数据科学实践

陈跃国 覃雄派 主编
2024.06.30

序言

这里是序言内容

编者 2024.06.30

章节目录

图目录

表目录

4 金融业

4.1 智慧金融与量化交易问题

4.1.1 智慧金融的内涵

智慧金融利用人工智能技术，分析金融数据，为金融业务赋能，在这里我们也把智慧金融称为数字金融。

数字金融，是数字经济领域的一个最活跃的研究分支，它利用大数据技术和现代信息技术，实现金融服务的数字化、自动化、智能化和个性化，实现业务优化和风险管理，从而提高金融机构的运营效率。

数字金融的核心是数据驱动的。通过收集大量的数据，然后进行分析，可以更好地了解客户需求，预测市场的趋势，制定更加科学的决策。数字金融依赖于大数据、人工智能、区块链技术等先进的技术手段，实现自动化运营和智能化决策。

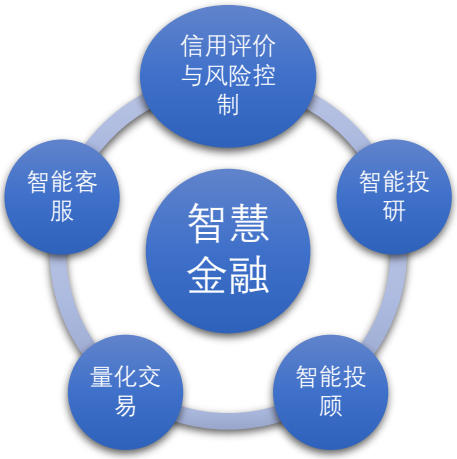

图 4-1 数字金融(智慧金融)的内涵

数字金融的内涵丰富，具体如下：

1.智能客服：利用人工智能和自然语言处理等技术手段，为金融机构提供自动化客户服务和交互式聊天功能。可以根据客户的需求和偏好，为客户提供个性化的产品和服务推荐。

2.信用评价与风险管理：依靠大数据和人工智能技术，构建客户信用评级体系，为评估模型提供更为丰富和准确的数据输入，建立合理的评价模型，从而降低误判和漏判的风险，全面掌握客户的信用情况。此外，通过分析大量的交易数据和客户行为，可以检测异常模式和风险信号，提高金融欺诈的检测和预防能力。在此基础上，建立完善的风险管理体系，制定风险管理制度，加强金融机构风险控制能力。

3.智能投研：利用计算机技术和人工智能算法，对金融市场数据进行自动化分析，实现更高效、更准确、更前瞻的投资研究。其核心是通过机器学习、自然语言处理、知识图谱等技术，对互联网数据、财经资讯、观点逻辑等信息进行自动化分析，从而有效提高分析师、基金经理等金融从业人员的工作效率和投研能力，为管理层提供高质量的研究报告。

4.智能投顾：将人工智能技术导入传统的理财顾问服务，依据客户的需求设定投资目标及风险承受度，通过计算机程序的算法，提供自动化的投资组合建议。通过分析金融市场数据，可以研究股票、债券、期货、外汇等金融资产的价格走势和波动性，进行投资组合管理和风险控制。基于人工智能的智能投顾，可以为投资者提供符合其风险偏好和投资目标的个性化投资建议。

5.量化交易：以数学模型为基础的决策模型，替代人为的主观判断，利用计算机技术从庞大的历史数据中筛选出能带来超额收益的大概率事件，制定交易策略，自动进行金融资产的交易，以期实现盈利。量化交易的主要优势是高度的一致性和纪律性，它能够极大地减少投资者情绪波动的影响，避免在市场狂热或悲观的情况下，做出非理性的投资决策。

4.1.2 智慧金融与量化交易历史简况

信息技术的发展，给各行各业提供了提高工作效率、增强决策智能的潜力。每一项新的信息技术推出来以后，金融行业是率先加以运用的行业。金融行业通过信息技术实现了数字化，目前在 AI 技术的加持下，正在向智能化迈进。

图 4-2 幻方量化私募基金(https://www.high-flyer.cn/)

在量化交易方面，世界范围内的各大银行、对冲基金(hedge fund)、投资机构广泛采用算法交易技术和模型。根据调查，2000 年，仅仅在美国，由华尔街的算法交易系统发出的订单(order)，占到全美股票交易量的 40%。到 2008 年，这个数字攀升到 60%；而到了 2014 年，这个数字超过了 70%。这些交易系统属于各大投资银行、对冲基金、机构投资者等。他们各展其能，研发了专有的量化交易系统，而对于技术细节则秘而不宣。

21 世纪初以来，国内的机构投资者，包括公墓基金、私募基金等，也开始采用量化交易系统、技术和策略。国内的头部量化私募基金，其管理的资金规模达到几百亿甚至上千亿。

图 4-3 微观博易私募基金(https://www.microtrading.com/page85)

4.1.3 量化交易问题的凝练

从上文对智慧金融的内涵的论述可以看出，我们需要基于数据分析的结果进行决策，即智慧金融的本质是数据驱动的决策。

其中，量化交易在现有数据基础上进行分析，尝试对接下来的价格运动做出提前预判，以便适时买卖某些标的物(包括股票、期货，本章主要介绍股票交易，而且只有做多，不做空)，以期在后续进行交易了结时获得一定的盈利。

量化交易具体要解决 2 个问题。一个是选股，也就是选择什么股票进行接下来的交易，并且给这些股票分配资金。另外一个问题是择时，即确定什么时候买入，什么时候卖出，仓位如何控制等。本文的量化交易实践，主要研究择时。

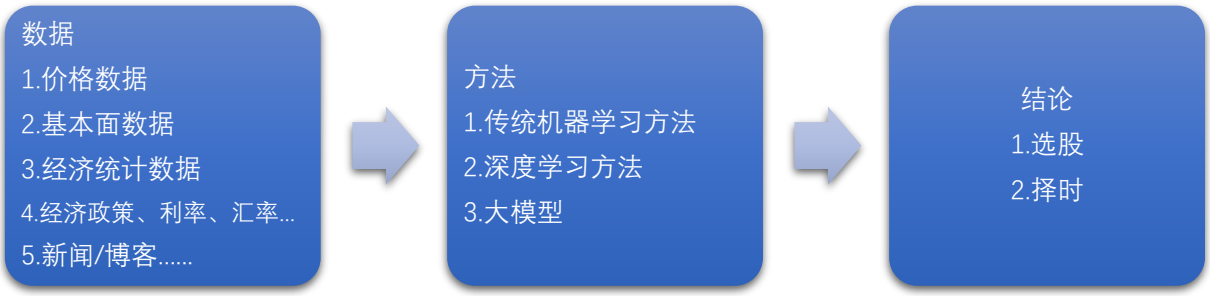

图 4-4 量化交易问题的凝练

一般地，我们能够获得的数据包括价格和交易量数据、基本面数据、经济统计数据、以及相关的新闻/博客等互联网数据。这些数据表达了价格以及其他经济变量的运动，在这些数据的分析基础上，我们需要建立这些数据到股票价格是涨还是跌、具体涨跌幅度的联系。这样，问题可以建模为一个分类问题，即建立输入数据到涨/跌的联系；或者把问题建模为一个回归问题，即建立输入数据到涨跌幅度的联系。根据股票价格的涨跌、或者涨跌幅度，确定什么时间点进行买入和卖出。

更进一步地，我们也可以直接建立输入数据到交易动作(买入、卖出、不做动作)的联系，而不是先预测涨/跌或者涨跌幅，再决定买入还是卖出，这也是一个分类问题。

单纯地在原始数据上进行预测，效果一般不会太好。一般需要在原始数据上计算一些导出的特征，也称为因子。好的因子，对目标变量具有较强的解释作用，建立的模型预测效果好，泛化能力强。结合股票交易，寻找合适的因子是一项重要的工作，这些因子包括量价因子、基本面因子、宏观和微观经济因子等。

4.1.4 量化交易系统

量化交易系统，一般包含 4 大模块，分别是数据收集、数据分析与决策、订单执行、模型监控与优化等。

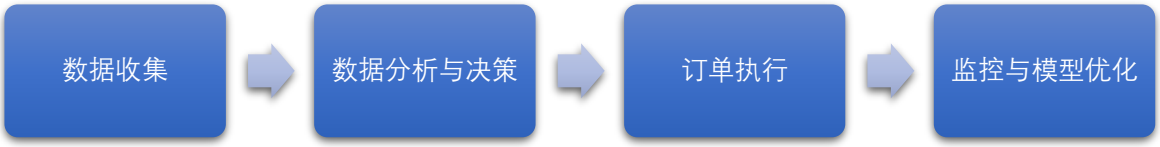

图 4-5 量化交易系统

1.数据收集

收集必要的数据，准备进行分析，以帮助决策，即生成交易信号。可以收集的数据包括历史价格和交易量数据、新闻、博客/微博、公司的季报/年报等财务数据、国内宏观经济数据、国际宏观经济数据、上下游厂商数据等。在这些数据基础上，提炼因子，研发量化交易策略，编写模型。生成投资组合，配合适当的仓位管理。

其中，历史价格和交易量数据可以从财经网站(比如 Google Finance)获得，而实时的价格和交易量数据可以从交易所获得。新闻、博客/微博等数据可以从互联网上获取。公司的季报/年报、国内宏观经济数据、国际宏观经济数据可以从权威的官方网站获取。

2.数据分析与决策

利用交易模型，分析相关的数据，决定两件事情，一个是选股，一个是择时。选股是选择相关股票进行交易，择时是对什么时候进行买卖、买卖多少进行决策，即生成交易信号(trading signal)。主要的分析方法包括统计分析、数据挖掘与机器学习方法等。近年来，大数据技术和深度学习技术被人们应用到量化交易领域，并且取得良好的效果。

在互联网时代，人们可以方便地获取大量的数据。从海量数据中分辨和识别模式，需要机器学习技术和工具的帮助，这是研发更好的量化交易策略的可行路径。

人们发现，基于单一价格数据建立的交易策略，由于信息的不完整，带有很大的局限性。于是开始引入新闻、博客、名人讲话、政策发布、社交网络等，从中获得价格变化前景的线索。甚至有些公司运用卫星拍摄的港口集装箱图像以及农田农作物图像进行分析，预测贸易和农产品价格的未来变化。这些数据包括结构化数据(比如价格信息)，也包括非结构化数据(比如新闻(文本数据)、以及社交网络(图数据)等。

3.订单执行

负责把交易模型生成的交易信号发送到交易所，真正地执行交易订单(order execution)，返回订单执行结果。订单的执行需要考虑市场冲击(market impact)、滑点(slippage)等因素的影响。

一般可以通过把大的订单分解成一系列小订单来减少市场冲击。订单拆分策略以 TWAP(time weighted average price)、VWAP(volume weighted average price)和 POV(percent of volume)为代表。TWAP 将大订单在规定的时间内按照一定的交易频率分割成小订单。VWAP 按照交易量的历史分布分割订单。POV 则将小订单以固定比例混入订单流以降低对市场的影响。订单拆分仅仅作用在订单执行部分，它不是量化交易的核心。量化交易的核心问题是买/卖什么、买还是卖、什么时候以及多少等。

考虑到滑点问题，即订单执行的时候，价格已经在不断变化，目的是希望按照预期的价格拥有金融资产头寸(position)。

4.监控与模型优化

该模块负责监控系统运行。市场瞬息万变，不断演进，假设某种相关性会持续很长时间是不现实的。量化交易系统需要对此做出反应，随着旧关系的衰减和新关系的出现，适时进化交易策略。

这种持续的优化和改进，可以是离线的，也可以是在线的。所谓离线(offline)优化，意思是交易时间使用交易策略，闭市时间优化策略。而在线(online)优化，意思是模型可以在线更新，具有及时自我进化的能力。

4.1.5 量化交易系统的评价

1.年化收益率

对于量化交易系统(或者模型)的评价，最重要的指标的是其年化利润率(annualized profit ratio) ，也称为年化收益率。假设我们投资的初始资金为$10000 元，一年以后资产净值(包括现金和股票)为$11000，那么利润率为 10%。如果投资周期不是一年，需要转化成年化利润率，计算公式为：年化收益率=(投资收益/本金)/(投资天数/365)×100%。

需要注意的是，如果一个交易模型不能在无风险的定期存款以外创造额外的收益，是没有用的。倘若如此，还不如直接把款项存为定期存款。

人们往往追逐更高的利润率而忽视了其他方面。我们认为利润率不是越高越好，判断交易模型的优劣还需要看其他指标。人们希望在保证稳健盈利的基础上，将风险和回撤降低到一个安全可控的范围内。除了年化收益率之外，还有两个重要的性能指标，即最大回撤(max drawdown, MDD)和夏普指数(Sharpe ratio)。

2.最大回撤

最大回撤指标，评价一个交易模型及其投资组合的风险度，最大回撤越小越好。最大回撤的计算公式为 MDD =(A-B)/A，其中 A 为最大下跌之前的资产最大净值(peak asset value before largest drop)，B 为资产净值创出新高之前的最低的资产净值(lowest asset value before new high established)。

比如，一个投资组合的开始净值为$10000，这个净值经过一系列股票交易以后发生变化，第一天为$17000，第二天为$8000，第三天为$13000，第四天为$7000，第五天为$19000。那么，最大回撤为(17000-7000)/ 17000=58.8%。最高净值$19000 没有用来计算最大回撤，因为这里的回撤是从最高点$17000 开始的。

3.夏普指数

夏普指数是由威廉.夏普(William F. Sharpe)发明的一个指标。该指标用标准差(standard deviation)以及超额收益(excess return)来计算每单位风险获得的收益，这里的风险指的是波动性。

假设投资组合的年化收益率为 12%，波动性为 10%，无风险的定期存款利率为 5%，那么夏普指数为(0.12-0.05)/0.1=0.7=70%。投资模型的夏普指数越高，显示投资模型越稳健，收益越好。注意，在这里也强调了量化交易追求定期存款收益之外的超额收益。

4.2 量化交易的关键技术

4.2.1 传统机器学习技术

在智慧金融与量化交易应用场景中，问题可以建模为有监督的机器学习、无监督的机器学习问题，人们可以充分利用现有的统计分析、数据挖掘、机器学习技术解决问题。比如，我们从庞大的历史数据中，利用数据分析方法，判断各种经济变量和股票价格变动的联系，提前预判有利的交易机会，进而自动化地进行股票等金融资产的交易。

4.2.2 深度学习技术与大模型

近年来，以大模型为代表的深度学习技术发展迅猛，涌现出惊人的能力。深度学习技术以及大模型也逐渐被人们运用到到智慧金融以及量化交易中。

2022 年 OpenAI 发布 ChatGPT 以来，大模型的潜力为人们所认识。人们开始考虑利用金融领域的语料，训练领域大模型，为智慧金融赋能。

假设我们把有史以来所有上市公司的股票价格变化、基本面数据、微观经济数据、宏观经济数据、统计数据、经济政策/汇率/利率、财经新闻等，集中起来形成一个大规模的预料，训练一个大模

型，那么这个大模型能够涌现什么智能，多大程度上帮助投资经理、研究员、交易员完成其工作，值得期待。

1.大模型(Large Language Model, LLM)简介

大模型，是大语言模型的简称，是指具有大规模参数和复杂计算结构的机器学习模型。这些模型通常由深度神经网络构建而成，拥有数十亿甚至数千亿个参数。大模型的设计目的，是为了提高模型的表达能力和预测性能，能够处理更加复杂的任务。大模型通过海量训练数据来学习复杂的模式和特征，具有强大的泛化能力，可以对未见过的数据做出准确的预测。

随着训练数据和参数规模不断扩大，达到一定的临界规模后，神经网络模型表现出了一些未能预测的、复杂的能力和特性。大模型能够从原始训练数据中自动学习并发现新的、更高层次的特征和模式，称为“涌现能力”。目前，大模型不仅具有文本生成和对话的能力，还具有逻辑推理、数学推导、编写程序等能力，多模态大模型甚至能够理解视频和图片，经过特殊设计的大模型比如 Sora，可以根据用户的提示，直接生成一定长度的视频。

自从 2022 年底 OpenAI 发布 ChatGPT 以来，大模型从一个玩具，变成了实用的技术。大模型在各种领域都有广泛的应用，包括自然语言处理、计算机视觉、语音识别和推荐系统等。直到目前(2024 年)，主流的大模型，大部分是基于 2017 年发表的 Transformer 神经网络架构进行构建的。

随着技术的进步，可以预见大模型将继续发展，变得更加高效和智能。同时，研究人员也在探索如何减少大模型的资源消耗，提高其可解释性和公平性。

2.大模型在金融领域的应用潜能

在大模型众多的能力中，强大的上下文(Context)处理和理解能力是其中一种重要的能力。我们可以把整部《红楼梦》输入大模型，然后就可以询问任何关于红楼梦的问题，大模型都可以给出准确合理的回答。这种强大的上下文处理和理解能力，有时候可以给我们带来一些匪夷所思的结果。比如，成吉思汗发动了蒙古西征，导致了黑死病的蔓延，最后一次黑死病爆发造成了伦敦大瘟疫，牛顿到乡下躲避瘟疫，在苹果树下被苹果砸中，引起了他深深的思考，最后发现了万有引力定律。这个案例经常被用作图数据库通过路径分析得到有趣结论的一个例证。我们可以想象，比图数据库强大得多的大模型，当它经过大规模预料的训练之后，结合思维链技术，应该能够给我们提供很多有意思的、以前没有注意到的事件的关联关系。在股票交易中，众多的经济变量驱动了股票价格的运动，当我们把相关的资料，包括历史价格、基本面数据、统计数据、宏观经济/微观经济数据、财经新闻、经济政策等输入到大模型，那么大模型可以从比较宽的山下文环境中，去识别出重要的因素，帮助我们预判价格的运动。

在 OpenAI 推出 ChatGPT 以后，大模型展示了强大的威力，人们开始在金融业务中运用大模型。比如，在智能客服方面，大模型给出的对话结果，比传统的对话系统的结果更加自然，更加具有针对性，客户的满意度更高。当大模型经过领域语料的训练和微调，我们可以把大模型运用到智能投研中来，我们可以向大模型询问诸如“请用 500 个字介绍某公司最近半年来的财务表现。”，大模型即可给出相关的结果，这些结果是在大量的事实和数据的分析基础上得出的。在这个场景中，大模型甚至能够从图片(比如当前财年各个季度的营收和利润)中识别文字，从视频(比如公司领导发表的演讲)中理解事件。在智能投顾领域，我们可以向大模型询问“请问对于风险厌恶型投资者来讲，最近应该投资什么金融资产？”，大模型可以给出一个可行的建议，财务顾问可以在这个建议的基础上，进一步和客户充分沟通，最后给出投资建议。在量化交易领域，已经有很多的投资者尝试运用大模型帮助选股。在交易择时方面，大模型能够注意到众多经济变量和价格运动的关系，帮助我们选择合适的买入和卖出的时机。由于有思维链技术的加持，大模型不仅能够给出结论，还能够把其决策过程涉及的关键步骤和主要依据给出来，形成一个推理链条，帮助我们判定这样的结论是否可靠。

3.金融领域大模型

- BloombergGPT

BloombergGPT 是由彭博社(Bloomberg)开发的一个大语言模型，专门为金融领域设计，这个模型拥有 500 亿个参数。彭博社构建了一个包含 3630 亿个 token 的金融数据集，结合通用数据集，训练这个模型。由于其训练数据来自彭博社四十多年的金融数据积累，模型的输出结果在金融领域的准确性和实用性方面，达到较高的水平。

BloombergGPT 提高了各类金融数据的分析处理能力，包括情感分析、命名实体识别、新闻分类和问题回答等。模型的评估结果显示，BloombergGPT 在通用任务上能与现有模型相媲美，在金融任务上的表现显著优于现有模型。使得金融从业人员可以应用自然语言处理技术，提高工作效率。

由于 BloombergGPT 基于数十年的彭博私有数据进行训练，其信息具有敏感性，BloombergGPT 将不会公开发布，仅限于彭博社内部和其合作伙伴使用。

- 无涯金融大模型

星环科技公司凭借其在大数据平台、大模型方面的积累，推出了“无涯金融大模型”。通过预训练、提示微调、增强学习、建立思维链推导范式等手段，星环科技完成了金融行业智能投研大模型无涯 Infinity 的训练。这个大模型，能够实现对事件语义刻画、定价因子挖掘、时序编码、异构关系图卷积传播，进而构建包含事件冲击、时序变化、截面联动和决策博弈等多个维度的智能投研辅助功能。该大模型在研报生成、资讯解读分析、金融知识图谱构建、事件传播分析、投研问答、量化因子挖掘与策略构建等方面发挥巨大作用。

在量化因子挖掘方面，Infinity 大模型覆盖量价因子、政策、舆情、产业链、风险因素等因子，帮助量化交易策略开发人员挖掘和构建有解释力的因子。

图 4-6 无涯金融大模型(https://www.transwarp.cn/product/infinity)

- 财跃 F1 金融大模型

2023 年，姜大昕博士创立了“阶跃星辰”。2024 年 3 月，阶跃星辰发布了 Step 系列通用大模型，包括 Step-1 千亿参数语言大模型，Step-1V 千亿参数多模态大模型，以及 Step-2 万亿参数 MoE 语言大模型——国内初创公司里面的首个万亿参数大模型。

在研发基础大模型的同时，阶跃星辰同时尝试在 C 端和 B 端的落地应用。2024 年 3 月，阶跃星辰与上海报业集团旗下界面财联社共同打造“财跃 F1 金融大模型”，以“财跃星辰”的身份推出。

财跃 F1 金融大模型具备三大能力。首先是金融知识问答能力，它能够理解和回答关于金融领域的各种问题，包括概念、术语、原理、市场动态等。其次是金融图表理解能力，它能够解析和解释各类金融图表，包括识别图表类型、理解数据含义、分析趋势和模式等。第三是金融计算能力，它能够进行金融计算和建模，包括利率计算、投资组合优化、风险评估等。

该大模型可以在智能运营、智能风控、智能投顾、智能营销、智能客服等场景加以运用，引起了国泰君安、广发证券、汇添富基金等 30 多家金融机构的关注。

财跃星辰CEO贾宝龙：财跃F1做最懂金融的大模型

2024年03月23日 20:26 来源： 财联社

国内首个千亿参数多模态金融大模型——“财跃F1金融大模型”今日在上海面市。这款由上海财跃星辰智能科技有限公司研发的金融大模型，将围绕金融信息服务、智能投顾、智能投研等场景，助力金融机构打造新质生产力。

财跃F1金融大模型基于万亿级金融语料预训练，具备强大的通用图像处理和图表理解能力，相比GPT-4等通用大模型在金融知识理解方面更为突出。 财跃星辰由上海报业集团旗下界面财联社与国内头部通用大模型公司阶跃星辰联合创办，前微软全球副总裁姜大昕任首席科学家。

图 4-7 财跃 F1 金融大模型发布(https://finance.eastmoney.com/a/202403233021958148.html)

- 度小满金融大模型

度小满即原百度金融。2023 年 5 月，度小满正式开源了千亿级中文金融大模型，名为“轩辕”。轩辕大模型是基于庞大的 1760 亿参数的 Bloom 大模型进行训练的，它在金融领域的任务表现相对于通用大模型有了显著的提升。度小满积累了垂直领域千亿 tokens 的中文预训练数据集，该数据集涵盖了金融研报、股票、基金、银行、保险等各个方面的专业知识。经过清洗和标注的高质量数据集，在通用性方面有望与 ChatGPT 比肩，同时显著提升了大模型在金融垂直领域应用的性能。

基于大模型等底层技术，度小满开始在信贷、财富管理、支付、保险、个人金融科技和供应链金融科技等六大领域尝试落地，为不同需求的人群提供值得信赖的金融服务。

4.3 量化交易实践

本章的量化交易实践包括 2 个阶段，第一个阶段训练一个预测模型，第二个阶段利用这个训练好的模型进行量化交易，具体描述如下。

首先利用已经下载的 10 只股票的 Daily Price 价格数据，进行数据标注，标注出合适的买入(Buy)、卖出(Sell)、不做动作(No Action)的标签(Label)。需要注意的是，这时候可以针对整个数据集进行处理和分析，以标定合适的标签(Label)。

然后，根据需要从原始数据计算一些导出的特征(Factor)，这些特征有可能对上述标签具有更强的解释作用。在原始数据以及导出的特征上，加上标签，建立样本。需要注意的是，这时候样本的构建，只能利用到目前为止的数据即历史数据，不能利用未来的数据。

利用样本进行模型训练。模型训练完成后，把模型存盘。该模型具有根据历史上到目前为止的价格数据(以及自行构造的特征)，做出买入、卖出、不做动作的预测的能力。

到这里，第一个阶段结束。

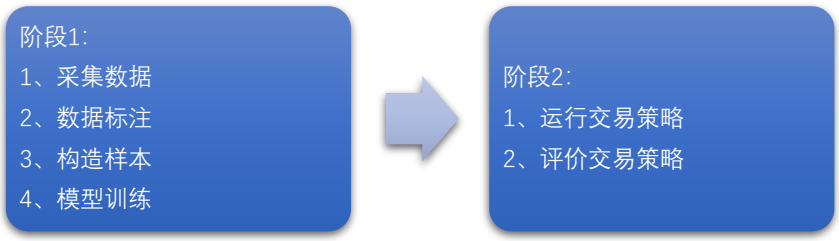

图 4-8 量化交易实践的两个阶段

在第二个阶段，装载训练好的模型，并且流式地接收和处理依次到达的新的 Daily Price Bar，利用到目前为止的原始数据计算必要的导出特征，基于到目前为止的原始数据和导出特征(不能利用未来数据)，构造新样本，馈入模型，得到买入、卖出、不做动作的输出。

根据模型的输出，执行买入、卖出、不做交易等动作。针对 10 只股票进行测试，得到在这 10 只股票上的年化利润率、最大回撤、夏普指数、交易次数等指标。最后对 10 只股票的测试结果进行平均，得到平均的年化利润率、最大回撤、夏普指数、交易次数。

4.3.1 数据下载

在这里，通过 Tushare 下载 A 股的 Daily Price 数据，每一行价格数据称为一个 Price Bar，包含 Datetime、Open、High、Low、Close、Volume、Adj Close 等字段，表示日期、开盘价、最高价、最低价、收盘价、交易量、临近收盘价等。

- 下载股票代码

首先，下载 A 股的股票代码。导入 tushare 库，然后指定 token，访问 tushare pro 的 API，查询到所有的股票代码。

```
import tushare as ts

token='a953fe53f83c71c6eb63558aa20d83d7cb5a9fdbe40cc6d49027a3ce'
ts.set_token(token)
pro=ts.pro_api()

stock_list = pro.query('stock_basic', exchange='', list_status='L',
    fields='ts_code,symbol,name,area,industry,list_date')
stock_list.head(20)
```

注意，上述代码中的 token 是本章作者自己注册获得的，请读者自行到 https://tushare.pro/网站进行注册，然后登录以后，按照“头像->个人主页->接口 TOKEN”的操作步骤，提取自己的 token，替换上述代码中的 token。

该代码的输出如图 4-9 所示。

	ts_code	symbol	name	area	industry	list_date
0	000001.SZ	000001	平安银行	深圳	银行	19910403
1	000002.SZ	000002	万科A	深圳	全国地产	19910129
2	000004.SZ	000004	国华网安	深圳	软件服务	19910114
3	000005.SZ	000005	ST星源	深圳	环境保护	19901210
4	000006.SZ	000006	深振业A	深圳	区域地产	19920427
5	000007.SZ	000007	*ST全新	深圳	其他商业	19920413

图 4-9 所有的股票代码

各个列分别表示股票代码(ts_code)、编号(symbol)、股票名称(name)、交易所在地(area)、所属行业(industry)、上市日期(list_date)等。

接着把所有股票代码保存到 stock_list.csv 文件里。stock_list.csv 文件的内容，即上述所有股票代码及相关信息。

```
stock_list.to_csv("stock_list.csv",    header=True, index=None)
```

重新读取 stock_list.csv 文件，形成内存的 Data Fame，并且显示。

```
import pandas as pd
from matplotlib import pyplot as plt

df = pd.read_csv('stock_list.csv',    sep=',', dtype = 'str')
df.head()
```

通过查看 Data Frame 的行数，得到股票的数量为 5360。

```
row_count_old = df.shape[0]
print( "row_count_old", row_count_old)
```

遴选 20100101 之前上市的股票，得到新的 Data Frame。

```
df_new = df[ df['list_date'] <= '20100101']
df_new.head()
```

显示 20100101 之前上市的股票的数量为 1576。

```
row_count_new = df_new.shape[0]
print( "row_count_new", row_count_new)
```

- 分别选择 10 只股票作为训练集和测试集

遴选 10 只股票代码，放入列表 stock_list_00 中，准备下载其价格数据作为训练集。每次生成一个随机数作为行号，提取股票代码 Data Fame（即 df_new）的某一行。

```
import random

stock_list_00 = []
for i in range(10):
    one_num = random.randint(0,row_count_new)
    stock_list_00.append( df_new.iloc[one_num,0])

print(stock_list_00)
```

遴选 10 只股票代码，放入列表 stock_list_01 中，准备下载其价格数据作为训练集。每次生成一个随机数作为行号，提取股票代码 Data Fame（即 df_new）的某一行。

```
import random

stock_list_01 = []
for i in range(10):
    one_num = random.randint(0,row_count_new)
    stock_list_01.append( df_new.iloc[one_num,0])

print(stock_list_01)
```

定义一个函数 down_load_one_stock，用于下载一只股票的价格数据，注意这里下载的是 Daily Price。时间范围是'20100101'到'20230101'，即 2010 年 1 月 1 日到 2023 年 1 月 1 日。

```
import tushare as ts

def down_load_one_stock( stock_name, stock_file_name):
    token='a953fe53f83c71c6eb63558aa20d83d7cb5a9fdbe40cc6d49027a3ce'
```

```
    ts.set_token(token)
    pro=ts.pro_api()

    df = pro.query('daily', ts_code=stock_name, start_date='20100101', end_date='20230101')

    df.to_csv(stock_file_name)
```

再定义一个函数 read_write_CSV，目的是把下载的数据转换为 PyAlgoTrade 所需要的格式。

```
from pyalgotrade.feed import csvfeed
import pandas as pd

#read_write_CSV
def read_write_CSV( from_file, to_file):
    feed = csvfeed.Feed("trade_date", "%Y%m%d")
    feed.addValuesFromCSV(from_file)

    df = pd.DataFrame(columns=['Date Time','Open','High','Low','Close','Volume','Adj Close'])
    file = open(to_file,"w")
    file.write("Date Time,Open,High,Low,Close,Volume,Adj Close")
    file.write("\n")
    for dateTime, value in feed:
        #print dateTime,value['open'],value['high'],value['low'],value['close'],value['vol'],value['close']
        strdatetime = dateTime.strftime("%Y-%m-%d %H:%M:%S")
        file.write("%s,%.2f,%.2f,%.2f,%.2f,%.2f,%.2f"%(strdatetime,
            value['open'],value['high'], value['low'],value['close'],value['vol'],value['close']))
        file.write("\n")

    file.close()
```

下文将给出从 tushare 下载的数据以及转换以后的数据的格式，读者自然了解其区别，并且理解上述代码。

现在下载训练集的股票价格数据，并且对其进行格式转换。在这里调用了上述两个函数，分别是 down_load_one_stock 和 read_write_CSV。

```
for stock_name in stock_list_00:
    file_name = './' + stock_name + '.CSV'
    file_name_new='./' + stock_name + '_new.CSV'
    print("downloading ", stock_name,file_name,file_name_new)

    down_load_one_stock( stock_name, file_name)
    read_write_CSV(file_name, file_name_new)
```

比如我们现在下载 000650.SZ 股票的价格数据，首先把价格数据下载到 000650.SZ.CSV 文件 ，然后进行转换，写入 000650.SZ_new.CSV 文件。000650.SZ.CSV 和 000650.SZ_new.CSV 的具体内容，请参见表 4-1。

表 4-1 价格数据文件

000650.SZ.CSV	,ts_code,trade_date,open,high,low,close,pre_close,change,pct_chg,vol,amount

	0,000650.SZ,20221230,6.3,6.31,6.19,6.19,6.3,-0.11,-1.746,401075.4,249880.397 1,000650.SZ,20221229,6.25,6.36,6.21,6.3,6.24,0.06,0.9615,557533.45,351396.479 2,000650.SZ,20221228,6.31,6.34,6.21,6.24,6.33,-0.09,-1.4218,464250.55,290823.827 3,000650.SZ,20221227,6.63,6.63,6.18,6.33,6.63,-0.3,-4.5249,823987.43,520895.504 4,000650.SZ,20221226,6.73,6.78,6.58,6.63,6.67,-0.04,-0.5997,471248.43,313777.311 5,000650.SZ,20221223,6.71,6.8,6.63,6.67,6.73,-0.06,-0.8915,430645.4,289064.678
000650.SZ_new.CSV	Date Time,Open,High,Low,Close,Volume,Adj Close 2010-01-04 00:00:00,21.00,21.00,20.66,20.70,13550.88,20.70 2010-01-05 00:00:00,20.68,21.10,20.52,21.03,18405.70,21.03 2010-01-06 00:00:00,21.03,21.26,20.76,21.13,17975.33,21.13 2010-01-07 00:00:00,21.13,21.24,20.25,20.40,15147.33,20.40 2010-01-08 00:00:00,20.35,20.48,19.83,20.10,9659.04,20.10

000650.SZ.CSV 文件包含了 PyAlgoTrade 交易策略回测平台所不需要的字段，而 000650.SZ_new.CSV 文件只保留了 PyAlgoTrade 交易策略回测平台所需要的字段，包括 Date Time、Open、High、Low、Close、Volume、Adj Close，分别表示日期时间、开盘价、最高价、最低价、收盘价、交易量、临近收盘价等。并且对数据的格式进行了修改，即 Date Time 字段采用“yyyy-mm-dd hh:mm:ss”的固定格式，其内容如“2010-01-04 00:00:00”所示，表示 2010 年 1 月 4 日 0 时 0 分 0 秒。

同样道理，下载测试集的股票价格数据，并且对其进行格式转换。

```
for stock_name in stock_list_01:
    file_name = './' + stock_name + '.CSV'
    file_name_new='./' + stock_name + '_new.CSV'
    print("downloading ", stock_name,file_name,file_name_new)

    down_load_one_stock( stock_name, file_name)
    read_write_CSV(file_name, file_name_new)
```

- 可视化价格数据

如下代码，定义了一个函数 show_one_stock，对价格变动情况进行可视化(线图)。

```
def show_one_stock(file_name):
    df = pd.read_csv(file_name, parse_dates=['Date Time'], index_col=['Date Time'], sep=',')
    plots = df[['Close']].plot(subplots=False, figsize=(8, 3))
    plt.show()
```

调用 show_one_stock 函数，对训练集和测试集的价格数据进行可视化。

```
for stock_name in stock_list_00:
    file_name_new='./' + stock_name + '_new.CSV'
    show_one_stock(file_name_new)
```

```
for stock_name in stock_list_00:
    file_name_new='./' + stock_name + '_new.CSV'
    show_one_stock(file_name_new)
```

价格数据的可视化结果如图 4-10 所示。

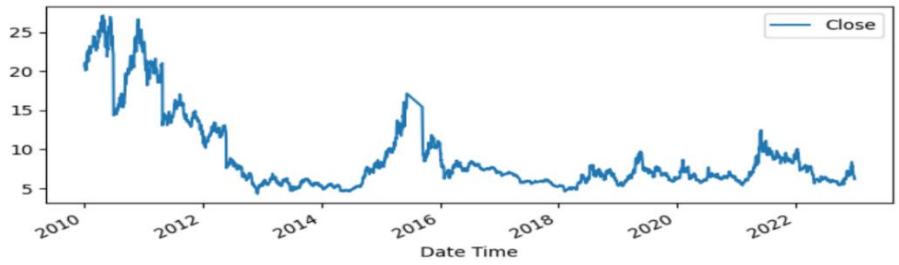

图 4-10 价格数据可视化(部分)

到这里，我们把训练集的股票价格数据文件(注意是经过转换的文件)，命名为“stock01.csv”到“stock10.csv”，把测试集的股票价格数据文件，命名为“stock11.csv”到“stock20.csv”，以便进行后续实验。当读者自行下载数据时，请自行重新命名。

4.3.2 数据标注

接下来，在价格数据上，针对每个价格数据点进行标注，标注在该价格数据点上是进行买入、卖出、还是不做动作。

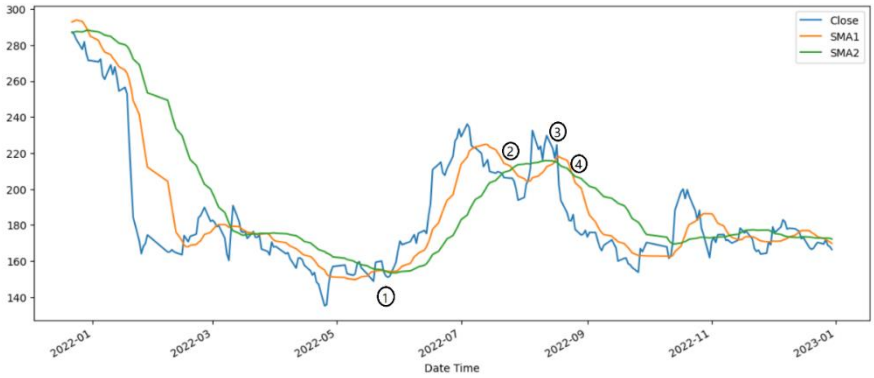

图 4-11 短周期移动平均和长周期移动平均的交叉

在这里，采用一种简单的判断策略。我们针对价格数据计算一个短周期的移动平均，和一个长周期的移动平均。当短周期的移动平均从下边往上穿越长周期的移动平均，我们认为价格将要上涨，适合买入，如图 4-11 上数字 1、数字 3 对应的位置。当短周期移动平均从上边往下穿越长周期的移动平均，我们认为价格将要下跌，适合卖出，如图 4-11 上数字 2、数字 4 对应的位置。其他时候，我们持有现金，或者持有股票，不做动作。

- 装载数据

首先，装载 stock01.csv 价格数据，并且显示开始的 5 行。

```
import pandas as pd
from matplotlib import pyplot as plt

df = pd.read_csv('stock01.csv', parse_dates=['Date Time'], index_col=['Date Time'], sep=',')
df.head()
```

这段代码的运行结果如图 4-12 所示。

Date Time	Open	High	Low	Close	Volume	Adj Close
2010-01-04	26.99	27.18	26.35	26.96	17902.49	26.96
2010-01-05	26.88	28.01	26.75	28.01	31438.24	28.01
2010-01-06	27.98	29.19	27.81	28.66	33094.95	28.66
2010-01-07	28.48	28.90	27.21	27.30	23864.32	27.30
2010-01-08	27.00	27.29	26.65	27.20	15488.45	27.20

图 4-12 stock01.csv 价格数据

- 计算移动平均

计算短周期和长周期的移动平均，并且可视化。短周期的天数为 13，长周期的天数为 33。这两个参数可以由用户自行进行修改和优化。

```
import numpy as np

short_term =13
long_term = 33
df['SMA1'] = df.Close.rolling(short_term).mean()
df['SMA2'] = df.Close.rolling(long_term).mean()

plots = df[['Close', 'SMA1', 'SMA2']].plot(subplots=False, figsize=(10, 4))
plt.show()
```

这段代码的运行结果如图 4-13 所示。

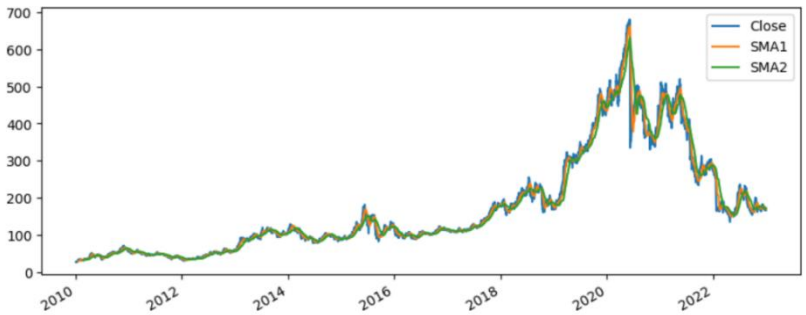

图 4-13 短周期和长周期的移动平均及其可视化

删除 Adj Close 列，其他数据列在后续计算其他指标时用到。

```
del df['Adj Close']
df.head()
```

- 构造 signal 列

构造一个新的数据列 signal，如果短周期移动平均大于长周期移动平均，那么 signal 为 1 否则为 0。构造 signal 列的目的，是准备构造真正的 Signal 列，该列就是数据的标注，即买入、卖出、和不做动作。signal 列是临时数据列。

```
df['signal'] = np.where(df['SMA1'] >df['SMA2'], 1.0, 0.0)
plots = df[['Close', 'SMA1', 'SMA2',   "signal"]].plot(subplots=True, figsize=(10, 10))
plt.show()
```

这段代码的运行结果如图 4-14 所示。

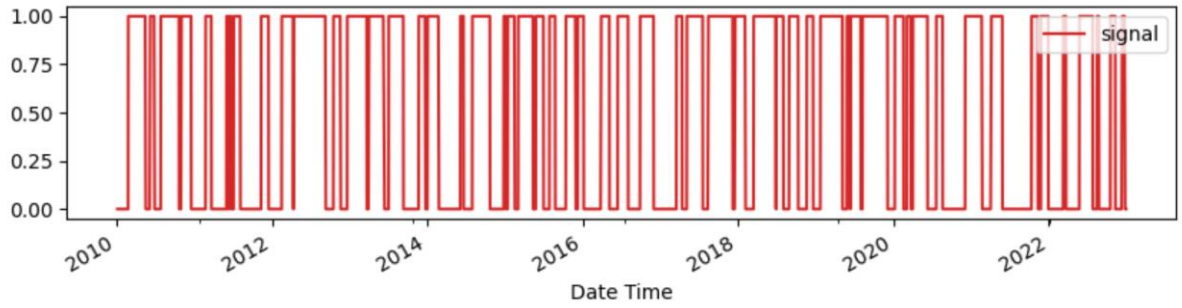

图 4-14 signal 数据列的可视化(部分结果)

展示 Close、SMA1、SMA2、signal 等数据列的部分数据（最后 250 行）。

```
df[['Close', 'SMA1', 'SMA2',  "signal"]].iloc[-250:].plot(figsize=(14,6),secondary_y=['signal'])
plt.show()
```

这段代码的运行结果如图 4-15 所示。

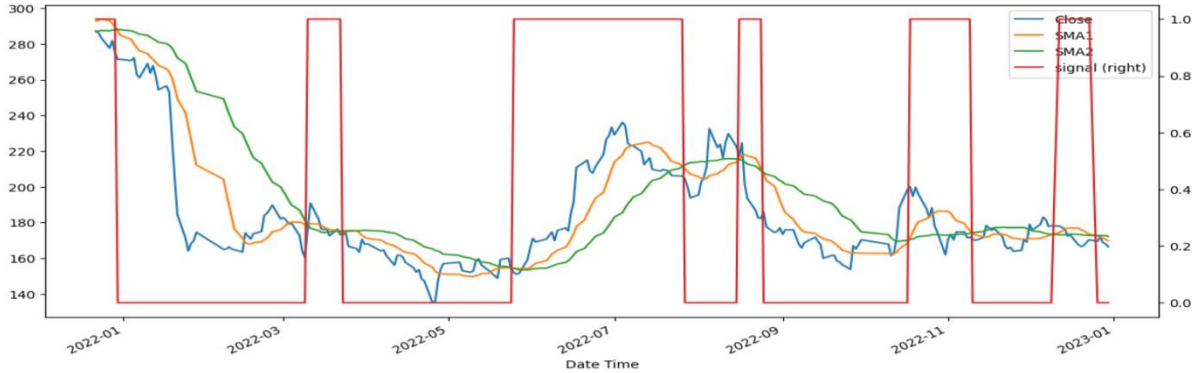

图 4-15 部分数据可视化(Close、SMA1、SMA2、signal)

可见，当短周期移动平均大于长周期移动平均，那么 signal 为 1，否则 signal 为 0。

- 构造 Signal 列

如上文所述，我们需要在两个移动平均发生交叉的时候进行买卖，在其他时候持有现金或者股票，不做动作。这时候，我们需要从 signal 数据列，生成 Signal 数据列。

转换规则为，从第二天开始到最后一天，如果判断数据列 signal 昨天是 0 今天为 1，那么 Signal 为 1 即买入，如果判断数据列 signal 昨天是 1 今天为 0，那么 Signal 为-1 即卖出，其他情况即 signal 昨天是 1 今天为 1，昨天是 0 今天为 0，那么 Signal 为 0 即持有或者不做动作。具体代码如下。

```
for i in range(1,3106):
    row_index = df.index[i]
    row_index_last = df.index[i-1]

    if df.loc[row_index,'signal'] ==0 and df.loc[row_index_last,'signal'] ==0:
        df.loc[row_index,'Signal'] = 0
    if df.loc[row_index,'signal'] ==1 and df.loc[row_index_last,'signal'] ==1:
        df.loc[row_index,'Signal'] = 0

    if df.loc[row_index,'signal'] ==1 and df.loc[row_index_last,'signal'] ==0:
        df.loc[row_index,'Signal'] = 1
    if df.loc[row_index,'signal'] ==0 and df.loc[row_index_last,'signal'] ==1:
```

```
            df.loc[row_index,'Signal'] = -1
```

df 的总行数为 3106，上述代码中 range(1,3106)得到 1,2,…,3105，即从第二行开始到最后一行的下标；我们根据本日 signal 和昨日 signal 做判断计算 Signal，所以第一行即下标为 0 的行无需处理。

接着对结果进行可视化，判断代码运行的结果是否正确。

```
df[['Close', 'SMA1', 'SMA2']].iloc[-250:].plot(figsize=(14,6))
plt.show()

df[['Signal']].iloc[-250:].plot(figsize=(14,6))
plt.show()
```

这段代码的运行结果如图 4-16 所示。由此可见，当短周期的移动平均从下边往上穿越长周期的移动平均，我们认为价格将要上涨，适合买入，Signal 为 1。当短周期移动平均从上边往下穿越长周期的移动平均，我们认为价格将要下跌，适合卖出，Signal 为-1。其他时候，我们持有现金，或者持有股票，不做动作，Signal 为 0。

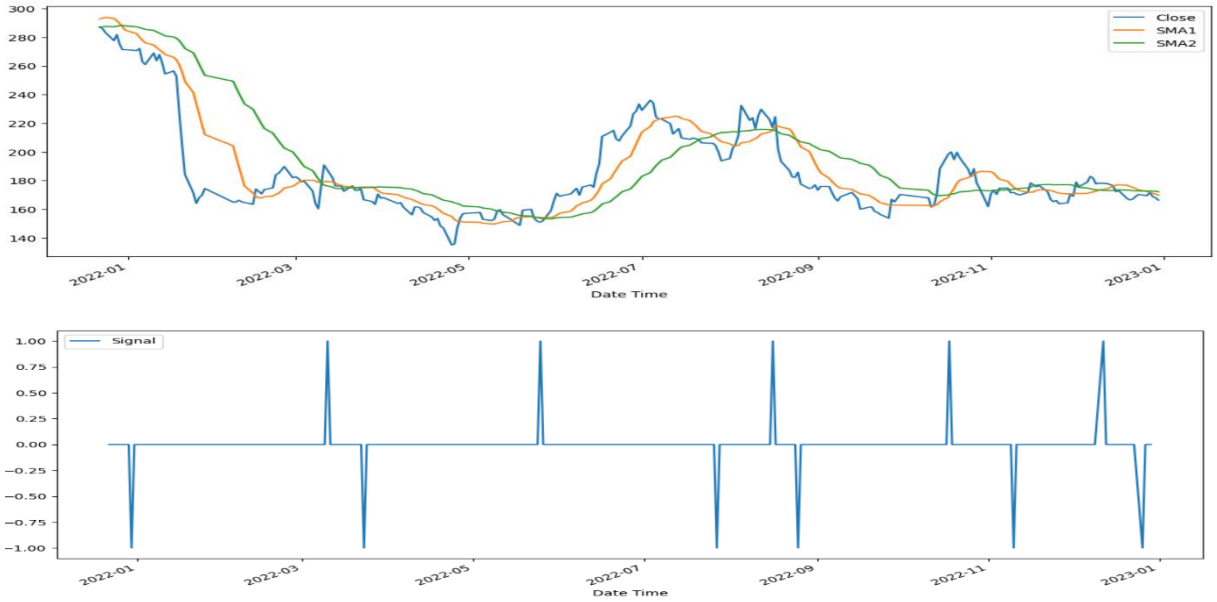

图 4-16 部分数据可视化(Close、SMA1、SMA2、Signal)

这时候由于 Signal 已经生成，signal 列可以删除。

4.3.3 特征构造与样本生成

- 构造差分数据列

首先，构造 DIFF_1 数据列，该列数据是 Close 数据列的差分，即今天收盘价减去昨天收盘价的价格差，第一天没有价格差，从第二天开始，都有今天收盘价减去昨天收盘价的价格差。

```
import numpy as np

df['DIFF_1'] = df["Close"].diff()
df.head()
```

接着，构造 DIFF_2 和 DIFF_3 数据列，这两个数据列分别是 DIFF_1 和 DIFF_2 平移一天得到的。这段代码的运行结果如图 4-17 所示。

```
df['DIFF_2'] = df["DIFF_1"].shift(1)
```

```
df['DIFF_3'] = df["DIFF_2"].shift(1)

df.head()
```

Date Time	Open	High	Low	Close	Volume	SMA1	SMA2	Signal	DIFF_1	DIFF_2	DIFF_3
2010-01-04	26.99	27.18	26.35	26.96	17902.49	NaN	NaN	NaN	NaN	NaN	NaN
2010-01-05	26.88	28.01	26.75	28.01	31438.24	NaN	NaN	0.0	1.05	NaN	NaN
2010-01-06	27.98	29.19	27.81	28.66	33094.95	NaN	NaN	0.0	0.65	1.05	NaN
2010-01-07	28.48	28.90	27.21	27.30	23864.32	NaN	NaN	0.0	-1.36	0.65	1.05
2010-01-08	27.00	27.29	26.65	27.20	15488.45	NaN	NaN	0.0	-0.10	-1.36	0.65

图 4-17 差分数据列

从图 4-17 可以看出，2010-01-07 这一天的 DIFF_1、DIFF_2 和 DIFF_3，分别表示 2010-01-07 和 2010-01-06、2010-01-06 和 2010-01-05、2010-01-05 和 2010-01-04 的差分值。现在把 DIFF_1、DIFF_2 和 DIFF_3 放在同一行上，目的是方便后续构造样本，即一行数据是一个样本。

- 构造技术指标(因子)

在这里，利用 TA Lib 技术分析库，构造了 6 组技术指标，作为除了 Open、High、Low、Close、Volume 之外的因子。这些指标的名称和含义，请参见表 4-2 的说明。

```
import talib as ta
import talib

df['MA13']=ta.MA(df.Close,timeperiod=13)
df['MA33']=ta.MA(df.Close,timeperiod=33)
df['MA13_MA33'] = df['MA13']- df['MA33']

df['EMA10']=ta.EMA(df.Close,timeperiod=10)
df['EMA30']=ta.EMA(df.Close,timeperiod=30)
df['EMA10_EMA30'] = df['EMA10']- df['EMA30']

df['MOM10']=ta.MOM(df.Close,timeperiod=10)
df['MOM30']=ta.MOM(df.Close,timeperiod=30)
df['MOM10_MOM30'] = df['MOM10']- df['MOM30']

df['RSI10']=ta.RSI(df.Close,timeperiod=10)
df['RSI30']=ta.RSI(df.Close,timeperiod=30)

df['K10'],df['D10']=ta.STOCH(df.High,df.Low,df.Close, fastk_period=10)
df['K10_D10'] = df['K10']- df['D10']

df['K30'],df['D30']=ta.STOCH(df.High,df.Low,df.Close, fastk_period=30)
df['K30_D30'] = df['K30']- df['D30']

df.head()
```

表 4-2 技术指标(因子)

序号	指标	含义
1	MA13、MA33 MA13_MA33	13 天的移动平均、33 天的移动平均 13 天的移动平均和 33 天的移动平均的差值 ● 移动平均代表价格运动趋势
2	EMA10、EMA30 EMA10_EMA30	10 天的指数移动平均、30 天的指数移动平均 10 天的指数移动平均和 30 天的指数移动平均的差值 ● 移动平均代表价格运动趋势
3	MOM10、MOM30 MOM10_MOM30	10 天的动量指标、30 天的动量指标 10 天的动量指标和 30 天的动量指标的差值 ● 动量可以视为一段时间内股价涨跌变动的比率
4	RSI10、RSI30	10 天的相对强弱指标、30 天的相对强弱指标 ● 相对强弱指标，一般用来衡量市场的超买和超卖状态，超买是过度买入，价格虚高，有可能下调，超卖是过度卖出，价格很低了，有可能上调
5	K10、D10 K10_D10	10 天的 K 线、10 天的 D 线 K10 和 D10 的差值 ● STOCH 函数返回 K 线和 D 线，K 线是快速确认线，当数值在 90 以上为超买，数值在 10 以下为超卖；D 线是慢速主干线，当数值在 90 以上为超买，数值在 10 以下为超卖
6	K30、D30 K30_D30	同上(周期为 30 天)

现在显示 Data Frame 的前 35 行，行下标从 0 到 34。

```
df_show = df.iloc[0:35]
df_show.head(35)
```

显示结果如图 4-18 所示，这里仅仅显示最后的第 33 行、34 行、35 行。我们发现，由于计算技术指标的时候，需要积攒一定的天数才能计算，到第 34 天，这行数据的各个数据列才得到非空的数值，也就是这行才能作为一个有效的样本。

第33行，下标为32	-4.54	61.703400	61.698536	52.750840	37.161652	15.589188	NaN	NaN	NaN
第34行，下标为33	-6.30	63.304972	62.203006	78.027101	55.961813	22.065288	81.239000	71.401810	9.837190
第35行，下标为34	-4.70	61.831576	61.771391	82.580077	71.119339	11.460737	83.773087	78.305684	5.467404

图 4-18 Data Fame 的第 33 行、34 行、35 行(部分列)

从 df 复制出 df2，两者都是 Data Frame，两者的结构和数据一模一样，然后剔除空值，准备构造样本。

```
df2=df.copy()
df2=df2.dropna()
df2.head()
```

删除 SMA1 列和 SMA2 列，这两列是为建立 Signal 列用到的，在构造样本的时候用不到。

```
del df2['SMA1']
del df2['SMA2']
```

显示各个数据列和 Signal 数据列的相关性，可以一定程度上让我们看到各个数据列多大程度上解释了 Signal 数据列。

```
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.graph_objects as go
import plotly.express as px

import seaborn as sb
corr = df2.corr()

corr = corr['Signal']
print(corr)

plt.rcParams.update({'font.size': 20}) # must set in top
ax = corr.plot.bar(rot=0,figsize=(26,6))
plt.xticks(rotation=270)
plt.show()
```

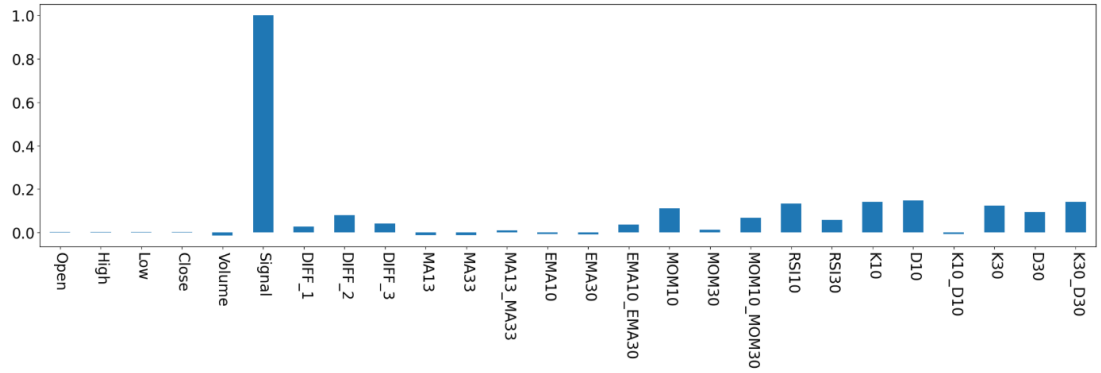

图 4-19 各个数据列和 Signal 的相关性

代码的执行结果如图 4-19 所示。从图中可以看出，DIFF_2、DIFF_3、MOM10、MOM10_MOM30、RSI10、RSI30、K10、D10、K30、D30、K30_D30 等数据列和 Signal 的相关性比较高，具有良好的解释作用。

- 构造样本

查看样本集的行数，为 3073 行。

```
row_count = df2.shape[0]
print("row_count", row_count)
```

构造样本集的 X 部分。

```
X = df2.copy()
del X['Signal']
X.head()
```

X 部分的主要字段包括 Open、High、Low、Close、Volume、DIFF_1、DIFF_2、DIFF_3、MA13、MA33、MA13_MA33、EMA10、EMA30 、EMA10_EMA30、MOM10、MOM30、MOM10_MOM30、RSI10、RSI30、K10、D10、K10_D10、K30、D30、K30_D30 等，总共 25 个字段。

构造样本的 y 部分，即 Signal 数据列。

```
y = df2['Signal'].copy()
y.head()
```

4.3.4 机器学习模型训练、评价与超参数调优

- 机器学习模型的训练与特征重要度

首先，对数据集进行划分，分为训练集和测试集，训练集占 80%，测试集占 20%。

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split( X,y,test_size=0.2)
```

设定固定参数即设定 n_estimators 为 100，使用训练集训练一个随机森林分类器模型。

```
from sklearn.ensemble import RandomForestClassifier

n_estimators=100
model = RandomForestClassifier(n_estimators,random_state=0, n_jobs=-1)
fit = model.fit(X_train,y_train)
```

利用随机森林预测器，对各个特征的重要度进行评价，并且可视化。

```
# feature importance
df_ret = pd.DataFrame(model.feature_importances_,index=X.columns, columns=['RandForest_Importance'])
print( df_ret)

plt.rcParams.update({'font.size': 20}) # must set in top
ax = df_ret.plot.bar(rot=0,figsize=(26,6))
plt.xticks(rotation=270)
plt.show()
```

这段代码的可视化效果如图 4-20 所示。

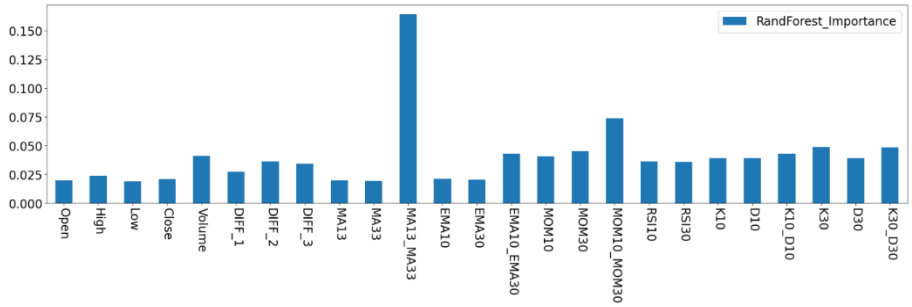

图 4-20 各个特征的重要度(基于随机森林分类器)

- 超参数优化与模型评价

随机森林分类器有一些超参数可以进行调整，比如随机森林中决策树的数量等。在这里，我们设定 n_estimators 为 50、100、或者 200，设定 max_features 为 auto 或者 sqrt，设定 max_depth 为 10、20、30、40、50 或者 None，使用 GridSearchCV 寻找优化的参数组合。这些参数的意义，请参考 scikit learn 库的在线文档。

```
from sklearn.model_selection import GridSearchCV

# Define parameter grid
```

```
param_grid = {'n_estimators': [50, 100, 200],
      'max_features': ['auto', 'sqrt'],
      'max_depth' : [None, 10, 20,30,40,50] }

# Initialize the Random Forest classifier
rfClassifier = RandomForestClassifier(random_state=42)

# Initialize GridSearchCV
gs_rfClassifier = GridSearchCV(estimator=rfClassifier, param_grid=param_grid, cv=5, n_jobs=-1, verbose=2)

model = gs_rfClassifier
# Fit model
model.fit(X_train, y_train)

# Best parameters
best_params = model.best_params_
print( best_params)
```

得到最优的参数组合为“{'max_depth': None, 'max_features': 'auto', 'n_estimators': 50}”。

在训练集上，评价模型的预测正确率，得分为 1.0。

```
y_train_predict = model.predict(X_train)

from sklearn.metrics import accuracy_score
score = accuracy_score(y_train.values,y_train_predict)
print("score train", score)
```

以热力图的方式显示混淆矩阵。

```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score, recall_score
cm = confusion_matrix(y_train.values, y_train_predict)

labels=["Sell", "NoAction", "Buy"]
sns.heatmap(cm,annot=True ,fmt="d",xticklabels=labels,yticklabels=labels)
plt.title('confusion matrix')   # 标题
plt.xlabel('Predict lable')   # x 轴
plt.ylabel('True lable')   # y 轴
plt.show()
```

这段代码的运行结果如图 4-21 所示。

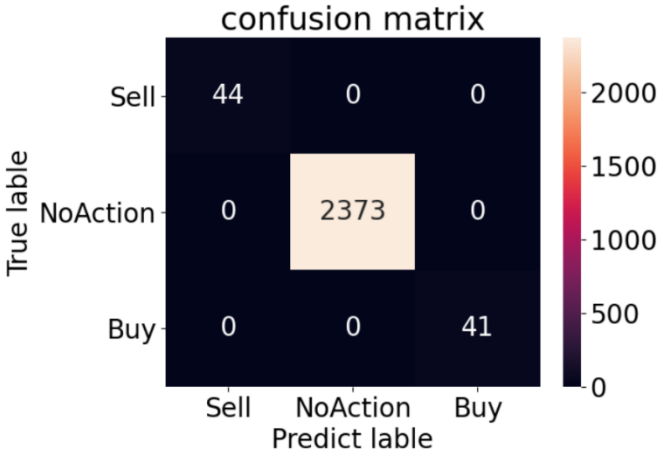

图 4-21 混淆矩阵的可视化(Train)

计算每个类别的 Precision、Recall、F1 分数。

```
from sklearn.metrics import classification_report
target_names = ['Sell', 'No Action', 'Buy']
print(classification_report(y_train.values, y_train_predict, target_names=target_names))
```

结果如图 4-22 所示。

	precision	recall	f1-score	support
Sell	1.00	1.00	1.00	44
No Action	1.00	1.00	1.00	2373
Buy	1.00	1.00	1.00	41
accuracy			1.00	2458
macro avg	1.00	1.00	1.00	2458
weighted avg	1.00	1.00	1.00	2458

图 4-22 每个类别的 Precision、Recall、F1 分数(Train)

在测试集上，评价模型的预测正确率，得分为 0.9642。

```
y_test_predict = model.predict(X_test)
score = accuracy_score(y_test.values,y_test_predict)
print("score test", score)
```

以热力图的方式显示混淆矩阵。

```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score, recall_score
cm = confusion_matrix(y_train.values, y_train_predict)

labels=["Sell", "NoAction", "Buy"]
sns.heatmap(cm,annot=True ,fmt="d",xticklabels=labels,yticklabels=labels)
plt.title('confusion matrix')  # 标题
plt.xlabel('Predict lable')  # x 轴
plt.ylabel('True lable')  # y 轴
plt.show()
```

这段代码的运行结果如图 4-23 所示。

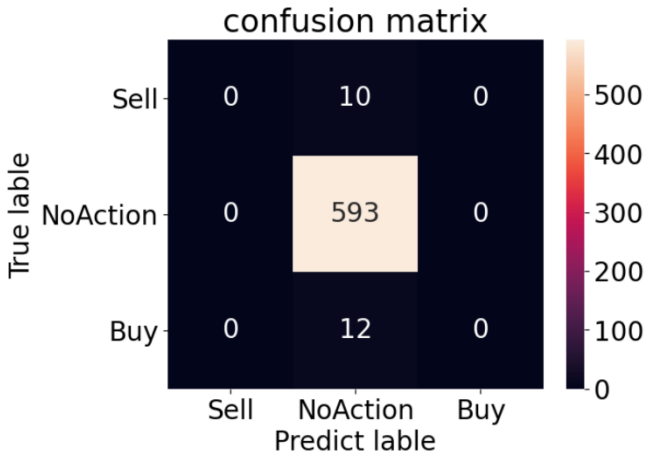

图 4-23 混淆矩阵的可视化(Test)

计算每个类别的 Precision、Recall、F1 分数。

```
from sklearn.metrics import classification_report
target_names = ['Sell', 'No Action', 'Buy']
print(classification_report(y_test.values, y_test_predict, target_names=target_names))
```

结果如图 4-24 所示。

	precision	recall	f1-score	support
Sell	0.00	0.00	0.00	10
No Action	0.96	1.00	0.98	593
Buy	0.00	0.00	0.00	12
accuracy			0.96	615
macro avg	0.32	0.33	0.33	615
weighted avg	0.93	0.96	0.95	615

图 4-24 每个类别的 Precision、Recall、F1 分数(Test)

- 模型持久化

使用最优参数建立模型，并且使用训练集对模型进行训练。

```
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(n_estimators=50,max_features='auto',max_depth=None, n_jobs=-1)
fit = model.fit(X_train,y_train)
```

把模型保存到 stock.pkl 文件中，并且重新装在到内存中。

```
# save model
import pickle
filename = 'stock.pkl'
pickle.dump(model, open(filename, 'wb'))

# load model
filename = 'stock.pkl'
loaded_model = pickle.load(open(filename, 'rb'))
```

然后在训练集和测试集上评价刚刚装载的模型预测的正确率，其结果和存盘前的模型是一模一样的。

```
# model on train set
score = loaded_model.score(X_train, y_train)
print("score", score)

# model on test set
score = loaded_model.score(X_test, y_test)
print("score", score)
```

4.3.5 交易策略构建与评价

- PyAlgoTrade 库

我们已经训练好机器学习模型，接下来基于 PyAlgoTrade 库，构建交易策略。PyAlgoTrade 是一个量化交易回测 Python 库。

PyAlgoTrade 有 6 大组件，分别是策略(strategy)、数据源(feed)、经纪商(broker)、数据序列(data series)、技术分析指标(technical)、以及优化器(optimizer)。其中：(1) 策略是交易模型的核心类，它决定何时买、何时卖、买卖多少等关键信息。(2) 在数据源方面，可以从 CSV 文件中加载数据推送给策略，对策略进行回测。也可以利用 Twitter 数据源，读取 Twitter 的事件信息进行分析，继而辅助做出交易决策。(3) 经纪商模块负责执行订单。(4) Data Series 是用于管理时间序列的抽象类，价格数据就是时间序列信息。(5) Technical 模块用于计算技术分析指标，比如简单移动平均线(SMA)、相对强弱指标(RSI)等。(6) Optimizer 可以利用多线程、多节点，提高策略回测的效率。

- 交易策略构建

我们继承 PyAlgoTrade 库的 BacktestingStrategy 类，派生出一个交易策略类 MyTestStrategy，装载上文所述训练好的模型，进行交易决策，即基于历史价格数据和各种技术指标（因子），决定买入、卖出、还是不做动作。

下面对“MyTestStrategy.py”的代码进行分析，该代码首先定义 MyTestStrategy 交易策略类，然后使用 stock01.csv 等文件进行历史回撤，获得性能指标。

- 导入 Python 库

首先，导入必要的 Python 库。这些库、模块、具体的类、具体的方法的用途，请参见代码的注释。

```
from pyalgotrade.barfeed.csvfeed import GenericBarFeed #价格数据流
from pyalgotrade.bar import Frequency #Price Bar 的频率

from pyalgotrade.technical import ma #移动平均

from pyalgotrade import strategy #交易策略
from pyalgotrade.stratanalyzer import returns #收益
from pyalgotrade.stratanalyzer import sharpe #夏普指数
from pyalgotrade.stratanalyzer import drawdown #最大回撤
from pyalgotrade.stratanalyzer import trades #交易次数
from pyalgotrade import plotter #历史回测结果的可视化

import pandas as pd #二维表处理库
import numpy as np #数组处理库
```

```
import talib as ta #技术分析库
import pickle #模型持久化库
```

- MyTestStrategy 的__init__方法

接着定义 MyTestStrategy 类。第一个成员函数为初始化方法__init__，其参数有 self、feed、instrument、smaPeriod1、smaPeriod2 等，分别是初始化的对象本身、输入的测试数据（Daily Price）、股票代码、第一个移动平均的周期即天数、第二个移动平均的周期即天数。

在__init__方法中，首先调用父类的初始化方法，其中的 100000 为初始资金。记录 instrument 到成员变量__instrument 中，设定成员变量__position 为空，即股票为空，也就是目前没有购入任何股票。从输入数据 feed 中，找出 instrument 股票代码对应的价格数据，记录到__prices 成员变量中，并且基于__prices 成员变量计算短周期和长周期移动平均，放到成员变量__sma1 和__sma2 中。注意，价格数据是以一个 Price Bar 一个 Price Bar 的形式传送给交易策略的 onBars 回调函数的。

接下来，构造一个空的 Data Fame，有 Open、High、Low、Close、Volume 等数据列。把上文保存的模型装载到内存中，准备使用。

```
class MyTestStrategy(strategy.BacktestingStrategy):
    def __init__(self, feed, instrument, smaPeriod1, smaPeriod1):
        super(MyTestStrategy, self).__init__(feed, 100000)
        self.__instrument = instrument
        self.__position = None
        # We'll use adjusted close values instead of regular close values.
        self.setUseAdjustedValues(True)
        self.__prices = feed[instrument].getPriceDataSeries()
        self.__sma1 = ma.SMA(self.__prices, smaPeriod1)
        self.__sma2 = ma.SMA(self.__prices, smaPeriod2)

        self.df = pd.DataFrame( { 'Open': [], 'High': [], 'Low': [], 'Close': [], 'Volume': [] })
        #self.df    = pd.DataFrame(columns=['Open', 'High', 'Low','Close','Volume'])
        print("df",self.df.head())

        # load model
        filename = 'stock.pkl'
        self.model = pickle.load(open(filename, 'rb'))
```

- MyTestStrategy 的一些回调函数

```
    def onEnterCanceled(self, position):
        self.__position = None

    def onExitOk(self, position):
        self.__position = None

    def onExitCanceled(self, position):
        # If the exit was canceled, re-submit it.
        self.__position.exitMarket()
```

onEnterCanceled 函数在买入股票的订单被取消的时候，由 PyAlgoTrade 运行时(Runtime)回调。onExitOk 函数在卖出股票的订单被完全执行后，由 PyAlgoTrade 运行时(Runtime)回调。这两个函数都把当前持有的股票设置为空。onExitCanceled 函数在卖出股票的订单被取消的时候，由 PyAlgoTrade 运行时(Runtime)回调，在这里重新提交这个订单，即把被取消的订单再提交一次。

注意代码里的 getSMA1 和 getSMA2 成员函数，分别返回__sma1 和__sma2 成员变量，即短周期移动平均和长周期移动平均的数据序列。

```
def getSMA1(self):
    return self.__sma1
def getSMA2(self):
    return self.__sma2
```

- MyTestStrategy 的 onBars 回调函数

```
def onBars(self, bars):
    bar = bars[self.__instrument]

    _datetime = bar.getDateTime()
    #print("_datetime",_datetime)
    _open = bar.getOpen()
    _high = bar.getHigh()
    _low = bar.getLow()
    _close = bar.getClose()
    _volume = bar.getVolume()

    action = 0
    row_count = self.df.shape[0]
    if(row_count ==0):
        one_row_list = [_open, _high, _low, _close, _volume]
        one_row_array = np.asarray( one_row_list)
        one_row_array = one_row_array.reshape(1,5)
        self.df = pd.DataFrame(one_row_array,
                               index=[_datetime],
                               columns=['Open','High','Low','Close','Volume'])

        print("self.df + one row",self.df.head())
    else:
        one_row_list = [_open, _high, _low, _close, _volume]
        one_row_array = np.asarray( one_row_list)
        one_row_array = one_row_array.reshape(1,5)
        df_one = pd.DataFrame(one_row_array,
                               index=[_datetime],
                               columns=['Open','High','Low','Close','Volume'])

        self.df = pd.concat([self.df, df_one])
```

```
            if(self.df.shape[0] == 6):
                print("self.df",self.df.head())

            if(self.df.shape[0] >=34):
                df = self.df
                df_test = df.iloc[-34:,:].copy()
                #del df_test['Adj Close']

                df_test.loc[:,'DIFF_1'] = df_test.loc[:,"Close"].diff()
                df_test.loc[:,'DIFF_2'] = df_test.loc[:,"DIFF_1"].shift(1)
                df_test.loc[:,'DIFF_3'] = df_test.loc[:,"DIFF_2"].shift(1)

                df_test.loc[:,'MA13']=ta.MA(df_test.Close,timeperiod=13)
                df_test.loc[:,'MA33']=ta.MA(df_test.Close,timeperiod=33)
                df_test.loc[:,'MA13_MA33'] = df_test.loc[:,'MA13']- df_test.loc[:,'MA33']

                df_test.loc[:,'EMA10']=ta.EMA(df_test.Close,timeperiod=10)
                df_test.loc[:,'EMA30']=ta.EMA(df_test.Close,timeperiod=30)
                df_test.loc[:,'EMA10_EMA30'] = df_test.loc[:,'EMA10']- df_test.loc[:,'EMA30']

                df_test.loc[:,'MOM10']=ta.MOM(df_test.Close,timeperiod=10)
                df_test.loc[:,'MOM30']=ta.MOM(df_test.Close,timeperiod=30)
                df_test.loc[:,'MOM10_MOM30'] = df_test.loc[:,'MOM10']- df_test.loc[:,'MOM30']

                df_test.loc[:,'RSI10']=ta.RSI(df_test.Close,timeperiod=10)
                df_test.loc[:,'RSI30']=ta.RSI(df_test.Close,timeperiod=30)

                df_test.loc[:,'K10'],df_test.loc[:,'D10']=ta.STOCH(
                    df_test.High,df_test.Low,df_test.Close, fastk_period=10)
                df_test.loc[:,'K10_D10'] = df_test.loc[:,'K10']- df_test.loc[:,'D10']

                df_test.loc[:,'K30'],df_test.loc[:,'D30']=ta.STOCH(
                    df_test.High,df_test.Low,df_test.Close, fastk_period=30)
                df_test.loc[:,'K30_D30'] = df_test.loc[:,'K30']- df_test.loc[:,'D30']

                X = df_test.copy()
                X = X.iloc[-1:,:]
                y_predict = self.model.predict(X)
                #print(y_predict)
                action=y_predict[0]
                if action==1 or action ==-1:
                    print(action)

        if(action == 0):
```

```
            pass
        elif (action == 1): #  如果没有头寸，买入
            if self.__position is None:
                shares = int(self.getBroker().getCash() * 0.9 / bars[self.__instrument].getPrice())
                # Enter a buy market order. The order is good till canceled.
                self.__position = self.enterLong(self.__instrument, shares, True)
        elif (action == -1): #  如果有头寸，卖出
            if self.__position is None:
                pass
            elif not self.__position.exitActive():
                self.__position.exitMarket()
```

onBars 函数首先取得__instrument 股票的当前 Price Bar，然后提取日期、开盘价、最高价、最低价、收盘价和交易量等。

接着判断 df(Data Frame)成员变量的行数。如果行数为 0，那么用刚刚提取的日期、开盘价、最高价、最低价、收盘价和交易量构造一个 Data Fame，替换 df 成员变量。

如果 df 的行数不为 0，意味着 df 已经积攒了一定的价格数据行。那么用刚刚提取的日期、开盘价、最高价、最低价、收盘价和交易量构造一个临时 Data Frame，合并到已有的成员变量 df 的后边，也就是为 df 增加一个价格数据行。

如果成员变量 df 的价格数据行已经超过 34 行，那么把最后 34 行(请回顾 4.3.3 节，我们需要积攒一定量的价格数据，才能计算各个技术指标，即在下标为 33 的行，也就是第 34 行，才是第一个有效的样本)拷贝出来，作为一个新的 Data Frame 即 df_test。

计算 DIFF_1、DIFF_2、DIFF_3、MA13、MA33、MA13_MA33、EMA10、EMA30、EMA10_EMA30、MOM10、MOM30、MOM10_MOM30、RSI10、RSI30、K10、D10、K10_D10、K30、D30、K30_D30 等指标，加上 Open、High、Low、Close、Volume 等字段，总共 25 个字段。这个和训练模型的时候构造的技术指标是一样的，以便利用训练好的模型进行预测。

把 df_test 的最后一行即第 34 行切割下来作为 X，输入装载好的模型 model，让它做出预测，预测的结果为 1 买入、-1 卖出、或者 0 不做动作。

根据预测结果，如果是 0，就什么都不做。如果预测结果是 1，那么在没有股票的情况下，利用现有资金的 90%除以股票价格，看看能够买多少股，然后买入股票，即做多。如果预测结果是-1，那么如果目前有股票，那么把股票全部卖掉。

买入和卖出是配对的，即先买入后卖出(即做多)，如表 4-3 所示。这是程序运行以后，在终端上输出的一些提示信息。

表 4-3 买卖配对

序号	预测的 action	是否买卖
1	SelL Signal = -1.0	不卖出，因为开始没有股票
2	SelL Signal = -1.0	不卖出，因为开始没有股票
3	BUY Signal = 1.0 BUY Action	买入，用现金买股票
4	BUY Signal = 1.0	不买入，已经有股票，不再买入
5	SelL Signal = -1.0	卖出，卖出股票

	SELL Action	
6	SelL Signal = -1.0	不卖出，因为股票已经卖出
7	BUY Signal = 1.0 BUY Action	买入，用现金买股票
8	BUY Signal = 1.0	不买入，已经有股票，不再买入
9	SelL Signal = -1.0 SELL Action	卖出，卖出股票

- 全局函数 test_one_stock

定义全局函数 test_one_stock，这个函数有两个参数，分别是 stock_name 即股票代码，stock_file_name 即股票价格数据文件。

在这个函数中，首先创建 GenericBarFeed 类的对象 feed，然后从 stock_file_name 文件装载价格数据。设定短周期为 13 天，长周期为 33 天，创建 MyTestStrategy 类的对象 myTestStrategy。

创建收益分析器 returnsAnalyzer、夏普指数分析器 sharpeRatioAnalyzer、最大回撤分析器 drawdownAnalyzer、交易次数分析器 tradesAnalyzer，并且挂接(attach)到 myTestStrategy 上。PyAlgoTrade 运行时不断地把价格数据一个 Price Bar 一个 Price Bar 地馈入 myTestStrategy，myTestStrategy 交易策略开始运行起来，这些分析器会记录这些性能指标所需的基础数据。

```
def test_one_stock( stock_name, stock_file_name):
    feed = GenericBarFeed(Frequency.DAY, None, None)
    feed.addBarsFromCSV(stock_name, stock_file_name)
    smaPeriod1 = 13
    smaPeriod2 = 33
    myTestStrategy = MyTestStrategy(feed, stock_name, smaPeriod1, smaPeriod2)

    # Attach analyzers to the strategy
    returnsAnalyzer = returns.Returns()
    myTestStrategy.attachAnalyzer(returnsAnalyzer)
    sharpeRatioAnalyzer = sharpe.SharpeRatio()
    myTestStrategy.attachAnalyzer(sharpeRatioAnalyzer)
    drawdownAnalyzer = drawdown.DrawDown()
    myTestStrategy.attachAnalyzer(drawdownAnalyzer)
    tradesAnalyzer = trades.Trades()
    myTestStrategy.attachAnalyzer(tradesAnalyzer)

    # Attach the plotter to the strategy.
    plt = plotter.StrategyPlotter(myTestStrategy)
    # Include the SMA in the instrument's subplot to get it displayed along with the closing prices.
    plt.getInstrumentSubplot(stock_name).addDataSeries("SMA1", myTestStrategy.getSMA1())
    plt.getInstrumentSubplot(stock_name).addDataSeries("SMA2", myTestStrategy.getSMA2())
    # Plot the simple returns on each bar.
    plt.getOrCreateSubplot("returns").addDataSeries("Simple returns", returnsAnalyzer.getReturns())

    # Run the strategy.
    myTestStrategy.run()
```

```
#print ("Final portfolio value1: $%.2f" % (myTestStrategy.getBroker().getEquity())    )
print ("Final portfolio value2: $%.2f" % (myTestStrategy.getResult())    )
print ("Cumulative returns: %.2f %%" % (returnsAnalyzer.getCumulativeReturns()[-1] * 100)    )
print ("Sharpe ratio: %.2f" % (sharpeRatioAnalyzer.getSharpeRatio(0.03))    )
print ("Max. drawdown: %.2f %%" % (drawdownAnalyzer.getMaxDrawDown() * 100)    )
print ("Trade Count: %d" % (tradesAnalyzer.getCount())    )
#print ("Longest drawdown duration: %s"% (drawdownAnalyzer.getLongestDrawDownDuration())    )

# Plot the strategy.
plt.plot()
```

接着，为 myTestStrategy 创建一个绘图对象 plt。绘图对象的第一个子图，显示 Close 价格序列，以及买点、卖点。在 Close 价格子图上，显示 SMA1 即短周期移动平均，以及显示 SMA2 即长周期移动平均。第二个子图为 Simple returns，即每次买卖的盈亏。第三个子图为浮动的资产净值。

最后调用 myTestStrategy 的 run 方法运行交易策略，策略运行结束后，显示最后的资产净值，收益率、夏普指数、最大回撤、交易次数等指标。并且把创建的绘图对象 plt 显示出来。

- 执行交易策略

```
test_one_stock("stock01", "./stock01.csv")
```

调用 test_one_stock 函数，参数为"stock01"(股票代码)和"./stock01.csv"(股票价格数据)。使用 stock01.csv 对交易策略进行回测。

- 交易策略评价

该策略的运行结果如图 4-25 和图 4-26 所示。

图 4-25 包含三个子图，分别是价格子图、收益子图(盈亏)、资产净值子图(累计盈亏)。价格子图上，还显示了 SMA1 和 SMA2 移动平均线，并且标记了策略运行以后的买入和卖出操作的时间点。买入用尖角向上的三角形表示，卖出用尖角向下的三角形表示。

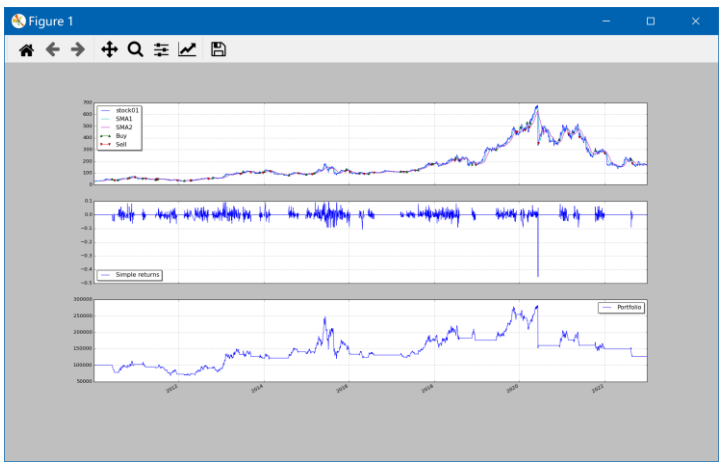

图 4-25 myTestStrategy 策略运行的可视化结果(stock01.csv)

图 4-26 显示策略运行结束后的最后资产净值、收益率、夏普指数、最大回撤和交易次数等指标。在 stock01.csv 价格数据上，该策略的模拟运行结果为：

(1) 资产净值为$126325.79。

(2) 收益率为 26.33%，收益较好。

(3) 夏普指数为 0.14，比较差。

(4) 最大回撤为 55.42%，最大回撤太大，该交易策略的风险较大。

(5) 交易次数为 32。

```
Final portfolio value2: $126325.79
Cumulative returns: 26.33 %
Sharpe ratio: 0.14
Max. drawdown: 55.42 %
Trade Count: 32
```

图 4-26 myTestStrategy 策略运行的输出(stock01.csv)

- 从 stock01.csv 到 stock02.csv

上文的模型是在 stock01.csv 上进行训练的，现在构造好交易策略以后，也在 stock01.csv 上进行测试，这样的测试说服力不强。

现在我们把交易策略在 stock02.csv 上进行回测，看看它的泛化能力怎么样。

该策略的运行结果如图 4-27 和图 4-28 所示。

图 4-27 包含三个子图，分别是价格子图、收益子图(盈亏)、资产净值子图(累计盈亏)。价格子图上，还显示了 SMA1 和 SMA2 移动平均线，并且标记了策略运行以后的买入和卖出操作的时间点。买入用尖角向上的三角形表示，卖出用尖角向下的三角形表示。

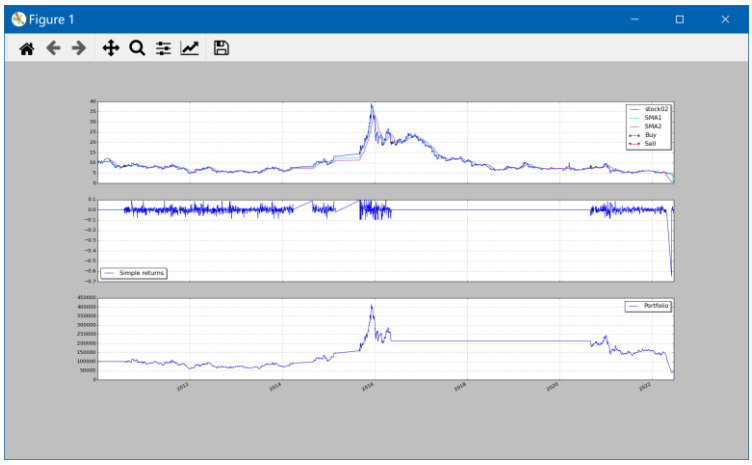

图 4-27 myTestStrategy 策略运行的可视化结果(stock02.csv)

图 4-28 显示策略运行结束后的最后资产净值、收益率、夏普指数、最大回撤、交易次数等指标。在 stock02.csv 价格数据上，该策略的模拟运行结果为：

(1) 资产净值为$40519.68。

(2) 收益率为-59.48%，收益很差。

(3) 夏普指数为-0.06，很差。

(4) 最大回撤为 90.57%，最大回撤太大。

(5) 交易次数为 1。

```
Final portfolio value2: $40519.68
Cumulative returns: -59.48 %
Sharpe ratio: -0.06
Max. drawdown: 90.57 %
Trade Count: 1
```

图 4-28 myTestStrategy 策略运行的输出(stock02.csv)

从结果来看，这个交易策略在 stock02.csv 上的历史回测结果并不好，说明其泛化能力有限。我们关心的问题是，原因到底是什么呢？有没有改进的余地呢？将在 4.4 节讨论。

4.4 总结与展望

1.数据标注

前文所述的数据标注方法，是一种简单的方法。它基于这样的假设，当短周期的移动平均从下边往上穿越长周期的移动平均，我们认为价格将要上涨，适合买入。当短周期移动平均从上边往下穿越长周期的移动平均，我们认为价格将要下跌，适合卖出。这样的操作并不一定是盈利的，当价格处在总体上涨趋势中的时候，比如在图 4-29 中，在数字 1 对应的点买入，在数字 2 对应的点卖出，是盈利的。但是如果价格处在总体下跌的趋势中，比如在图 4-29 中，在数字 3 对应的点买入，在数字 4 对应的点卖出，则是亏损的。我们应该考虑其他的数据标注方法。

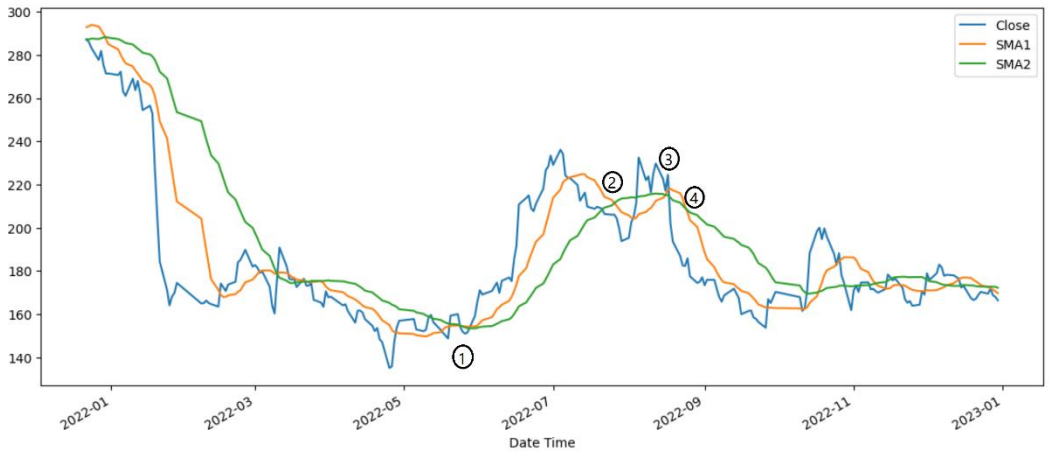

图 4-29 短周期移动平均和长周期移动平均的交叉(原图 4-11)

- 均值与阈值

我们可以计算一段时间内的价格的均值，并且设定一个阈值。当价格在均值加上一个阈值之上时，适合卖出。当价格在均值减去一个阈值之下时，适合买入。

如果我们已经判断到一段时间内，价格的变化处在横盘整理中或者总体上涨趋势中，相信这样的标注方法会取得比较好的盈利效果。

- 局部最高点和局部最低点

价格的变化千变万化，即便在总体上涨和下跌的趋势中，也有一些反向的调整。可以考虑对价格数据进行平滑处理，去掉细碎的价格变化，保留总体趋势。然后找出平滑后的价格曲线的局部最高点和局部最低点，把局部最低点作为买入点，把局部最高点作为卖出点。如图 4-30 所示。

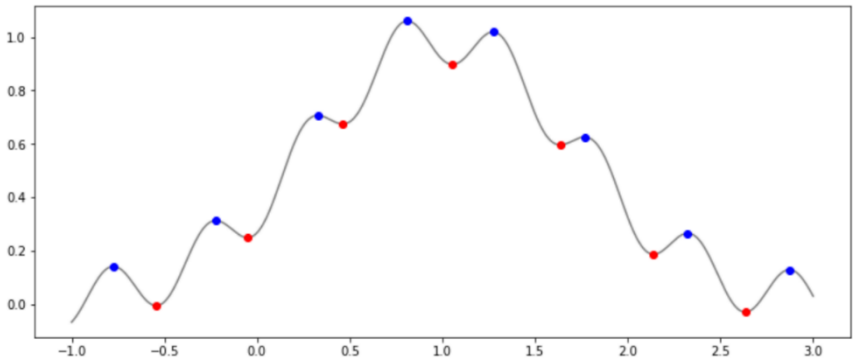

图 4-30 价格数据的平滑与局部最高点/局部最低点

- 局部最高点和局部最低点的改进

人们一般对总是能够踩中高点和低点，完成逃顶和抄底表示怀疑。价格运动处在上涨趋势、下跌趋势和横盘整理中，从下跌转换为上涨和从上涨转换为下跌，称为反转。一般地，反转需要一些指标的确认，即确认反转是真正发生了。

我们可以选择在局部最高点和局部最低点之后一段时间之后，选择这样的时间点作为买入和卖出点。结合我们从数据上计算的一些技术指标，以及其他渠道收集的一些变量(比如交易量的放量)，作为解释因子确认这些趋势反转，这样构建的模型可能更加具有可信度。

- 模仿人类高手进行操作

每年一些头部金融机构举办股票交易和期货交易大赛。选手们通过实盘模拟，完成选股和交易，最后决出优胜者。这些优胜者的操作，即他们选择了什么股票，什么时候买入，什么时候卖出，可以看作是人工标注。

大赛的举办者掌握了这些操作信息，是否可以加以应用，以及如何使用，目前在法律方面还存在空白，没有规定。

2.数据的规范化：从绝对值到相对值，提高模型泛化能力

在 4.3.5 节中，我们在 stock02.csv 上测试交易策略。由于模型是在 stock01.csv 上进行训练的，交易策略在 stock02.csv 上的测试结果很差。到底是什么原因呢？真正的原因是，我们训练的模型拟合到具体的价格上，而不是识别价格的运动趋势。

记得我们在构造样本的时候，我们大量地使用了绝对值，即 Open、High、Low、Close、Volume、DIFF_1、DIFF_2、DIFF_3 等使用的是绝对的价格和价格差。这样构造的样本，使得模型适配到具体数值上，而不是识别出价格运动的模式。可以考虑使用相对值，即对这些数值进行滑动窗口式的相对值处理，即在一个滑动窗口内，计算价格的相对变化，包括日内 Open、High、Low、Close 之间的相对变化，以及日间 Close 价格之间的相对变化，即 DIFF_1、DIFF_2、DIFF_3 等特征改成相对变化率，相信模型的表现会更好。

在概论一章中，我们介绍了一系列数据预处理方法，在这里都可以尝试，目的是在样本构造的时候，样本表达的是最近的价格以及特征的变化模式，而不是绝对值。

具体的实现，有赖于读者在“实验二”中，去探索和尝试。

3.各类技术分析指标的运用、因子设计与选择

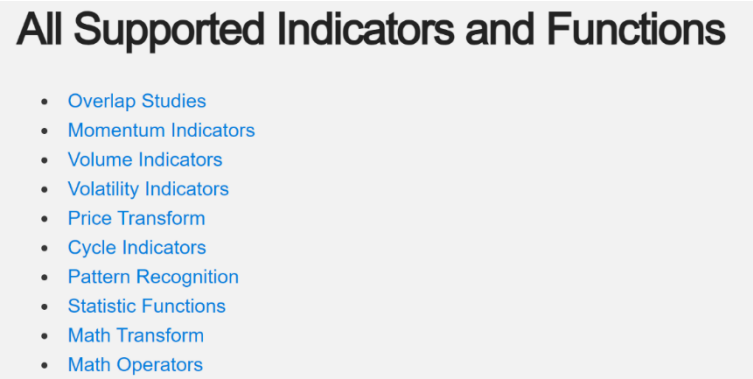

图 4-31 Ta-Lib 库提供的指标类别(https://ta-lib.github.io/ta-lib-python/funcs.html)

TA-Lib 库提供了 200 个左右的技术指标的计算方法。Ta-Lib 库提供的指标，可以分为趋势性指标(Trend or Overlaps Indicators)、动量指标(Momentum Indicators)、交易量指标(Volume Indicators)、周期性指标(Cycle Indicators)、波动性指标(Volatility Indicators)、形态识别指标(Pattern Recognition)等类别，如图 4-31 所示。此外，读者也可以自行设计有用的因子。

4.其他经济变量

在进行量化交易的时候，人们不仅关注价格的变化情况，还参考基本面数据、宏观经济数据、微观经济数据、财经新闻和博客等数据。在此基础上，进行综合考虑，判断合适的买点和卖点。

这些数据的引入，需要有效的方法对其进行分析，提取有用的信息，作为交易决策的参考。也就是从这些数据上概括和计算因子，作为现有自变量的补充。

4.5 习题

1.请介绍智慧金融的内涵。

2.什么是量化交易，量化交易有什么优势。

3.如何把量化交易凝练为机器学习的问题？

4.请介绍量化交易系统的主要模块。

5.如何评价量化交易系统，什么是年化收益率、最大回撤、夏普指数？

6.请解释什么是量化交易中的选股，什么是量化交易中的择时？

4.6 实验

4.6.1 实验环境

表 4-4 列出了运行本章代码需要的 Python 版本、以及各个 Python 库的版本。读者也可以安装更新的版本，本章代码一般不需要修改，即可运行。

表 4-4 Python 以及 Python 库的版本

序号	Python 以及 Python 库	版本号
1	Python	Python 3.8.13
2	pandas	1.2.4
3	numpy	1.19.5
4	scipy	1.6.0
5	scikit-learn	0.22
6	statsmodels	0.12.2
7	Keras	2.4.3
8	tensorflow	2.2.0
9	PyAlgoTrade	0.20
10	tushare	1.2.64
11	TA-Lib	0.4.19

4.6.2 实验一

请读者运行和分析本章提供的代码。

任务 1、使用 Jupyter 打开 training.ipynb 文件，参考本章 4.3.1~4.3.4 节的描述，分析代码，并且运行代码。了解股票代码下载、测试集数据下载、训练集数据下载、装载数据集、进行数据标注、特征构造、样本构造、训练模型、模型评价、模型存盘与重新装载等功能。

任务 2、使用 Spyder 打开 MyTestStrategy.py 文件，参考本章 4.3.5 节的描述，分析代码，并且运行代码。了解特征构造、样本构造、模型预测与交易决策等功能，并且对交易策略的运行结果进行分析。

4.6.3 实验二

请读者模仿本章提供的代码，自行编写代码，完成如下任务。

任务 1、数据标注：请读者自行考虑数据标注方法，在训练集上标注买入点、卖出点、不做动作的点。

任务 2、特征构造：请读者自行考虑，构造更加具有解释作用的特征。请参考 4.3.3 节的描述。

任务 3、模型选择与模型的超参数优化：请读者自行考虑采用不同的分类方法，利用样本进行训练，并且对模型进行超参数优化，已达到最优的预测效果。模型训练完成以后，存盘。

任务 4、交易策略评价：构建交易策略，装载已经训练好的模型，利用新数据构造特征，生成新样本，输入模型，获得买入、卖出、持有的输出，据此进行交易。

运行交易策略，分析该策略的性能表现。

5 知识图谱与危化品管理

6 智慧医疗

7 直播电商与智能推荐