) = \frac{1}{m}$，$i = 1, 2, ..., m$。

2. 对于 $t = 1, 2, ..., T$，进行以下步骤：

   - 使用当前权重分布 w_t 训练一个弱分类器 $h_t(\mathbf{x})$。
   - 计算弱分类器在训练集上的误差率：$\varepsilon_t = \sum_{i=1}^{m} w_t(i) \cdot \mathbb{I}(h_t(\mathbf{x}_i) \neq y_i)$，其中 $\mathbb{I}(\cdot)$ 是指示函数。
   - 计算弱分类器的权重：$\alpha_t = \frac{1}{2} \ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$。
   - 更新样本权重：$w_{t+1}(i) = w_t(i) \cdot \exp\left(-\alpha_t y_i h_t(\mathbf{x}_i)\right)$，其中归一化使得 w_{t+1} 成为概率分布。

最终的强分类器可以表示为：

$$H(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})\right)$$

AdaBoost 通过不断调整样本权重和弱分类器的加权组合，能够将多个弱分类器提升为一个强分类器，并在实际应用中取得了较好的效果。

4 选股策略

多因子选股的策略如下，我们这里用 SVM + 逻辑回归的策略来介绍，其他策略组合同理。

算法流程见下表：

表 1: 股票交易策略算法

算法：股票交易策略

输入： T-3 交易日 A 股所有股票的 22 个因子数值作为训练集，T-2、T-1、T 交易日的所有 A 股三日总计收益率作为训练标签结果，T 日的 A 股所有股票的 22 个因子数值作为测试集。

步骤 1： 选择 T-3 交易日的因子数值作为训练集。

步骤 2： 将 T-2、T-1、T 交易日的三日总计收益率作为训练标签，大于 2 **步骤 3：** 选择 T 交易日的因子数值作为测试集。

步骤 4： 数据处理，包括因子数值标准化、处理停牌或非交易日数据，剔除涨停股票。

步骤 5： 使用 SVM 模型（clf）对步骤 1 和步骤 2 中的训练集进行训练。

步骤 6： 将步骤 3 中的数据输入 clf 模型中，预测标签为 1 的股票作为下一步的股票池。

步骤 7： 以 SVM 预测上涨的股票作为股票池，再用逻辑回归算法计算这些股票的上涨概率，选取概率最大的 20 只股票作为持仓，并在收盘时买入。

输出： 选定的交易股票。

这里我们选取了包含技术因子和基本面因子一共 22 个因子，较为全面、准确。

这些因子具体含义如下表所示：

表 2: 股票因子说明

PB	市净率
PE	市盈率
PS	市销率
LCAP	对数市值
CTOP	现金流市值比
DebtEquityRatio	产权比率
ROE	权益回报率
ROA	资产回报率
EPS	每股收益
CurrentRatio	流动比率
DAVOL10	10 日平均换手率与 120 日平均换手率之比
DAVOL5	5 日平均换手率与 120 日平均换手率之比
MA5	5 日移动平均
MA10	10 日移动平均
MA20	20 日移动平均
MFI	资金流量指标
ATR6	6 日均幅指标
BIAS10	10 日乖离率
CCI5	5 日顺势指标
BollDown	下轨线（布林线）指标
BollUp	上轨线（布林线）指标
ROC6	6 日变动速率

5 实验结果及分析

5.1 回测结果

5.1.1 SVM + 逻辑回归

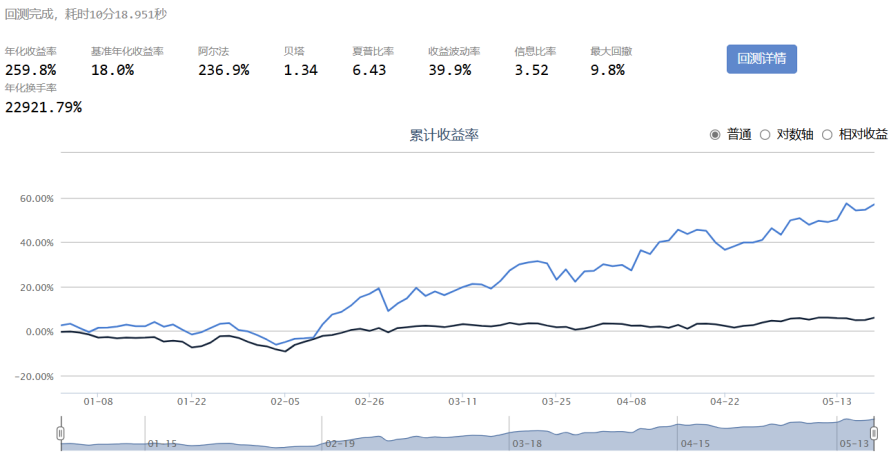

图 1: SVM + 逻辑回归

5.1.2 MLP + 逻辑回归

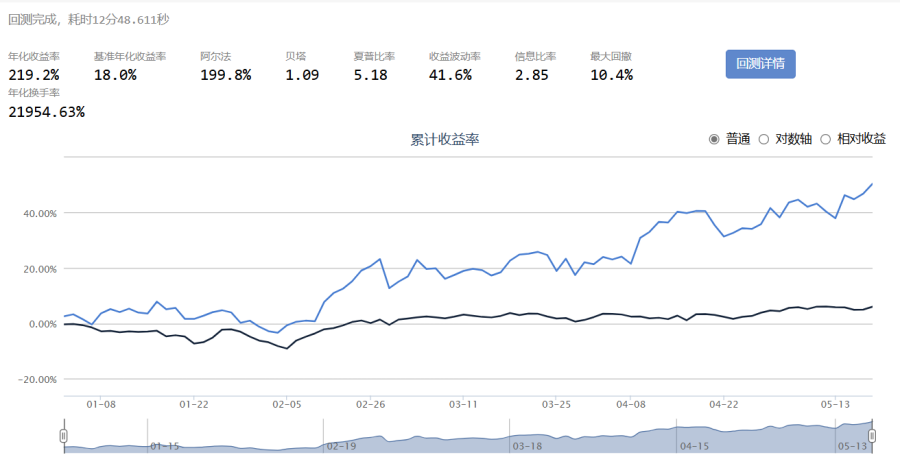

图 2: MLP + 逻辑回归

5.1.3 随机森林 + 逻辑回归

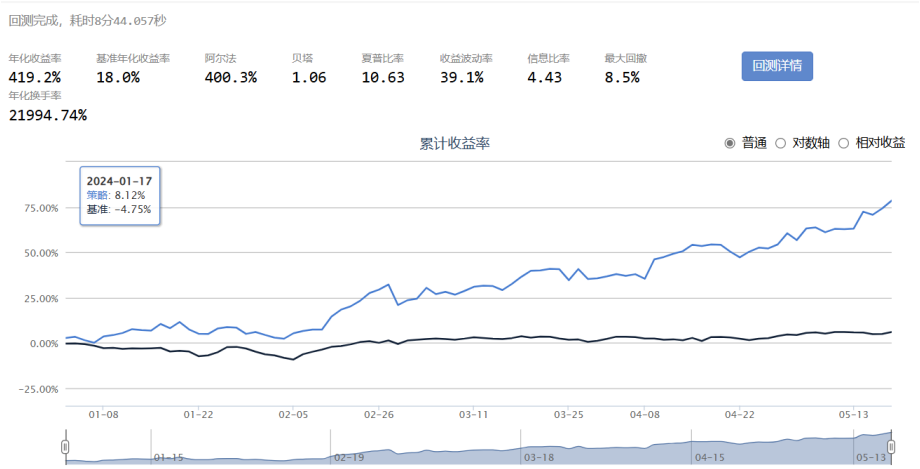

图 3: 随机森林 + 逻辑回归

5.1.4 朴素贝叶斯 + 逻辑回归

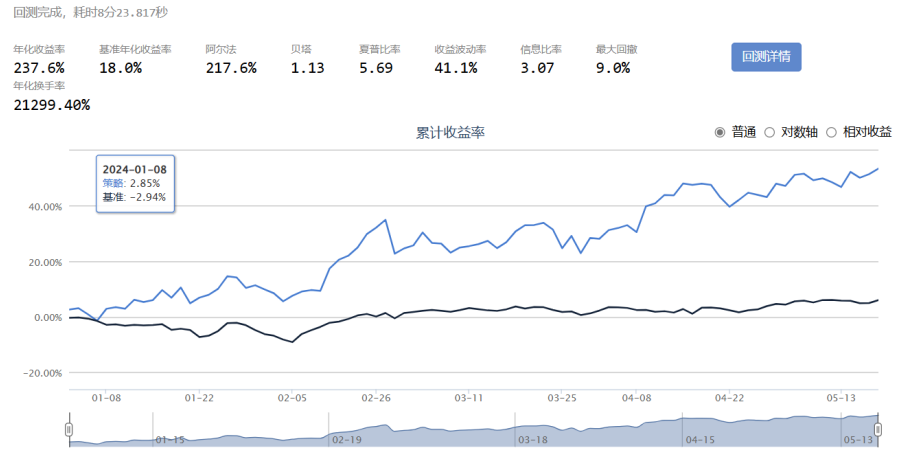

图 4: 朴素贝叶斯 + 逻辑回归

5.1.5 极端随机树 + 逻辑回归

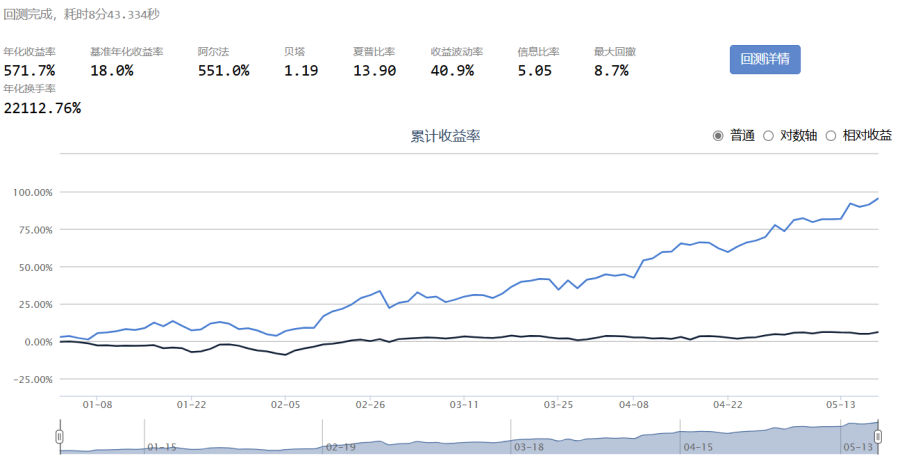

图 5: 极端随机树 + 逻辑回归

5.1.6 AdaBoost + 逻辑回归

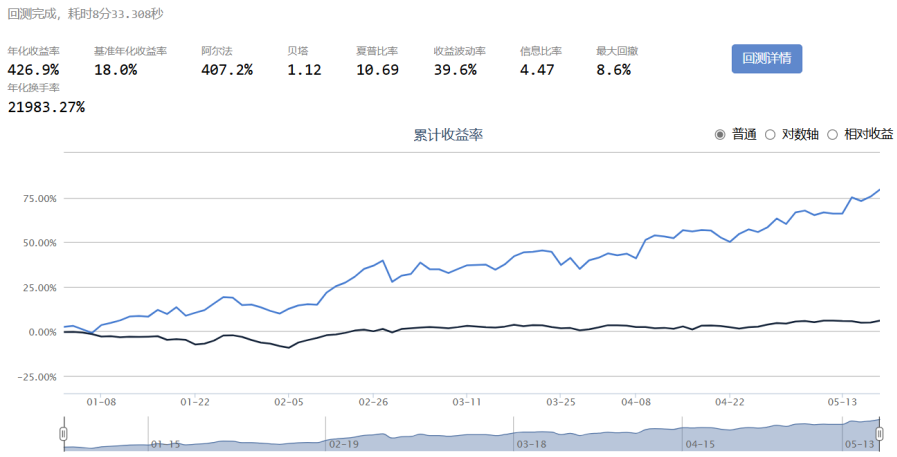

图 6: AdaBoost + 逻辑回归

5.2 结果分析

分析表 3 显示，各策略年化收益率对比中，极端随机树 + 逻辑回归以高达 571.7% 的年化收益率领先，表现出色。随机森林 + 逻辑回归和以 419.2% 紧随其后，同样显著。这表明非线性模型在处理复杂的金融数据中

表 3: 不同策略的年化收益率对比

策略名称	年化收益率 (%)
SVM + 逻辑回归	259.8
MLP + 逻辑回归	219.2
随机森林 + 逻辑回归	419.2
朴素贝叶斯 + 逻辑回归	237.6
极端随机树 + 逻辑回归	571.7
AdaBoost + 逻辑回归	426.9

具有优势，特别是随机性树和随机森林因其高度多样性，对非线性适应性强，能捕捉更多特征间的复杂交互。朴素贝叶斯 + 逻辑回归以 237.6% 的年化收益率显示在统计方法的实用性，虽然假设条件独立性简化，但在某些场景下依然有效。MLP+ 逻辑回归的 219.2% 则体现了多层神经网络在处理非线性问题的能力。

SVM+ 逻辑回归与 259.8% 的年化收益率，以及 AdaBoost+ 逻辑回归的 426.9% 表明，支持向量机和自适应增强技术在量化投资中的稳健性与准确性，尽管不如非线性模型极端随机性树和随机森林，但仍然取得了不错的效果。

综上，实验分析指出，不同策略各有利弊，选择取决于具体需求，如追求高收益可优先考虑极端随机树，偏好稳定和解释性则 SVM 或逻辑回归，而资源有限可能偏好逻辑回归或 SVM。未来工作可进一步细化调优参数，结合实际交易成本和策略执行效率，优化综合评估。

5.3 模拟炒股结果

模拟炒股结果如下，赚了一点点：

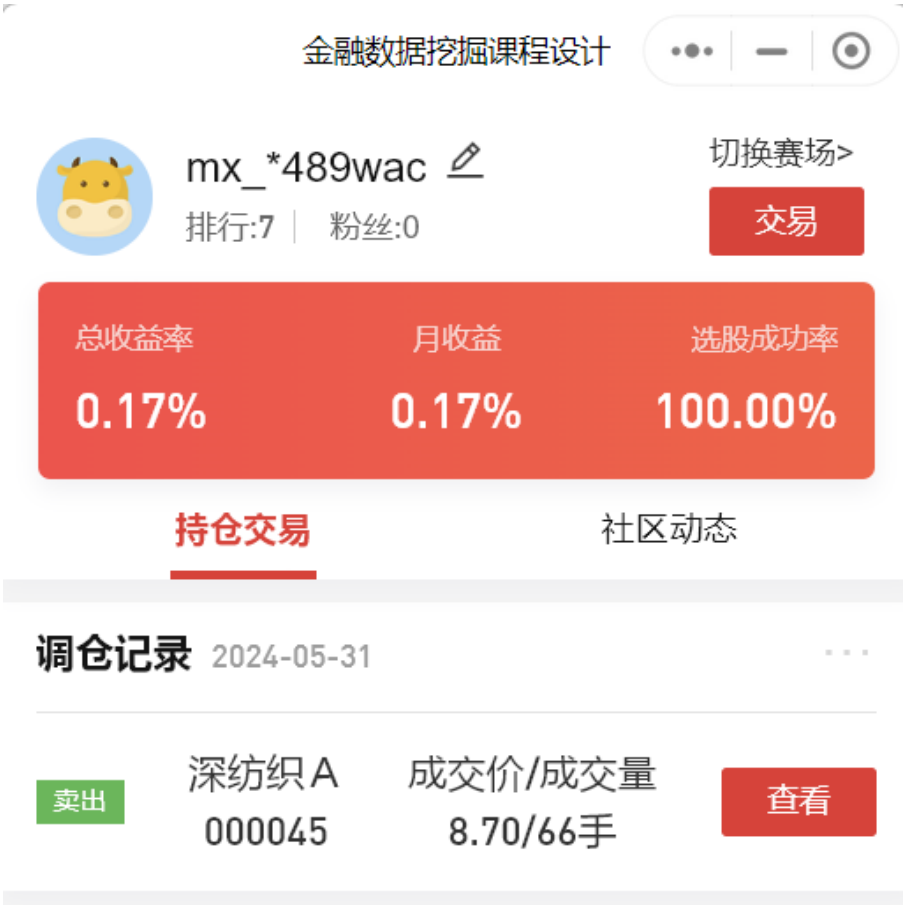

图 7: 模拟炒股结果图

6 总结展望

- **模型的稳定性与随机性：**极端随机树和随机森林的高泛化能力突出，适合波动大的市场环境。
- **模型更新频率：**模型更新要求，考虑到市场的快速变化，模型如 AdaBoost 的动态权重调整机制有利。
- **复杂性与解释性：**逻辑回归、SVM 的解释性较强，适合监管要求严格的环境，而 MLP 和随机森林较难解释。
- **资源消耗：**计算成本，MLP 和随机森林资源需求较高，需评估硬件支

持。

7 附录

本实验中实验代码如下：

```
#加载库函数
import datetime #加载时间函数库
import numpy as np
import pandas as pd
from sklearn.cross_validation import train_test_split
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import classification_report
from sklearn.linear_model.logistic import LogisticRegression
from sklearn.svm import SVC


#初始化回测环境
start = '20240101' # 回测起始时间 注：支持两种日期表述形式（ '2015-01-01'
    , '20150101'）
end = '20240517' # 回测结束时间
benchmark = '000002.ZICN' # 策略参考标准为A股指数
universe = set_universe('A') # 证券池：可供选择的股票的范围为A股.（由于选
    股条件比较苛刻，所以为了保证有充足的可供选择股票，因此以整个A股作为投资
    范围）
freq = 'd' # 用日线回测的策略
refresh_rate = 2 # 每10日调一次仓，即每个交易日都会运行第三部分的
    handle_data函数

def trading(buylist,context):
    #数据获取（通用部分）：投资者账户，可供投资股票
    account = context.get_account('fantasy_account') #获取投资者的股票账户
        （fantasy_account）
    current_universe = context.get_universe(asset_type = 'stock',
        exclude_halt=True) #获取当前除停牌外的所有可供投资股票（universe）
    security_position = account.get_positions() #字典型数据，上一K线结束后
        的有效证券头寸，即持仓数量大于0的证券及其头寸
    cash = account.cash #获取股票账户可用于投资的现金额度
```

```
    #交易执行部分：卖出
    for sec in current_universe:
        if sec not in buylist and sec in security_position: #不在buylist
            中，且有持仓，卖出
            order_to(sec,0) #执行卖出指令后,会自动更新股票账户中的金额
    #交易执行部分：买入
    if len(buylist) > 0: #判读本期是否有需要买入股票（buylist>0）
            weight = min(Max_Position_per,1.0/len(buylist)) #每只股票所分配
                的金额相同，但最多为总金额的10%
    else:
        weight = 0 #若本期没有要购入股票，设置分配权重为0
    for sec in buylist: #判断股票是否为本期要购入股票且尚未持有
        if sec not in security_position:
            order_pct_to(sec,weight) #购入指定比例的股票
def stock_sellection_stock(context):
    """
    前一天的历史数据（前一天的x）训练今天的数据(今天的y)，再用今天的数据（今
        天的x）预测明天的数据（明天的y）。
    """
    buylist=[]
    #数据获取（通用部分）：投资者账户，可供投资股票
    hist=DataAPI.MktIdxdGet(tradeDate=u"",indexID=u"000001.ZICN",ticker=u
        "",beginDate=start,
                            endDate=end,exchangeCD=u"XSHE,XSHG",field=u""
                                ,
                            pandas="1")[(["tradeDate","closeIndex"])]
    previous_date = context.previous_date #获取上一交易日的时间
    current_date = context.current_date #获取本交易日的时间

    pre3_date = context.current_date-datetime.timedelta(days=3)
    pre2_date = context.current_date-datetime.timedelta(days=2)
    previous_date = previous_date.strftime('%Y%m%d') #将日期格式调整为%Y%m
        %d形式（20240101），方便读取因子数据
    account = context.get_account('fantasy_account') #获取投资者的股票账户
        （fantasy_account）
    current_universe = context.get_universe(asset_type = 'stock',
        exclude_halt=True)

    #获取T-3日的所有HS300成分股的部分有效因子作为(训练集)
```

```
yes_factor = DataAPI.MktStockFactorsOneDayGet(tradeDate=pre3_date,
    secID=current_universe,field=['secID','PB','PE','PS','LCAP','CTOP
    ','DebtEquityRatio','ROE','ROA','EPS','CurrentRatio','DAVOL10','
    DAVOL5','MA5','MA10','MA20','MFI','ATR6','BIAS10','CCI5','
    BollDown','BollUp','ROC6'],pandas="1")
yes_factor2 = DataAPI.MktStockFactorsOneDayGet(tradeDate=pre2_date,
    secID=current_universe,field=['secID','PB','PE','PS','LCAP','CTOP
    ','DebtEquityRatio','ROE','ROA','EPS','CurrentRatio','DAVOL10','
    DAVOL5','MA5','MA10','MA20','MFI','ATR6','BIAS10','CCI5','
    BollDown','BollUp','ROC6'],pandas="1")
while(yes_factor.shape[0]==0):
    if (yes_factor.shape[0]==0):
        pre3_date =pre3_date - datetime.timedelta(days=1)
        yes_factor = DataAPI.MktStockFactorsOneDayGet(tradeDate=
            pre3_date,secID=current_universe,field=['secID','PB','PE',
            'PS','LCAP','CTOP','DebtEquityRatio','ROE','ROA','EPS','
            CurrentRatio','DAVOL10','DAVOL5','MA5','MA10','MA20','MFI'
            ,'ATR6','BIAS10','CCI5','BollDown','BollUp','ROC6'],pandas
            ="1")


while(yes_factor2.shape[0]==0):
    if (yes_factor2.shape[0]==0):
        pre2_date = pre2_date - datetime.timedelta(days=1)
        yes_factor2 = DataAPI.MktStockFactorsOneDayGet(tradeDate=
            pre2_date,secID=current_universe,field=['secID','PB','PE',
            'PS','LCAP','CTOP','DebtEquityRatio','ROE','ROA','EPS','
            CurrentRatio','DAVOL10','DAVOL5','MA5','MA10','MA20','MFI'
            ,'ATR6','BIAS10','CCI5','BollDown','BollUp','ROC6'],pandas
            ="1")
#数据处理部分（数据结构部分）：  去掉无效数据并设定索引
yes_factor = yes_factor.dropna() #去掉无效数据
#数据处理部分（逻辑部分）:将所有因子数据标准化
yes_factor.set_index(['secID'],inplace=True,drop=True)
yes_factor_train=yes_factor[['PB','PE','PS','LCAP','CTOP','
    DebtEquityRatio','ROE','ROA','EPS','CurrentRatio','DAVOL10','
    DAVOL5','MA5','MA10','MA20','MFI','ATR6','BIAS10','CCI5','
    BollDown','BollUp','ROC6']]
```

```
yes_factor_train = yes_factor_train.apply(lambda x: (x-x.mean())/x.
    std(), axis=0)
#拿出今天的收盘价作为(训练集的结果验证)
cur_price=DataAPI.MktEqudGet(secID=current_universe,tradeDate="",
    beginDate=pre2_date,endDate=current_date,isOpen="",field=u"",
    pandas="1")#先拿出前天、昨天、今天的三天数据，然后用分组
cur_price=cur_price.groupby(['secID'])['chgPct'].sum()
cur_price = pd.DataFrame(cur_price, columns = ['chgPct'])
cur_price['label']=0
cur_price = cur_price.dropna()
#cur_price.set_index(['secID'],inplace=True,drop=True)

cur_price.loc[cur_price['chgPct']>0.02,'label']=1
cur_price.loc[cur_price['chgPct']<=0.02,'label']=0

cur_price=cur_price.loc[cur_price.index.isin(yes_factor_train.index)
    ,:]
yes_factor_train=yes_factor_train.loc[yes_factor_train.index.isin(
    cur_price.index),:]
#print(yes_factor_train)
#获取T日的所有HS300成分股的部分有效因子作为(测试集)
yes_factor_test = DataAPI.MktStockFactorsOneDayGet(tradeDate=
    current_date,secID=current_universe,field=['secID','PB','PE','PS'
    ,'LCAP','CTOP','DebtEquityRatio','ROE','ROA','EPS','CurrentRatio'
    ,'DAVOL10','DAVOL5','MA5','MA10','MA20','MFI','ATR6','BIAS10','
    CCI5','BollDown','BollUp','ROC6'],pandas="1")
#数据处理部分（数据结构部分）： 去掉无效数据并设定索引
yes_factor_test = yes_factor_test.dropna() #去掉无效数据
#数据处理部分（逻辑部分）:将所有因子数据标准化
yes_factor_test.set_index(['secID'],inplace=True,drop=True)
yes_factor_test=yes_factor_test[['PB','PE','PS','LCAP','CTOP','
    DebtEquityRatio','ROE','ROA','EPS','CurrentRatio','DAVOL10','
    DAVOL5','MA5','MA10','MA20','MFI','ATR6','BIAS10','CCI5','
    BollDown','BollUp','ROC6']]
yes_factor_test = yes_factor_test.apply(lambda x: (x-x.mean())/x.std
    (), axis=0)
# 剔除T+1日涨停的股票（测试集的T+1日去预测T+2日，T+1日涨停的股票在T+1的
    尾盘没法买入T+2日的股票，直接剔除），series可以直接和某个数值比较
yes_factor_test_avilible=cur_price[cur_price['chgPct']<0.095].index
```

```
yes_factor_test=yes_factor_test.loc[yes_factor_test.index.isin(
    yes_factor_test_avilible),:]

# 1.
#用SVM开始训练类别
clf = SVC(C=80, kernel='rbf', gamma=0.2,
      decision_function_shape='ovr') # 高斯核，gamma值越小，分类界面
          越连续；gamma值越大，分类界面越“散”，分类效果越好，但有可
          能会过拟合
clf.fit(yes_factor_train.values,cur_price['label'].values)

# 2.
# from sklearn.neural_network import MLPClassifier

# # 实例化一个多层感知器分类器
# clf = MLPClassifier(hidden_layer_sizes=(100, 50), max_iter=300,
    alpha=1e-4,
# solver='sgd', activation='relu', random_state=1,
# learning_rate_init=.1)

# clf.fit(yes_factor_train.values, cur_price['label'].values)

# 3.
# from sklearn.ensemble import RandomForestClassifier

# # 实例化随机森林分类器
# clf = RandomForestClassifier(n_estimators=100, max_depth=10,
    random_state=0)

# clf.fit(yes_factor_train.values, cur_price['label'].values)

# 4.
# from sklearn.naive_bayes import GaussianNB

# # 实例化高斯朴素贝叶斯分类器，适用于连续特征
# clf = GaussianNB()

# clf.fit(yes_factor_train.values, cur_price['label'].values)
```

```
# 5.
# from sklearn.ensemble import ExtraTreesClassifier

# # 实例化ExtraTrees分类器
# clf = ExtraTreesClassifier(n_estimators=100, random_state=0)

# clf.fit(yes_factor_train.values, cur_price['label'].values)

# 6.
# from sklearn.ensemble import AdaBoostClassifier

# # 实例化AdaBoost分类器
# clf = AdaBoostClassifier(n_estimators=50, learning_rate=1.0,
    random_state=0)

# clf.fit(yes_factor_train.values, cur_price['label'].values)

# 进行预测并保存数据，predict_proba是可以得到每行预测的值落入某个类别的
    概率。所有行取第1列，代表所有行涨的概率
prediction1 = clf.predict(yes_factor_test.values)
yes_factor_test['prediction'] = prediction1
yes_factor_test2=yes_factor_test[yes_factor_test['prediction']==1]
yes_factor_test2=yes_factor_test2[['PB','PE','PS','LCAP','CTOP','
    DebtEquityRatio','ROE','ROA','EPS','CurrentRatio','DAVOL10','
    DAVOL5','MA5','MA10','MA20','MFI','ATR6','BIAS10','CCI5','
    BollDown','BollUp','ROC6']]

#SVM训练出来为1的再用logistic模型训练，为1概率最大的20只
clf2 = LogisticRegression()
clf2.fit(yes_factor_train.values, cur_price['label'])
# 进行预测并保存数据，predict_proba是可以得到每行预测的值落入某个类别的
    概率。所有行取第1列，代表所有行涨的概率
prediction2 = clf2.predict_proba(yes_factor_test2.values)
yes_factor_test2['prediction'] = prediction2[:,1]

# 逻辑回归策略买入了所有预测为上涨的股票，这里改进为买入上涨概率最大（
    prediction列）的20只股票；按降序排列
yes_factor_test2.sort_values(by='prediction', inplace=True, ascending
    =False)
```

```
    #i=a['secID'].tolist()
    buylist = yes_factor_test2[:20].index.tolist() #选择前10只作为选择的股
        ,生成buylist
    #print (buylist)
    return list(buylist) #返回buylist

#初始化投资者（账户）参数
#accounts为字典类型，代表投资者所有的账户，而字典中每一个键代表一个账户，而
    每一个键对应的值为该账户的初始情况，如本程序中的键为fantasy_account（股
    票账户），值为相应配置
accounts = {
    'fantasy_account': AccountConfig(account_type='security',
        capital_base=10000000) #初始化投资者的股票账户： 投资品种为股票，初
        始投资金额为1千万
}

#初始化策略参数：
Max_Position_per = 0.1 #每只股票购入的最高比例为10%

#初始化回测环境，指明创建账户时的工作，全局只运行一次
def initialize(context):
    pass

#handle_data函数是策略的核心函数，包含了所有策略算法的内容，包括数据获取，
    交易信号生成，订单委托等逻辑。
#handle_data函数无论是回测还是模拟交易场景，这个函数会根据回测频率 freq 的
    设置被调用。当freq='d'时，每天被调用一次，当freq='m'时，每分钟被调用一
    次。
def handle_data(context):
    buylist = stock_sellection_stock(context) # 选股策略
    trading(buylist,context) # 基于固定投资比例的仓位管理策略
```