目 录

基于 Stacking 方法的量化交易策略研究

摘要

量化投资是指利用数学及计算机知识辅助进行自动金融交易的过程。在机器学习等人工智能算法发展并广泛运用之前，量化投资领域中绝大部分策略都是依靠时间序列分析等计量模型或者依靠技术指标如 ATR、BOLL 等趋势跟踪指标进行构建。然而，由于金融市场的非平稳性，大部分传统模型适用性较窄，投资效果低下。机器学习算法的发展为量化投资提供了新的方案，机器学习算法对非平稳序列具有高度适应性，能够凭借模型优势充分挖掘金融数据中的非平稳信息。而从机器学习在量化金融领域的应用来看，单一简单模型的预测效果有上限，在实证中，很难通过单独调整参数而大幅提高准确率，综合准确率均在 50%上下浮动，因此获取收益的能力有限，且风险也较高，此时，Stacking 模型作为集成算法框架走入视野，作为一种融合学习算法，能够有效提高预测模型的准确率，能够将弱学习器通过合并交叉形成一个强学习器，有效提升量化交易过程中的总体收益。

本文基于 Stacking 算法，利用历史数据预测股票的涨跌情况，并根据预测结果构建日频量化交易策略。随后，基于对贵州茅台（600519）股票的历史数据，建立了包括标准化数据及 MA、MACD、CCI 等技术性指标的数据集，以保证数据内包含了关于股票涨跌情况的大部分信息。其次构建了由 SVM、MLP、RF 及 Logistic 构成的两层 Stacking 学习模型，采用 400 日移动滑窗对股票历史数据进行学习，并预测下一日涨跌情况。为了提升预测效果，基于五折交叉验证，并分别采用网格搜索法、贝叶斯优化和随机搜索对模型参数进行自动寻优，以达到最优的预测效果。回测结果表明，所构建的 Stacking 模型能够显著提高对股票涨跌的预测效率，综合正确率达到 90%上下浮动，显示出更为优越的性能。

基于 Stacking 模型预测结果，本文对贵州茅台股票历史数据构建了数个参数组合及交易策略，并使用 python 中的 Backtrader 框架进行了回测。回测结果表明，在预测性能上，平均正确率最低为 83.99%，最高达到 91.24%；在盈利能力上，构建出了一个低风险，适合进行资产保值的交易策略，夏普比率为 0.3747，年化收益率为 3.91%，构建出了一个中低风险，适合进行资产增值的交易策略，夏普比率达到 1.48，年化收益率达到 9.19%。

本文的研究为机器学习在量化金融领域的应用做出了进一步探索，为量化投资新人提供了新的学习借鉴。

【关键词】价格预测　量化交易策略　Stacking 模型

Research on Quantitative Trading Strategies Based on Stacking Method

Abstract

Quantitative investment refers to the process of using mathematical and computer knowledge to assist in automated financial transactions. Before the development and widespread application of artificial intelligence algorithms such as machine learning, the vast majority of strategies in the field of quantitative investment relied on econometric models such as time series analysis or trend tracking models based on technical indicators such as ATR and BOLL. However, due to the non-stationary nature of financial markets, most traditional models have limited applicability and low investment effectiveness. The development of machine learning algorithms provides new solutions for quantitative investment. Machine learning algorithms have high adaptability to non-stationary sequences and can fully explore non-stationary information in financial data with model advantages. From the perspective of the application of machine learning in the field of quantitative finance, there is an upper limit to the predictive performance of a single simple model. In empirical research, it is difficult to significantly improve accuracy by adjusting parameters separately, and the overall accuracy fluctuates around 50%. Therefore, the ability to obtain profits is limited, and the risk is also high. At this time, the Stacking model, as an integrated algorithm framework, comes into view. As a fusion learning algorithm, it can effectively improve the accuracy of the prediction model, merge and cross weak learners to form a strong learner, effectively improving the overall returns in the quantitative trading process.

This article is based on the Stacking algorithm, with the aim of predicting the rise and fall of stocks and constructing a daily frequency quantitative trading strategy based on the predicted results. Then, based on the historical data of Kweichow Moutai (600519) stock, a data set including standardized data and technical indicators such as MA, MACD and CCI was established to ensure that the data contained most of the information about the rise and fall of the stock. Secondly, a two-layer Stacking learning model consisting of SVM, MLP, RF, and Logistic was constructed. A 400 day moving sliding window was used to learn historical stock data and predict the next day's ups and downs. In order to improve the prediction performance, based on five fold cross validation, grid search method, Bayesian optimization, and random search were used to automatically optimize the model parameters, in order to achieve the optimal prediction effect. The backtesting results show that the constructed Stacking model can significantly improve the efficiency of predicting stock price fluctuations, with a comprehensive accuracy of up to 90%, demonstrating superior performance.

Based on the prediction results of Stacking model, this paper constructs several parameter combinations and trading strategies for the historical stock data of Kweichow Moutai, and uses the Backtrader framework in python for backtesting. The backtesting results show that in terms of predictive performance, the average accuracy is the lowest at 83.99% and the highest at 91.24%; In terms of profitability, a low-risk trading strategy suitable for asset preservation has been constructed, with a Sharpe ratio of 0.3747 and an annualized return of 3.91%. A medium to low-risk trading strategy suitable for asset appreciation has been constructed, with a Sharpe ratio of 1.48 and an annualized return of 9.19%.

This study further explores the application of machine learning in the field of quantitative finance and provides new learning references for newcomers in quantitative investment.

【Key words】Price forecast　Quantitative trading strategy　Stacking model

第 1 章 绪论

1.1 研究背景

随着互联网以及人工智能技术的迅猛发展，中国 A 股市场迎来了一个全新的发展阶段，越来越多的资金持有者倾向于使用量化投资的手段来实现对风险的有效控制，并取得稳定且高效的收益效果。量化投资是指利用计算机技术和科学的交易策略实现高频自动交易的一种投资方式，能够帮助投资者更有效的掌握市场的发展趋势，避免交易过程中人性的干扰，快速把握转瞬即逝的套利机会，及时快速地跟踪市场的变化，帮助投资者制定更准确的买卖决策。

一方面，量化投资能够高效、自动分析市场信息，极大地避免了传统主动性投资手段中投资者的主观偏见以及其他偶然因素可能带来的风险，有效提高投资效率；另一方面，机器学习、自然语言处理等人工智能的快速发展和广泛应用，量化投资能够专注于数据领域，利用数据挖掘技术和复杂的模型算法分析等手段，挖掘隐藏在市场中的投资机会。

2010 年 4 月 16 日，我国第一个股指期货沪深 300 股指期货上市，国内从此开始研究股指期货的实际应用；2015 年 4 月 16 日，中证 500 指数期货上市，这意味着量化金融行业发展出更多的空间，随着可融券的股票数量增加，量化投资策略也逐步发展到精细化和高频化时代。此外，随着深度学习算法的发展与完善，机构投资者开始进入“金融+科技”领域，开始结合各种人工智能模型，不断拓展和运用不同的量化投资策略。

1.2 研究目的与意义

国内大型金融机构都将部分资金投入到量化交易市场。目前来看，随着科技的不断创新和金融市场的改革，量化投资在中国 A 股市场中的应用越来越广泛。而由于现代金融数据呈现多样化、复杂化，传统因子模型难以处理大量金融数据中所蕴涵的信息；使用随机过程模型和时间序列模型进行投资决策分析也存在着较大的风险，其是在拟合复杂模型时，可能导致对历史数据的过度敏感而在未来表现不佳。并且传统模型原理较为复杂，需要较高的数学和统计知识，可能对不具备数学背景的投资者或分析师不太友好。因此，相当一部分金融从业者将视线投入到迅猛发展中的机器学习技术。

在机器学习研究领域，大部分学者聚焦于研究单一模型的应用，利用单一模型进行学习和预测并构建交易策略，往往通过参数调优等方法优化模型，提高模型预测准确率，但由于模型受到诸多限制，准确率提高到一定程度难以继续优化。研究表明，Stacking 方法能够显著提高单一模型的预测性能，从而提升策略的准确性和收益。

1.3 研究内容

本文的目的是建立一种集成算法模型，在总结有关文献内容的基础上，理论研究与实证研究相结合，通过集成多种分类算法，实现对股票趋势的预测，并在预测结果的基础上构建具有一定能力的交易策略，并进行日频量化投资交易回测。在经过数据分析和处理后，我们选择了几个预测性能较好的机器学习算法，包括 BF、XGBoost、SVM 和 KNN 等模型，提出了一种基于 Stacking 算法的股票涨跌预测模型。以沪深 300 中成分股中的部分股票作为研究对象，选取了十个股票基本数据：开盘价、收盘价、最高价、最低价、成交量、成交额、振幅、涨跌幅、涨跌额和换手率，以及数个技术分析指标：6 日均线 MA、MACD 和 CCI 指标作为模型的输入参量，验证 Stacking 算法的预测性能。

1.4 研究方法

本文采用的研究方法有：

（1）文献研究法

通过广泛阅读国内外期刊论文，充分了解关于量化投资交易策略、机器学习策略等文献，对其进行分析和归纳，以获取和总结关于机器学习算法在量化交易领域应用的现状和趋势。通过对这些文献进行分析和综述，提炼出本文的研究模型和实现手段，为后续将模型应用于实证分析提供坚实的理论基础和方向指引。

（2）定量研究法

本文将构建的 Stacking 量化交易模型应用于沪深 300 中的部分成分股，进行定量化分析，对目标股进行定制化调整，以达到模型的最佳性能，并验证模型的性能表现。

（3）实证研究法

本文通过搜集国内金融市场数据，采用 Backtrader 回测框架，以回测的方式验证策略的有效性，并计算夏普比率等作为回测结果的评价指标。

1.5 技术路线

本文的技术路线如下。首先结合有关文献以及实证分析，构建常见因子池，选取出与股票趋势相关的数据及指标。然后，建立两层 Stacking 模型，并且寻找模型最优参数。其次，对模型和量化交易策略进行回测和评价，在此基础上，对本文进行归纳整理，得出相关的结论，并指出研究的问题、缺陷以及未来的改进方法。

第 2 章 文献综述与理论基础

2.1 文献综述

2.1.1 量化交易发展历史

上世纪 70 年代，Fisher Black 和 Myron Scholes(1973)在此时期提出了著名的 Black-Scholars 期权定价模型，为后来的衍生品交易和量化金融领域奠定了基础。80 年代中期，华尔街传奇人物 James Simmons 首次将数学模型应用于投资交易，创立的文艺复兴科技公司成为量化交易领域的先驱者，通过利用数学模型和大数据分析进行交易，在他的管理下，旗下的大奖章基金在 1989-2009 年的平均年回报率约为 30%，取得了巨大的成功，开创了量化交易时代。在 90 年代，随着计算机技术的飞速发展，高频交易逐渐开始崭露头角，大型金融机构纷纷开始尝试运用快速的计算机算法进行交易，开始追求微秒级别的利润。

量化交易的核心是套利，其核心是 Stephen Ross(1976)所提出的套利定价理论(APT)，该理论假设证券的收益率主要受一个或多个市场因子的影响，但是并没有解释这些因子是如何影响未来的证券收益率的。为了研究这一议题，Fama 和 French(1996)提出了三因子模型，该模型基于 APT 和 CAPM 模型，提出了市场贝塔、规模因子和价值因子，旨在解释不同股票之间的收益率差异。基于三因子模型，Carhart(1995)在模型中加入动量因子，得到 Fama-French-Carhart 四因子模型，动量因子被定义为最优投资组合和最劣投资组合一年收益率的差值，其实证结果表明，在样本区间内，四因子模型相较于三因子模型显著降低了平均定价误差，能够更有效地解释证券之间收益率的差异，很大程度上弥补了三因子模型中市场异常的局限性。此外，欧阳志刚(2016)等人研究了四因子资产定价模型在中国股市的适用性，其研究结果表明，加入 6 个月动量因子的四因子资产定价模型在中国股票市场能够有效实现盈利，相比于三因子模型和 CAPM 模型，对股价的变化具有更高的解释能力[1]。

进入新世纪，世界对量化金融领域的研究更加系统和深入，金融机构开始使用机器学习算法精确识别交易窗口，互联网和大数据技术的普及为量化交易提供了更多的数据和信息，飞速发展的计算机行业让金融机构有能力凭借高性能的计算机和复杂的算法在海量数据中挖掘出有效的信息。Fama 和 French 在 2015 年提出五因子模型，在原先三因子模型的基础上重新挖掘了两个新因子。但是，在巫迪(2022)等学者关于五因子模型在中国 A 股市场适用性分析的研究中，结果表明，我国新冠时期股市中存在极强的规模效应和价值溢价效应，并通过实证检验和分析，说明五因子模型不完全适用于中国 A 股市场，没有表现出较好的解释能力[2]。

这些因子模型旨在更全面地解释证券收益率的变化。通过引入各个方面的数据，试图更准确地捕捉影响资产价格变动的因素，这些模型在资产定价、投资组合管理和风险管理等领域中有着广泛的应用，推动了量化金融的研究更深入的发展，为量化投资领域提供了更为广阔的应用前景，大力推动了金融市场的发展。

传统定价模型有很大的缺点。第一，这些传统因子模型通常基于线性假设，然而在实际市场中这种关系是未知的，可能是非线性的，这导致传统模型无法准确捕捉到所有的市场变化；第二，传统因子模型中的因子通常基于理论或经验选择，具有很大的主观性、偏见和局限性；第三，传统模型通常是静止的，而如今的市场越来越大，越来越复杂，这导致同一个模型无法处理大量复杂的数据并且难以适应市场的飞速变化；第四，传统因子模型更加注重解释资产收益的过程，而在预测方面表现的不尽如人意。

2.1.2 机器学习模型发展历史

近年来，随着计算机技术的不断迭代发展，决策树、支持向量机、人工神经网络、XGBoost 等模型已成为机器学习领域的主流算法。自从机器学习这个概念问世以来，这些算法一直受到社会各界的广泛关注，各行各业的研究者都在尝试将他们应用到各自的领域，通过跨多学科结合解决各自

的问题。在早期阶段，金融领域主要是使用统计模型和数学方法进行风险管理和决策支持，主要使用线性回归等传统方法进行预测，效果并不理想。直到上世纪 90 年代左右，机器学习算法飞速发展，美国学者 Robert Hecht Nielsen(1987)提出了对偶传播神经网络模型推动了神经网络的发展，为其在金融领域的应用奠定了基础，随和，神经网络开始在期权定价和信用评分等领域得到应用。进入大数据时代，Vladimir Vapnik(1995)提出完整的 SVM 支持向量机模型，用于解决分类和回归问题，被广泛应用于金融领域，在信用评分和市场预测中取得了巨大成功。Geoffrey Hinton、Yoshua Bengio、Yann LeCun 被认为是深度学习领域的奠基人，他们的工作在卷积神经网络(CNN)和长短时记忆网络(LSTM)等方面推动了金融领域的创新。桥水基金创始人 Ray Dalio 尝试将机器学习应用于宏观经济和市场预测，他的投资方法使得桥水基金成为全球最大的对冲基金之一，以对宏观经济和市场的研究、系统化的投资方法和风险管理而闻名。

在国外，机器学习算法在金融领域的应用已经有数十年的历史，有着非常成熟的理论体系和制度体系。而国内起步较晚，仍在追赶国外进度，程曦(2021)运用 Logistic 回归模型,对上证 50 指数和沪深 300 指数历史股票的价格趋势进行研究和分析, 得到结果预测准确率均要高 50%[3]；赵梦娜(2021)采用 SVM 和 BP 神经网络两种算法对沪深 300、中证 500 和上证 50 三只指数建立量化交易策略，并进行实证回测研究，收益效果显著[4]；付嘉华(2023)运用 KNN 算法对 SVM 进行改进，并将改进模型对沪深 300 指数进行回测，结果表明显著提高 SVM 算法的预测精度[5]。还有众多国内学者将各种模型进行优化和相互结合，以求得更高的准确性和稳定性，并研究各种模型在国内金融市场上的适用性。

2.1.3 Stacking 集成算法发展历史

尽管单一模型在金融领域预测方面取得了较好的效果，但是想要进一步提高准确性，不可避免的会出现过拟合等问题，导致准确性难以进一步提高，而集成算法能有效缓解过拟合问题。集成算法通过融合不同学习器的准确性和差异，尽可能的挖掘出数据中所蕴含的所有有价值的信息。Dasarathy(1979)提出集成系统的思想，并通过复合模型进行训练，得到了比单一分类器训练更好的预测效果。Schapire(1990)从理论上证明了一个算法是弱可学习的，那这个算法必定也是强可学习的，这为我们通过集成算法实现将一个弱学习器提升为强学习器提供了可能。田冬梅(2022)、蒋爽(2022)、张一帆(2023)等学者使用 Stacking 集成算法，分别集成 BP 神经网络、SVM 支持向量机、随机森林等单一模型组合，并在我国金融市场上进行了实证分析，结果均表明集成算法能显著提高预测准确率[6][7][8]。

2.2 经典单一模型

本节分别介绍了四个常用于股票趋势预测的机器学习分类器，包括：MLP 神经网络、RF 随机森林、SVM 支持向量机以及 Logistic 逻辑回归。

2.2.1 MLP 多层感知器

多层感知器（MLP）是一种最基本的前馈神经网络模型，灵感来源于对人类大脑的研究，类似于大脑皮层与神经元细胞的结构，MLP 构建了数层由仿神经元组成的层级结构。如下图 1 所示，最简单的 MLP 通常由三层组成：输入层、隐藏层和输出层，每个层次均分别由单个或多个节点组成，每个节点层会链接到下一层。

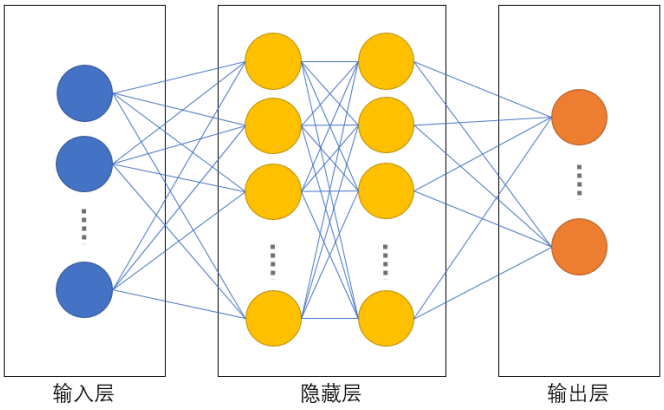

图 1 MLP 模型训练过程

输入层：输入层接收来自数据集的特征向量。每个特征向量对应一个输入神经元，输入层的神经元数量由数据集中的特征数量决定，负责输出对原始数据进行学习、处理和分析后信息。

隐藏层：隐藏层负责对输入数据进行非线性变换和特征提取，一个神经网络可以包含多个隐藏层，每个隐藏层包含多个如图 2 所示神经元。

输出层：输出层负责合并隐藏层的输出内容。输出层的神经元数量通常由问题的类型决定。例如，本文中将每日股票数据预处理成涨、跌、平三种状态，则输出层就有三个神经元

MLP 神经网络算法通过学习实现输入信息和输出结果之间的映射关系，为了提高预测准确性，在模型的训练过程中常加入激活函数，以引入非线性变换。每个神经元都采用激活函数，这样可以学习非线性模式和复杂特征。

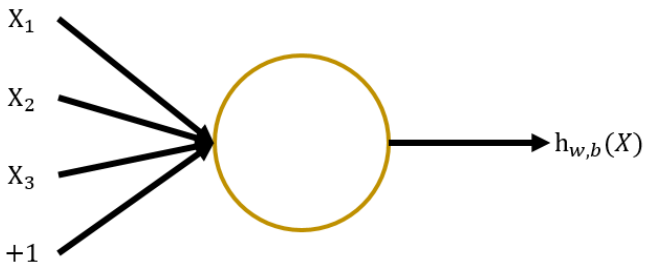

图 2 MLP 神经元示意图

图 2 中，“+1”表示偏置，该神经元进行了如下的计算：

$$h_{W,b}(x) = f(Wx + b) = f(\sum_{i=1}^{n} W_i x_i + b) \tag{2.1}$$

其中，W 表示权重，b 表示偏置值，f(·)即为激活函数。

MLP 神经网络的层与层之间是相互联系的，数据在层与层之间主要通过矩阵乘法和激活函数完成传递，MLP 神经网络的学习过程一般包括前向传播和反向传播两个重要步骤。

（1）**前向传播**。数据从输入层传播到输出层，经过线性变换后再通过激活函数进行非线性转换，得到每个神经元的激活值，再传递到下一层的神经元，直到传播到输出层。

（2）**反向传播**。计算损失函数对参数的梯度，利用梯度下降算法更新参数。从输出层开始逐层向后传播梯度，并根据链式法则计算每个参数对损失的贡献。在传播过程中，根据激活函数的导数计算梯度并更新参数。

通过前向传播和反向传播的交替执行，模型参数不断进行优化，使预测结果逐渐接近真实标签，实现对数据的准确建模和预测。

2.2.2 RF 随机森林

随机森林算法是集成学习 Bagging 类方法中的一种，它通过构建多个决策树来提高单个决策树的性能和泛化能力，即将固定数量和不同特征的决策树集成起来，使其具有较高的预测性能和鲁棒性，具有较高的准确性，能够处理大量的输入特征和样本数据。并且对于股票数据，其特点包括数据量大、数据维度广，容易受到不可知因素的影响，具有较高的偶然性，容易出现异常值，而 RF 算法在面对异常情况或不可预测事件时，仍能保持稳定和可靠运行的能力，因此在构建 Stacking 模型时选择 RF 算法作为基模型之一，以期望捕捉股票数据中所蕴含的非线性关系。

RF 随机森林算法可用于对股票数据涨跌情况进行分类。在分类问题中 ，每一个决策树的输出结果作为一个类别标签。随机森林模型核心是由多课决策树所构成，模型如图 3 所示。

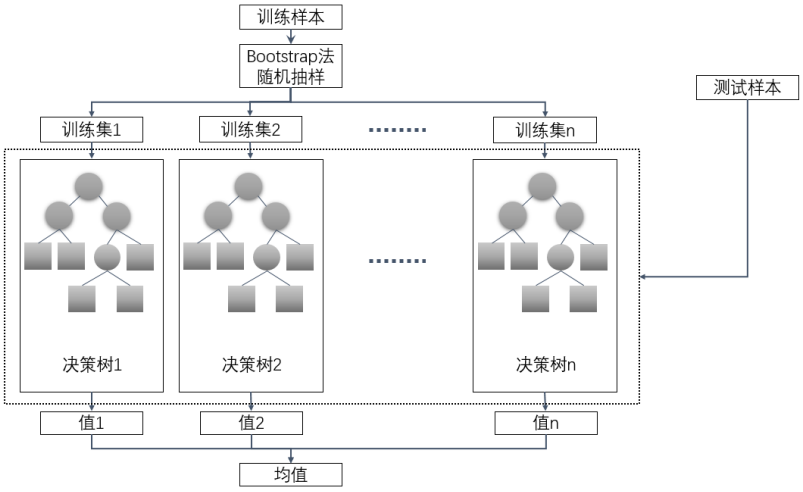

图 3 RF 模型训练过程

随机森林模型训练过程如下：

（1）从训练集中随机选择一个子集作为每棵决策树的训练集。

（2）针对每个节点，随机选择一个特征子集作为候选特征，并选取最佳特征进行分裂，递归地构建决策树。

（3）重复步骤（1）（2），生成 T 棵树

为了确保决策树的充分随机化，随机森林引入了以下两种方法：

随机抽样样本：从训练集中用随机抽样的方法抽取一个子集用于训练，使得每棵树都在略微不同的数据集上进行训练。

随机选择特征：在每个节点上，随机森林仅考虑一个随机选择的特征子集来进行分裂，这种方法使得每棵树的结构也有所不同。

最后将每一棵树的结果合并取平均值得到最终输出结果。

2.2.3 SVM 支持向量机

支持向量机是一种监督学习算法，其原理是在数据集中找到一个超平面，能够有效地将不同类别的数据分隔开。SVM 是基于监督学习的二元广义分类器，通过对数据的学习构建模型，使用该模型可以将输入的多维数据映射到已经划分好的空间，实现对未知类别数据进行预测分类的功能。

对于给定的数据集，将特征向量（二维向量为例）映射为空间（平面）中的一些点，如图 4 所示为一二分类模型。SVM 的目标是计算出一条曲线，以最优的方式划分出两个空间，使得能够以最佳的方式区分出这两类样本点。对于新的需要预测的数据，经过空间映射到划分好的空间上，判断与曲线的关系，就能得到模型的预测结果。

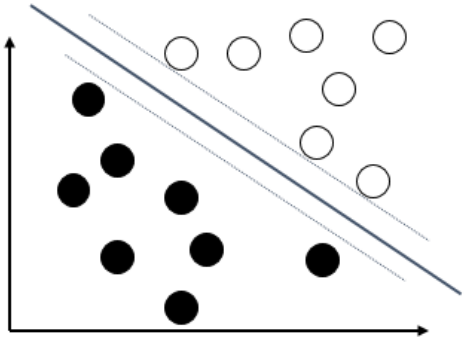

图 4 线性 SVM 模型

根据样本集的性质，SVM 可分为如下两种类型：

（1）**线性可分**型指存在一个线性超平面可以将两类样本完全分开，如图 4 所示。

对于给定的训练集$\{(x_1,y_1),(x_2,y_2),\cdots(x_n,y_n)\}$，其中$x\in R^n$，$y\in\{-1,1\}$，假设该训练集可以被一个超平面线性划分，记该超平面为$(w\cdot x)+b=0$。

在训练集中，若所有数据均可被超平面正确划分且最近的异类数据间距离最大，则称为最优超平面。最近的异类向量称为支持向量，它们唯一确定了超平面，并且两类支持向量之间的距离即为分类间隔。构建最优超平面并最大化分类间隔的问题可表述为以下最优化问题：

$$\text{Minimize } \Phi(w,b)=\frac{1}{2}||w||^2 \tag{2.2}$$

$$s.t.\ y_i(w\cdot x_i)+b-1\geq 0\ i=1,2,\cdots,l \tag{2.3}$$

构造拉格朗日函数：

$$L(w,\alpha,b)=\frac{1}{2}||w||^2-\sum_{i=1}^{l}\alpha_i y_i(x_i\cdot w+b)+\sum_{i=1}^{l}\alpha_i\ \alpha_i\geq 0,i=1,2,\cdots,l \tag{2.4}$$

进而求得最优解，即最优超平面。

（2）**线性不可分**型指不能使用一条直线或者一个直面把样本分为两类，如图 5 所示。。

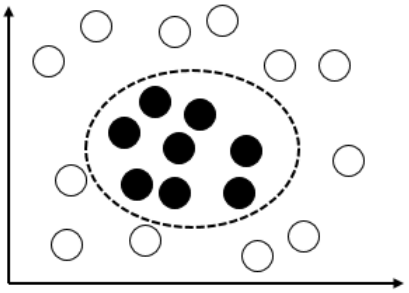

图 5 非线性 SVM 模型

理论上，可通过非线性映射，如选用不同的核函数，将数据映射到高维特征空间，然后在此空间中，再次寻找线性分类规则，并构建出线性广义最优分类超平面。

目前常用的非线性映射核函数有如下三种：

（1）多项式核函数：$K(x,y)=[(x\cdot y)+1]^d$。

（2）高斯径向基核函数：$K(x,y)=e^{-||x-y||^2/2\sigma^2}$。

（3）Sigmoid 函数：$K(x,y)=\tanh(k(x\cdot y)+\delta)$。

核函数的原则对于模型的准确性有着至关重要的左右，本文主要通过高斯径向基核函数作为模型的核函数生成支持向量机模型，进行分类计算。

2.2.4 Logistic 逻辑回归

逻辑回归是一种广义的线性回归。逻辑回归的模型表示为：

$$\mathrm{p} = \mathrm{g}(\mathrm{w}^T\mathrm{X} + \mathrm{b}) = \frac{1}{1+e^{-(wx+b)}} \tag{2.5}$$

经过对数化可以得到：

$$\ln\frac{p}{1-p} = w^T X + b \tag{2.6}$$

其中$\ln\frac{p}{1-p}$被称为对数几率，因此有：

$$\ln\frac{p(y=1|x)}{p(y=0|x)} = w^T\mathrm{X} + \mathrm{b} \tag{2.7}$$

为了求得最佳的参数选择，采用极大似然估计法对参数进行估计。对于样本i，若标签为 1，则设其概率为$\mathrm{h}(\mathrm{x_i})$，若标签为 0，则设其概率为$1-\mathrm{h}(\mathrm{x_i})$。构造极大似然函数：

$$\mathrm{L}(\mathrm{w}) = \prod_{i=1}^{n} h(x_i)^{y_i}\,(1-h(x_i))^{1-y_i} \tag{2.8}$$

为方便求解，将其对数化，并且除以样本总数 n，乘以-1 转化为求极小值的优化问题，得：

$$L = -\frac{1}{n}\sum_{i=1}^{n}[y_i(w^T x + b) - \ln(e^{w^T x+b} + 1)] \tag{2.9}$$

然后采用梯度下降法，求解得到上式最小时是w，得到逻辑回归函数。

第 3 章 基于 Stacking 模型的量化交易策略构建

3.1 集成学习算法

集成学习是机器学习方法的一种，它通过结合多个模型的预测结果，可以获得比单个模型更优的预测结果。目前，集成学习主要有以下三种类型：Bagging（装袋法）、Boosting（提升法）和 Stacking（堆叠法）。

Bagging 算法是并行式学习的典型代表。其核心思想在于利用自助采样方法，从训练集中采用随机抽样方式均匀且有放回地抽取多个子集。然后，针对每个子集独立地训练一个学习器，最终将所有学习器的预测结果结合起来，采用取平均或投票的方式得到最终的预测结果。典型的 Bagging 算法例如前文提到的 RF 算法。

Boosting 算法是一种串行式的集成学习方法，其基本思想是利用错误来不断提升模型的性能。首先，使用初始权重训练出一个弱学习器 1，然后根据该弱学习器的学习误差重新调整训练样本的权重，使得学习错误的样本获得更高的权重。接着，在调整后的权重下再次进行训练，得到弱学习器 2，依此类推，直到达到指定数量的弱学习器。最后，综合所有弱学习器的结果得到最终的预测结果。典型的 Boosting 算法包括 AdaBoost、XGBoost 等。

Stacking 算法是一种分层式的集成学习方法。其基本思想是，综合多个模型的预测结果进行二次学习，形成更为强大的新模型，进而做出预测。

Stacking 过程通常分为两个阶段，如图 6 所示，第一阶段是训练阶段，由若干个弱学习器组成，在这个阶段中，训练数据被分为K个折叠，对于每一个基模型进行五折交叉验证，输出对数据集的一个预测结果作为新的特征；第二阶段是利用第一阶段每一个基模型所产生出来的预测结果作为新的训练集，进行再次训练。最后得到预测结果。

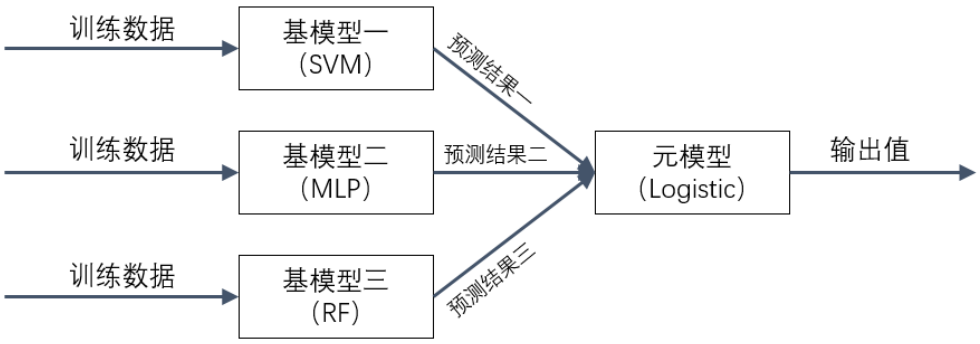

图 6 Stacking 模型训练过程

而对于每一个基模型，本文采用五折交叉验证法进行训练，其训练过程如图 7 所示。在每一个基模型的训练中，将训练集拆分成五份，其中四份用于模型的训练，剩下一份输入到训练好的模型中，对此部分数据进行预测并输出相应的标签。如上过程依次循环 5 次，得到了关于训练集的一列预测的标签，将数个基模型输出的标签合并作为新的训练集传递到元模型中再次训练，这就完成了一次 Stacking 学习过程。

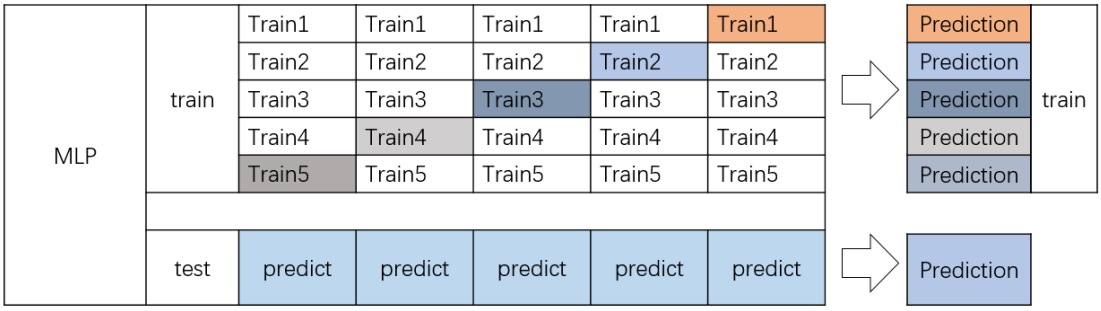

图 7 五折交叉验证

3.2 基于 Stacking 集成算法的预测模型构造

在选择基模型和元模型时，需要考虑它们的性能、复杂度、以及彼此之间的差异性，要求基模型足够强大，能够充分挖掘并学习数据中的所有特征；元模型要求尽量简单，可解释性强，能够融合基模型做出的假设，并最终输出融合模型的结果。以确保 Stacking 模型能够获得良好的泛化能力。

（1）SVM 是最典型的分类模型，在处理分类问题时具有良好的泛化能力，特别是在处理复杂数据集和高维数据时表现出色。而股票市场的数据往往是高维的，SVM 能够有效地捕捉其中的复杂关系。

（2）MLP 作为一种强大的深度学习模型，能够有效地学习复杂的非线性关系，在股票预测中，MLP 能够更好地捕捉可能存在许多非线性的因素影响股价变动。

（3）RF 是一种集成学习算法，借助多个决策树的组合进行预测，展现出良好的鲁棒性和泛化能力，在股票预测中，RF 能够处理大量特征和样本，并且不容易受到噪声的影响。

元模型的选择在于对基模型输出的概率进行整合，而 Logistic 回归是一种简单而有效的分类方法，尤其适用于处理概率输出，因此，选择 Logistic 逻辑回归作为基模型。

3.3 基于 Stacking 集成算法的量化交易策略构造

3.3.1 交易信号识别过程

对于股市来讲，如何对股票进行分类是一个重要并且复杂的问题，以预测未来一周市场状况为例，未来一周的股票市场情况可能有：持续上涨、持续下跌、先涨后跌，先跌后涨，持续震荡五种情况。尽管在视觉上，人们可能能够主观地对图形进行大致分类，但从计算机对图形的识别角度来看，确定这些类别是相当困难的。因此，需要从实际有利于代码实现的角度出发，将股票分类定义为上涨（涨），震荡（平）和下跌（跌）三种状态。

对于涨、跌、平三种状态的界定，本文主要通过均线比较来区分，主要尝试使用两种方案。第一，比较短期均线（5-20 日均线）和长期均线（20-60 日）的相对位置来进行分类，若短期均线上穿长期均线时，为股票上涨信号，此时应当买多或者空方平仓；若短期均线下穿长期均线时，为股票下跌信号，此时应当卖空或者买方平仓；主要适用于大盘趋势稳定的股票，识别长期趋势。第二，比较当日成交价与短期均线（5-20 日均线），若当日成交价大于短期均线，为股票上涨信号，此时应当买多或者空方平仓；若当日成交价小于短期均线，为股票下跌信号，此时应当卖空或者买方平仓；主要适用于股价变化幅度小、速度快的股票，抓住短期机遇。

为了避免股票震荡导致频繁交易，应当设置一个阈值，例如短期均线上穿或下穿长期均线的 1.005 倍时，才会被识别为上涨或下跌信号，否则将被归类为震荡。

3.3.2 模型训练过程

通过移动滑窗，记当前日期为 T 日，样本期为 T-400 日到 T-1 日，分别使用网格搜索法、贝叶斯搜索法和随机搜索法对 SVM、MLP 和 RF 模型参数进行自动寻优，并使用最佳的参数组合进行模型训练，构建 Stacking 模型，最后利用 T 日的输入指标预测输出指标。

而由于模型可调整参数众多，因此需要根据每一支股票的大盘趋势、特性特征、投资者的风险喜好等动态调整如长短均线期限的组合、模型参数搜索范围、止盈止损额度、进场退场时机等。并且由于投资者个人风险喜好等原因，没有统一的优劣评价标准，因此需要依照每一支股票和投资者的风险喜好及风险承担能力等，定制化调整参数范围，依次进行组合尝试。

3.3.3 交易策略构建过程

通过上诉模型学习历史 400 日的股票数据，并预测得到了当日的涨跌情况后，构建如下交易策略，其中，参数T，x，n均需要根据实际情况进行调整：

（1）如果预测分类为涨，则在市场行情低于 T 日收盘价时买入做多，如果涨幅超过x%则卖出，否则到T + n日平仓；

（2）如果预测分类为跌，则在市场行情高于 T 日收盘价时卖出做空，如果跌幅超过x%则卖出，否则到T + n日平仓；

（3）如果预测分类为平，则不进行交易，继续持有或者保持观望。

第 4 章 基于 Stacking 模型的量化交易策略实证分析

本章基于贵州茅台股票（600519）2016 年 01 月 01 日到 2020 年 12 月 31 日的每日交易数据进行实证分析。计算了 MA、MACD、CCI 等技术指标作为趋势信息辅助，与预处理后的数据一并输入到 Stacking 模型中，随后对模型与策略进行实证研究并回测。

4.1 数据预处理

本文通过 python 中调用 TuShare 第三方库获得贵州茅台（600519）的日频数据，包括但不限于：开盘价、最高价、最低价、收盘价和成交量。依据收盘价计算股票的 14 日平均真实波幅 ATR 指标、

20 日顺势指标 CCI、（12，26，9）异同移动平均线 MACD 指标，计算 4、5、6、10、15、20、40 日均线作为备用。

此外，为了增加模型的通用性，由于不同股票处于不同行业，数据中可能存在巨大差距，不同特征的量级存在差异，这会导致模型预测效果差异过大，极其不稳定，因此有必要对数据进行标准化处理，从而保证最终的预测效果。

根据赵梦娜等人的研究，标准化处理方式不采用传统的均值、标准差的形式，因为此种标准化方式会导致数据带有大量历史信息，进而影响模型的预测效果。因此依据其研究，本文使用当天数据与前一天对应数据做比值方式进行，并将结果减 1 使得落在-1 到 1 之间。选取了 9 个变量指标，如表 1 所示。

表 1 数据标准化

输入参量	标准化处理方式
X1	昨日开盘价/最新收盘价-1
X2	昨日收盘价/最新收盘价-1
X3	昨日最高价/最新收盘价-1
X4	昨日最低价/最新收盘价-1
X5	最新成交量/昨日成交量-1
X6	最新波动率/昨日波动率-1
X7	今日开盘价/最新收盘价-1
X8	最新最高价/最新收盘价-1
X9	最新最低价/最新收盘价-1

这种标准化方式，不仅仅能够让数据处于同一数量级之间，而且可以很好地反应股票价格的趋势变化，更好地保留了股票的趋势信息，有效提高模型的预测准确率和模型的适用性。

4.2 单一模型实证结果分析

本节尝试使用单一模型对股票涨跌情况进行预测，并且构建合适的交易策略，主要说明单一模型的回测结果以及构建交易策略的过程中可能存在的问题，为下文构建 Stacking 模型提供参考。

4.2.1 SVM 模型回测结果及分析

在 Backtrader 回测框架中调用 SVM 函数，并且使用网格搜索对模型参数进行大致寻优，采用滚动 400 日的贵州茅台股票价格信息，对其涨跌情况进行预测，并根据预测结果构建了合理的交易策略。

表 2 SVM 回测结果

指标	结果
平均正确率	35.60%
夏普比率	1.30
最大回测率	7.65%

总收益率	35.99%
平均每笔收益率	0.0295%
年化收益率	7.73%
胜率	88%

回测结果表明，模型预测平均正确率仅 35.60%，说明 SVM 模型对股票涨跌情况的学习能力较差。观察正确率变化图如图 8 所示，短期来看模型正确率变化较为平滑，说明单一 SVM 模型抗干扰能力较强，不易受到异常值干扰。而由于正确率较低，因此需要采用宽松的交易策略，依据股票的大盘趋势特点设置较大的阈值，对策略参数进行多次调整期望能够实现在历史数据上的盈利，如上表结果所示，策略年化收益率达到 7.73%。但是由于预测正确率太低，因此在不可预料的未来，在变化莫测的股票市场必然会导致大额亏损，很难使用该策略进行实际交易。

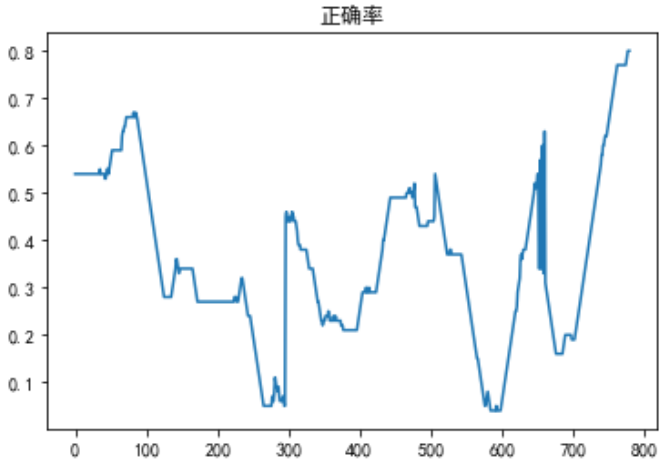

图 8 SVM **正确率变化图**

4.2.2 MLP 模型回测结果及分析

改用 MLP 模型对贵州茅台股票涨跌情况进行预测，尝试使用贝叶斯优化算法尝试对模型参数进行调优，回测结果如下表所示。

表 3 MLP **回测结果**

指标	结果
平均正确率	85.29%
夏普比率	0.83
最大回测率	6.27%
总收益率	25.68%
平均每笔收益率	0.0211%
年化收益率	5.46%
胜率	76.39%

回测结果表明，模型预测平均正确率为 85.29%，说明 MLP 模型能够很好的捕捉股票价格的变化趋势，能够充分挖掘出股票价格数据以及指标中所蕴含的趋势信息。其预测正确率变化图如图 9

所示，显示出该模型在一般情况下具有较优秀的预测效果，但是预测效果及其不稳定，使得预测正确率出现较为频繁的波动。因此，由于综合正确率较高，长期来看正确率较为稳定，可以实行较为大胆的交易策略，依据股票的大盘趋势特点设置更为敏感的阈值，可以对模型预测正确率提出类似于大于60%的要求，只有当正确率大于某个阈值时才能够进行下一步交易，能够有效提高 MLP 策略交易成功的概率。

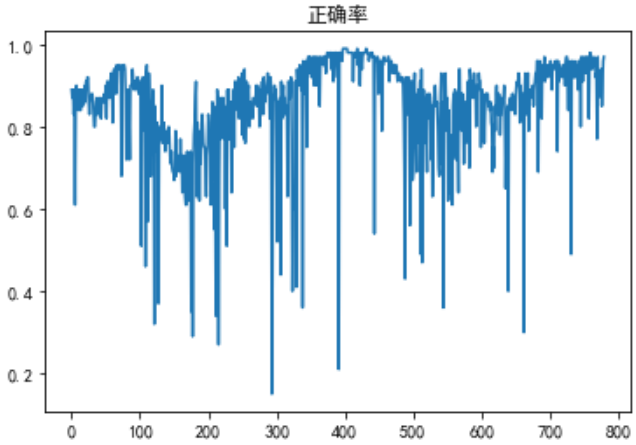

图 9 MLP 正确率变化图

4.2.3 RF 模型回测结果及分析

改用 RF 模型对贵州茅台股票涨跌情况进行预测，尝试使用随机搜索算法尝试对模型参数进行调优，回测结果如下表所示。

表 4 RF 回测结果

指标	结果
平均正确率	81.56%
夏普比率	0.45
最大回测率	11.07%
总收益率	20%
平均每笔收益率	0.0163%
年化收益率	4.21%
胜率	71.83%

回测结果表明，模型预测平均正确率为 81.56%，说明 RF 模型能够有效地学习股票涨跌情况。而预测正确率变化图如图 10 所示，极其不稳定，短期来看波动较大，长期趋势依旧不够稳定。因此需要降低标志变化的频率，在进行标志划分时，需要扩大划分阈值，在构建交易逻辑时采用较为宽松的策略，调整交易参数，最终使得总收益率达到 20%。

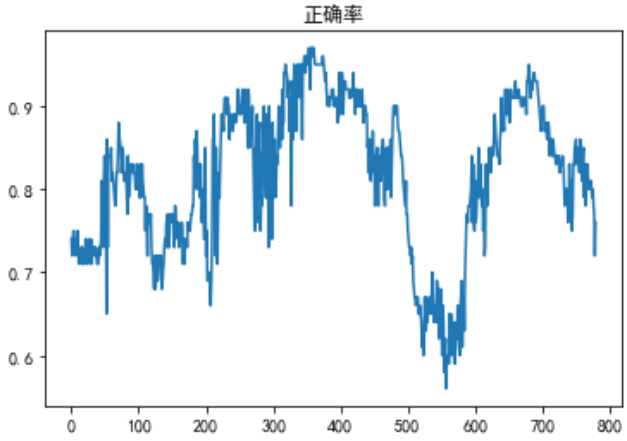

图 10 RF 正确率变化图

4.2.4 模型及策略可能存在的问题总结

从以上实证结果及分析可知，无论是简单分类模型如 SVM，亦或是复杂分类器如 MLP，在构建预测模型以及交易策略时均面临以下问题。

（1）风险较高。单一模型均没有很好的应对异常情况的机制，导致策略很难处理受到的突发情况，表现在股市中，即突发事件如：战争、重大政策变化、自然灾害、公司内部问题等。

（2）准确率不稳定。模型预测准确率较低，或者准确率波动较大，导致需要对交易策略的参数进行更为详细且严格的要求，进一步使得策略的适用性降低，因此随着时间的发展，股票的风格会发生偏移变化，旧参数的模型便不在适用新的形势，需要时刻对参数进行调整，这就偏移了量化投资的初始目的。

4.3 模型训练与超参数寻优

4.3.1 预测模型参数自动寻优

根据本文第三章所构建的两层 Stacking 模型中，第一层基学习器有 SVM 支持向量机、MLP 神经网络和 RF 随机森林组成，第二层元学习器由 Logistic 逻辑回归构成。

通过广泛阅读文献及查找资料，获得各个模型常用的参数范围如表 5。

表 5 超参数搜索范围

模型	参数范围
支持向量机	C：0.1~10
	Gamma：0.1~10
	核函数：径向基函数核
多层感知机	隐藏层：50，100，150
	Alpha：-6~0
	初始学习率：-4~0
	最大迭代次数：100，200，300
随机森林	树的数量：50，100，150
	最大深度：None，10，20
	最小样本数：2，5，10

常见的参数寻优算法包括：网格搜索法、随机搜索法、贝叶斯优化、遗传算法、粒子群算法等，考虑各个模型参数搜索范围及搜索时长等因素，查阅相关资料，在模型实现过程中，对 SVM 采用网格搜索法，对 MLP 采用贝叶斯优化，对 RF 采用随机搜索法，以求获得较高的学习性能及较快的训练时间。

4.3.2 交易策略参数选择

在第三章关于交易策略的构建中，有关股票分类的方法、不同均线比较的组合可以随意选择，分类阈值大小、最大持仓时间、开仓时机和最小买卖单位需要根据股票特性、投资者的投资喜好、风险喜好及风险承担能力来自由选择。经过多次尝试各种参数组合，得到相对优秀的范围。

而对于贵州茅台股票来说，如图 11 所示，测试时间内大盘趋势为上涨，小范围内波动不明显，适合长线投资跟随大趋势。因此阈值设置为±0.05%~ ± 5%，短期均线围绕周线，选择 4、5、6 日均线；长期均线围绕季线，选择 30，40，50 日均线；考虑预测风险和预期收益，止损设定最大持仓时间为 5~7 日，止盈设定最大涨跌幅为 2%。

图 11 贵州茅台（600519）K 线图

4.4 模型与策略结果分析

标的资产：贵州茅台（600519）
回测时间：2016-01-01 到 2020-12-31
买卖单位：2000 股/手
手续费：0.2‰
初始资金：10000000 元
最大涨跌幅：2%
相关回测指标部分结果如表 6 所示。

表 6 回测指标数据对比

指标	策略 1	策略 2	策略 3	策略 4
剩余资金	13178013.02	15295625.77	12038706.38	12599006.20
平均正确率	90.13%	90.01%	83.99%	81.82%
夏普比率	0.88	1.48	0.38	0.65
最大回测率	13.17%	9.92%	10.88%	6.67%
最大回测额	1887629	1552116	1462380	870956
总收益率	27.60%	42.98%	18.55%	23.10%
平均每笔收益率	0.0227%	0.3489%	0.0154%	0.0189%
年化收益率	5.88%	9.19%	3.91%	4.90%
胜率	73.45%	76.42%	75.00%	76.54%

对于策略 2，采用 4 日 MA 与 40 日 MA 相对位置作为股票分类标准，阈值设置为±0.5%，最大持仓时间设置为 7 日。在滚动训练过程中，贵州茅台分类正确率如图 12 所示。

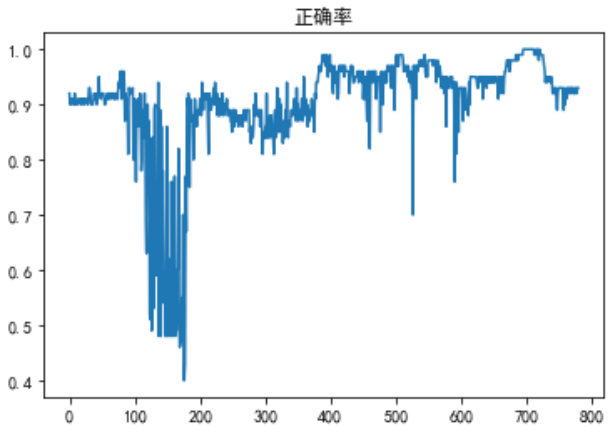

图 12 策略 2 正确率变化图

如上图所示，预测正确率维持在 90%上下，但是在 2018 年 6 月至 7 月，预测正确率暴跌至 50%，并且有较大幅度波动，观察日 K 线图发现，2018 年 6 月之前贵州茅台历史数据大盘均保持上涨趋势，至 2018 年 6 月开始出现震荡，因此此时历史数据仅包含较少的股价震荡时的特征，因此预测正确率大幅下跌。但是经过一段时间的震荡，训练集逐渐包含震荡时的历史数据，模型也成功学习并掌握了此种特征，最终使得正确率回涨，并且在未来再次出现震荡时能够成功进行预测。

如图 13 所示，为回测过程中可用资金的变化图像，显示资金在排除买卖产生的波动后，整体趋势呈上升状态，表明策略能够保证持续盈利。

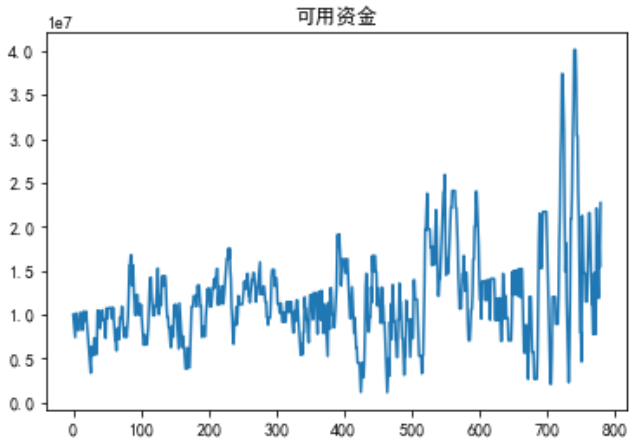

图 13 策略 2 可用资金变化图

如图 14 所示，第一个图为当前的经纪人状态，cash 表示当前可用现金金额，value 表示当前账户中的总价值；第二个图为交易记录中的盈利和亏损情况，蓝点表示交易中实现的盈利，表明该笔交易实现了正收益，红点表示交易中实现的亏损，意味着该笔交易产生了负收益；第三个图为股票价格及买入卖出情况，并且有各期 MA 均线，下方的柱状图显示了每笔交易的交易量；第四、五、六个图分别为 CCI、MACD 和 ATR 指标。

观察策略 2 的回测结果。夏普比率达到 1.48，表明单位风险下所获得的超额收益较高，认为是一个比较好的投资表现；最大回撤率为 9.92%，表明策略在最坏情况下可能面临的损失程度较小；总收益率为 42.98%，年化收益率为 9.19%，表明执行策略的预期收益高于银行存款。综上所述，策略 2 可以被认为是一个中低风险的投资策略。

图 14 策略 2 可视化回测结果

对于策略 3，夏普比率 0.3747，意味着该种参数组合盈利能力略高于银行存款，适用于希望对资产进行保值的投资者。观察其正确率变化，如图 15，在 2018 年 6 月至 7 月出现暴跌，原因与策略 2 相同。

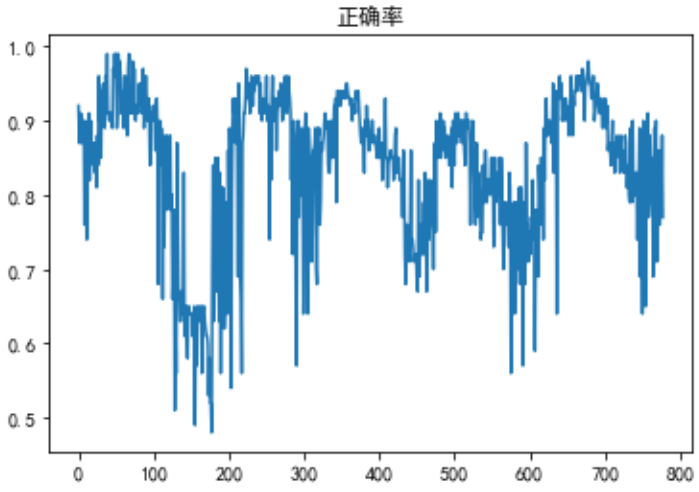

图 15 策略 3 正确率变化图

第 5 章 研究结论与展望

5.1 研究结论

本文旨在对股票的涨跌情况进行预测，并基于预测结果进行模拟交易，构建了一套完整的、可应用于实际交易的日频量化交易策略，对于量化交易的初学者具有一定的指导意义。

本文基于机器学习领域有关量化交易策略的相关文献，构建了预测模型和交易策略。基于 Stacking 模型，采用三种参数优化方法对模型参数进行寻优；然后，使用贵州茅台股票日频数据进行回测验证分析，并不断尝试各种交易参数组合，寻求最佳的回测效果，验证交易策略是否具有一定的效果及获取收益的能力。

使用普通单个模型进行预测时，预测准确率均为 50%上下浮动，对模型参数进行寻优，也很难大幅度提高模型的正确率。而使用 Stacking 融合学习模型时，预测准确率为 75%上下浮动，这表明优化的 Stacking 模型具有更好的性能，能够有效提高模型的预测准确率，有效提高交易策略的胜率，期望获得更高的收益。

其次，本文针对贵州茅台股票进行了实证分析，对贵州茅台股票进行了模型参数的寻优以及策略参数的组合，寻找到了数个性能优秀的参数组合，能够在设定的时间中实现盈利，取得较好的交易效果。

最后，本文所构建的模型经过微调，可适用于类似于贵州茅台，股价长期趋势稳定的股票，若要拓展到其他股票，则需要继续对股票分类方法、参数及策略进行更多的针对性优化。

5.2 研究不足及对策

（1）除了基本的股票日频数据和常见的技术指标之外，还可以考虑进行特征工程，添加一些其他的指标和特征，量化各个指标对股票涨跌的贡献率，寻找对股票涨跌贡献较大的指标；

（2）所构建的模型基于历史数据进行预测，没有考虑突发情况，突然出现历史数据中未包含的信息，会导致预测准确率突然暴跌，对资金造成短期的大量损失；

（3）模型参数自动寻优所消耗的时间过长，因此未进行大范围的参数寻优和更好的寻优方法，如粒子群算法、遗传算法等，可以在实证时针对特定的股票，先通过大范围寻优后缩小参数范围，确定数个相对较优的参数选择，期望能够大幅降低模型的训练时间；

（4）本模型仅采用一支股票作为交易标的，但实际上有大量股票可以进行组合交易，通过分散化投资能够显著降低非系统性风险；

投资有风险，理财需谨慎。

参考文献

[1] 欧阳志刚,李飞.四因子资产定价模型在中国股市的适用性研究[J].金融经济学研究,2016,31(02):84-96.
[2] 巫迪.新冠疫情下 Fama-French 五因子模型在中国 A 股市场适用性分析[D].对外经济贸易大学.2022.
[3] 程曦.基于 Logistic 回归模型对股票趋势的预测[D].山东大学. 2021.
[4] 赵梦娜.基于 SVM 和 BP 神经网络的量化策略研究[D].大连理工大学.2021.
[5] 付嘉华.基于 KNN 改进的 SVM 股票趋势预测方法及应用[D].河南大学.2023.
[6] 田冬梅,刘家鹏,张芹等.基于 Stacking 的集成算法在证券趋势预测中的应用[J].现代电子技术.2022.45(19):115-121.
[7] 蒋爽.基于 Stacking 集成学习的商品期货价格预测与交易策略研究[D].西南财经大学.2022.
[8] 张一帆.基于 Stacking 模型的量化交易策略研究[D].西南大学.2023.
[9] 表二苏.股票价格预测方法研究[D].天津大学.2007.
[10] 王珏.基于三层 Stacking 集成学习的商品期货量化交易策略研究[D].华中师范大学.2022.
[11] 徐慧丽.Stacking 算法的研究及改进[D].华南理工大学.2018.
[12] 陈凯,朱钰.机器学习及其相关算法综述[J].统计与信息论坛.2007.(05):105-112.
[13] 谢琪,程耕国,徐旭.基于神经网络集成学习股票预测模型的研究[J].计算机工程与应用.2019.55(08):238-243.
[14] Bo A Z, Tingting C. Stock return prediction: Stacking a variety of models[J]. Journal of Empirical Finance. 2022. 67288-317.
[15] Jiang M, Liu J, Zhang L, et al.An improved Stacking framework for stock index prediction by leveraging tree-based ensemble models and deep learning algorithms[J]. Physica A: Statistical Mechanics and its Applications. 2020. 541(C): 122272-122272.
[16] Wang, Zhuowen. Prediction of stock market prices using neural network techniques[D]. University of Ottawa (Canada). 2004.

附录

```
import pandas as pd
import numpy as np
from datetime import datetime
import backtrader as bt
import matplotlib.pyplot as plt
import tushare as ts
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from backtrader.feeds import PandasData
import backtrader.indicators as btind
from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from hyperopt import fmin, tpe, hp, STATUS_OK, Trials
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import StackingClassifier
import multiprocessing
from pyswarm import pso
from xgboost import XGBRegressor
from backtrader.analyzers import SharpeRatio, DrawDown, TradeAnalyzer
plt.rcParams["font.sans-serif"]=["SimHei"]  # 设置画图时的中文显示
plt.rcParams["axes.unicode_minus"]=False  # 设置画图时的负号显示

def get_data(code='600519',starttime='2016-01-01',endtime='2020-12-31'):
    df=ts.get_k_data(code,start=starttime,end=endtime)
    df.index=pd.to_datetime(df.date)
    df['openinterest']=0
    df=df[['open','high','low','close','volume','openinterest']]  # 整合数据至满足 backtrader 框架
    return df
stock_df=get_data()
fromdate=datetime(2016,1,1)
todate=datetime(2020,12,31)
data=bt.feeds.PandasData(dataname=stock_df,fromdate=fromdate,todate=todate)  # 加载数据
import warnings
warnings.filterwarnings("ignore")
# 构建策略
```

```
accuracies = [] #记录正确率
total_money = [] #记录资金
class MyStrategy(bt.Strategy):
    params = (
        ('lookback_period', 300),
        ('split_index', 225),
        ('s_time', 4),
        ('m_time', 6),
        ('l_time', 15),
    )
    def __init__(self):
        self.order=None
        self.start_cash = 10000000
        self.stop_loss = 0.05  # 设置止损
        self.take_profit = 0.10  # 设置止盈
        self.data_close = self.datas[0].close
        self.model = SVC(kernel='linear')  # 使用线性核的支持向量机模型
        self.cci = btind.CommodityChannelIndex(self.data, period=20)
        self.macd = btind.MACD(self.data, period_me1=12, period_me2=26, period_signal=9)
        self.ma4 = bt.indicators.SimpleMovingAverage(self.data, period=self.params.s_time)
        self.ma6 = bt.indicators.SimpleMovingAverage(self.data, period=self.params.m_time)
        self.ma15 = bt.indicators.SimpleMovingAverage(self.data, period=self.params.l_time)
        self.ma20 = bt.indicators.SimpleMovingAverage(self.data, period=20)
        self.ma40 = bt.indicators.SimpleMovingAverage(self.data, period=40)
        self.atr = bt.indicators.AverageTrueRange(self.data, period=14)
        self.trade_analyzer = bt.analyzers.TradeAnalyzer()
        self.entry_price = 0
        self.price_record = []
    def next(self):
        # 如果历史数据不足以进行预测，则不执行任何操作
        if len(self.data) < self.params.lookback_period:
            return
        # 获取历史数据
        hist_open = self.data.open.get(size=400)
        hist_high = self.data.high.get(size=400)
        hist_low = self.data.low.get(size=400)
        hist_close = self.data.close.get(size=400)
        hist_volume = self.data.volume.get(size=400)
        hist_cci = self.cci.get(size=400)
        hist_macd = self.macd.get(size=400)
        hist_atr = self.atr.get(size=400)
        hist_ma4 = self.ma4.get(size=400)
        hist_ma6 = self.ma6.get(size=400)
        hist_ma15 = self.ma15.get(size=400)
```

```
            hist_ma20 = self.ma20.get(size=400)
            hist_ma40 = self.ma40.get(size=400)
            hist_data_df =
pd.DataFrame(data=zip(hist_open,hist_high,hist_low,hist_close,hist_volume,hist_atr,hist_cci,hist_macd,hist_ma4,hist_ma6,hist_ma15,hist_ma20,hist_ma40),
columns=['open','high','low','close','volume','atr','cci','macd','ma4','ma6','ma15','ma20','ma40'])
            #print(hist_data_df)  # 得到历史 n 日的交易数据
            if hist_data_df.isna().any().any() or hist_data_df.empty:
                return
            st_data_df = data_deal(hist_data_df)
            #print(st_data_df)
            try:
                signal = svm_predict_Stacking(st_data_df)
            except:
                print(f"模型训练失败，时间：{self.datas[0].datetime.datetime(0)}")
                signal=0
            #signal=1
            if self.order:  # 如果有挂单，则不执行后续操作
                if self.order.status in [bt.Order.Submitted, bt.Order.Accepted]:
                    print("等待订单成交")
                    return  # 等待订单成交
                elif self.order.status in [bt.Order.Partial, bt.Order.Completed]:
                    print("清空订单状态")
                    self.order = None  # 清空订单状态
                elif self.order.status in [bt.Order.Canceled, bt.Order.Rejected]:
                    print("取消订单并重新发出")
                    self.order = None  # 取消订单并重新发出
                    return
                else:
                    print("交易失败")
            # 根据预测结果生成买卖信号
            count = 0
            if len(self.price_record) > 0:
                for i in range(len(self.price_record)):
                    self.price_record[i][2] += 1
                if self.price_record[0][2] == 9:
                    count-=self.price_record[0][1]  # 持有 n 日者平仓
                    self.price_record.pop(0) #踢出第 n 日前的数据，表示平仓
                    print(f'持有达期限，平仓，时间：{self.datas[0].datetime.datetime(0)}")
            if len(self.price_record) > 0:
                j=[]
                for i in range(len(self.price_record)):
                    if self.price_record[i][1] == 1 and self.data_close[0]/self.price_record[i][0] > 1.02: #如果持多仓且涨幅超过 2%
```

```
                        count -= 1 #卖多平仓
                        print(f"卖多平仓，位置{i}")
                        j.append(i)
                    if self.price_record[i][1] == -1 and self.data.close[0]/self.price_record[i][0] < 0.98: #如果持空仓且跌幅超过 2%
                        count += 1 #买空平仓
                        print(f"买空平仓，位置{i}")
                        j.append(i)
            if len(j) != 0:
                self.price_record=[self.price_record[idx] for idx in range(len(self.price_record)) if idx not in j]
        if signal == 1:
            if self.data_close[0] < self.data_close[-1]:  # 如果市场行情低于当日收盘价
                count += 1 #买多开仓
                self.price_record.append([self.data_close[0],signal,0])
            else:
                print(f"无交易时机，时间：{self.datas[0].datetime.datetime(0)}")
        if signal == -1:
            if self.data_close[0] > self.data_close[-1]:  # 如果市场行情高于当日收盘价
                count -= 1 #卖空开仓
                self.price_record.append([self.data_close[0],signal,0])
            else:
                print(f"无交易时机，时间：{self.datas[0].datetime.datetime(0)}")
        if count > 0:
            size=2000*count
            self.order = self.buy(size=size)
            self.entry_price = self.data.close[0]  # 记录开仓价格
            print(f"当日执行买入，时间：{self.datas[0].datetime.datetime(0)}")
        elif count < 0:
            size=2000*count
            self.order = self.sell(size=size)
            self.entry_price = self.data.close[0]  # 记录开仓价格
            print(f"当日执行卖出，时间：{self.datas[0].datetime.datetime(0)}")
        else:
            print(f"当日未交易，时间：{self.datas[0].datetime.datetime(0)}")
        if len(self.price_record) > 0:
            print(f"当前持仓状态：{self.price_record}")
        print('当前可用资金{}，当前持仓量{}，当前持仓成本{}，开仓价格{}，当前价格{}'.format(self.broker.getcash(),self.broker.getposition(self.data).size,
self.broker.getposition(self.data).price, self.entry_price,self.data.close[0]))
        total_money.append(self.broker.getcash())
    def notify_order(self, order):
        if order.status in [order.Completed, order.Canceled, order.Margin]:
            self.order = None
```

```
def data_deal(data):
    sign=[]
    X1=[]
    X2=[]
    X3=[]
    X4=[]
    X5=[]
    X6=[]
    X7=[]
    X8=[]
    X9=[]
    for i in range(len(data)):
        if i == 0:
            X1.append(1)
            X2.append(1)
            X3.append(1)
            X4.append(1)
            X5.append(1)
            X6.append(1)
            X7.append(1)
            X8.append(1)
            X9.append(1)
        else:
            X1.append(data['open'][i-1]/data['close'][i]-1)
            X2.append(data['close'][i-1]/data['close'][i]-1)
            X3.append(data['high'][i-1]/data['close'][i]-1)
            X4.append(data['low'][i-1]/data['close'][i]-1)
            X5.append(data['volume'][i]/data['volume'][i-1]-1)
            X6.append(data['atr'][i]/data['atr'][i-1]-1)
            X7.append(data['open'][i]/data['close'][i]-1)
            X8.append(data['high'][i]/data['close'][i]-1)
            X9.append(data['low'][i]/data['close'][i]-1)
        #收盘均线比较
        if data['ma4'][i] > data['ma40'][i]*1.005:
            sign.append(1)
        elif data['ma4'][i] < data['ma40'][i]*0.995:
            sign.append(-1)
        else:
            sign.append(0)
    d=pd.DataFrame(data=zip(X1,X2,X3,X4,X5,X6,X7,X8,X9,data['cci'],data['macd'],data['ma6'],sign),
                   columns=['X1','X2','X3','X4','X5','X6','X7','X8','X9','cci','macd','ma','sign'])
    return d
```

```
def svm_predict_Stacking(data):
    X_train = data.iloc[:-100, :-1]
    y_train = data['sign'][:-100]
    X_test = data.iloc[-100:, :-1]
    y_test = data['sign'][-100:]
    # 定义基础模型
    svm = SVC()
    mlp = MLPClassifier()
    rf = RandomForestClassifier()
    xgb = XGBRegressor()
    # 定义 mlp 目标函数
    def objective(params):
        clf = MLPClassifier(**params)
        clf.fit(X_train, y_train)
        y_pred = clf.predict(X_test)
        accuracy = accuracy_score(y_test, y_pred)
        return {'loss': -accuracy, 'status': STATUS_OK}
    # 定义参数网格
    param_grid_svm = {'C': np.linspace(0.1,10,num=10),
                      'gamma': np.linspace(0.1,10,num=10),
                      'kernel': ['rbf']}
    param_grid_mlp = {'hidden_layer_sizes': hp.choice('hidden_layer_sizes', [(50,), (100,), (150,)]),
                      'alpha': hp.loguniform('alpha', -6, 0),
                      'learning_rate_init': hp.loguniform('learning_rate_init', -4, 0),
                      'max_iter': hp.choice('max_iter', [100, 200, 300])}
    param_grid_rf = {'n_estimators': [50, 100, 150],
                     'max_depth': [None, 10, 20],
                     'min_samples_split': [2, 5, 10]}
    param_grid_xgb = {'n_estimators': [100, 200, 300],
                      'max_depth': [3, 5, 7],
                      'learning_rate': [0.05, 0.1, 0.2],
                      'subsample': [0.6, 0.8, 1.0],
                      'colsample_bytree': [0.6, 0.8, 1.0],
                      'gamma': [0, 0.1, 0.2]}
    # 搜索优化
    # 网格搜索
    svm_grid_search = GridSearchCV(estimator=svm, param_grid=param_grid_svm, n_jobs=-1)
    svm_grid_search.fit(X_train, y_train)
    # 贝叶斯搜索
    trials = Trials()
    best = fmin(fn=objective,
                space=param_grid_mlp,
                algo=tpe.suggest,
```

```
                    max_evals=50,
                    trials=trials)
    best_params = {'hidden_layer_sizes': [(50,), (100,), (150,)][best['hidden_layer_sizes']],
                   'alpha': best['alpha'],
                   'learning_rate_init': best['learning_rate_init'],
                   'max_iter': [100, 200, 300][best['max_iter']]}
    mlp_bayes_search = MLPClassifier(**best_params)
    mlp_bayes_search.fit(X_train, y_train)
    # 随机搜索
    rf_random_search = RandomizedSearchCV(rf, param_grid_rf, n_iter=10, n_jobs=-1)
    rf_random_search.fit(X_train, y_train)
    # 获取最优模型
    best_svm = svm_grid_search.best_estimator_
    best_mlp = mlp_bayes_search
    best_rf = rf_random_search.best_estimator_
    logistic_regression = LogisticRegression()
    base_models = [
        ('svm', best_svm),
        ('mlp', best_mlp),
        ('rf', best_rf),
    ]
    meta_model = LogisticRegression()
    stacking_model = StackingClassifier(estimators=base_models, final_estimator=meta_model)
    # 训练 Stacking 模型
    stacking_model.fit(X_train, y_train)
    # 预测测试集
    y_pred = stacking_model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    accuracies.append(accuracy)
    print(f"Stacking 模型的准确率为: {accuracy}")
    prediction = stacking_model.predict([data.iloc[-1, :-1]])
    print("预测结果是：{}，实际涨跌情况：{}".format(int(prediction),int(data.iloc[-2,-1])))
    return int(prediction)
# 策略设置
cerebro=bt.Cerebro() # 创建引擎
# 将数据加入回测系统
cerebro.adddata(data)
# 加入自己的策略
cerebro.addstrategy(MyStrategy)
startcash=10000000
cerebro.broker.setcash(startcash)  # 经纪人
cerebro.broker.setcommission(0.0002)  # 设置手续费
s=fromdate.strftime("%Y-%m-%d")
t=todate.strftime("%Y-%m-%d")
```

```
# 添加分析器
cerebro.addanalyzer(SharpeRatio, _name='sharpe')
cerebro.addanalyzer(DrawDown, _name='drawdown')
cerebro.addanalyzer(TradeAnalyzer, _name='trade')
cerebro.addanalyzer(bt.analyzers.Returns, _name='returns')
print(f"初始资金：{startcash}\n 回测时间：{s}-{t}")
result = cerebro.run()  # 执行回测
portval=cerebro.broker.getvalue()
print(f"剩余资金：{portval}\n 回测时间：{s}-{t}")
if len(accuracies):
    overall_accuracy = sum(accuracies) / len(accuracies)
    print(f"\n 平均正确率：{overall_accuracy}")
else:
    print("\n 无正确率，回测失败")
# 获取分析器结果
sharpe_ratio = result[0].analyzers.sharpe.get_analysis()
max_drawdown = result[0].analyzers.drawdown.get_analysis()
trade_analyzer = result[0].analyzers.trade.get_analysis()
all_yield = result[0].analyzers.returns.get_analysis()
total_trades = trade_analyzer.total.total
profitable_trades = trade_analyzer.won.total
win_rate = profitable_trades / total_trades if total_trades > 0 else 0
# 打印结果
print("夏普比率:", sharpe_ratio)
print("最大回撤:", max_drawdown)
print("收益率: ",all_yield)
print("胜率:", win_rate)
%matplotlib inline
fig = cerebro.plot(style='candlestick')
show = fig[0][0]
show.set_size_inches(10,10)#调整大小
show
plt.plot(accuracies)
plt.title('正确率')
plt.show()
plt.plot(total_money)
plt.title('可用资金')
plt.show()
```

致 谢

感谢指导我的各位老师，感谢帮助我的朋友和同学，感谢教导我的母校湖北大学！
书山有路勤为径，学海无涯苦作舟。