中国期货市场量化交易(R 与 C++ 版)

李尉 著

清華大學出版社
北 京

内 容 简 介

本书主要介绍如何运用统计分析和机器学习等方法对中国期货市场量化交易进行建模分析。不仅覆盖了最基础的数据获取、数据清理、因子提取、模型构造以及最后的动态投资组合优化、C++ 编程实现等方面，而且有丰富的代码方便读者临摹学习和修改提升。本书中的数据首先是交易所最原始的期货分笔数据，在此基础上整合成 5 分钟 K 线，然后再计算预测因子，最后套入统计预测模型。在交易层面，采用严谨的滚动优化方式，充分考虑了滑点和手续费，严格测试。另外本书还覆盖了中低频的趋势策略以及高频的短趋势策略，最后也详细介绍了跨期套利策略，以及对读者择业就业的建议。

本书内容的广度和深度都是国内市场上少见的，适合相关专业人士和感兴趣的投资爱好者阅读，如高校数理类和经管类师生及证券、期货、私募证券、公募基金等量化交易相关从业人员，以及对机器学习在金融方面运用的相关人士和对量化交易感兴趣的各行各业人士。

图书在版编目(CIP)数据

中国期货市场量化交易 : R与C++版 / 李尉著. — 北京：清华大学出版社，2018
ISBN 978-7-302-50322-4

Ⅰ. ①中… Ⅱ. ①李… Ⅲ. ①期货市场－研究－中国 Ⅳ. ①F832.5

中国版本图书馆 CIP 数据核字（2018）第 114984 号

责任编辑：刘志彬
封面设计：汉风唐韵
版式设计：方加青
责任校对：宋玉莲
责任印制：丛怀宇

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　**邮　　编：**100084
社 总 机：010-62770175　　**邮　　购：**010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 装 者：三河市国英印务有限公司
经　　销：全国新华书店
开　　本：170mm×240mm　　**印　　张：**19.75　　**字　　数：**319 千字
版　　次：2018 年 11 月第 1 版　　**印　　次：**2018 年 11 月第 1 次印刷
定　　价：89.00 元

产品编号：078896-01

前言

期货市场是一个有着悠久历史的金融市场，早在几百年前，芝加哥一带的农民聚集在一起，商量一个有中央结算性质的交易场所，最终成立了芝加哥期货交易所。后来随着电子计算机技术的发展，期货交易所日益电子化，交易更为便捷，交易大厅的交易员也逐渐演变成计算机前的量化交易员和程序员。

相比期货交易，量化交易是一个更新鲜的概念。传统的期货交易有很多技术分析的书籍，如经典的《日本蜡烛图技术》《期货市场技术分析》等，一般更着重于使用技术图表分析 K 线形态，从而给交易员提升买卖点位，辅助交易员主观交易。

然而，量化交易不大一样，或者说是期货交易的升华。量化交易更多是运用现代统计学模型，包括机器学习、深度学习等模型来预测市场的价格变化，从而编写计算机程序，实现自动交易。更广泛地说，投资的整个过程，包括品种的选择、价格变化的预测、投资组合权重的分配、最小化交易成本地下单等，都可以使用相应的量化模型来分析，并且提供一套系统性、科学性的测试方法。因此，量化交易跟传统意义上辅助交易员下单的技术分析还是很不一样的。

对于一些基本面信息，本质上也可以融入量化模型中，因此，量化分析和基本面分析并不矛盾。并且现在市场上也有很多期货和股票方面的基本面量化的书籍供读者参考翻阅。

目前，国内期货市场蓬勃发展，量化交易方兴未艾。然而，目前国内很多私募量化基金交易的期货策略都是传统的程序化交易方法，与国外基于统计分析、机器学习模型的方法相比存在较大差距。然而国内却没有有关方面的书籍，即使有也是在股票投资方面，期货方面仍属空白。因考虑到广大理工科学生和科研人员对金融量化交易有着极大的热情，且本人有国内外期货量化交易多年

的经验，于是写作了本书。

本书特色

1. 国内率先系统性运用统计和机器学习模型研究中国期货市场的书

国外用机器学习模型研究股票与期货市场的书确实存在，但比较新，如《Machine Trading: Deploying Computer Algorithms to Conquer the Markets》，主要运用 Matlab，分析的主要是美国股票市场日线数据。本书分析的是国内期货市场，使用的是分笔数据和 5 分钟 K 线数据，频率上要比市场上同类书籍高出不少。另外本书的模型都是本人实战多年的成果，有着良好的实盘交易记录，并且还给出了研究用的 R 代码和实盘用的 C++ 代码，方便读者学习。能做到这点的，市面上无论中国还是美国，以本人的经验看，尚不存在。

2. 理论结合实际，由浅入深，娓娓道来

本书从最基本的分笔数据出发，如从如何获得数据、如何合成 K 线等，到最后的 C++ 实盘交易程序，应该说量化交易的内容都有所涵盖。从最基本的基于买卖规则的策略，到最后基于深度学习、增强学习的策略都有所涉及，而且有详细的 R 和 C++ 代码，方便大家自主学习。本人也有着丰富的国内外量化交易经验，不仅在美国对冲基金公司全职工作过，而且也在国内多家期货公司和私募基金工作过。本书里面的代码经历过多年实盘交易的检验，另外也会穿插介绍本人的职场经历，可以供各位参考。

3. 覆盖高频与中低频交易

绝大多数的量化交易书籍都不会涉及高频交易，本书却给出了研究高频交易模型的框架，同时检验了多种经典的机器学习预测模型。一般来说，相对于中低频交易，高频交易数据量更大，就可以训练更复杂的模型，因此本书也探讨了很多非线性的模型。但对于中低频交易的训练，还是以传统线性模型为主。

本书内容及体系结构

第 1 章　期货基本策略概要。简单介绍了目前国内流行的股票对冲、商品 CTA、高频交易等策略，以及常见的程序化交易平台，对比了 R、Python、Matlab 等常见的分析语言，并且对全书进行了概括性的介绍，结合本人的经历发表了对国内量化交易市场的看法。

第 2 章　数据处理。详细介绍了国内商品期货分笔数据的数据结构、获取的方式、处理的方法等，以及如何从分笔数据合成 K 线数据，如何提取主力合约，如何编写更高效率的处理程序等。其中包括 R 与 C++ 相结合的编程方式，如何在 R 里面编译及调用 C++ 程序，如何使用多核并行计算等，而且有详细的 R 与 C++ 代码，为以后的建模做准备。

第 3 章　预测因子。任何模型本质上都是因子的组合方式。机器学习模型又被称为统计预测模型，因此里面用到的因子自然被称为预测因子，当然也有人称为特征因子。本章介绍了构造因子的方法，给出了一些常用的因子，并且还给出了测试因子的基本方法。这里的因子既有基于 K 线信息的因子，也有基于分笔数据的高频因子，方便各种策略使用。

第 4 章　基础统计模型。本章在第 3 章的基础上，运用一些经典的统计模型进行预测分析，并且使用了训练集、验证集和测试集的概念，严谨建模。本章使用的模型以线性模型为主，因为对于绝大多数情况，采用线性模型已经足够了。在模型测评方面，采用样本外的 R^2 作为主要依据，这种方法跟样本内的 R^2 和调整后的平方都不一样。

第 5 章　复杂统计模型与机器学习。本章讨论了更为复杂的统计模型，一般也被称为机器学习模型，包括决策树、随机森林、神经网络、深度学习等，并且对比了不同模型之间的表现。由于金融数据的高噪声、高维度特征，因此复杂的模型很多时候未必会比简单的模型更好。在中低频交易中，如果条件允许，花更多精力收集信息或许更为有效。

第 6 章　从预测到交易。有了预测模型之后，还要落实到交易才有意义。把预测结果转成交易信号有很多方法，本章会进行比较。当然还要结合品种的买卖价差和手续费，以及交易的频率等。对于股票配置型的策略和期货择时型的策略，会有不一样的处理方法。

第 7 章　策略模型深化。本章是对前面几章的总结和提炼，主要是在结果一致的情况下探讨一些提高计算速度的方法，从而提高研究效率。很多时候，量化研究过程需要很多次的搜索、迭代等运算，这会消耗大量时间。如果能提高计算速度，那么就可以大大提高研究的效率。事实上，最近神经网络、深度学习等方法重新流行起来，更多是依靠 GPU 等计算技术的发展。

第 8 章　投资组合优化。有了交易信号和资金曲线之后，下一步就是对各

个策略、各个品种的投资组合进行优化工作。事实上，这部分工作也可以用量化模型来完成。与之前的统计预测、机器学习不同，这部分更多是传统的运筹优化方面的模型，如均值－方差模型、Black-Litterman 模型等。本章对比不同的投资组合优化的方法，并给出测评的结果及相关的程序。

第 9 章　投资组合优化深入研究。本章主要介绍了风险评价策略和增强学习（近似动态规划）等在投资组合里面的应用。其中近似动态规划属于动态投资组合优化的内容。另外，本章也介绍了策略的滚动优化和动态调整，对比了滚动优化和全局最优化的结果。事实上，如果处理得当，滚动优化可以取得比全局最优更好的效果。

第 10 章　C++ 实现策略。本章主要介绍了如何把 R 语言转换成 C++，从而实现自动交易。本章还介绍了 CTP 接口的基本原理，以及转换策略的基本步骤，包括处理行情数据、K 线数据、计算指标、计算仓位、合并策略等。本章主要是基于 Linux 的 C++ 编程，系统默认是 Ubuntu 16.04 LTS，读者掌握后就能自主编写全自动交易程序了。

第 11 章　实盘交易管理。上一章介绍了用 C++ 实现实盘交易的程序。事实上交易过程中其实会遇到各种各样的问题，特别是自己用 C++ 写程序，各类错误都要自己调试改正。本章系统介绍实盘交易会遇到的各种问题，并给出相应的解决方案。

第 12 章　套利交易。前面介绍的都是关于投机型趋势策略。因为没有做空的限制，所以商品期货从本质上都可以交易。本章就专门讨论套利类的策略，先从最基本的跨期套利开始，然后再简单介绍一些跨品种套利。

第 13 章　求职与工作。前面章节讲的都是量化交易研究与技术方面的问题。现实中，我们技术人员还要去工作，无论是担任期货的资管还是从事私募证券投资基金。因此，在学习了前面的知识之后，本章有关求职与工作相关的话题，作者有些经验可以和读者们分享。

本书读者对象

- 数学、统计、信息与计算科学、计算机、金融工程等专业本科生、研究生
- 高校数理类和经管类教师和科研人员

- 证券、期货、私募证券、公募基金等量化交易相关从业人员
- 对量化交易感兴趣的各行各业人士
- 对机器学习在金融方面运用的相关人士
- 人工智能方面想从事量化交易的相关人士
- 学习R语言或C++的相关人士
- 其他对中国期货市场量化交易感兴趣的人

感　谢

本书写作过程中，本人的妻子张妮洁女士正处于怀孕中，还要不断给本人写作提供各种各样的帮助。在此，本人表示对妻子衷心的祝愿。希望我的妻子能一切顺利，同时希望我们的宝宝能平安出生，成为新一代的“baby quant”。对于本书的更新和未来进展，以及相关程序、数据资料的获取，本人会在知乎“baby quant 谈量化金融”专栏里发布，敬请各位读者留意。

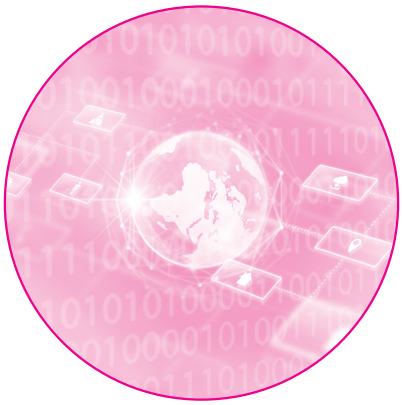
2018 年 1 月

baby quant

目录

第一章

期货基本策略概要



本章主要介绍期货量化交易策略的基本类型及发展的历程。目前国内的期货品种已经有数十个，其中金融期货包括股指期货和国债期货。商品期货则覆盖农产品、能源化工、有色金属、黑色金属、贵金属等各个板块。曾经股指期货的交易额占全部期货品种的90%，但2015年股指交易受限制之后，股指交易量下降了99%。目前的期货市场由商品主导，特别是螺纹钢、铁矿石等黑色系商品。下面分别介绍各品种对应的策略。

1.1 股指日内策略

在 2012-2015 年，最受欢迎的期货量化策略是股指日内策略。国内有很多期货程序化交易平台，如 TB 开拓者、金字塔交易系统等，很多本土的量化交易团队都会使用这些平台。这些平台的优点主要是回测、优化、模拟、交易都是同一个程序，不需要修改，而且平台会自动维护数据库，另外只需要支付很低的费用（如金字塔当时是 1800 元 / 年）就可以实现自动交易。以至于很多初创公司都会使用这类平台。

1.1.1 日内策略简单介绍

这类平台的期货历史数据主要是连续合约和指数合约。所谓连续合约，比如螺纹钢现在成交量最大的是 rb1305，那么连续合约对应的数据就是 rb1305 的数据，如果过了一段时间螺纹钢成交量切换到 rb1310，那么连续合约的数据就是 rb1310。所谓指数合约，指的是该品种上市的所有合约的加权合约。因此，实际上并不存在指数合约这个交易标的。之所以使用指数合约，是因为商品策略一般是隔夜策略，而连续合约在换月上会有较大跳空，回测的时候不准确，因此使用指数合约可以缓解跳空带来的影响。

我们可以看一个连续合约跳空的例子，如图 1-1 所示。

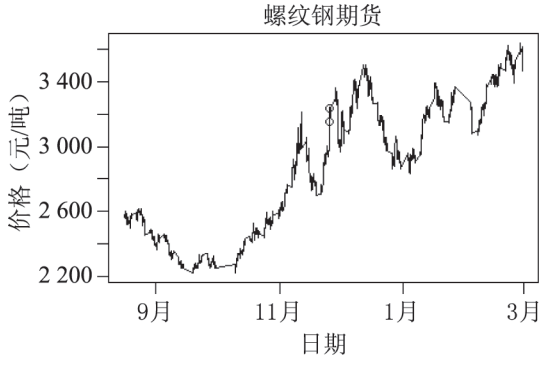


图 1-1 螺纹钢连续合约

图 1-1 中价格曲线的两个圆圈表示合约换月的日子，可以看到合约价格有较大的跳空。如果回测的时候没有注意到，就很可能捕捉到这个虚假盈利。如果要更精确地进行回测，则需要在换月之前平仓。第三方平台要将此写入程序里，不是那么的方便，如果用 R 语言等更通用的统计分析语言则方便得多。

有些人可能会问，现在很多微信号和淘宝店铺都有销售基于第三方平台的程序化交易策略，售价几十元到几百元不等，资金曲线也挺好看，既然如此，为何还要使用这么复杂的基于机器学习的量化交易呢？

事实上，很多上述策略设置了极低的手续费，比如万分之一，而且没有设置滑点，只能用最新价交易，这对商品来说每交易一次其实还能获得不少便宜。另外它们使用的是指数合约，对跳空没做专门处理，很可能策略捕捉的是不存在的行情。

正是因为使用第三方平台写隔夜策略有这些麻烦，所以很多人喜欢做日内策略。日内策略由于强制性日内平仓，因此就不会有赚取换月虚假跳空获利的情况。然而，国内上市期货品种虽然大约有 50 个，但绝大多数不适合日内策略。日内平仓可以看作一种风险控制手段，规避了隔夜的风险，放弃了潜在的利润，也避免了潜在的损失。然而，要规避这种风险是需要付出代价的。每天的平仓操作就是代价：一来需要支付手续费；二来需要牺牲滑点。因为平仓操作是必须成交的，所以一般会主动成交。

说到成交方式，一般有两种：一种称为被动成交；另一种称为主动成交。一般来说，如果需要立即成交，则会采取主动成交的方式，比如市价成交，或者加入很大滑点的限价成交。对于被动成交，则一般都是限价成交，比如说要买入，一般是当前的买 1 价，也可以是比买 1 价低的价。对于流动性比较差的品种，买卖价差非常大，被动成交可能是买 1 与卖 1 之间的某个价位挂单。

如果是第三方平台的程序化交易，一般都是趋势型策略，不会涉及频繁的挂撤单操作，因此基本可以假设是市价成交。对于上海期货交易所这种没有市价指令的交易所，一般是加了 3 个价位的限价指令，它的目的是立即成交。

好了，下面我们更仔细地介绍股指日内策略。

1.1.2 曾经辉煌

前面说过第三方平台非常适合股指日内策略。由于程序化交易、私募基金等是在 2012 年开始大规模兴起的，所以当时规模都还很小，几百万到几千万的水平，而股指日内策略的容量一般也是这么大。一个比较好的股指日内策略其实交易频率并不高，一般 1 周 1 次，即 5 天 1 次，如果一个团队储备了 50 个比较好的策略，那么平均下来每天有 10 个策略触发，可以做 10 手。当时股指合约的价值为 50 ～ 100 万元，因此，10 个策略对应的容量是几千万元。日内策略的收益回撤比较隔夜策略好，如果每个策略的收益 / 最大回撤有 1 ～ 2 倍，50 个策略合起来应该可以达到 3 ～ 5 倍，这已经是非常好的策略组合了。

如图 1-2 所示，这是一个股指日内策略组合的效果图。

图 1-2 股指日内策略组合

这是 5 个策略的组合，每个策略配 50 万元资金，初始资金 250 万元，总收益 66.83%，最大回撤 5.38%，收益回撤比超过 10 倍。由此可以看出，策略在 2013 年 8—12 月持续振荡，因为在经历大跌之后的那个时期股指每天均窄

幅波动，真正获利多的是 2014 年年底开始的大牛市，一直持续了很长时间。一般股指日内策略回测标准是最新价加合约价值万分之三的成本。

2015 年 8 月开始股指实盘，其中 2015 年 10 月之后持续稳定，实盘结果如图 1-3 所示。

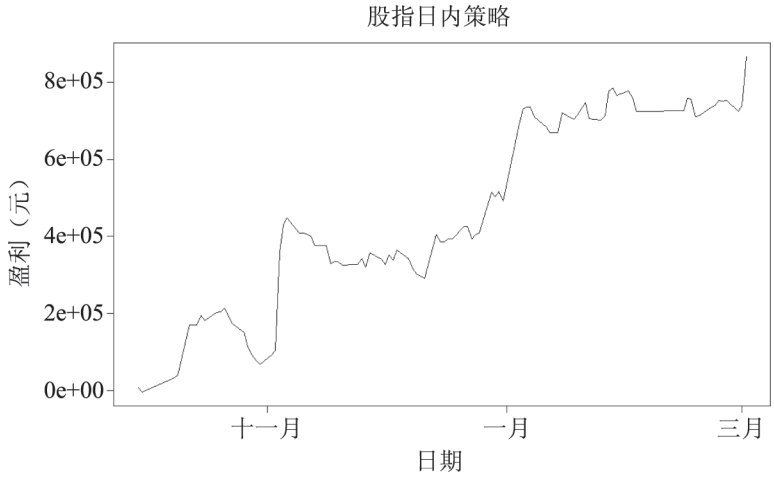


图 1-3　股指日内策略实盘交易

2015 年 10 月—2016 年 3 月：夏普比 3 倍，累计收益 / 最大回撤 5.68 倍。应该说这段时间的股指日内表现都还是很不错，主要原因是在股指受到限制后很多厉害的量化团队放弃了这块业务，因此市场有效性降低，盈利变得更加容易。

1.1.3　近期发展

然而，2015 年 8 月，证监会已经对股指进行了开仓限制，每天只能开仓 10 手，之后买卖价差变大，流动性变低，使股指日内策略的盈利难度变大。其实 2015 年 8 月—2016 年 3 月还是可以的，毕竟当时的波动还很大，基本上还可以每个月都能盈利，但 2016 年 1 月经历了熔断，之后波动开始下降，2016 年 3 月之后股指日内盈利就比较困难了，于是很多人采取了折中的办法，比如日内收盘不再硬性平仓，毕竟日内的波动已经无法覆盖平仓的成本了。

2015 年 8 月之后日内平仓手续费增加了 100 倍，要规避这个限制，只能采取锁仓的方式。国内的交易方式跟国外不大一样，国内有“开仓”“平仓”

的概念。比如现在没有仓位的话，买入操作其实是买入开仓，给交易所增加新的持仓量；如果现在已经有了仓位，比如现在是空仓，那么可以选择买入平仓，相当于为交易所减少当前的持仓量。然而，即使现在已经有了空仓，也可以选择买入开仓，这样就规避了日内平仓高额的手续费，只需要在第二天再买入平仓和卖出平仓，即可平掉仓位，实现日内交易。

一开始交易所对保证金的措施是“单向较大金额”，即买入开仓和卖出开仓都要占用保证金，但只收取更大的那一边。后来交易所为了抑制日内交易，取消了这一规定，而且把保证金调整至 40%，双向就是 80%，几乎没有杠杆了，这进一步降低了日内交易的吸引力。

2017 年开始了大盘股的牛市，俗称“漂亮 50”和“悲惨 3000”，或者说一九行情。即大盘股开启了持续上涨的趋势行情，这也给股指日内交易提供了机会。很多在 2016 年长期不盈利的策略在 2017 年也开始盈利。随着股指限制从 10 手到 20 手，保证金比例也相应下调，相信未来会越来越好。

1.2 商品趋势策略

商品趋势策略是最传统的量化交易策略，一般人们称为程序化交易策略。下面简单介绍一下。

1.2.1 程序化交易的挚爱

传统的程序化交易者都是交易商品起家的，因为股指期货 2010 年才上市。例如，TB 开拓者陈剑灵就是做商品趋势策略起家，然后反过来收购了 TB 开拓者，这种商品趋势策略又统称为 CTA 策略。

一开始的 CTA 策略一般基于日线指标，因为商品波动太低，持仓时间需要比较长，另外第三方平台处理数据、优化策略的速度实在太慢，很难处理更高频率的数据，因为有数十个品种，如果每个品种每个策略都优化一下，再放到几十个品种重复做一遍，计算量会非常大。因此，CTA 策略一般以日线为主，每天收盘前下单。

其实现在很多提及的机器学习、现代统计模型，实际上他们的建模过程比传统的基于灵感和规则的程序化交易模型死板很多，也正因为建模分析过程很死板，因此容易规模化，也更方便严谨测试。实际上，很多灵活处理的交易策略未必能很好地转成机器学习的模型。但从大规模生产和高速计算的角度来看，机器学习优势明显，此内容后面会提到。

下面看一下商品日线策略的例子，如图 1-4 所示。

名称	利润率	年回报	胜率	交易数	成功率	最大回撤...	MAR比率
全部综合	433.31%	22.31%	48.28%	116	13.79%	11.22%	1.99
S 螺纹钢指数	433.31%	22.31%	48.28%	116	13.79%	11.22%	1.99

图 1-4　螺纹钢日线策略

从 2009 年 3 月 27 日螺纹钢期货上市开始，至 2017 年 7 月 18 日，单手螺纹钢初始资金 3 万，由此可见年回报达到 22.31%，累计回报 433.31%，最大回撤 11.22%，交易次数只有 116 次，相当于每年 15 次左右。交易成本是交易所手续费加两个滑点，但由于交易次数非常少，因此加多一些滑点也不会差很多。

一般商品的买卖价差比较大，比如螺纹钢，买卖价差一般占合约价值的 0.3% ～ 0.6%，而手续费只有万分之一左右，因此对于商品低频交易，成本更多指的是买卖价差，而不是手续费。

还有一项叫作 MAR 比率，它实际上是年回报与最大回撤的比值。由于最大回撤一直增加，而年回报一般来说每年会有波动，但整体来说不会随着时间的增加而增加，因此 MAR 比率对时间长的资金曲线不利。比如一些国际知名的 CTA 产品，它的 MAR 或许只有 1 ： 1，因为它成立了 20 多年，总有回撤

大的时候。因此，利润率与最大回撤的比值更为合理。CTA 策略的利润率 / 最大回撤约 38 倍，是非常好的策略了。

这个策略是 2014 年 8 月 10 日前做的，之后没有修改，如果看完全样本外的情况，那么近 3 年表现，如图 1-5 所示。

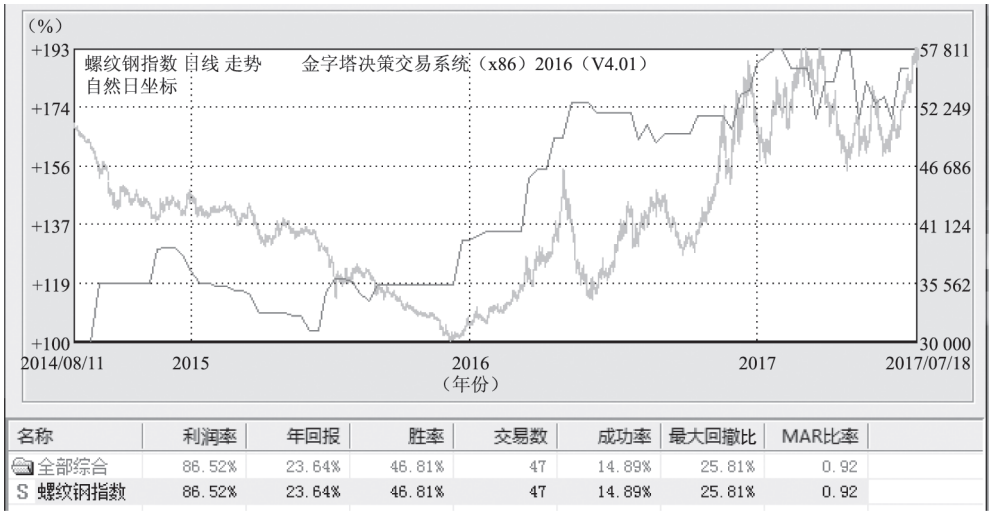


图 1-5　螺纹钢日线策略回测

从图 1-5 可以看出年回报没有多少改变，利润率 / 最大回撤也有 3.35，应该说对单策略品种来说是不错的。其实最大回撤的计算也按实际发生时的曲线值作为分母，如果最大回撤发生时间较晚，此时曲线值已经增长比较高了，计算出的结果就比较小。

可以再看看样本外测试的一些指标，如图 1-6 所示。

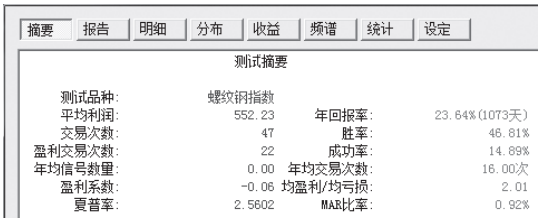


图 1-6　螺纹钢日线策略统计

胜率是 46.81%，盈亏比是 2.01，一般趋势策略的胜率都不会很高，40% ～ 50% 是比较理想的，而盈亏比会比较高，这里也有 2 倍以上。另外夏

普率 2.56 也是比较高的，一般来说 CTA 策略的夏普比都不会很高，1 ～ 2 倍比较常见。螺纹钢是目前国内最活跃的商品期货，因此一个 CTA 组合能否扩大盈利，很大程度上取决于它在这种活跃品种上的盈利程度。有些策略难以扩大容量是因为它盈利的都是小品种，而大品种亏钱。

1.2.2　2016 年的辉煌

2015 年股灾之后，很多股民选择把资金转向商品市场，并且由于股指限仓，把原来交易股票的资金也转去交易商品。基于股民炒股票只能做多不能做空的特点，这些股民进入商品市场后也维持了只做多不做空的风格，导致一时间多空失衡，因而引起了 2016 年商品市场的暴涨。很多原来做商品的大户，包括现货商和投机大户，他们从基本面角度出发，维系着原来偏空的思路，随着行情的上涨不断做空，因此出现了散户赚钱大户亏钱的景象。由于散户的进入，商品单边趋势行情明显，因此 CTA 基金表现良好，很多第三方评价机构称商品 CTA 策略是 2016 年的最佳策略。智道金服研究院统计数据显示，CTA 等权平均收益为 9.40%，在所有大类资产配置策略中排名第一。

2016 年虽然全年表现良好，但中间也出现了一些波折。比如 7 月、8 月、9 月很多 CTA 出现了回撤，10 月、11 月的大涨又恢复了过来。2016 年 11 月 11 日星期五晚上，商品期货价格全面大跌，大多数基金产品净值出现了大幅度的回撤。由于此前的行情很好，大家仓位都比较重，因此这次回撤带来的损伤也是非常严重的。此后的 11 月、12 月大多数商品趋势策略没有太好的表现。

1.2.3　2017 年的惨淡

很多基金产品从“11.11 大屠杀”开始到 2017 年 7 月中旬一直回撤，甚至看不到净值曲线有好转的迹象。少数基金产品由于 CTA 基金控制了仓位，在连续七八个月的时间里窄幅振荡，然后在 2017 年 6 月、7 月的大行情中重新创了新高，结束了长达 8 个月的平台期。

仔细分析这段时间，其实还是有很多值得思考的地方。比如著名的“11.11”行情，发生在周五晚上，商品集体跳水，原来持有多头的 CTA 策略肯定是亏钱的，

但如果反应及时，还是可以止损平仓出来，甚至反手做空，然后在 11 月 14 日即周一把周五晚上的亏损弥补回来。由于周五晚上和周一白天是同一个交易日，因此，如果处理得当是不会出现大幅亏损的。那段时间，商品日内策略和持仓 3 ～ 5 天的策略表现还是不错的，但持仓 7 ～ 10 天的长线策略则不尽如人意。

在接下来的 2017 年 1 月、2 月，持仓 3 ～ 5 天的策略则持续亏损，但一些更长期持仓的策略则表现良好，不少持仓 1 个月以上的策略不受短期波动的影响，在 1 月、2 月表现良好。这类基金一般容量很大，达 30 亿以上。日内交易的商品，表现也不错。事实上，商品日内策略在 2016 年之后才有比较好的表现，因为之前商品波动太小，日内波动不足以覆盖手续费，所以持续亏钱的。高频和日内策略都与波动性高度相关。

一般持仓 1 个月以上的策略都要基于基本面的数据，国内目前存在的问题是基本面数据缺乏有效的来源，数据质量也很差，并且很难保证每天准时更新数据。万德对每个品种有上中下游三类指标，每类指标都有数百个，因此每个品种可以有上千个基本面数据，除去月度、季度、年度的数据，日频数据也有数百个。如何从数百个指标中找到有预测力的指标属于统计学问题，毕竟日线的样本不会很多。而且每个指标也可以通过一些公式衍生出不同的技术指标，总体来说预测因子的数目 p 特别大，而样本数量 n 却很小，毕竟每年才 240 多天，因此，这属于 $p>>n$ 的问题，属于高维统计（high-dimensional statistics）问题，应该用一些稀疏性（sparsity）的模型来求解，如 lasso 模型，而不是那些神经网络、深度学习等复杂的模型。

然而，3 ～ 6 月，持仓 3 ～ 5 天的策略逐渐变好，而持仓 1 个月以上的策略却损失惨重。这或许是因为持仓时间过长的策略被频繁止损出局，由于这段时间的总体趋势并不明显，无论多空都有可能亏损严重触及止损，而一般趋势策略让盈利充分奔跑，不会有强制的止盈措施，因此对长线持仓策略不利。

值得庆幸的是，2017 年 6 月、7 月商品又开启了新一轮的大涨行情，商品 CTA 结束了长期的徘徊期，不少产品也创了历史新高，开始迎接新的征程。

CTA 作为一种策略类别，其本身或许会有很长的平台期、回撤期，但由于它跟股票、债券、货币等其他投资品种相关性很低，因此，作为资产配置的一种手段，分散风险是非常必要的。对于个人投资者而言，期货的高杠杆型本质上是零息贷款，并且杠杆可达 5 ～ 10 倍，比如个人可以放 20 ～ 30 万元作为

保证金进行全自动交易，每年预期收益 100%，最大回撤 30% ～ 50%，10 年获得上千倍的回报，也不过二三亿元，即使打个折扣，10 年后 100 倍，也有两三千万，相当于合约价值 1 个亿，对期货市场影响非常小，完全在策略容量允许的范围内。第二个 10 年，从两三千万元起家，如果是 20 倍的收益，也有 5 亿元左右的收入。第三个 10 年，从 5 亿元起家，如果是 10 倍收益，则是 50 亿元的身家。30 年时间，进可得 50 亿元，退也不过损失 20 万～ 30 万元，其实是值得考虑的。CTA 策略 1980—2010 年这 30 年的表现与标准普尔 500 指数的对比，如图 1-7 所示。

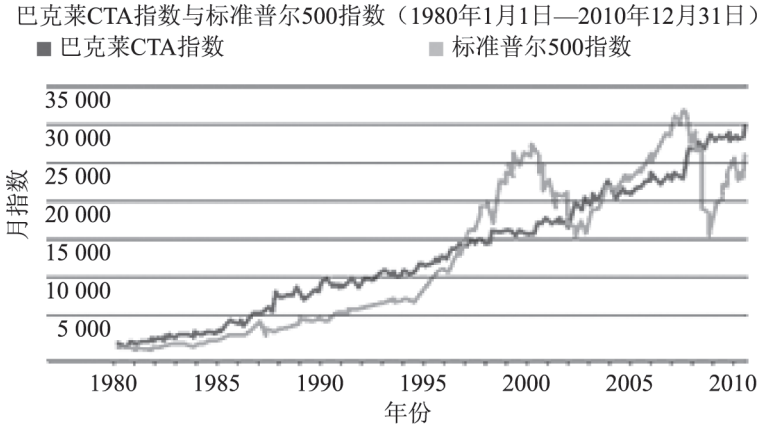


图 1-7　巴克莱 CTA 指数与标准普尔 500 指数

可见，从 30 年的时间段来看，相比于标准普尔 500 指数来说，CTA 指数的收益增长要稳定得多，表现为收益更高，波动更小，回撤也不大。当然，CTA 策略的总容量只有 3 千亿美元，而标准普尔 500 的总市值高达几十万亿美元，因此 CTA 更适合小众市场，对个人投资者和小型机构投资者来说足够了。

1.2.4　在公募机构做商品交易

一般来说，做商品 CTA 的大多以初创私募团队和个人散户为主，因为商品期货具有高杠杆特征，可以获得较高的收益。对于小团队来说，募集资金比较困难，只能依靠高收益来吸引客户，比如 15% ～ 20% 的年化收益对客户而言才有吸引力。如果是 10% ～ 15% 的收益，股票量化对冲就可以实现，没必

要做高杠杆的期货，很多大机构都在做，小私募没有竞争力。

但是，对于商品 CTA，公募基金很多都没有涉足。主要是合规、风控方面的原因所导致。公募基金即使有专户，也是偏向于频率很低的股票对冲策略，因为公募基金自己的公募部门就有股票，对股票比较熟悉，对股票的风控也比较有经验。但是对于商品，很多有夜盘。传统的风控系统要加入夜盘模块的话需要额外增加一二百万元，加入商品本身也需要一二百万元，如果是国债期货可能又需要额外的一二百万元。有人会说公募基金每年收入几十亿元，加入几百万元的成本似乎并不算什么。但每块业务单独核算，对于商品 CTA 而言，公募在募集资金方面跟私募是一样的，并没有优势。因此，想要覆盖数百万元的成本，必须有几个亿的管理规模，这对于一开始成立商品的团队来说是非常困难的。另外，公募对全自动交易非常抵触，只允许出了信号后由交易员手动下单，这对期货而言可以说是噩梦，毕竟波动大起来远超股票，账户一多更是难以处理。很多在公募做 CTA 策略的都苦不堪言。另外，公募在量化人才方面储备严重不足，在 IT 方面投入更加滞后，这些均远远比不上私募。因此，如果想在 CTA 方面发展，最好的选择是去私募基金。

1.3 高频交易策略

高频交易策略给人的感觉是交易频率非常高的策略。但有一个疑问就是如果手续费很高，但波动很小，不足以覆盖手续费，那么该如何实现高频呢？在美国股票市场，被动成交是不需要交手续费的，而且还有返佣，因此成交的越多赚得越多，另外被动成交还能赚取买卖价差。因此，在美国股票市场，一些大盘股成交非常活跃，每天数据量达到几个 G，确实可以出现每分每秒都在成交，数据精度基本要达到微秒级才能分析出来，而且做市商每次成交既不用交手续费，又不会损失滑点，甚至两者都有赚头，因此确实可以实现高频交易高频赚钱。

当然，有时会出现单边趋势的行情，比如一边的单子成交了另一边却没成交。由于美国市场一只股票在多个交易所上市，每个券商自己都还有暗池，而价格的趋势变化不可能在各个地方同时发生，毕竟“同时”这个概念在高频中

需要细化，比如对普通人，同一天、同一小时、同一秒都可以看成同时，但对高频来说，可能毫秒级、微秒级的差异都可以区分开来。因此，大趋势来临，只是数十个交易所的某个交易所的该股票先变化，其他交易所的该股票还没来得及改变，高频交易商完全有时间撤单，甚至可以追踪趋势。

一般来说，跟多个交易所同时交易的策略属于套利策略，套利策略对速度要求比较高，如果能够在速度上获得优势，盈利的可能性大很多，而对单个品种进行交易的策略，更多属于趋势策略，只是趋势策略持仓时间较短，属于短趋势。此策略对趋势的预测非常重要，一般来说越短期的趋势越容易预测，但还是有一定概率的，哪怕速度再快如果做反了也没用，因此，趋势策略对速度的要求没有套利策略这么高。

1.3.1　人工炒手 vs 机器高频

国内有很多人工炒手的培训，我也有幸参加过一期，费用两万元左右，主要是了解人工炒手与机器程序化之间有什么区别。

对人工炒手来说，预测方向反而是最次要的，最重要的是止损、资金管理、心态调整这些。而对机器来说，是不需要调整心态的，因此这是两者之间一个很大的区别。

人工炒手不可能托管机房，哪怕在上海期货大厦里面交易也是如此，其实跟高频交易商比起来人工炒手在速度上也是没有任何优势的，另外人也有反应时间。在分析上，高频策略都是经过历史上几千几万次回测优化得出来的，而人工炒手哪怕复盘次数再多也远远比不上，更不要说分析能力了，计算机运算速度比人工快得多，因此，人工炒手靠以上这些难以取胜。

那么人工炒手有什么优势呢？首先，人工炒手可以盯着很多个屏幕，看各种各样的信息，而高频交易由于托管机房是没法获得这些信息的。即使说那种模仿人工炒手，不用托管机房的，在处理文字信息上，机器高频也没有优势，国内自然语言处理的能力还没那么强。其次，对于一些信息的发布，不管利多还是利空，人工炒手肯定早于机器知道，因此无论机器反应速度多快，他们都是被动型的，而人工炒手可以主动根据信息做交易。最后，在对交易节奏的把握上，人工炒手可以很容易调节，比如手续费高了就要降低频率，对此，每次

调整对机器高频都是噩梦，需要重新优化测试策略，如果时间太赶，测试不大严谨，就很可能出错。

在大商所，由于过去品种的价差比较大，排队挂单量很多，资金量大的人工炒手可以通过操纵盘口获利，此时以预测为主的机器高频则没办法用这类方法。这两种本质上是竞争关系。人工炒手希望可以利用资金优势诱导机器高频出错，从而获利；而机器高频也要识别出人工炒手制造的噪音，避免上当。这类博弈，人工炒手具有资金和灵活度的优势，变化多端，且监管层对此一直没有采取有效措施，而机器高频则会受到严格监管。

从美国的经验看，机器高频会越来越多，人工高频会越来越少，毕竟不只是交易这个行业，任何行业都有被机器取代的趋势。当然，很多人工炒手也在寻求转型，很多自营交易公司都是人工炒手创办，然后跟一些程序化背景的人合作。美国也一样。之前美国很多高频交易公司都是半自动的，比如人工交易量占 30%，机器占 70%，如著名的 Getco，也是高频做市商。但后来这类公司的发展遇到了瓶颈，逐步让位给全自动交易公司。

在美国，传统的交易员很多本科读商科出身，然后读了 MBA，或者一直做交易没读 MBA。他们更依赖都市生活方式（street smart），即使很多招聘要求说“数学好”，也只限于速算和简单的概率比较厉害，跟陶哲轩那种数学好完全是两回事，也跟最近兴起的机器学习、人工智能风马牛不相及。这类人工交易员也会把想法和策略让程序员写成程序，这种公司的量化（quant）一般跟程序员（developer）同属一个部门。而真正的量化交易一般是 quant 和交易员（trader）是同一类角色，因此管理分工上会很不一样，这也决定了传统公司转型会非常困难，因为传统的交易员已经占据了领导的岗位，或者公司就是他们开的，他们不可能把自己位置让出来给量化的人。因此做量化的人一般喜欢去更纯粹的量化交易公司，而不是从中途转型过来的。在中国香港那种“重商轻理轻工”的社会，这类现象会更为严重。

中国内地反而好很多，因为中国内地的传统交易员一般都是学历很低的个体户，入不了大公司的“法眼”，即使历史业绩再好，大的资管、公募、私募也不会招他们并委以重任，而这些人即使自己开了公司，真正高学历的人也不大愿意给他们打工，因为这类公司基本工资不高，名校毕业生很看重基本工资。因此，国内大的量化资管、量化私募极少见到传统交易员出身的人，很多是招

了名校硕士生来自己摸索和培养。参与人工炒手的也是个体户居多，因为在大公司工作很难抽出两个月时间去参加培训。因此，量化交易员的对手也并不是十分厉害的人，在零和博弈的市场中，大家要对自己有信心。

1.3.2　2016 年以前最暴利的策略

伊始顿是一家贸易公司（trading company），因为外国资本是不允许交易中国期货的，它必须想办法规避这个限制，于是便开了一家贸易公司，然后以贸易公司的名义交易期货。贸易公司本质上是交易现货的，按常理来说用期货对冲也无可非议，但是当期货的交易量远超现货，这就有点不大正常了。

伊始顿就是这么一家公司，老板是两个俄罗斯人，请了一些莫斯科大学数学系的学生，主要交易国内的股指期货。2012 年开始，他们的本金是 600 万，到 2015 年 7 月，积累到 20 亿，获利超过 300 倍，主要用的就是股指高频交易策略，因此，我称它为 2016 年以前最暴利的策略。

事实上，做股指高频的远不止伊始顿一家，最终对伊始顿的判词，也不是因为高频交易方面，而是其他原因。比如伊始顿跟期货公司技术部勾结，绕开了期货公司的风控系统。在中国不存在直连市场（Direct Market Access），所有人必须把单子报给期货公司，期货公司审核保证金等要求合格后，再报给交易所，交易所也只核查期货公司的保证金是否充足，不会核查到个人。如果跟期货公司技术部合作，绕开了期货公司这一端的核查，确实可以比市场上其他交易者快很多。而且，期货公司有很多闲置资金，伊始顿甚至可以利用这些闲置资金来获利，这样他动用的资金就不止几百万，而是几十亿。考虑到他的策略几乎能确定每天赚钱，因此风险是非常小的。

另外还有关于撤单方面的问题。2015 年之前监管股指套利编码账户并没有撤单限制，每天几万次也没问题，因此伊始顿不会有撤单方面的限制。

在高频速度方面，有两个速度非常重要：一个是机器到交易所的速度，另一个是机器内部的速度。机器到交易所的速度可以通过托管机房等优化提高，机器内部的速度可以通过 FPGA 提高行情速度，通过更高效的程序编写提高运算速度，通过对操纵系统的优化来提高系统运行速度。这些都是非常专业的计算机方面的内容，与本书关于量化交易的内容不大相关，只是简单介绍一些。

机器到交易所的时间一般是毫秒级别的，机器内部的时间一般是 7 ～ 10 微秒（最简单的策略），因此机器到交易所的传输时间是最为重要的，哪怕它的一点误差也远远超过机器内部的时间，因此有人说机器内部的时间不大重要。

正常人需要从本机传输到期货公司柜台，然后再从柜台到交易所，这有两段，每段都是毫秒级别的。如果伊始顿节省了其中一段，那么相当于比其他人节省了毫秒级别的时间，哪怕别人请全世界最厉害的系统优化大师，也只能省下几微秒，比起伊始顿的速度差距甚远。因此，大家不难明白为什么伊始顿是印钞机，也不难明白为什么就抓他而不抓其他人。伊始顿事件之后，监管当局开始对高频交易进行全面从严监管。

1.3.3 全面从严监管

自 2015 年 8 月起，对股指期货的一系列严格监管的规定相继出台，比如每次报单需要一块钱的费用，取消套利编码账户的无限撤单优惠，股指日内平仓手续费先是 10 倍，然后很快增加至 100 倍，股指保证金增加至 40%，很快锁仓保证金双向计算等；商品方面，对各开户人在不同期货公司开的账户进行撤单的合并计算，违反者停止交易一个月；另外对那些不以成交为目的的挂单也进行了限制，这主要打击了人工炒手操纵盘口，事实上对高频交易者影响不大。曾经有一段时间要求提交量化交易源代码，但后来暂缓实施。

股票方面的监管更为严厉，比如不允许新的程序化交易接入，旧的账户则没有限制。这实际上保护了已有账户的利益，不少股票日内回转策略在 2015 年 8 月之后大放异彩，正是因为缺乏竞争对手的缘故。按常理说，股票日内交易的难度远小于期货高频，毕竟股票日内一天最多也就是卖一次然后买回来，分析难度不大，而期货高频几百几千次的却很常见。但由于期货交易之前太暴利，看不上股票日内回转的收益，等期货没法做的时候又不能开股票的户，导致股票日内回转缺乏强有力的竞争对手，因此从业者获利丰厚，但整体而言其实他们水平并不高。比如很多日内回转都是人工工厂，即使有机器做的，从业者也大多不是海外回来的，而是本土成长的，跟期货高频几乎被海外巨头垄断的情景不一样。

2016 年年底以来，股票指数连续大涨，因此很多量化团队转去做指数增强，

比如沪深 300 指数增强、中证 500 指数增强等。由于指数本身就很强，一年涨幅为 20% ～ 25%，再增强就更厉害了，所以这类策略是比较受欢迎的，而且容量也很大。但是，这本质上并不是市场中性的策略，2015 年那种腰斩行情未来也很可能出现，加上很多量化产品宣传未必客观，而且很多依靠代销机构，就算客观，如果真的回撤 40%、50%，哪怕增强个 4%、5%，恐怕客户也难以接受。

这类策略运行时间还不是很长，还没见过大场面，所以也不能因为曲线好看而觉得他们有多厉害。并且最近深度学习、阿尔法狗概念持续发酵，不少人使用了深度学习模型，很多国外顶级的量化对冲基金几十年来都是以传统多因子模型为主，对复杂一些的模型持观望态度，非线性模型逻辑性比较差，导致赚钱不知道是怎么赚的，亏钱也不知道是怎么亏的。

在此期间，证监会高层也发生了大的变动，新任领导似乎并没有把工作重心放在程序化交易的监管上，程序化交易处于被遗忘的状态，这对从业者而言反而是好事。

1.4　本节介绍

本节主要介绍本书的编程语言及主要研究的策略，也跟其他主流的编程语言做了对比，并且介绍了本书要讲的策略类型，以及其他一些策略类型。

1.4.1　为什么使用 R 语言

很多人会问为什么使用 R 语言？因为现在国内最受欢迎的分析类语言是 Matlab，国际上 Python 越来越流行，而且介绍 Matlab 和 Python 的书喜欢用 R 语言作为反面例子。整体而言，一个语言是否适合做量化交易，有以下几个方面原因。

- 运算速度。其实这几个都是解释性语言，或许有一定的速度差异，但一般是10%～20%这个幅度，对于计算量大的处理，要么已经封装了软件包，里面是C++写的，要么自己可以写成C++编译，然后用分析语言

调用，整体速度就差不多。本书会介绍如何使用Rcpp来在R语言里面使用C++。

- 研究与交易结合。客观说这不能算是优势，因为绝大多数正规的量化团队还是会用C++把交易程序重新写一遍，Python在国内期货也需要调用C++的CTP接口才能交易，其实正规的团队不大喜欢用第三方的接口，因为有错误的话难以调试。
- 研究效率。其实这是R语言的优势，在可视化方面R语言应该是最好的，而且R语言很多package都是开源的，网上可以搜到代码，有不懂的可以上stackoverflow上面问，一般作者都会在24小时内回答，非常方便。
- 其他方面。国内使用Matlab多是因为可以免费试用，但国外Matlab是收费的，而R语言是免费的，且没有版权的问题。很多学术界的书籍都是基于R语言的，Python更多是复杂的机器学习模型上有优势，但统计学领域的时间序列、非参数统计等基本上只有R语言才会有对应的package。

基于上述的理由，加上本人在研究生阶段就是学习 R 语言为主，而且并没怎么使用深度学习模型，因此本书使用 R 语言作为研究的语言。

1.4.2 重点介绍中低频趋势

本书重点介绍的是中低频策略，基于 5 分钟 K 线，持仓三五天。如果是更高频率的策略，比如基于分笔数据的策略，数据量比较大，回测、优化时间较长，不适合初学者。如果是更低频的日线数据，最好结合基本面数据才能研究出好的策略，因此这里也没有涉及。基于 5 分钟 K 线的数据，一般不需要基本面数据，也不会对交易速度有很高的要求，各项都比较适中，而且容量也比较大，10 ～ 15 亿都没问题，适合中小型私募和个人投资者。

1.5 未来展望

在写作本书的时候我主要交易的是商品趋势策略，商品日内策略也刚刚完成，目前正在研究其他的一些策略，这些策略留在以后有机会再写。

1.5.1　跨期、跨品种套利策略

跟趋势策略相对应的是跨期、跨品种的套利策略。所谓“套利”，并不是指赚取无风险的利润，而是指做多一个品种的同时做空另一个品种，然后通过赚取价差的变化来盈利。因此，这种策略跟趋势策略有相似的地方，只不过从预测单品种的趋势变成预测品种价差的趋势，从买卖期货变成买卖价差。大商所有专门的组合合约，但这里我们并没有交易那种合约，而是自己构造合约。一般原品种的流动性会比组合合约好很多。

构造价差合约的K线时也有需要注意的地方，比如两个合约的时间戳（time stamp）不一定完全对应，在计算价差时要先把时间对齐了才能计算。不同交易所的时间也不一样，因此跨品种的时候最好是同一个交易所的，当然跨期处理可以更容易一些。另外如果是合成5分钟K线，计算高低价格的时候，也需要从分笔数据的价差合约来计算，不能用已经合成的各个合约的5分钟数据相减计算，因为各自的最高价相减是没有什么意义的，跟价差的最高价没有关系。

构造好价差的5分钟K线后，还有一个需要注意的问题是价差可能是负数，负数不存在对数，没法计算对数收益，因此只能用实际价格的变化作为因变量。另外价差合约的成交量也需要处理，可以是两个合约成交量较少的那个，因为整体价差合约的流动性由较小的那个决定。另外价差合约的持仓量似乎很难定义，因此研究策略的时候可以忽略跟持仓量有关的指标。

跨期、跨品种套利的思路大致如此，本书不做深入展开，以后有机会再写。

1.5.2　商品市场中性策略

另外一种策略叫作商品市场中性策略，也有人称为宏观对冲策略。套利策略一般是买一个卖一个进行配对交易，而市场中性策略则只需要保持多空市值相等即可，然后每隔一段时间调整一次。这属于配置型策略，而不是择时策略。一般来说基于机器学习预测模型的策略都是择时策略，而这种配置型的策略由于时刻保持多空仓位平衡，比如多空各一个亿的合约价值，因此并不是择时的。

有一个可行的办法是选一些技术指标，然后每个技术指标都可以作为

排序的依据，各个品种按强弱排序，强的一半做多，弱的一半做空。如果市场出现整体上涨或下跌，其实对组合是没有太大影响的，因此这也是“市场中性”一词的含义。然后再把各个因子得到的多空组合汇总起来，得到整体的组合。

另外一种方法也可以用回归的方法来做，此时因变量不再是某个品种自身的价格变化，而是该品种相对于市场整体的价格变化。至于市场整体的价格变化如何加权计算，即商品指数如何加权计算，也是一个复杂的问题。无论如何，相对于指数的价格变化，肯定一部分是正的，一部分是负的，只需每次做多正的做空负的即可。调仓频率可以比较低，比如一个星期乃至一个月调整一次，预测的频率也比较低，触发交易的阈值不必敏感。

国内一些基于基本面的商品私募基金就是这么做的，比如在 2016 年年底做多黑色系做空农产品，虽然农产品亏了一点钱，但黑色系的收入远远超过农产品亏的钱，总体而言表现也不错。

1.5.3 基于基本面数据的策略

之前的数据都是基于行情数据的，这对于比较短线的策略来说还是可以的，但对于持仓时间比较长，比如一个月以上的策略，或许就不大合适了。为此，可以考虑基于基本面数据的策略。

基本面数据的获得是一个问题。每个品种每个版块在网上都可以找到相关的基本面数据，但过于分散，可以借助万德等第三方平台。比如万德里面每个商品都有几百甚至几千个基本面指标，频率从日到年都有。考虑到国内商品上市时间很多都不长，用日频数据比较好，这样也有数百个指标。

然后可以构建简单的线性回归模型，利用这些因子来预测价格未来的变化，由于因子数目较多，可以运用 lasso 等稀疏模型来求解，筛选其中重要的因子。当然，最好有基本面方面的知识，防止选择很多意义不大但恰好很有预测力的因子，这是由于因子数目太多因此选择了噪音部分。

选择好基本面因子，之后的交易过程就跟基于 5 分钟 K 线的策略差不多了。对冲策略也可以用基本面因子来建模。

1.5.4　算法交易

算法交易是量化交易的另外一块内容，一般指把一个大单分拆下单从而减少交易成本。比如一个 1 万手的买单，由于市场上买一价挂单一般只有几百手，而且超出 5 档的挂单会非常稀薄，因此 1 万手一次性下单可能会把价格推到涨停，俗称“乌龙指”，产生巨大的买入成本，这是流动性风险。而如果把单子分拆得很细的话，比如每次只下 1 手，需要一万次，假设每 1 秒下一次，也需要近 3 个小时，可能价格变化也非常厉害了，因此这有价格波动的风险；那么，如何在流动性风险与价格波动之间实现平衡也是一个问题。另外，如果下单过于有规律，会被市场察觉，这会带来逆向选择的风险，总的来说整套体系也是挺复杂的。

一般衡量执行效果的指标有成交量加权平均价（Volume Weighted Average Price，VWAP），即这段时间交易 1 万手买单的平均价格跟这段时间市场的 VWAP 对比，如果低于 VWAP 则是比较好的。事实上，如果买入的时候一直是对手价抢单，成交量跟当前成交量相当，那么最终的平均价会比 VWAP 略高一些。但如果买入的时候一直挂单，虽然成交价格更优，但也存在不能成交的风险。因此，如何用挂单与抢单结合的方法来降低平均成交价，也是一门学问。

国外这方面技术在股票用得比较多，比如把要交易的量分到一天内的各个时段来成交，每分钟成交多少，每秒钟成交多少都有对应的任务，时间快到了就抢单，时间没到就挂单等待，然后挂单成交的概率也需要利用实际成交情况来估计。

一般规模比较大的私募需要这些方法，规模小的私募还是以预测模型为主。

1.6　本章小结

本章抛砖引玉，介绍了中国市场量化交易的一些基本情况，也介绍了几种常用的策略。由于篇幅有限，本书重点介绍期货中低频策略，从最基础的地方开始讲起。

第二章

数 据 处 理

本章主要介绍期货数据的获得与处理。由于最常用的程序化交易接口CTP接到的行情一般是一档的500毫秒截面数据，因此这里主要讨论的也是这种数据。我们从这种数据出发，构造出5分钟的K线数据，以后的章节会在这种5分钟K线数据上建模。事实上，国内商品期货的波动相对于交易成本（买卖价差和手续费）而言并不大，太高频的分笔数据很难分析出长期有效的短趋势策略，即使有也是高度依赖挂单被动成交。但如果使用5分钟数据的话，可以很容易分析出较长期的策略，波动远远超过交易成本，反而可以有比较好的策略。

2.1　期货分笔数据

国内商品期货有 40 多种，金融期货有 5 种，目前（2017 年 10 月）由于股指期货受到限制，现在最活跃的大都是商品期货，特别是黑色系的商品期货，比如螺纹钢、铁矿石等。为统一讨论，本书一般以国内成交量最大的螺纹钢期货作为例子来研究。

螺纹钢在上海期货交易所交易，代码为 rb，2009 年上市至今已经超过 8 年。螺纹钢价格在 1 500 ～ 5 000 元波动，近期一般在 3 000 ～ 4 000 元，最小买卖价差是 1，每手是 10 吨，因此合约价值大约是 3 万～ 4 万元每手。

这里给出 rb1710 在交易日为 2017 年 7 月 21 日的数据，如图 2-1 所示。

时间	最新	持仓	增仓	成交额	成交量	买一价	卖一价	买一量	卖一量	
2017-07-20 18:30:26.500	3496	3954248	3954248	0	0	0	0	0	0	
2017-07-20 20:59:00.500	3510	3950524	-3724	291470400	8304	3509	3510	5	134	
2017-07-20 21:00:00.500	3507	3949310	-1214	117806300	3358	3505	3507	106	55	
2017-07-20 21:00:01.000	3508	3948646	-664	80454200	2294	3508	3509	31	544	
2017-07-20 21:00:01.500	3507	3948962	316	109051660	3110	3506	3507	19	70	
2017-07-20 21:00:02.000	3506	3949594	632	47328520	1350	3504	3506	378	919	
2017-07-20 21:00:02.500	3505	3949598	4	27125660	774	3504	3505	247	79	
2017-07-20 21:00:03.000	3505	3950170	572	54952600	1568	3505	3506	133	3286	
2017-07-20 21:00:03.500	3505	3950012	-158	32595820	930	3504	3505	132	152	
2017-07-20 21:00:04.000	3504	3950032	20	29158040	832	3503	3504	390	368	

图 2-1　螺纹分笔数据

可以看出，虽然说交易日是 7 月 21 日，但它实际上是从 7 月 20 日晚上开始。一开始的 18:30 的价格可以看成是前一天的结算价，然后 20:59:00 的价格为集合竞价的价格，然后 21:00:00 之后的价格才是真正开始交易的价格。主要的信息包括：

时间——即那笔行情发布的时间，虽精确到 500 毫秒，但不一定每个 500 毫秒都有，存在更新信息的时候才会有；

最新——即最新的成交价，如果在非交易时间的这个价格，但它是前一天的结算价；

持仓——当前该合约的市场总持仓量，商品是买卖双倍计算；

增仓——当前时间新增加的持仓量，商品是买卖双倍计算；

成交额——当前时间的成交金额，即成交量乘以合约乘数，商品是买卖双倍计算；

成交量——当前时间的成交合约数量，商品是买卖双倍计算；

买一价——最高的买价；

卖一价——最低点卖价；

买一量——在买一价上的挂单量；

卖一量——在卖一价上的挂单量；

第一行的很多数字是零，那是因为它发布的是结算价和前一天的成交量，买价卖价等信息不存在，所以都是零。

螺纹钢有夜盘，因此集合竞价发生在夜盘，白天开始的时候是不存在集合竞价的，如图 2-2 所示。

时间	最新	持仓	增仓	成交额	成交量	买一价	卖一价	买一量	卖一量
2017-07-20 23:00:00.000	3492	3871238	-70	6773700	194	3492	3493	5	187
2017-07-20 23:00:00.500	3492	3871238	0	0	0	3492	3493	5	187
2017-07-21 09:00:00.500	3479	3879074	7836	906039440	26022	3479	3480	20	1063
2017-07-21 09:00:01.000	3478	3875176	-3898	345417900	9932	3476	3478	3	283
2017-07-21 09:00:01.500	3479	3870682	-4494	348284080	10016	3478	3479	140	547
2017-07-21 09:00:02.000	3476	3869180	-1502	350905700	10094	3476	3478	22	739
2017-07-21 09:00:02.500	3477	3869396	216	130298500	3748	3477	3478	49	193
2017-07-21 09:00:03.000	3478	3870578	1182	100202880	2882	3478	3479	113	275
2017-07-21 09:00:03.500	3475	3871468	890	75917980	2184	3474	3475	1	10
2017-07-21 09:00:04.000	3474	3872016	548	134043880	3858	3472	3473	272	1

图 2-2　螺纹钢白天价格

2.1.1　获得数据的途径

很多人想收集历史行情信息，其实有很多途径。比如有些人只需要 5 分钟数据，或许就可以从第三方的程序化交易平台上下载。但有几点需要注意。

（1）信息完整性。第三方平台的 5 分钟数据往往只有高开低收等 K 线信息，没有挂单量和买一、卖一等微观结构的信息，而这些信息对精准回测而言比较重要。

（2）构造复杂因子。如果有微观信息，可以利用 5 分钟内的这些信息构造出更复杂的因子，虽然这些因子仍然是 5 分钟频率，但由于用到了更微观的信息，所以构建出来的因子会比单纯的 5 分钟高开低收更有效一些。

（3）换月数据。公开平台的连续合约在换月时没有新合约的历史数据，因此在计算因子的时候很不方便。比如最简单的 20 均线，新合约只能在出现 20 根 K 线之后才能准确算出，前面的均线多少都要用到旧合约的信息。但如果是自己构造的 K 线，则不会有这个问题。或许很多人认为这只是一个小部分，

但很多时候机器学习模型就是由一个个模块叠加而成，如果每一个模块都引入一些误差，那么这些误差是逐渐叠加的，而不是误差越多越能互相抵消的，导致最终的结果跟设想的很不一样。

因此，最好从最基本的分笔数据开始，自己合成每个合约的 5 分钟 K 线，计算相应的因子，这样就不会有跨合约计算同一个因子的问题，在源头上减少误差。

获得分笔数据的几个途径有以下几种。

（1）自己用 CTP 下载。这个当然是免费的，而且最及时，可以得到一手的数据，也可以对比不同行情服务器地址的数据之间的差异，以本人经验来看，上海和大连的不同地方得到的数据基本一致，但郑州的数据不大一致，有细微的差别。

（2）万德资讯。这是国内最常用的金融信息软件，里面有商品的高频数据，而且支持 R、Matlab、Python 等接口，比较方便，但是需要收费，并且高频数据有流量的限制。

（3）国泰安。这是国内比较庞大的一个数据提供商，费用也比较高，数据需要落地，因此需要本地的服务器接收，一般通过 sql 导出。

（4）淘宝。本人购买的是淘宝大富翁提供的数据，盘后是 100 元 / 月，历史数据是 60 元 / 月，如果是 5 档行情会贵一些，180 元 / 月。

如果是小私募或者个人，建议在自己下载数据的同时也购买淘宝数据作为对照，且经济实惠。

有了数据获得的途径，就可以写一个程序，每天定时下载，自动处理，这样可以节约人力资源。其实现在很多新的发明主要作用就是节约人力资源，据说著名的高盛公司原来主要依靠银行家和销售人员，做的都是看起来比较高端的工作，觥筹交错间畅谈世界经济金融局势，但现在招聘了越来越多的程序员，越来越像一家科技公司。摩根士丹利也一样，原来交易大厅有 600 人，现在只有两个人。后面会讲到如何盘后自动获取数据。

2.1.2　数据储存

数据下载到计算机后，如何存储数据也是一个问题。如果存储成 csv 格式，

每次读取的速度是个大问题，我们可以来测试一下各种情况下数据读取的速度。

首先是用最简单的 read.csv 命令：

```
setwd("D:/liwei/tick/f_c201707d/sc/20170727") ## 设置工作路径
file.list <- list.files() ## 读取文件列表
system.time(for (file in file.list) ## 测试运行时间，并且使用循环
  data <- read.csv(file, header=TRUE, stringsAsFactors = FALSE)) ## 读
取csv数据
用户系统流逝 ## 运行结果
40.86  0.30 43.71
> length(file.list) ## 显示文件长度
[1] 161 ## 显示结果
```

可见需要 43.71 秒，一共 161 个文件。但如果我们使用速度更快的 fread 命令，则有：

```
library(data.table) ## 调用data.table程序包
system.time(for (file in file.list) ## 测试运行时间
  fast.data <- fread(file, header=TRUE, stringsAsFactors = FALSE)) ##
用fread高速读取
用户系统流逝 ## 运行时间
5.07 0.06 5.36
```

速度变成 5.36 秒，只有原来的 12.26%，提高了许多，而且结果是一致的：

```
> data[101:110,c(3:6,13:16)] ## 显示部分样本
时间最新持仓增仓买一价卖一价买一量卖一量
101 2017-07-26 21:00:51.000 22590   38    0  22340  22770      1      5
102 2017-07-26 21:00:51.500 22590   38    0  22340  22770      1      5
103 2017-07-26 21:00:52.000 22590   38    0  22345  22770      4      5
104 2017-07-26 21:00:52.500 22590   38    0  22350  22770      1      5
105 2017-07-26 21:00:53.000 22590   38    0  22350  22770      1      5
106 2017-07-26 21:00:53.500 22590   38    0  22355  22770      2      5
107 2017-07-26 21:00:54.000 22590   38    0  22360  22770      1      5
108 2017-07-26 21:00:54.500 22590   38    0  22360  22770      1      5
109 2017-07-26 21:00:55.000 22590   38    0  22365  22770      2      5
110 2017-07-26 21:00:55.500 22590   38    0  22370  22770      1      5
> fast.data[101:110,c(3:6,13:16)] ##显示高速运行的部分样本
时间最新持仓增仓买一价卖一价买一量卖一量
 1: 2017-07-26 21:00:51.000 22590   38    0  22340  22770      1      5
 2: 2017-07-26 21:00:51.500 22590   38    0  22340  22770      1      5
 3: 2017-07-26 21:00:52.000 22590   38    0  22345  22770      4      5
 4: 2017-07-26 21:00:52.500 22590   38    0  22350  22770      1      5
 5: 2017-07-26 21:00:53.000 22590   38    0  22350  22770      1      5
 6: 2017-07-26 21:00:53.500 22590   38    0  22355  22770      2      5
 7: 2017-07-26 21:00:54.000 22590   38    0  22360  22770      1      5
 8: 2017-07-26 21:00:54.500 22590   38    0  22360  22770      1      5
 9: 2017-07-26 21:00:55.000 22590   38    0  22365  22770      2      5
10: 2017-07-26 21:00:55.500 22590   38    0  22370  22770      1      5
```

可以看出结果是一致的。

当然，csv 文件还只是文本的格式，如果保存成 R 语言自己的二进制格

式，理论上说读取速度还可以更快。为此，我们首先把 csv 文件存成二进制的 RData 文件：

```
for (file in file.list) { ## 循环逐个处理文件
  data <- fread(file, header=TRUE, stringsAsFactors = FALSE) ## 高速读取
   save(data, file=paste("d:/liwei/binary tick/20170727/", substr(file,
1, nchar(file)-3), "RData", sep="")) ## 保存二进制格式的文件
}
```

然后就可以直接调用这些二进制文件了：

```
>system.time(for (file in file.list)
+   load(file)) ## 直接调用二进制文件
用户系统流逝
4.74 0.01 4.76
```

可见调用二进制文件可以比 fread 还要更快一些。但如果保存成 RData 格式，其实多占用了一份空间，因为 csv 文件方便人工查看，因此直接使用 fread 可以节省保存数据的空间。当然，如果只是希望在 R 语言里面使用数据，只保留 RData 文件也是可以的，也不会因为不小心失误修改了数据而导致意外发生。因此，实际工作中可以将 RData 和 csv 文件各保存一份，平时调用 RData 文件，需要增减数据也容易，如果不小心篡改了，再从 csv 文件中重新读取即可。

2.1.3　盘后自动获取

虽然说交易过程中也可以输出收到的分笔数据，但难免发生意外，专业的事情可以交给专业的机构做，费用也并不高。比如可以使用淘宝大富翁上的数据服务，每天盘后商品一档数据是 100 元 / 月。如果自己本身是有一定规模的私募，IT 团队比较厉害，也可以自己做。但是事实上，现在程序员工资也挺高，如果仅仅做这种期货量化，确实不太需要专门的程序员。更何况很多程序员都是 C++ 和 Java 厉害，对 R 语言也不大懂。

例如，每天下载数据的程序可以这么写：

```
> address <- "ftp://account:password@down.licai668.cn/" ## 下载的网址，其
中account和password
## 需要自己购买填写
> MAIN.CONTRACT.PATH <- "d:/liwei/main contract" ## 主力合约所在的路径
>exchanges <- c("dc","sc","zc") ## 三个商品交易所列表
download.exchange.data <- function(commodity.latest.date, exchange) {
## 函数头
## 需要最新日期和交易所的代码作为参数
  setwd("d:/QQMiniDL") ## 设置工作目录
```

```
  library(RCurl) ## 调用RCurl包，可以进行网络传输
  file.dest <- paste(address, exchange,"/",sep="") ## 目标文件地址
  all.files <- getURL(file.dest) ## 获取目标文件列表
  names <- str_split(all.files, pattern=" ")[[1]] ## 分割文件名
chosen<- names[which(!is.na(str_match(names, paste("^", exchange,
"_.*", sep=""))))]
## 选择符合命名规则的文件
  date.list <- str_sub(chosen, 4,11) ## 读取文件日期部分
  new.date <- date.list>commodity.latest.date ## 选取需要更新的日期
  file.list <- str_sub(chosen,1,15)[new.date] ## 读取需要更新的文件名
  dire.list <- str_sub(chosen,1,11)[new.date] ## 读取需要更新的文件目录
  date.list <- str_sub(chosen, 4,11)[new.date] ## 读取需要更新的日期列表
  month <- str_sub(date.list,1,6)[1] ## 需要更新的月份
  dir.create(paste("d:/liwei/tick/f_c",month,"d",sep=""),showWarnings =
FALSE) ## 创建文件夹
  dire.path <- paste("d:/liwei/tick/f_c",month,"d/",exchange,sep="") ##
下载的文件夹
   target.path <- paste("d:\\liwei\\tick\\f_c",month,"d\\",exchange,s
ep="")## 目标文件夹
  dir.create(dire.path, showWarnings = FALSE) ##创建目标文件夹
  for (j in 1:length(file.list)) { ## 逐个文件处理
    file <- file.list[j] ## 读取文件名
    file.url <- paste(file.dest, file, sep="") ## 文件下载地址
    download.file(url=file.url, cacheOK=FALSE,mode="wb",quiet=FALSE,destfi
le=file) ##下载文件
command<- paste("unrar e -y d:\\QQMiniDL\\", file, " ",target.path,
"\\",date.list[j], sep="")
##  解压命令
dir.create(paste(dire.path,"/",date.list[j],sep=""), showWarnings =
FALSE)
## 创建目标目录
    system(command) ## 在操作系统中执行命令
  }
  return (date.list)  ## 返回更新的日期列表
}
```

关键点如下。比如我们需要下载的是某个日期之后的数据，毕竟一般是每天更新，上一交易日的日期保存在 commodity.latest.date 中：

```
  new.date <- date.list>commodity.latest.date
```

由于国内是三个交易所，每次下载其中一个交易所的数据即可，比如下载上期所的数据，上期所 sc 是第二个交易所，可以用：

```
download.exchange.data(commodity.latest.date, exchanges[2])
```

另外，数据格式可能是压缩的 rar，需要运用 unrar 命令解压，在 R 语言里面可以直接运行 windows 命令行的命令，比如把命令写成字符串，然后用 system() 调用，这是以下部分：

```
download.file(url=file.url, cacheOK=FALSE,mode="wb",quiet=FALSE,destfile=fi
le)
```

```
command<- paste("unrar e -y d:\\QQMiniDL\\", file, " ",target.path,
"\\",date.list[j], sep="")
dir.create(paste(dire.path,"/",date.list[j],sep=""), showWarnings =
FALSE)
system(command)
```

这样就可以每天自动把数据下载到计算机了。如果使用万德等第三方平台，他们会整合其他很多功能，整体收费会比较高，因此只需要行情数据的话没必要使用太复杂的集成化信息系统。但如果需要其他基本面信息，或许那些平台更好一些。

下载完数据，接下来就可以用来分析日内策略或高频策略，如果想研究中低频的策略，可以使用 5 分钟 K 线，后面会有介绍。

2.2　合成 5 分钟数据

有些人问为什么需要合成 5 分钟数据？其实这更多是从研究效率来考虑。5 分钟 K 线有以下几个好处。

（1）过滤了分笔数据的很多噪音。分笔数据的量非常大，但行情起起伏伏，很多是噪声交易者所为，并不能预示未来的趋势。因此，如果能把这些信息整合起来，可以起到减少噪声的作用，而且太微观的变化对持仓长线的策略并没有太大意义，性价比不高；

（2）提高计算速度。500 毫秒的分笔数据 5 分钟可以有 5×60×2=600 个数据，如果使用 5 分钟 K 线，数据量只有原来的 1/600，可以大大减少计算量；

（3）频率也不会太低。如果用频率更低的 15 分钟线，则可能会因为频率太低而失去很多交易的机会。国内商品市场上午 10:15—10:30 有个空档，而且上午是到 11:30，下午 1:30 开始，因此划分 K 线最好是 15 的约数，10 分钟其实也不大好，比如 15 分钟的空档很难处理；如果是半小时、一小时，则会出现跨越很多空档时间段，因此 5 分钟是比较好的选择。其实 15 分钟的因子都可以用 5 分钟合成。但 1 分钟或许太过密集，很多商品不活跃，1 分钟也不会有太多成交。

合成 5 分钟 K 线数据主要包括开始价、最高价、最低价、结束价、成交量、成交额、持仓量等信息，这些都是 500 毫秒数据包含的，只需简单处理一下。

但需要注意的地方包括：

（1）并不一定在整 5 分钟结束的时候有行情。比如这段 rb1709 的行情：

```
    date.time price cum.open.int open.int  bid  ask bid.qty ask.qty
341  2017-07-26 21:04:57.500   3664          12724           0  3662  3664
1      10
342  2017-07-26 21:04:58.500   3664          12724           0  3662  3664
1       8
343  2017-07-26 21:05:00.500   3664          12724           0  3662  3664
1       8
344  2017-07-26 21:05:01.000   3664          12724           0  3662  3665
2       1
345  2017-07-26 21:05:01.500   3664          12724           0  3663  3665
2       9
346  2017-07-26 21:05:02.000   3664          12724           0  3664  3665
1       9
347  2017-07-26 21:05:02.500   3664          12722          -2  3662  3664
3       2
348  2017-07-26 21:05:03.000   3664          12722           0  3662  3665
4       9
349  2017-07-26 21:05:03.500   3664          12722           0  3664  3665
2       9
350  2017-07-26 21:05:06.000   3664          12722           0  3664  3665
2       9
```

在加粗部分，直接从 21:04:58.500 跳到了 21:05:00.500，而应该存在的 21:05:00.000 却不存在。因此，在进行 K 线分割时，并不能机械地查找整 5 分钟的时间点，而应该顺着行情一个一个检查。

（2）每个合约开始结束时间并不一样。比如黄金、白银是凌晨 2：30 结束，而很多品种没有夜盘，有的品种夜盘的时间在历史上修改过，因此，程序要能灵活识别这些情况。比如把数据分段，如果该段不存在行情，则应该知道该品种在那个时段是没有交易的。在这里，把夜盘分成 5 段，因为白天都是一样的，所以只有一段，因此，程序可以这么写：

```
night.1 <- which(data$time>"20:59" & data$time<"23:00:01") ## 夜盘第一段
时间
night.2 <- which(data$time>"23:00:01" & data$time<"23:30:01") ## 夜盘第
二段时间
night.3 <- which(data$time>"23:30:01") ## 夜盘第三段时间
night.4 <- which(data$time<"01:00:01") ## 夜盘第四段时间
night.5 <- which(data$time>"01:00:01" & data$time<"02:30:01") ## 夜盘第5
段时间
day.time <- which(data$time>"08:59:00" & data$time<"15:00:01") ## 白天时
段
```

注意到，比如是 23:00:00.500 这个时间，它会属于 night.2，某些品种夜盘会在这个时刻结束，因此，如果用“<23:00:01”的条件可以把它包含进来。每

天结束时有时会有15:00:00.500的时刻，因此用“<15:00:01”可以把它包含进来。

（3）程序运行速度。无论R语言、Matlab，还是Python，其实都是解释性语言，运行速度较慢，如果处理一些难以向量化的运算则会力不从心，此时可以考虑用C++来写。

2.2.1　R语言的版本

比如上一小节说的找出每个5分钟的切割点，如果用R语言来写，可以写成如下函数：

```
get.time.split <- function(data.time, split.time) { ## 切割5分钟K线
  j <- 1 ## 跟踪切割位置
  total.bar <- length(data.time) ## 总的行情数目
  n.bar <- length(split.time) ## 总的K线数目
  chosen.line <- rep(0,n.bar) ## 切割位置
for (i in 1:n.bar) {
    while (j<=total.bar && data.time[j]<=split.time[i]) ## 寻找下一个切割
位置
      j <- j+1
    chosen.line[i] <- j-1 ## 设置切割位置
  }
return(chosen.line)
}
```

可以重复1000次考察其运行时间：

```
system.time(for (i in 1:1000)
+     bb <- get.time.split(data$time[night.1], night.1.split) +night.1
[1]-1) ## 记录运行时间
用户系统流逝
39.61  0.00 39.87
```

由此可以看出，效率并不是十分高。主要原因在于R语言本质上是解释性语言，对于循环语句来说运行速度很慢，而这类切割K线存在路径依赖的问题，很难并行化处理。因此，要想提高速度，只能使用其他速度更快、更低级的编程语言来实现。

2.2.2　结合Rcpp提速

目前最常见的低级语言是C和C++，它们速度差不多，但C++更容易上手，而且有着面向对象的功能，在R语言里面也可以直接调用，无缝连接。

R 语言有一个专门跟 C++ 程序结合的包叫做 Rcpp，使用这个包可以直接在 R 语言里面编译 C++ 的程序，甚至可以自己专门建立一个 library，每次方便调用。因此，如果读者熟悉 C++ 的话，不妨尝试一下。

如果这部分用 C++ 来写，结合 Rcpp，可以新建一个程序文件 getTimeSplit.cpp：

```
#include <Rcpp.h> ## 包含Rcpp.h的头文件
using namespace Rcpp; ## 使用Rcpp命名空间

#include <vector> ## 使用C++的vector库
#include <algorithm> ##使用C++的algorithm库

// [[Rcpp::export]] ## Rcpp特有的注释，表明这是要被R调用的cpp文件

NumericVector getTimeSplit(CharacterVector dataTime, CharacterVector
splitTime) {
## dataTime和splitTime都是字符型向量，与R语言程序一致
  int j=0; ## C++开始位置是0
  int totalBar=dataTime.size(); ## 行情数目
  int nBar=splitTime.size(); ## K线数目
  Rcpp::NumericVector chosenLine(nBar); ## 切割位置
for (int i=0; i!=nBar; ++i) { ## 寻找下一个位置
while (j<totalBar && dataTime[j]<=splitTime[i]) j++;
    chosenLine[i]=j; ## 设置切割位置
  }
  return chosenLine; ## 返回切割为主
}
```

如果要调用里面的命令，需要先用 sourceCpp() 命令来编译：

```
> library(Rcpp) ## 调用Rcpp包
> sourceCpp("d:/liwei/rcode/getTimeSplit.cpp") ## 编译.cpp文件
```

然后可以看看用 Rcpp 的效果：

```
> system.time(for (i in 1:1000) ## 循环1000次
+   aa <- getTimeSplit(data$time[night.1], night.1.split)+night.1[1]-1)
## 调用Rcpp编译好的命令
用户系统流逝## 运行结果
1.13 0.00 1.14
```

我们可以对比一下 R 和 Rcpp 运行的结果：

```
aa
 [1]   602  1202  1802  2402  3002  3602  4202  4802  5402  6002  6602
7202  7801  8400  8999  9598 10194 10794
[19] 11394 11994 12594 13194 13794 14394
>bb
 [1]   602  1202  1802  2402  3002  3602  4202  4802  5402  6002  6602
7202  7801  8400  8999  9598 10194 10794
[19] 11394 11994 12594 13194 13794 14394
```

可见，两种语言运行的结果是一样的，但 Rcpp 的速度相比 R 语言要提高数十倍。

因此，提高计算速度对提高研究速度是非常有帮助的。比如80年代的神经网络，虽然模型拟合能力非常厉害，但由于当时的数据集并不是很大，很容易过度拟合。而且复杂的模型需要的参数比较多，即使有大量的数据集，运算一遍非常耗时，以至于研究者缺乏足够的耐心进行参数优化。90年代的支持向量机等算法虽然在拟合能力上不如神经网络，但是却采用了大量的数值计算、数值优化技巧，使运算速度大为提高，反而能够在那个年代脱颖而出。

然而，2010年以后，随着GNU等并行计算的大规模普及，以及互联网等大规模数据的产生，很多计算量大的模型反而重新获得了生命力。由于传统神经网络模型的名声不是那么好，神经网络的研究者采用了一个新的名字“深度学习”来继续他们的研究。

事实上，深度学习是在神经网络上发展起来的，比如著名的深度学习模型卷积神经网络（Convolutional Neural Network，CNN）则更是跟神经网络高度相关，所不同的地方在于普通的神经网络要自己找出因子，然后放入模型中学习，而卷积神经网络可以自己主动去寻找因子。这主要应用于图像分类中，比如要识别出一只动物是猫还是狗，一开始的分层更多是一些边界轮廓，比如识别出头部，然后逐步深入，可以识别出眼睛、嘴巴等，卷积类似于信号处理的滤波，可以把图像不同频率的特征识别出来，一开始是一些比较低频的特征，然后是更为高频的特征，最后再把这些特征放入神经网络模型中去拟合。

那么，这类模型能否用来分析金融数据呢？这个问题后面的章节会提到，这里只提一些思路。比如金融时间序列也是有低频和高频之分的，低频的更能显示出趋势，高频的则更类似于数据中的“毛刺”。比如一个5分钟K线，开盘价和收盘价构成主体，类似于主要趋势部分；上尾和下尾则类似于毛刺部分。或许一些厉害的高手可能把整个金融时间序列进行分解，比如是把价格变化分解成高频交易员产生的部分、低频交易员产生的部分、基本面产生的、技术面产生的等几个部分，类似于猫的头部、身体、尾部等然后再加总起来。

各个部分的交易员一般不会频繁改变自己的交易思路，并且市场中各部分成员的比例在短时间内也不会改变，即使缓慢改变，我们也可以通过渐进式（adaptive）的模型去拟合，这种方式或许比目前简单的线性回归模型更好一些。当然，这类模型需要的数据量也会更多，或许可以把数十个品种的数据加起来进行拟合。其实，每个分笔数据类似于图像处理的每个像素点，或许可以把图

像处理的模型移植到金融中。这些内容已经超出了本书的范围，不再深入研究，它的计算量会比目前的模型大得多。我们接下来讲解运用多核并行计算来提高计算速度。

2.2.3 多核并行再次提速

如果是处理单个品种的数据，用 Rcpp 进行提速似乎已经很不错了。但商品期货有数十个品种，而且品种数目一直在增加，我们应寻求其他提高速度的办法。

现在的计算机普遍采用了多核 CPU，每核一般有两个线程。比如本人现在使用的笔记本计算机是 2 核 4 线程，使用的台式机是 10 核 20 线程，当然服务器可以有几百个线程，国外厉害的公司估计几千个线程都没问题。使用这些来进行编程，自然可以更好地提高速度。

比如，在 R 语言里面可以使用 parallel 包来实现并行计算：

```
library(parallel) ## 调用并行的程序包
product.list <- c("ag", "au","bu","cu","hc","ni","rb","ru","zn",
                  "a","c","cs","i","j","jm","jd","l","m","p","pp","v",
"y",
                "CF","FG","MA","OI","RM","SR","TA","ZC") ## 活跃商品列表
n.core <- 4 ## 核的数目
system.time({
  cl <- makeCluster(n.core) ## 建立集群
   results <- parLapply(cl,product.list,parallel.process.5m.data) ## 并
行处理
stopCluster(cl)
})
用户系统流逝
  1.03    0.03 676.99
```

如果使用更多的核来并行，可以进一步提高速度。这里 makeCluster() 是使用核的命令，n.core 是核的数目，这里是 4。另外，parLapply() 是并行的命令，par 开头的此类处理命令在 parallel 包里面。

下面看看 parallel.process.5m.data() 的程序：

```
parallel.process.5m.data <- function(product, prefix="f_c201703d") { ##
并行处理5分钟数据
  cat(product,"\n") ## 输出品种名称
  path <- "d:/liwei/tick" ## 分笔数据的路径
  setwd(path) ## 设置路径
  all.dire <- list.dirs(full.name=FALSE,recursive=FALSE) ## 查找文件夹
  dire.list <- all.dire[grep(prefix, all.dire)] ## 满足前缀条件的文件夹
```

```
  library(Rcpp) ## 调用Rcpp程序包
  library(inline) ## 调用inline程序包
  source("d:/liwei/rcode/commodity.5m.helper.r") ## 调用相关辅助函数
  sourceCpp("d:/liwei/rcode/getTimeSplit.cpp") ## 编译.cpp函数
  library(data.table) ## 调用data.table
  chosen.month <- paste(contract.month[[product]],collapse="|") ##选择品
种活跃月份
if (!grepl("^[[:upper:]]+$", product)) { ## Shanghai or Dalian上海或大连
交易所
      pattern <- paste(".*",product,"[[:digit:]]{2}(?:",chosen.
month,")",sep="")
  } else { ## Zhengzhou郑州交易所
      pattern <- paste(".*",product,"[[:digit:]]{1}(?:",chosen.
month,")",sep="")
  }
for (dire in dire.list) {
cat(dire,"\n")
    parse.dire.fast(dire,pattern,product) ## 处理一个文件夹的问价
  }
}
```

之所以对郑州单独处理是因为大连和上海的合约月份都是 4 位，而郑州是 3 位。这里面还调用了一个函数 parse.dire.fast()，它是对一个文件夹里面的所有合约进行处理，代码如下：

```
parse.dire.fast  <- function(dire,pattern,product,object.dire="d:/
liwei/binary/com5m/") {
## 处理文件夹的程序
   file.list <- list.files(dire, recursive=TRUE,pattern=pattern) ##查找所
有满足条件的文件
   file.list <- file.list[grep(paste("/",product,sep=""), file.list)] ##
过滤掉一些错误的文件
   start.prod <- as.numeric(regexpr(paste("/",product,sep=""),file.
list[1])) ## 查找到期日位置
   check.dup <- substr(file.list, start.prod+nchar(product)+1,start.
prod+nchar(product)+1) ##提取到期日
  file.list <- file.list[check.dup>="0" & check.dup<="9"] ## 继续过滤文件
  if (length(file.list)==0) return(1) ## 如果没有满足要求的文件则返回
  for (i in 1:length(file.list)) { ## 逐个处理文件
    file <- file.list[i] ## 提取文件
cat(file,"\n")
     if (file.info(paste(dire,file,sep="/"))$size==0) next ## 文件错误则下
一个
    data <- get.data.5m.fast(dire,file) ## 快速整理5分钟数据
    if (length(data)==1) next ## 数据错误则下一个
    contract <- data$contract[1] ## 提取合约
      dir.create(paste(object.dire,contract,sep=""),showWarnings=FALSE)
## 创建文件夹
      new.file <- gsub("_","/",paste(substr(file,10,nchar(file)-
3),"RData",sep="")) ##  新文件名
    heaven <- as.numeric(gregexpr("/",new.file)[[1]]) ## 特殊情况处理
if (length(heaven)>1) new.file <- substr(new.file, heaven[1]+1,
nchar(new.file))
```

```
      save(data,file=paste(object.dire,new.file,sep="")) ## 保存新文件
    }
  }
```

这里面调用了函数 get.data.5m.fast()，它就是利用了 Rcpp 的函数来把分笔数据整理成 5 分钟数据，速度比较快，代码如下：

```
get.data.5m.fast<- function(dire,file) { ## 快速处理5分钟数据
   data <- fread(paste(dire,file,sep="/"),stringsAsFactors=FALSE) ## 快速读取文件
  data <- as.data.frame(data) ## 整理成数据框
  trade.date <- substr(file, nchar(file)-11, nchar(file)-4) ## 提取交易日
   colnames(data) <- c("market","contract","date.time","price","cum.open.int","open.int","turnover",
"qty","open.symbol","close.symbol","type","dire","bid","ask","bid.qty","ask.qty") ## 列的名称
   data$date<- paste(substr(data$date.time,1,4),substr(data$date.time,6,7),substr(data$date.time,9,10),sep="")
## 提取日期
  data$time <- substr(data$date.time,12,23) ##提取时间
  total.bar <- nrow(data) ## 总的行情数
  pre.time <- c(data$time[1], head(data$time,-1)) ## 前一个时间
  night.1 <- which(data$time>"20:59" & data$time<"23:00:01") ## 夜盘第一段数据
  night.2 <- which(data$time>"23:00:01" & data$time<"23:30:01") ## 夜盘第二段数据
  night.3 <- which(data$time>"23:30:01") ## 夜盘第三段数据
  night.4 <- which(data$time<"01:00:01") ## 夜盘第四段数据
  night.5 <- which(data$time>"01:00:01" & data$time<"02:30:01") ## 夜盘第五段数据
  day.time <- which(data$time>"08:59:00" & data$time<"15:00:01") ## 白天数据
  chosen.line <- rep(0,120) ## 分割位置
  split.time <- rep("",120) ## 分割时间
  cur.i <- 0
  if (length(night.1)>0) { ## 处理夜盘第一段数据
chosen.line[(cur.i+1):(cur.i+24)] <- getTimeSplit(data$time[night.1],night.1.split)+night.1[1]-1
chosen.line[cur.i+24] <- tail(night.1,1)
split.time[(cur.i+1):(cur.i+24)] <- night.1.split
    cur.i <- cur.i+24
  }
  if (length(night.2)>0) { ## 处理夜盘第二段数据
chosen.line[(cur.i+1):(cur.i+6)] <- getTimeSplit(data$time[night.2],night.2.split)+night.2[1]-1
chosen.line[cur.i+6] <- tail(night.2,1)
split.time[(cur.i+1):(cur.i+6)] <- night.2.split
    cur.i <- cur.i+6
  }
  if (length(night.3)>0) { ## 处理夜盘第三段数据
chosen.line[(cur.i+1):(cur.i+6)] <- getTimeSplit(data$time[night.3],night.3.split)+night.3[1]-1
chosen.line[cur.i+6] <- tail(night.3,1)
split.time[(cur.i+1):(cur.i+6)] <- night.3.split
```

```
      cur.i <- cur.i+6
    }
    if (length(night.4)>0) { ## 处理夜盘第四段数据
  chosen.line[(cur.i+1):(cur.i+12)] <- getTimeSplit(data$time[night.4],
  night.4.split)+night.4[1]-1
  chosen.line[cur.i+12] <- tail(night.4,1)
  split.time[(cur.i+1):(cur.i+12)] <- night.4.split
      cur.i <- cur.i+12
    }
    if (length(night.5)>0) { ## 处理夜盘第五段数据
  chosen.line[(cur.i+1):(cur.i+18)] <- getTimeSplit(data$time[night.5],
  night.5.split)+night.5[1]-1
  chosen.line[cur.i+18] <- tail(night.5,1)
  split.time[(cur.i+1):(cur.i+18)] <- night.5.split
      cur.i <- cur.i+18
    }
    if (length(day.time)>0) {## 处理白天数据
  chosen.line[(cur.i+1):(cur.i+45)] <- getTimeSplit(data$time[day.time],
  day.split)+day.time[1]-1
  chosen.line[cur.i+45] <- tail(day.time,1)
  split.time[(cur.i+1):(cur.i+45)] <- day.split
      cur.i <- cur.i+45
    }
    chosen.line <- chosen.line[1:cur.i] ## 提取有效的部分
  chosen.line[chosen.line==0] <- 1
    data.5m <- data[chosen.line,selected.column] ##提取有意义的行和列
    data.5m$time <- substr(split.time[1:cur.i],1,8) ## 设置时间
    data.5m$trade.date <- trade.date ## 设置交易日期
  start<- 1
     while (data$bid[start]==0 & data$ask[start]==0) start <- start+1 ##
  过滤没意义的行情
    for (i in 1:cur.i) { ## 逐段计算K线数值
      range <- start:chosen.line[i] ## K线对应的行情范围
      data.5m$open.int[i] <- sum(data$open.int[range]) ## 持仓量
      data.5m$qty[i] <- sum(data$qty[range]) ## 成交量
      data.5m$open[i] <- data$price[start] ## 开盘价
      data.5m$high[i] <- max(data$price[range]) ## 最高价
      data.5m$low[i] <- min(data$price[range]) ## 最低价
      data.5m$close[i] <- data$price[chosen.line[i]] ## 收盘价
      start <- chosen.line[i]+1 ## K线开始位置
    }
    return(data.5m) ## 返回5分钟K线
  }
```

这里计算了 5 分钟 K 线的高开低收以及成交量、持仓量等，至此，整个把分笔数据合成 5 分钟 K 线的程序就完成了。如果该品种某段交易时间段不交易，则会自动跳过该段时间段，进入到下一个时间段。哪怕该段时间只有一次交易，也会生成该段所有的 5 分钟 K 线。

当然，这个程序也有一定的缺陷，比如交易所改变交易时间段，增加某些时间段或拆分已有的时间段等都需要修改程序。

另外，部分量化交易策略需要估算挂单成交的，除了高开低收的成交价以外，还需要挂单价，这里由于我们分析的是趋势策略，加滑点抢单成交，并没有分析被动挂单策略，因此不需要最高价和最低价的挂单价，有兴趣的读者可以自己整理一下。

2.3 异常处理

做机器学习或数据挖掘的人都知道，数据集往往有很多缺失、错误的地方，需要先人工进行处理。比如期货数据会有很多非交易时间段的数据需要专门处理，例如：

```
    时间最新持仓增仓买一价卖一价买一量卖一量
1 2017-07-26 19:12:02.500 3568 3421350 3421350    0    0    0    0
2 2017-07-26 20:59:00.500 3555 3421818    468 3554 3555  273  153
3 2017-07-26 21:00:00.500 3556 3422600    782 3556 3557  792  164
4 2017-07-26 21:00:01.000 3557 3423296    696 3557 3558  119  396
5 2017-07-26 21:00:01.500 3558 3423872    576 3557 3558  509  239
```

第一行的数据买一价、卖一价、买一量、卖一量都是零，而且交易时间是19：12：02.500，这是非交易时段，这个数据理应去掉。事实上，我们之前的数据处理程序已经考虑了这点：

```
while (data$bid[start]==0 & data$ask[start]==0) start <- start+1
```

从第一个 bid 价和卖价至少一个不为零的数据开始，有时候行情出现涨跌停板，比如涨停板时 ask 是零，因此，只有当两个都是零的时候才是异常的。

2.3.1 夜盘数据

现在不少期货品种开通了夜盘，一般是从晚上 9 点开始，但结束时间各不一样。如果机械记录每个品种的夜盘结束时间，其实也不大好，因为这个时间不是一成不变的，而是会随着一些新的规定一直改变。比如螺纹钢一开始夜盘交易到凌晨 1 点，现在交易到晚上 11 点。因此，我们需要一个更灵活的方式来处理夜盘数据。为此，我把夜盘分成以下几段：

```
night.1 <- which(data$time>"20:59" & data$time<"23:00:01")
```

```
night.2 <- which(data$time>"23:00:01" & data$time<"23:30:01")
night.3 <- which(data$time>"23:30:01")
night.4 <- which(data$time<"01:00:01")
night.5 <- which(data$time>"01:00:01" & data$time<"02:30:01")
```

最早结束夜盘的品种是 23:00:00，但很多时候会有 23:00:00.500 这笔行情，因此，用 data$time<“23:00:01”可以把这笔行情包含进来，其他时间也类似处理。

2.3.2　涨跌停处理

在涨停板，卖价是空值，一般行情软件用零代替，而在跌停板，买价是空值，也是用零代替。今后处理一些因子和因变量的时候要特别注意，因为这个时候的交易实际上并没有产生价格变化，而此时计算的预测指标，由于买量很大，一般都会预示有很强的上涨动能，因此会出现预测值与技术指标值的相关性极不正常的情况，这些就是异常值，在训练模型拟合系数的时候需要剔除。这里原始数据并没有做特别处理，这点留到以后再详细说明。当然，如果某些品种涨跌停板的比例很低，不会对模型参数拟合产生太大影响，也可以不用考虑。

事实上，涨跌停板数据占总数据量并不多，以螺纹钢为例：

```
> product <- "rb" ## 定义品种为螺纹钢
> all.contracts <- get.dates(product) ## 读取该品种的全部主力合约
> all.contracts ## 显示该品种的全部主力合约
 [1] "rb1210.RData" "rb1301.RData" "rb1305.RData" "rb1310.RData"
"rb1401.RData" "rb1405.RData" "rb1410.RData"
 [8] "rb1501.RData" "rb1505.RData" "rb1510.RData" "rb1601.RData"
"rb1605.RData" "rb1610.RData" "rb1701.RData"
[15] "rb1705.RData"
> total <- 0 ## 总的K线数目
> good <- 0 ## 非涨跌停板的K线数目
> for (contract in all.contracts) { ## 遍历所有合约
+   load(contract) ## 调用合约数据
+   total <- total+sum(data$continuous)  ## 总K线数累加
+   good <- good+sum(data$continuous & data$good) ## 非涨跌停板K线数累加
+ }
> good/total ## 好的K线的比例
[1] 0.9960169
> good ## 好的K线的数目
[1] 74268
> total ## 总的K线的数目
[1] 74565
```

由此可见，正常的无涨跌行情占 99.6%，因此涨跌停行情对数据影响不会很大。可能很多人会担心涨跌停板时期价格基本处于停滞状态，但技术指标的

取值往往却比较极端，这时候如果价格变化与技术指标取值很不一致，会影响拟合的效果。但因为这部分行情占比很小，所以对实际参数估计的影响并不大。

2.3.3 为什么保留买卖盘口

之前章节也提到过，虽然用最新价也可以较准确的回测策略，但这里我们还是保留了买卖盘口，哪怕是用 5 分钟 K 线做中低频的策略，原因有以下几点。

（1）很多品种买卖价差相对于价格来说很大，比如螺纹钢，价格一般在 1800 ～ 3600 元，买卖价差是 1，即买卖价差是盘口的 0.28% ～ 0.56%，其实挂单量也蕴含了很多信息，此时的最新价一般在二者之一，不能较准确反映当前的行情，如果用挂单量加权平均价或许更有意义。

（2）对于一些基于挂单价成交的策略，可以设置当前买一价买入，或者低于当前买一价若干价位买入，然后看 5 分钟内能否成交。事实上，回测时可以更严格一些，比如我们挂买一价，如果最低价低于买一价才算是成交，即使相等也不算，这样可以更为严格的估算挂单成交。

比如回测一个策略，可以出信号后立刻对手价 + 滑点成交，也可以假设挂单，如果下一根 K 线的最低价低于挂单价，则挂单价成交；如果下一根 K 线的最低价大于等于挂单价，则用下一根 K 线收盘价 + 滑点成交，这样可以跟原来的成交情况进行对比，或许有所改进。

由于涉及挂单的策略会更复杂一些，这里不作专门讨论，本书讨论的趋势策略默认都是加滑点即时成交。

2.3.4 跟其他行情软件对比

每个行情软件都有自己切割 K 线的方法，关于 500 毫秒应该放上一根还是下一根的问题每个软件处理方法都不一样。其实这些方法整体上都是大同小异的，没必要刻意强求，但是要保证自己实盘程序跟研究程序用到的 K 线是一致的。因为这类机器学习模型，对数值计算高度敏感，随着计算次数的增加，误差是一直累积的并不会抵消，由此可能造成实盘和回测优化很不一致，比如开平仓时间点不一样甚至缺少一些交易等。因此，最重要的是保持自己 K 线的一

致性，而不是跟某个行情软件的一致性。

另外，现在有很多免费或付费的基于第三方平台的策略程序，也可以用自己的 K 线去测试一下。一般规则型的策略对数值不会很敏感，事实上如果对数值过于敏感也可能由过度拟合所导致。总之，如果是 5 分钟 K 线的话应该差别不会很大，因此不必过于纠结。

对于夜盘结束等停止交易的时段要特别小心，交易所可能会把最后一条行情反复推送，或者自己的程序在夜盘结束后关闭，而白天重新启动时会再次接收到夜盘的数据，很可能把成交量等再次累积计算，这些都要被过滤掉。本人曾经因为类似的原因导致白天开盘 K 线总是跟回测对不上，而且总是容易触发交易而亏钱。

2.4　本章小结

本章介绍了分笔数据的处理，主要处理方法是把分笔数据整理成 5 分钟 K 线数据，以及各种提高速度的方法。R 语言本质上是解释性语言，虽对统计分析有丰富的函数包做支持，但处理一些计算时速度并不快。为此，本章使用了 Rcpp 与多线程相结合的办法，把速度提高了数十倍甚至上百倍，大大提高了研究的效率。接下来的一章，我们将用这些数据构造预测因子，进入统计分析的阶段。

第三章

预 测 因 子

本章主要介绍用于建模的预测因子。在机器学习这个领域，一般把构建因子称为特征工程（feature engineering），因为在机器学习中一般把统计学里面的参数（parameter）称为特征（feature），实际上特征、因子、参数都是一个含义，下面不再进行区分。当然，机器学习会帮另一类基础的因子成为属性（attribute），然后把attribute加工成的因子称为feature，所以才有特征工程一说。一般在统计学里没有这么麻烦，这里就不加以区分了。

3.1　技术指标来源

传统的直观的建模方法是规则型或者条件型的，满足一系列条件就买（卖），不满足就不进行买（卖）。这些条件都是由人脑主观构造的，但由于市场变化莫测，各种情况千差万别，很难用一些过于精确的条件来描述价格的变化。

3.1.1　建模背景介绍

举一个类似的例子，如自动驾驶。要教会一辆汽车自动驾驶，如果依靠规则系统，比如告诉它看见红灯就减速，看见绿灯就启动等，这些还算比较简单；但如果是看见前面的车就刹车、看到旁边的车就变道、看到后面的车就加速等，则会比较复杂，整套规则体系过于庞杂也不便系统化地研究，很多规则都是一些较为特殊的情况，并且规则之间多有重叠性，如何泛化又是一个问题。因此，如果顺着这种规则型、专家型的思路，很难进行系统化的研究。

20 世纪 80 年代最流行的模型是神经网络，也有人用神经网络的方法研究自动驾驶，可惜那时候不是很成功。神经网络的一个很大问题在于过度拟合，这主要是由模型参数过多而训练数据太少导致的。比如估计一个参数的数值，数据量每扩大 100 倍，精度可以提高 10 倍；而如果估计的参数非常多，类似于一个高维统计问题，数据点则会更多地分布在样本空间的边缘地带，从而严重影响参数估计的效果。详细的讨论可以参考《Elements of Statistical Learning second edition》第二章，这里不做展开。

如果数据量足够大，在样本空间内部各种取值都有，参数估计则更为准确。当年神经网络的模型在今天重新运用都取得了很好的成果，但因为神经网络的名声不是很好，所以现在的研究者把它命名为深度学习。

现在回到汽车驾驶的问题。我们可收集非常长的驾驶记录作为训练数据，然后把可以想象到的指标如车速、前车车速、后车车速、车距等作为因子，然后用复杂的深度学习模型，例如卷积神经网络，就可以训练出很好的自动驾驶

系统。事实上，人类学习驾驶也是通过观察别人驾驶，然后有个较为模糊的印象，而不是通过一条一条精确的规则来驾驶，这跟卷积神经网络类似，如果驾驶途中一条一条规则去套，显然反应不过来，容易出车祸。

因此，知道预测模型的威力之后，我们不再使用规则型的系统，在交易中也一样。本章主要讨论预测因子，下一章再讨论预测模型。

因子的来源有很多种，其实更多依赖数据的来源。如果只有行情数据，那么可以做的工作非常有限。更进一步地说，如果是持仓 3、5 天的中低频策略，那么过于高频的分笔数据用处不大，还要先整理成 5 分钟 K 线，然后在 5 分钟 K 线的基础上构建预测因子，这样可以变换的方式并不多见。

如果是持仓时间更长的策略，比如 2 ～ 4 周，那么一般需要用日线数据建模，可以使用很多基本面的数据。国内很多信息提供商会提供商品基本面的数据，比如有上游、中游、下游三种，数据有日频、周频、月频等，可以用来建模。但很多时候数据质量是一个问题，比如数据更新不及时，数据修正缺乏历史记录等，就会导致用这些数据建立的模型没说服力。行情数据虽然简单，但最起码数据质量是有保证的。如果数据质量没有保证，机器学习就会称之为垃圾进垃圾出（garbage in，garbage out），得不到有意义的结果。因此，我们这里只研究行情数据和 5 分钟 K 线。

我们以螺纹钢（rb）为例来说明。本地数据中 rb 的合约有如下几个：

```
> product <- "rb" ## 设置品种为螺纹钢
> all.contracts <- get.dates(product) ## 获得全部主力合约
> all.contracts
 [1] "rb1210.RData" "rb1301.RData" "rb1305.RData" "rb1310.RData"
"rb1401.RData" "rb1405.RData" "rb1410.RData"
 [8] "rb1501.RData" "rb1505.RData" "rb1510.RData" "rb1601.RData"
"rb1605.RData" "rb1610.RData" "rb1701.RData"
[15] "rb1705.RData"
> n.contract <- length(all.contracts) ## 主力合约数目
```

我们可以用 rb1701 作为例子：

```
> contract <- all.contracts[n.contract-1] ## 取其中一个
> CONTINUOUS.PATH ## 合约存放的路径
[1] "d:/liwei/continuous binary"
> setwd(CONTINUOUS.PATH) ## 设置当前工作路径为合约所在路径
> load(contract) ## 调用合约
> plot(data$date.time, data$price, type="l", xlab="date", ylab="price",
main=contract) ## 画合约全部时段的价格曲线
> points(data$date.time[data$continuous], data$price[data$continuous],
col=2, type="l") ## 画主力合约时期的价格曲线
```

图 3-1　螺纹钢 1701 合约

具体的价格趋势如图 3-1 所示，黑线部分是所有的价格，红线部分是该合约作为主力合约时的时间段。由此可以看出，每个合约都只有其上市时的一部分时间作为主力合约存在，其他都是作为非主力合约存在。一般来说，只有主力合约的流动性足够好，才能够用来交易趋势类策略，因此，我们这里构造因子也只针对主力合约进行。

一般来说，传统的技术指标并不多，下面我们逐个进行分析。

3.1.2　R 自带的 TTR 库

我们可以先看看 R 的 TTR（Technical Trading Rules）包里面的函数：

```
> library(TTR) ## 调用技术指标库
> lsp <- function(package, all.names = FALSE, pattern) { ## 列举package
所有函数
+   package<- deparse(substitute(package))
+   ls(
+     pos = paste("package", package, sep = ":"),
+     all.names = all.names,
+     pattern = pattern
+     )
+ }
> lsp(TTR) ## 列举TTR所有函数
 [1] "adjRatios"       "ADX"       "ALMA"       "aroon"       "ATR"
 [6] "BBands"    "CCI"    "chaikinAD"    "chaikinVolatility"    "CLV"
[11] "CMF"      "CMO"      "DEMA"    "DonchianChannel"      "DPO"
[16] "DVI"      "EMA"      "EMV"      "EVWMA"      "getYahooData"
[21] "GMMA"      "growth"      "HMA"      "KST"      "lags"
[26] "MACD"      "MFI"      "momentum"      "naCheck"      "OBV"
[31] "PBands"      "ROC"      "rollSFM"      "RSI"      "runCor"
```

```
[36] "runCov"      "runMAD"      "runMax"      "runMean"      "runMedian"
[41] "runMin"      "runPercentRank"      "runSD"      "runSum"    "runVar"
[46] "SAR"      "SMA"      "SMI"      "stoch"      "stockSymbols"
[51] "TDI"      "TRIX"      "ultimateOscillator"      "VHF"      "VMA"
[56] "volatility"      "VWAP"      "VWMA"      "wilderSum"    "williamsAD"
[61] "WMA"      "WPR"      "ZigZag"      "ZLEMA"
```

函数有 64 个之多，用它们构造技术指标应该是足够的。

例如我们可以先看第一个 ADX。考察一个因子好坏的一个简单方法是看它自身与未来收益的相关性是否稳定。正负并不重要，因为只需要加个负号就可以改变。

```
period <- 16 ## 研究周期
adx.5m <- function(high,low,close,period) { ## 5分钟的ADX
return (clean(ADX(cbind(high,low,close),n=period)[,3]/100-0.5))
}
fcum <- function(x, n, na.rm = TRUE, fill = 0) { ## 未来一段数据的和，用于
计算未来收益
if (na.rm) return(head(lag(cum(c(x, rep(fill, n)), n), -n), -n))
else return(lag(cum(x, n), -n, fill = fill))
}
cor.vec <- as.numeric(sapply(all.contracts, function(contract){ ## 指标
与收益的相关系数，每段
## 合约1个数字
load(contract)
  x <- with(data, adx.5m(high, low, close, period))[data$continuous]
  y <- with(data, fcum(wpr.log.ret, period))[data$continuous]
cor(x,y)
}))
cor.vec
[1]    0.0268267841  -0.0039409878    0.0317268921  -0.0544822096
-0.0030651075  0.0049134024 -0.0557897969
  [8]  -0.0980392969  -0.0293593289    0.0107987621  -0.0224249379
0.0589294998 -0.0076046940 -0.0008975012
[15] -0.0119682773
mean(cor.vec)/sd(cor.vec) ## 均值除以标准差
[1] -0.2655112
```

如果一个指标，在不同的合约上，跟未来的收益率都是正相关，或者都是负相关，那么这个指标是比较稳定的；但如果一个指标在一半合约上正相关，另一半合约上负相关，那么它就不是十分稳定。另外，考察稳定性还可以用均值 / 标准差的方法，类似于 Z-score 的概念。这里这个数值是 −0.266，只从数值上我们很难分辨它是高还是低，可以结合其他因子一起来看。

```
> aroon.5m <- function(high,low,period) { ## 5分钟K线的AROON指标
+   return (clean(aroon(cbind(high,low),n=period)[,3])/100)
+ }
> cor.vec <- as.numeric(sapply(all.contracts, function(contract){
+   load(contract)
```

```
+    x <- with(data, aroon.5m(high, low, period))[data$continuous]
+    y <- with(data, fcum(wpr.log.ret, period))[data$continuous]
+    cor(x,y)
+ }))
> cor.vec
  [1]   0.001599738   0.067477774   0.033246871   0.015651357 -0.037340366
0.047214472 -0.010770836  0.016546496
  [9] -0.013133586   0.026169464 -0.047167192   0.031344160   0.052498959
0.022193933  0.010627709
>mean(cor.vec)/sd(cor.vec)
[1] 0.451995
```

由此可见，这个指标的数值是 0.45，而且 15 个合约中有 11 个是正的，比之前的指标更为稳定，因此，我们认为这个指标更好。

```
> size.imb <- function(bid.size,ask.size) { ## 买卖挂单量不平衡的指标
+    return (clean((bid.size-ask.size)/(bid.size+ask.size)))
+ }
> cor.vec <- as.numeric(sapply(all.contracts, function(contract){
+    load(contract)
+    x <- with(data, size.imb(bid.qty, ask.qty))[data$continuous]
+    y <- with(data, fcum(wpr.log.ret, period))[data$continuous]
+    cor(x,y)
+ }))
> cor.vec
  [1]  -0.0139030213  -0.0339061757  -0.0148683773   0.0138486289
0.0104661120 -0.0015431636  0.0029864356
  [8]   0.0031128340  -0.0101103320   0.0001543563   0.0083883613
0.0063344118  0.0294441502  0.0247227111
[15]  0.0120725355
> mean(cor.vec)/sd(cor.vec)
[1] 0.1539977
```

这个指标仅仅用到买卖盘后，表现弱一些，因为买卖盘后跟未来 80 分钟的价格变化关系不大，但是跟未来短时间的价格变化关系比较密切。因此，同样一个指标，针对不同的预测周期，会有非常不一样的效果。

也许很多人会问，这种利用过去行情信息预测未来的方法，是否违反市场弱有效的假说？其实，这个问题可以这么考虑：首先，市场弱有效本来就是个假说，有事实支持它，也有事实反驳它。其次，每个人分析信息的能力不一样，同样的信息不可能同一时间反映到行情中，有人快有人慢，信息处理能力强的人会获得一定优势，毕竟他们也支付了更多的成本（脑力、劳力），因此理应获得更高的收益。最后，大多数投资大师都反对市场弱有效假说，甚至有很多小幽默调侃它，比如“资助商学院教市场弱有效假说方便自己赚钱”，但对市场保持敬畏之心还是有必要的。总之，我觉得正确的态度是相信市场存在无效的时候，同时也认可挖掘市场无效不是那么容易的事情。

我们接下来考察其他因子：

```
bbands.5m <- function(high,low,close,period) { ## 5分钟布林带指标
return (clean(BBands(cbind(high,low,close),period)[,4]))
}
cor.vec <- as.numeric(sapply(all.contracts, function(contract){
load(contract)
  x <- with(data, bbands.5m(high, low, close, period))[data$continuous]
  y <- with(data, fcum(wpr.log.ret, period))[data$continuous]
cor(x,y)
}))
cor.vec
mean(cor.vec)/sd(cor.vec)
[1] 0.4645663
```

这个因子表现更加稳定。有人会问构造这些因子的过程是否复杂，我个人的观点是没必要在构造因子上花太多的时间，原因在于因子本质上起着向量基的作用，但是在同一个线性空间下面，目标向量（因变量）确定后，用不同的基去表示本质上是一样的；其次，现在有 boosting 等各种 ensemble 方法，复杂的因子可以通过一定的方法由简单的因子合成，所以没必要去人工构造；最后，在信息源不变的情况下，构造的因子很多是高度相关的，如果精心构造一个因子，或许会过于复杂，预测效果反而不好，还不如把精力放在这些简单的因子上。综上所述，本人不会把太多精力放在构造因子上。这些因子都是运用 R 语言自带的函数稍加整理即可，一般只有一行代码（one liner）。

```
> cor.vec <- as.numeric(sapply(all.contracts, function(contract){
+   load(contract)
+   x <- with(data, cci.5m(high, low, close, period))[data$continuous]
+   y <- with(data, fcum(wpr.log.ret, period))[data$continuous]
+   cor(x,y)
+ }))
> cor.vec
 [1] -0.004578969  0.028667631  0.069166449 -0.021923355 -0.038857232
0.068293330 -0.001378321  0.025188173
 [9]  0.012752632  0.025800136 -0.029980450  0.014307613  0.046035234
0.009632681  0.015497611
>mean(cor.vec)/sd(cor.vec)
[1] 0.4590126
```

也有些人会问，如果把这些因子都公开了，会不会对其不利。事实上，既然认为特征工程不重要，自然也就不会认为这些因子有多重要。量化建模是一个庞大的系统过程，并不存在哪块特别重要之说，本人觉得各个环节都挺重要的，而且本人也一直处于不断进步中，期货中低频也未必是本人未来主要研究的方向。

```
clv.5m <- function(high,low,close) { ## 5分钟CLV指标
return (clean(CLV(cbind(high,low,close))))
}
cor.vec <- as.numeric(sapply(all.contracts, function(contract){
load(contract)
  x <- with(data, clv.5m(high, low, close))[data$continuous]
  y <- with(data, fcum(wpr.log.ret, period))[data$continuous]
cor(x,y)
}))
cor.vec
[1]   0.015157724   0.006745066   0.035790903   0.003152490   0.003642633
-0.005943928  0.011471704  0.025517315
 [9]   0.016062568   0.006616733   0.010025479   0.009541799 -0.002265039
0.003366260 -0.006107624
mean(cor.vec)/sd(cor.vec)
[1] 0.7873691
```

另外，从单个因子来看，即便相关性不是很一致，但很多个因子结合起来，预测效果就会好很多，这就是把几个弱指标合成一个强指标的意义。总之，没必要在单个指标上花太多时间。

```
> cmf.5m <- function(high,low,close,qty,period) { ## 5分钟CMF指标
+   return (clean(CMF(cbind(high,low,close),qty,period)))
+ }
> cor.vec <- as.numeric(sapply(all.contracts, function(contract){
+   load(contract)
+     x <- with(data, cmf.5m(high, low, close, qty, period))
[data$continuous]
+   y <- with(data, fcum(wpr.log.ret, period))[data$continuous]
+   cor(x,y)
+ }))
> cor.vec
 [1]   0.066274020 -0.000389366   0.003887651   0.078116275   0.015925439
-0.018772502  0.100707881  0.035378893
 [9]   0.026334896   0.012896424 -0.001722354 -0.009436509 -0.011295621
0.013112683  0.017397626
>mean(cor.vec)/sd(cor.vec)
[1] 0.630666
```

好了，关于 TTR 的指标就阐述到这里，剩下的内容大家可以自己尝试着练习一下。

3.1.3　一些技术指标的书籍

除了 R 语言自带的指标以外，还有一些技术指标方面的书籍大家可以参考学习一下，这里推荐一本书：《The Encyclopedia of Technical Market Indicators》，作者为 Robert W. Colby。推荐这本书并不是因为这本书里面的

技术因子有多厉害，而是因为这本书包含的因子数目非常多，一共有 800 多页，从 A 排到 Z，有数百个因子，因此不需要自己思考，直接把上面的因子拿来用即可。这本书网上有电子版，里面的技术指标很多也包含在 R 的 TTR 包里，如果没有包含，自己简单仿照着写也不困难，这里不再赘述。

另外一些是基于日本蜡烛图（即 K 线图）的预测因子，可以参考这本书《日本蜡烛图技术》（史蒂夫·尼森著，丁圣元译）。这些更像是一些技术形态，把它们翻译成技术指标还需要一些工夫。

如著名的乌云盖顶：

```
dark.cloud <- function(open,close,ret,ratio=0.25) { ## 乌云盖顶指标
  one.quar <- open+(close-open)*ratio
  pre.one.quar <- c(one.quar[1], head(one.quar,-1))
  pre.open <- c(open[1], head(open,-1))
  pre.ret <- c(0,head(ret,-1))
result<- (ret<0 & pre.ret>0 & close<pre.one.quar & close>pre.open)
return (result)
}
```

可以看出，这类信号的返回值只有 TRUE 和 FALSE 两种，并不是具体的数值，因此很难直接拿来使用。这些更适合传统的规则型策略，不大适合拿来用在机器学习上。

还有穿刺的形态：

```
piercing <- function(open,close,ret,ratio=0.25) { ## 穿刺指标
  one.quar <- close-(close-open)*ratio
  pre.one.quar <- c(one.quar[1], head(one.quar,-1))
  pre.open <- c(open[1], head(open,-1))
  pre.ret <- c(0,head(ret,-1))
result<- (ret>0 & pre.ret<0 & close>pre.one.quar & close<pre.open)
return (result)
}
```

另外还有启明星的形态：

```
morning.star <- function(open, close, body,thre) { ## 启明星指标
move<- close-open
  pre.move <- c(0,head(move,-1))
  pre.close <- c(close[1], head(close,-1))
jump<- open-pre.close
  pre.body <- c(body[1],head(body,-1))
ratio<- ifelse(pre.body==0,0,body/pre.body)
result<- rep(0,length(open))
chosen<- which(jump<0 & pre.move<0 & move>0 & ratio<thre)
result[chosen] <- -1
return(result)
}
```

有人会问这些K线形态的自动识别是不是就是我们常说的“模式识别”？其实，模式识别一词更多是针对图像分析领域，比如手写数字的识别，但运用的方法还是传统的统计模型，如logistic regression。现在也有人使用更复杂的深度学习模型，在手写数字的识别上，误差率又大为降低。但这类都是标准的机器学习模型，不是那种通过语言描述一种模式（比如先涨一段回调一点再涨起来）然后用计算机去识别的模式。很多人工炒手转量化失败就是因为尝试着去用计算机语言描述这些状态，然后回测验证，这个是失败的模式。毕竟炒手主观交易很多时候也是模糊的逻辑，类似开车，而统计学习这类复杂模型也没有太清晰的逻辑，本质上也是这种模糊的思路，反而能更好拟合。

因此，这类K线形态或许不能直接拿来使用，这也是量化研究跟技术分析的一大区别，技术分析更多是基于这些形态使用的。

美国WorldQuant公司的老板写了一本书《Finding Alphas： A Quantitative Approach to Building Trading Strategies》，里面介绍了构建预测因子及测试的基本步骤，还给出了很多具体因子的例子，读者可以参考。

3.1.4　常见的论文

除了这些书籍以外，还有一些常见的论文介绍了预测因子，如《Automated Trading With Boosting and Expert Weighting》，作者是German Creamer和Yoav Freund，其中Yoav Freund是adaboost的发明者，boosting领域的顶级研究者，而boosting是机器学习领域很常见的概念。

这篇文章使用了boosting的方法来研究交易，即把一些较弱的预测因子合成为较强的预测因子，然后再构建模型，另外还给出了风险管理、投资组合优化等方法，附录给出了全部因子的公式，这些都可以作为构建因子的素材。

另外一篇比较著名的论文是《101 Formulaic Alphas》，里面列举了101个因子的公式，大多数适用于股票，但很多时候用在期货也是可以的。一般来说，用于股票的因子涉及全市场股票之间的相关关系，因此有很多基于排序（rank）的指标，而期货的因子一般只在单一品种上根据价量构建，不需要这么复杂。

3.2 因变量的选择

在机器学习领域，很多人把精力放在构建预测因子上，因为机器学习出现的很多都是分类问题，因变量是显然的。比如手写数字识别，因变量就是 0 ～ 9 这 10 个数字。还有一些图像识别，因变量就是猫头、狗头等图像，这些都是相当显然的。既然因变量如此简单，那么具有研究意义的就只有自变量了，因此就会花更多时间在构造因子上。

3.2.1 因变量的重要性

在金融领域，因变量就不那么显而易见了，需要自己定义。很多人刚开始研究的时候喜欢套用分类模型，也就是把 y 变量分成上涨、不变、下跌 3 类，这样显然是不够精确的。而且如果仅有价格变化的方向而没有幅度，也不足以触发交易。很多时候，价格只在很窄的区间里面震荡，这个时候即便预测对了涨跌的方向，但由于价格波动强度不足以覆盖交易成本，即使交易也很难赚到钱。

因变量的选择多种多样，不同因变量对应不同的策略，通过改变因变量来增加策略的数目相对于改变因子来说是有很多优势的，比如改变因变量后不需要增加新的因子，模型计算量比较固定，特别是实盘中，如果需要开发新的因子和增加新的内存和计算时间的话，改变因变量则一般好很多，计算完就没用了，不需要保存；另外，如果因变量固定，仅仅改变自变量的话，策略并没有什么变化，因为拟合的目标变量没有变化，只是预测的准确度改变了，而如果改变了因变量，则真正改变了策略，这样得出的策略相关性更低。

3.2.2 价格未来变化的对数收益率

以下是较为常见的因变量，之前的研究也是用价格未来变化的对数收益率作为因变量。从数学的角度出发，对数收益率可以累加，方便处理，而且跟实际的收益率相似，因此常常用它来作为实际收益率的近似值。现在来研究一下对数收益率的性质。

```
> log.ret <- c() ## 对数收益
> contract.len <- c() ## 合约长度
> mean.ret <- c() ## 收益的均值
> sd.ret <- c() ## 收益的标准差
> for (contract in all.contracts) { ## 遍历所有合约
+   load(contract) ## 调用合约
+   y <- with(data, fcum(wpr.log.ret, period))[data$continuous] ## 计算
因变量
+   log.ret <- c(log.ret, y) ## 计算对数收益
+   contract.len <- c(contract.len, sum(data$continuous)) ## 合约长度
+   mean.ret <- c(mean.ret, mean(y)) ## 计算均值
+   sd.ret <- c(sd.ret, sd(y)) ## 计算标准差
+ }
> mean(log.ret)
[1] -7.017801e-05
> sd(log.ret)
[1] 0.00713672
```

平均值接近零，标准差是 0.007 136 72，但如果观察每个合约内部的均值和标准差，则有：

```
> mean.ret ## 均值
  [1]  -2.181444e-04  -5.537213e-04   6.646196e-04  -8.597087e-04
9.795995e-05 -3.757381e-04 -4.020932e-04
  [8]  -8.007577e-04  -1.596867e-04  -4.765981e-04  -3.565376e-04
3.421181e-04   4.724662e-04   6.773713e-04
[15]   2.954907e-04
>plot(mean.ret, type="l", main="return") ## 画均值
> sd.ret ## 标准差
  [1]  0.003184312  0.006897623  0.005594518  0.006479099  0.004021878
0.003706708 0.005024155 0.005728694 0.005136765
[10]  0.005325337  0.004549144  0.006057983  0.010973734  0.010639177
0.012857962
>plot(sd.ret, type="l", main="standard deviation") ## 画标准差
```

平均收益率如图 3-2 所示。

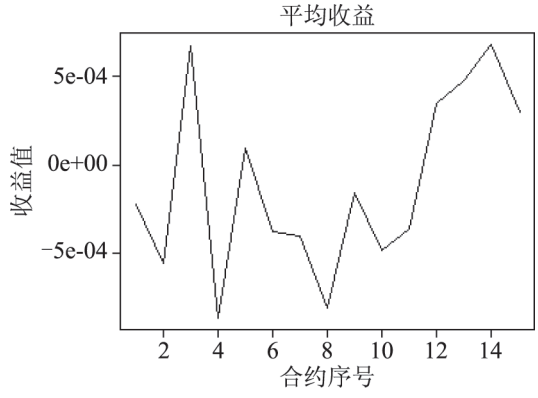


图 3-2　价格平均收益率

由图 3-2 可见价格变化的平均是基本平稳的，在零附近波动，毕竟短期来看价格上不会有明显的趋势，否则整体的趋势会相当明显，但实际上不会有这么明显的趋势。

对于价格波动的标准差，如图 3-3 所示。

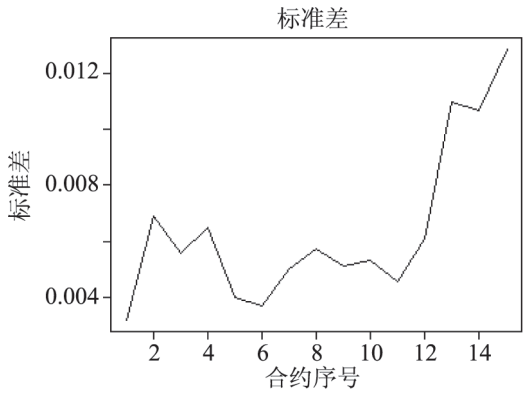


图 3-3　价格波动的标准差

由图 3-3 可见，标准差呈现逐步上升的形态，这说明价格变化的时间序列整体上是不平稳的。

对于不平稳的 y，使用回归模型会不会有问题呢？实际上，对一些趋势性很明显的 y 和 x 来说，有伪回归的嫌疑，因此会对它们进行差分后再进行回归。但现在 y 本身并没有趋势性，只是变化幅度逐渐变大。其实我们这里定义的 y 并不服从独立同分布的假设，并且我们做最小二乘拟合也并没有用到这些假设，我们本质上就是在预测因子所在的线性空间找一个跟因变量 y 距离最近的向量，这并不依赖独立性、平稳性等假设，只要不用到与统计量相关的经典统计学内容，我们做的回归分析都是可行的。

然而，正是由于 y 的幅度会随着时间的变化而变化，而根据波动率聚类的原理，用近期数据做拟合或许会更有意义。因此，如果能更频繁的更新模型，对我们其实是有帮助的。对于 y 的走势，如图 3-4 所示。

代码为

```
plot(log.ret, type="l", main="return")
```

可见，近期的价格波动确实比前面要大很多。

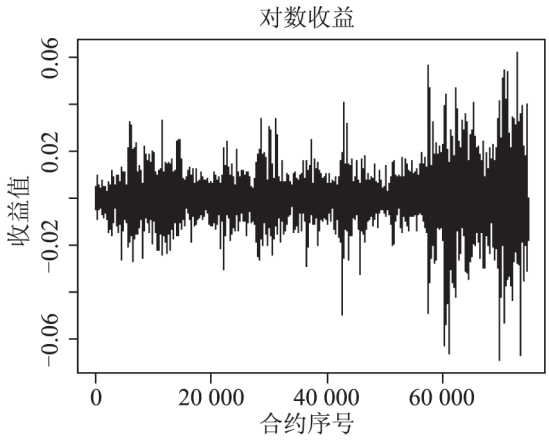


图 3-4　因变量的图形

3.2.3　价格变化的点数

用价格变化的点数代替对数收益率也有一定的道理。首先，与对数收益率类似，价格实际点数的变化也是可加的，这跟实际收益率不大一样。其次，在不考虑资金管理的情况下，研究期货策略一般都是默认固定 1 手，这时本质上获取的是单利而不是复利的收益，考虑到实际的盈亏跟实际的收益其实是一样的，用实际盈亏或许更直观一些。

```
> ret <- c() ## 收益率
> contract.len <- c() ## 合约长度
> mean.ret <- c() ## 平均收益率
> sd.ret <- c() ## 标准差
> for (contract in all.contracts) { ## 遍历所有合约
+   load(contract) ## 调用合约
+   change <- c(0, diff(data$wpr)) ## 价格变化
+    y <- with(data, fcum(change, period))[data$continuous] ## 获得未来收
益率
+   ret <- c(ret, y) ## 加入最新的收益率
+   contract.len <- c(contract.len, sum(data$continuous)) ## 合约长度
+   mean.ret <- c(mean.ret, mean(y)) ## 平均收益率
+   sd.ret <- c(sd.ret, sd(y))
+ }
> mean(ret) ## 收益均值
[1] -0.2581286
> sd(ret) ## 收益标准差
[1] 20.08028
> mean.ret
 [1] -0.9056318 -2.1057190  2.5574370 -3.3016951  0.3448239 -1.3034878
-1.3066634 -2.3057020 -0.4070499
```

```
[10] -1.0920553 -0.6853151  0.6665803  1.1098804  1.9193035  0.9871116
>plot(mean.ret, type="l", main="return")
> sd.ret
  [1]  13.260706  24.446920  21.224836  24.283626  14.752074  13.098001
16.178354 15.780021 12.998219 11.775168
[11]  9.055863 12.001606 25.538411 29.383808 41.622949
```

y 的各个合约的均值，如图 3-5 所示。

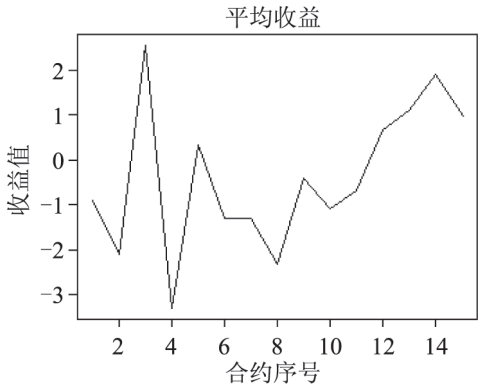


图 3-5　各合约因变量均值

由图 3-5 可见，价格变化的平均值基本不会有什么变化。对于标准差，如图 3-6 所示。

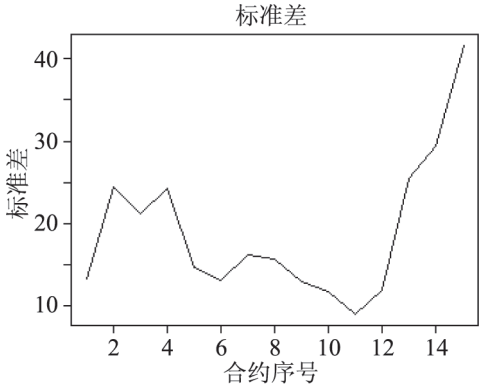


图 3-6　各合约因变量标准差

与之前对数收益率类似，按实际价格变化的标准差计算的话也是呈现出最近上涨的趋势，也不是平稳的。下面分析一下几个指标在这个 y 值的表现。

```
> cor.vec <- as.numeric(sapply(all.contracts, function(contract){ ## 计
算因子跟y的相关性
```

```
+    load(contract) ## 调用合约
+    x <- with(data, adx.5m(high, low, close, period))[data$continuous]
## 计算因子
+    change <- c(0, diff(data$wpr)) ## 价格变化
+    y <- with(data, fcum(change, period))[data$continuous] ## 计算因变量y
+    cor(x,y) ## 计算相关系数
+ }))
> cor.vec
  [1]    0.0273661966  -0.0057789437    0.0316919702  -0.0540117359
-0.0009170374   0.0040615262 -0.0555682043
  [8]  -0.0950403520  -0.0292980555    0.0123674084  -0.0243875696
0.0577538637 -0.0029906528 -0.0024970121
[15] -0.0124239078
>mean(cor.vec)/sd(cor.vec)
[1] -0.2610164
```

之前对数收益对应的结果是 −0.265 511 2，可见改变 y 之后两者相差不大。我们再看一个例子：

```
> cor.vec <- as.numeric(sapply(all.contracts, function(contract){
+    load(contract)
+    x <- with(data, clv.5m(high, low, close))[data$continuous]
+    change<- c(0, diff(data$wpr))
+    y <- with(data, fcum(change, period))[data$continuous]
+    cor(x,y)
+ }))
> cor.vec
  [1]   0.015110306   0.006024411   0.035825368   0.003805619   0.002934998
-0.005962322   0.011347802   0.025071703
  [9]   0.015903474   0.008110899   0.009623460   0.009239691 -0.002869892
0.004348989 -0.006186146
>mean(cor.vec)/sd(cor.vec)
[1] 0.7870768
```

之前对数收益的结果是 0.787 369 1，可见相差也不大。因此，这两个因变量的选择区别不大。

由此可见，简单改变一下 y 值或许并不会发生多大的变化。使用这种绝对金额的变化而不是对数收益率还有一个好处，就是对于做套利交易的策略，每次交易一个品种时，买一个卖一个，本质上就是交易价差合约。期货的价格一定是大于零的，但价差却可能是负数，负数没有对数，不存在对数收益率，因此这时只能用价格本身的变化作为因变量。

3.2.4　小波等去噪方法

前面章节提到过，改变因变量是比改变自变量更好的选择，其中另外一个

好处在于无论因变量的构造如何复杂，在实盘中我们是不需要实现的，实盘中我们只需要把自变量线性相加，得到因变量的预测值，而不需要构造因变量的实际值，况且实盘中也得不到因变量的实际值。

因此，我们可以用一些更高级的方法来处理因变量，而不需要对自变量进行处理。如果我们对自变量也这么处理，那么我们也必须在实盘中对自变量进行相应的处理，这样一来编程量会非常巨大，而得到的回报未必很多。

下面来看一下小波分析的影响。首先把原来的因变量 y 整理出来，如图 3-7 所示。

```
> library(wavelets) ## 调用小波包
> all.ret <- c() ## 所有收益
> len <- c() ## 合约长度
> for (contract in all.contracts) { ## 遍历所有合约
+   load(contract) ## 调用合约
+    x <- with(data, clv.5m(high, low, close))[data$continuous] ## 计算
clv指标
+   change <- c(0, diff(data$wpr)) ## 价格变化
+   y <- with(data, fcum(change, period))[data$continuous] ## 计算未来收益
+   all.ret <- c(all.ret, y) ## 加入新的y值
+   len <- c(len, sum(data$continuous)) ## 加入合约长度
+ }
> ret.ts <- as.ts(all.ret) ## 转成时间序列
>plot(ret.ts, type="l")
```

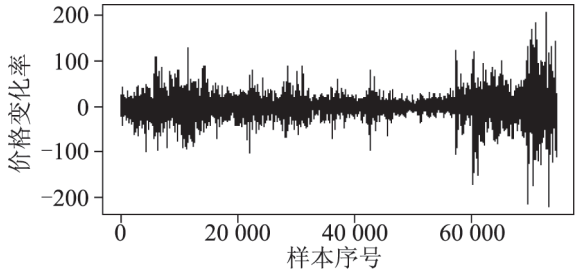


图 3-7　因变量的值

其中用到的软件包是 wavelets，如果大家对小波分析在时间序列的应用感兴趣，可以参考这本书《Wavelet Methods for Time Series》，作者是 Donald B. Percival 和 Andrew T. Walden，这个软件包跟这本书是相对应的。

小波算法本质上可以理解为滤波，即对信号进行过滤，希望把噪音过滤掉。对于简单的时间序列，或许本身噪音就不多，因此使用小波基过滤之后恢复的序列可以跟原序列一样。对于金融时间序列，噪音比较多，过滤后的序列或许会跟原来不一样，我们可以详细分析一下。下面的这条命令：

```
> ret.ts <- as.ts(all.ret)
```

是把向量转成时间序列，这样才能用小波函数进行分析。

```
> model<- dwt(ret.ts, filter="haar", n.levels=3, boundary="reflection")
## 用离散小波变换，使用
## haar滤波
> imodel<- idwt(model, fast=TRUE) ## 用逆变换返回生成的信号
>length(imodel)
[1] 74565
>length(ret.ts)
[1] 74565
> cor(imodel, ret.ts) ## 滤波前后的相关性
[1] 0.9070177
```

这里我们使用了 dwt() 这个函数进行分析，dwt() 是 discrete wavelet transform 的缩写，离散小波变换，这里使用的滤波是 haar 滤波，用了 3 层滤波，对边界点的处理是 reflection，它还有一个选项是 periodic，我们可以试试：

```
>model<- dwt(ret.ts, filter="haar", n.levels=3, boundary="periodic")
>imodel<- idwt(model, fast=TRUE)
>length(imodel)
[1] 74565
>length(ret.ts)
[1] 74565
>cor(imodel, ret.ts)
[1] 0.7274209
```

原来 reflection 是 0.907，现在 periodic 是 0.727，效果差很多，毕竟原来的时间序列不是周期函数，所以还是用 reflection 比较好，它相当于用近期的数据进行填充。一般收益序列有波动性聚类，用近期的数据进行填充会比较好。

我们这里先用了 dwt() 得到小波基的各项系数，然后用 idwt() 把信号还原，得到的时间序列长度跟原来是一样的，都是 74565，后来的数据相当于过滤之后的收益序列，它们的相关性高达 90%，我们希望的结果是把噪音过滤掉，保留信号。我们可以改变 n.levels 试一下：

```
>model<- dwt(ret.ts, filter="haar", n.levels=4, boundary="reflection")
>imodel<- idwt(model, fast=TRUE)
>length(imodel)
[1] 74565
>length(ret.ts)
[1] 74565
>cor(imodel, ret.ts)
[1] 0.4866929
```

可见多加一层滤波之后相关性下降了不少。我们现在只分析 n.levels=3 的情况。

```
> cor.vec <- c()
> start <- 1
>end<- 0
> i <- 1
>for (contract in all.contracts) {
+   load(contract)
+   x <- with(data, clv.5m(high, low, close))[data$continuous]
+   end<- end+len[i]
+   i <- i+1
+   cor.vec <- c(cor.vec, cor(x,imodel[start:end]))
+   start<- end+1
+ }
>mean(cor.vec)/sd(cor.vec)
[1] -2.070912
> cor.vec
 [1] -0.022735178 -0.031990655  0.002866996 -0.025221517 -0.029966868
-0.044399077 -0.020665077
 [8] -0.014877282 -0.015121824 -0.019714229 -0.019631380 -0.022820007
-0.043046340 -0.037073527
[15] -0.035691915
>sum(cor.vec<0)/length(cor.vec)
[1] 0.9333333
```

可见，均值 / 标准差是 −2.07，这是相当稳定的表现，比之前 0.78 的水平好很多。另外 93% 的数值都是负数。或许大家比较奇怪的是为什么原来是正数现在成了负数？要知道这两个目标向量的相关性高达 90%，为什么会出现这么大的差异？这也正是金融数据的特点，在极端高波动的情况下，往往跟低波动情况呈现出相反的特征，而波动太高的行情往往难以持续，而且风险也大，需要避免交易，而小波分析很好地考虑到了这点。

大家可以尝试使用其他的滤波，由于只是改变了因变量，预测因子没有改变，因此实盘程序不需要做任何修改，只需要保存系数的配置文件修改就可以了，这也是改变因变量的好处。低频的因子和因变量就介绍到这，下面章节介绍高频因子。

3.3 高频因子

高频因子指基于分笔数据的因子，国内是指基于 500 毫秒截面数据的因子，这类因子一般会用到买卖价格和挂单量，即盘口信息。一般来说，可以分为三种因子：基于买卖挂单量的因子、基于成交量和基于价格变化的因子。另外，如果有 5 档行情，还可以开发基于深度行情的因子。

我们拿螺纹钢2016年11月17日至2017年6月30日一共150个交易日的日内分笔数据作为例子来分析。高频因子分析的一个特点是计算量和存储量都特别巨大，耗时比较长。简单起见，我们只选取了少数几个因子来分析。

我们先选取前面40天的数据，一共11个因子，加上因变量y，一共12个。

```
> range <- 1:40 ## 取1-40天的数据
> product <- "rb" ## 品种是螺纹钢
> all.data <- prepare.tick.data(signal.list, y.str, product, all.dates[range]) ## 准备分笔数据
>dim(all.data)
[1] 1609232      12
```

足足有160多万个样本，由此可见分笔数据的训练量是非常大的。我们可以挑选其中几个因子来看。

3.3.1　基于买卖挂单量的因子

这是高频数据比较独特的地方，它对买卖盘口比较依赖，因此很多时候可以考虑用挂单量来构造因子。比如：

```
size.imb<- function(bid, ask, bid.size, ask.size) { ## 买卖不平衡指标
return ((bid.size-ask.size)/(bid.size+ask.size)/(ask-bid)/(ask+bid)*2);
}
```

由于数据庞大，我们按照每10万数据量一组，计算各组的指标与因变量的相关系数，如图3-8所示。

```
all.data$id <- ceiling(1:nrow(all.data)/100000) ## 分笔数据分组
library(plyr) ## 调用plyr包
sample.cor <- ddply(all.data, "id", summarise, corr=cor(size.dif, y)) ## 每组数据内部相关性
plot(sample.cor$corr, type="l")
```

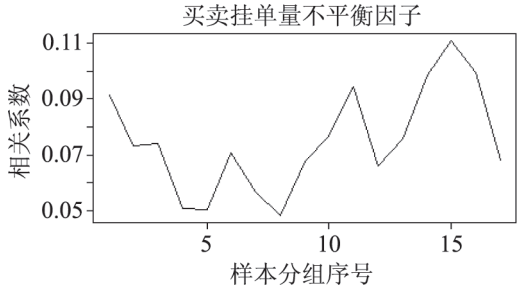


图3-8　挂单量因子相关性

由图3-8可见，相关系数在5%～11%，基本保持正相关性，还是非常稳定的。

3.3.2 基于成交量的因子

第二类是基于成交量的因子，我们也选取了其中一个：

```
trade.imb<- function(bid, ask, volume, turnover,period) { ## 成交量买卖不
平衡
    get.trade <- active.trade(bid, ask, volume, turnover)
    buy.trade <- cum(get.trade$buy.trade,period)
    sell.trade <- cum(get.trade$sell.trade,period)
buy.trade[1:period] <- 0
sell.trade[1:period] <- 0
volume[1] <- 0
    cum.volume <- cumsum(volume)/(1:length(volume))*period
signal<- (buy.trade-sell.trade) %0/% cum.volume
return (signal)
}
```

这是衡量主动买量与主动卖量对比的因子，我们可以看看它的表现，如图 3-9 所示。

```
> sample.cor <- ddply(all.data, "id", summarise, corr=cor(trade.imb.32,
y))
>plot(sample.cor$corr, type="l", main="trade.imb.32", ylab="cor")
```

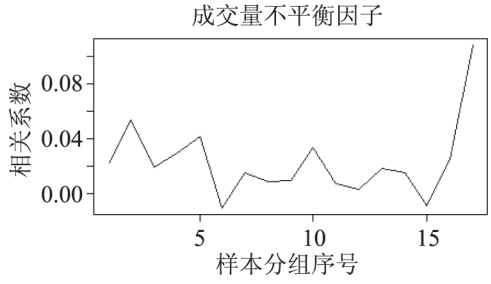


图 3-9 成交量因子相关性

由图 3-9 可知该因子在最近的行情中略有上升，整体也是正相关，但没有基于挂单量的因子表现好。一般来说基于挂单量的因子有最强的预测力，但预测出来的波动并不大。

3.3.3 基于价格变化的因子

对于中低频来说最常见的因子都是基于价格变化的，高频也可以用这类因子，比如最简单的基于指数加权平均线的因子。

```
ewma.move<- function(bid, ask, bid.size, ask.size, period) { ## 标准化的
指数加权平均因子
wpr<- clean.x((bid*ask.size+ask*bid.size)/(bid.size+ask.size))
signal<- clean.x((1-ewma(wpr, period)/wpr))
    first.negative <- which(signal<0)[1]
signal[1:first.negative] <- 0
return (signal)
}
```

可以看看这个因子的效果，如图 3-10 所示。

```
> sample.cor <- ddply(all.data, "id", summarise, corr=cor(ewma.move.32,
y))
>plot(sample.cor$corr, type="l", main="ewma.move.32", ylab="cor")
```

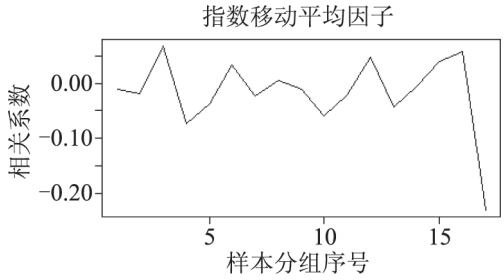


图 3-10　价格因子相关性

由图 3-10 可见这是一个负相关的因子，其实无论是正相关还是负相关对结果并没有什么影响，加入一个负号即可改变。

3.3.4　多因子预测模型

现在，我们给出一个多因子的预测模型。我们用 40 天的数据进行交叉验证来建模——即每 10 天的数据为一组，一共 4 组，每次预留一组作为验证集，然后其他 3 组作为训练集，这样一来我们可以获得 40 天的交叉验证结果，如图 3-11 所示。

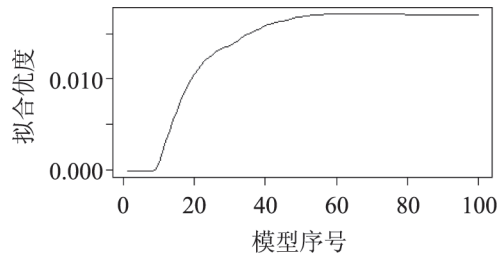


图 3-11　高频模型拟合优度

```
> system.time(tick.model <- get.tick.lasso.cross(range=1:40, product,
signal.list, y.str,chunk=chunk)) ## 用cross-validation的lasso模型进行建模
best= 66  ## 最佳模型的编号
用户系统流逝 ## 总共运行时间
38.08  1.98 40.18
```

由图 3-11 可以看出，最佳的是第 66 个模型。下面我们用这个模型来预测第 41 ～ 100 天：

```
> load(paste("d:/liwei/tick model/", product, ".", range[1], ".",
tail(range,1),".RData",sep=""))
## 调出模型
> tick.model$coef <- tick.model$coef.mat[,66] ## 设置模型系数为最佳模型的系数
> chosen <- which(tick.model$coef!=0) ## 过滤出非零系数
> signals <- tick.model$signals[chosen]
> coef <- tick.model$coef[chosen]
> length(coef)
[1] 11
> all.dates <- get.dates(product)
> r2 <- rep(0,n.dates)
> for (i in 1:n.dates) {
+   date <- all.dates[i]
+   data <- prepare.tick.data(signals, y.str, product, date)
+   pred <- as.matrix(data[,1:(ncol(data)-1)]) %*% coef
+   r2[i] <- R2(pred, data$y)
+    save(pred, file=paste("d:/liwei/pred tick/", product, "/", date,
".RData", sep=""))
+ }
```

系数的数目还是 11 个，由此可见所有因子都得以保留，我们可以考查一下样本内外的拟合优度：

```
> train <- 1:40
> test <- 41:150
> mean(r2[train])
[1] 0.01757111
> median(r2[train])
[1] 0.01687339
> mean(r2[test])
[1] 0.01136023
> median(r2[test])
[1] 0.01129559
```

可见样本内是 0.017 左右，样本外是 0.011 左右，样本外会比样本内差一些。事实上，很多时候并不一定选样本内交叉验证最好的模型，也可以选一些次优模型，这样的话在样本外或许会有更佳的表现，比如我们随机抽取一个次优的模型 best=40：

```
> tick.model$coef <- tick.model$coef.mat[,40]
> chosen <- which(tick.model$coef!=0)
> signals <- tick.model$signals[chosen]
```

```
> coef <- tick.model$coef[chosen]
> length(coef)
[1] 9
> all.dates <- get.dates(product)
> r2 <- rep(0,n.dates)
> for (i in 1:n.dates) {
+   date <- all.dates[i]
+   data <- prepare.tick.data(signals, y.str, product, date)
+   pred <- as.matrix(data[,1:(ncol(data)-1)]) %*% coef
+   r2[i] <- R2(pred, data$y)
+    save(pred, file=paste("d:/liwei/pred tick/", product, "/", date,
".RData", sep=""))
+ }
> train <- 1:40
> test <- 41:150
> mean(r2[train])
[1] 0.01512204
> median(r2[train])
[1] 0.01475278
> mean(r2[test])
[1] 0.01376843
> median(r2[test])
[1] 0.01289922
```

这时样本内降低为0.015 1，但样本外提高至0.013 8，这样就比较接近了，而注意到此时模型因子的数目为9，比之前少了一些，同时起到了筛选因子的作用。进一步地，如果我们选择best=35的模型，则有：

```
> tick.model$coef <- tick.model$coef.mat[,35]
> chosen <- which(tick.model$coef!=0)
> signals <- tick.model$signals[chosen]
> coef <- tick.model$coef[chosen]
> length(coef)
[1] 7
> all.dates <- get.dates(product)
> r2 <- rep(0,n.dates)
> for (i in 1:n.dates) {
+   date <- all.dates[i]
+   data <- prepare.tick.data(signals, y.str, product, date)
+   pred <- as.matrix(data[,1:(ncol(data)-1)]) %*% coef
+   r2[i] <- R2(pred, data$y)
+    save(pred, file=paste("d:/liwei/pred tick/", product, "/", date,
".RData", sep=""))
+ }
> train <- 1:40
> test <- 41:150
> mean(r2[train])
[1] 0.01393324
> median(r2[train])
[1] 0.01397686
> mean(r2[test])
[1] 0.01410278
> median(r2[test])
[1] 0.01319343
```

这时样本内数据是 0.0139，样本外数据是 0.0141（平均值），样本外数据甚至比样本内数据还好，这说明降低模型复杂度之后，模型在样本内的拟合程度下降了，但是在样本外的预测能力提升了。

3.3.5 对 R^2 的理解

如果是这类分笔数据建模，预测时间也只有未来几十个 tick 的话，那么 R^2 一般都会是正数。R^2 取值较低的原因一般是行情急剧暴涨暴跌，其实也不难理解，这种情况下人工也无法判断，或者是由于“乌龙指”使然，完全在模型预料之外，因此预测值会很低。

其它情况下，对于价格低位震荡时，R^2 会比较高，因为此时价格波动一跳都比较困难，整体 y 的取值变化不大，因此也更容易预测。但要注意的是哪怕预测得十分准确也无法赚钱，因为价格波动实在太小，无法覆盖手续费和滑点等成本。对于价格波动幅度稍大的情况，R^2 会略低一些，但是由于波动幅度可以覆盖手续费和滑点，因此这时候盈利也是可能的。

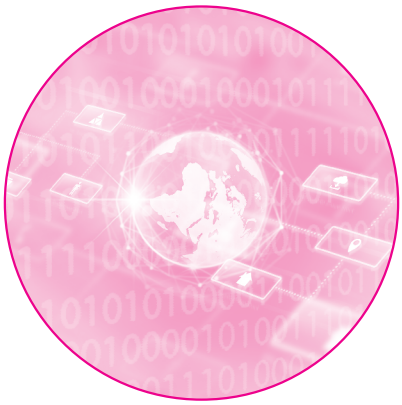
由此可见，现实中 R^2 高不一定就盈利高，只能说在同一段行情下，样本外的 R^2 是越高越好。

3.4 本章小结

本章主要介绍了预测模型中的因子，包括中低频因子以及高频因子，为下一章建模做准备。高频因子属于比较特殊的因子，由于本书重点是中低频交易，因此在介绍高频因子的同时，也顺带介绍了高频因子建模的一些例子，并给出了样本内和样本外的一些结果，读者也可以先对建模的基本步骤有个大概的了解，为下一章统计建模打下一定的基础。下一章将重点介绍各类统计建模的方法。

第四章

基础统计模型



上一章介绍了一些基本的预测因子，这一章介绍如何用这些因子构造模型。常用的预测模型包括线性回归、决策树、神经网络、深度学习等。由于金融数据的高噪音特性，这类预测模型的主要问题是过度拟合而不是欠拟合，因此，这里主要讨论普通的线性回归模型。

4.1 线性回归

线性回归模型一般描述如下：

$$Y=\beta_0+\beta_1X_1+\beta_2X_2+\cdots+\beta_pX_p+E_i$$

其中 Y 是因变量，X_i 是第 i 因子，β_i 是 i 因子的系数，它量化了自变量与因变量之间的关系。β_i 可以理解为在其它因子固定的情况下，X_i 每增加一个单位对 Y 的平均影响。

有了模型之后，我们可以把数据分为训练样本和测试样本两部分，其中训练样本用来拟合系数的值，而测试样本则用来检验拟合的效果。在这里，我们以螺纹钢期货（rb）为例来阐述建模的过程。

4.1.1 为什么用线性模型

很多人会问为什么使用线性模型，而不使用其他更复杂的模型，比如非线性模型，或者多个层次的神经网络模型？其实，我们可以从以下几个方面考虑。

首先，从上一章结尾对分笔数据建模的分析中可以看出，即便是预测短短几十个 tick 的价格变化，我们的样本外 R^2 也只有 1% 左右。很多初步研究量化金融预测模型的人甚至会以为自己弄错了，认为误差是 1%，R^2 是 99% 才对。但实际上，金融数据样本外能预测出来的 R^2 确实不高，一般预测几十 tick 或许加入几十个因子之后也只有 6% 左右，如果是中低频的会更低，可能千分之几。因此，金融问题跟其他问题很不一样，我们习惯于分类问题总是 99%、99.9% 的识别率，比如识别一幅图有没有猫，或者识别手写的数字等，这些问题都可以达到非常高的准确度，但金融问题会比较难。

其次，我们从金融问题的性质出发。有人说金融问题是时间序列的，可以用那种长短期记忆网络（Long Short-Term Memory Network，LSTM）来解决。但事实上，LSTM 更多运用在自然语言处理这种样本更为稳定的问题中，比如英语的说话习惯不会怎么变的话，过去的意思对未来的影响才能更好地体现。更本质地，人们说话的根本目的是让别人听得懂，尽可能地减少自己的噪音，

让信号更好地表达出来。但金融不同，人们交易的目的是为了自己赚钱，并且尽量隐藏自己的意图，不让别人清楚自己想干什么。因此，这也导致金融数据分析在本质上是一个博弈问题，需要不断变化，而不是一个客观的事物放在那里任由人们去分析。

最后，就是金融数据本身的高噪音低信号的特性，使复杂的模型很容易拟合了样本内数据的噪音部分。诚然，大部分好一些的模型，在拟合迭代过程中，一般都会先拟合信号部分，再拟合噪音部分。但是有可能噪音太多而信号太微弱，拟合起来也不是那么容易的。

还有就是统计学中的 bet on sparsity 的押注稀疏性原则，因为现实问题可能是稀疏的（参数较少，模型结构简单），也可能是稠密的（参数较多，模型结果复杂），但我们并不知道真实的数据产生机制，只能用模型去逼近它。如果真实模型是稀疏的，我们用稀疏去逼近，这是好的；如果真实模型是稠密的，我们用稀疏去逼近，最多欠拟合，也不会有太大问题，在金融里面表现为交易稀少但是能赚钱，只是赚的少而已。但如果真实是稀疏的，我们用稠密模型去逼近，那就很容易过度拟合了，在金融里面会更严重，可能导致频繁交易而亏钱；如果现实模型是稠密的，用稠密模型去逼近，由于模型结构复杂，参数较多，也未必能估计准，到时候效果还是可能不好。深度学习模型虽然也复杂，但是数据较多且分布稳定，所以没有上述这些问题。但金融领域往往没有这么多高质量的数据。

综上所述，我们先从线性模型出发来研究建模的问题，如果还学有余力，再慢慢过渡到其他更复杂的模型。

4.1.2　数据准备

我们的数据库是 rb1210 合约至 rb1705 合约，都是选取主力合约：

```
> all.contracts
[1] "rb1210.RData" "rb1301.RData" "rb1305.RData" "rb1310.RData"
"rb1401.RData" "rb1405.RData"
[7] "rb1410.RData" "rb1501.RData" "rb1505.RData" "rb1510.RData"
"rb1601.RData" "rb1605.RData"
[13] "rb1610.RData" "rb1701.RData" "rb1705.RData"
```

数据量为：

```
>dim(all.data) ## 看看行列的结构
[1] 74565    21
```

这主要是行情信息，并不是因子。这里的数据是 5 分钟的 K 线数据，而且把 K 线结束时的盘口信息也包含在内。此外，我们建模一般只考虑连续合约。

各变量含义如下：

```
>names(all.data)  ## 数据列名
[1] "contract"   "date"     "trade.date"  "time"     "price"    "open.int"
[7] "qty"        "bid"      "ask"         "bid.qty"  "ask.qty"  "open"
[13] "high"      "low"      "close"       "continuous" "good"   "wpr"
[19] "wpr.ret"     "wpr.log.ret" "date.time"
```

其中，各变量含义如表 4-1 所示。

表 4-1 变量含义

变　量	含　义
Contract	合约名称
Date	当前日期
trade.date	交易日期
Time	时间
Price	最新价
open.int	持仓量
Qty	成交量
Bid	买一价
Ask	卖一价
bid.qty	买一量
ask.qty	卖一量
Open	开始价
High	最高价
Low	最低价
Close	结束价
Continuous	TRUE 表示连续合约
good	TRUE 表示无涨跌停
wpr	挂单量加权价
wpr.ret	价格变化量
wpr.log.ret	对数收益率
date.time	日期时间

现在我们可以把数据集划分成训练集和测试集两部分，其中前 6 个合约是训练集，后面 7、8、9 合约是测试集：

```
> n.train <- 6 ## 训练集数量
> train.range <- 1:n.train ## 训练集范围
> test.range <- (n.train+1):9 ## 测试集范围
> train.contracts <- all.contracts[train.range] ## 训练集合约
> test.contracts <- all.contracts[test.range] ## 测试集合约
> train.mat <- prepare.data(model$signals, model$y, product, train.
contracts, over.night=-1)
## 训练集数据
> test.mat <- prepare.data(model$signals, model$y, product, test.
contracts, over.night=-1)
## 测试集数据
> colnames(train.mat) <- c(paste("x",1:19,sep="."),"y") ## 给训练集命名
> colnames(test.mat) <- c(paste("x",1:19,sep="."),"y") ## 给测试集命名
> dim(train.mat) ## 训练样本规模
[1] 21825    20
> dim(test.mat) ## 测试样本规模
[1] 12756    20
```

可见，有 21 825 个训练样本，12 756 个测试样本，19 个因子，1 个因变量。这里的因变量选取的是未来 16 根 K 线之后的价格对数收益率。对数收益率具有对称性，比如价格从 80 涨到 100，再从 100 跌回 80，如果使用传统收益率，价格先上涨 25%，再下跌 20%，关系无法累加，但如果用对数收益率，则 log(100)−log(80) 与 log(80)−log(100) 互为相反数，相加后为零，方便处理。训练样本因变量 *y* 的图形如图 4-1 所示。

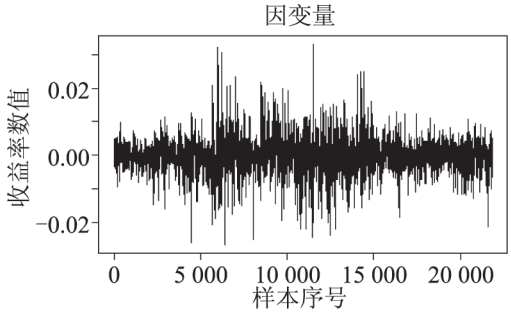


图 4-1　y 的取值

从图形看 *y* 的取值还是比较平稳的，即均值为零，方差有限。取值大小可以达到 2% 左右，这对于期货合约的价格变化而言应该算是非常厉害的，毕竟我们预测的是 16×5=80 分钟。

当然，这类价格收益率一般而言并不满足正态分布，有时甚至不是平稳的，而是肥尾分布。对于趋势类策略，很多时候就是为了在肥尾时候能赚钱，而不

是为了捕捉平稳时候的价格波动，因此处理的时候往往不会把波动大的行情过滤掉，有时也不会对指标进行相应的处理。

在期权定价领域，期权的卖方往往会担心肥尾效应，或者是更极端的“黑天鹅”事件，因为此类事件发生后，价格会剧烈波动，期权买方会行权获利了结，期权卖方则会亏钱，而且这种亏损理论是无上限的。历史上著名的量化对冲基金长期资本管理公司（LTCM，Long Term Capital Management）就是因为类似的策略导致损失惨重且最后倒闭。但对量化 CTA 而言却是完全不一样的。在价格剧烈变化时，比如导致长期资本破产的长期债券价格大幅上涨和短期债券价格大幅下跌等事件发生时，如果是做债券期货，例如国内的 10 年期债券期货价格大幅上升，5 年期债券期货价格大幅下跌，那么作为 CTA 趋势追踪策略，它大概会追着买入 10 年期债券期货，卖空 5 年期债券期货，从而两边获利，并且这类系统会有比较好的止损体系，如果价格反转或平稳的话就会平仓。

长期资本策略本质上是套利策略，除非加入强硬的资金止损，否则策略本身并没有很好的止损逻辑，哪怕他们是诺贝尔奖得主，或者华尔街交易老手，其实也是一样的，策略的逻辑来来去去都是那么几种，并不复杂。很显然长期资本一方面做套利类策略，另一方面却没有很好的止损逻辑，还有就是高杠杆，而且是流动性差的合约，想止损都难，因此一个月内亏损 90% 也不足为奇。但我们研究的期货市场完全是另外一个世界，甚至可以说是完全相反的两个世界。

期货合约在交易所交易，没有对手方违约的风险，正常情况下没有流动性风险，趋势策略也不惧怕价格上的“黑天鹅”。当然，如果是乌龙指一类的“黑天鹅”，瞬间把价格打到涨跌停板，策略很可能会判断失误，这需要在程序中单独处理。毕竟所有模型都是基于市场流动性好的情况下才成立的，如果流动性不好，则模型失效概率比较大，要做专门处理。

下面章节我们来看看样本内的情况。

4.1.3 样本内的情况

我们在 R 语言里面建立线性回归模型来分析样本内的情况：

```
> fit <- lm(y~.+0, data=train.mat) ## lm是拟合线性回归的函数
> summary(fit) ## 看看模型的总结
```

```
Call: ## 运行结果
lm(formula = y ~ . + 0, data = train.mat) ## 公式

Residuals: ## 残差统计
      Min        1Q    Median        3Q       Max
-0.027096 -0.002527  0.000029  0.002510  0.033085

Coefficients: ## 稀疏的结果
       Estimate Std. Error t value Pr(>|t|)     ## 估计值、标准误、t值、p值
x.1  -4.359e-04  2.129e-04  -2.047 0.040630 *
x.2  -5.329e-06  1.244e-04  -0.043 0.965841
x.3  -7.059e-04  3.969e-04  -1.778 0.075354 .
x.4   1.150e-03  1.297e-03   0.887 0.375320
x.5   2.227e-03  7.045e-04   3.162 0.001571 **
x.6   8.518e-05  3.940e-04   0.216 0.828855
x.7  -5.025e-04  4.497e-04  -1.117 0.263796
x.8  -1.274e-03  3.495e-04  -3.646 0.000267 ***
x.9  -6.205e-03  3.643e-03  -1.704 0.088483 .
x.10 -1.087e-03  4.930e-04  -2.204 0.027513 *
x.11 -3.040e-05  3.566e-05  -0.853 0.393906
x.12  1.859e-02  1.757e-02   1.058 0.290034
x.13 -4.233e-02  2.419e-02  -1.750 0.080124 .
x.14  3.110e-03  1.036e-03   3.001 0.002693 **
x.15  2.600e-04  3.857e-04   0.674 0.500247
x.16  1.606e-02  2.037e-03   7.885 3.28e-15 ***
x.17  1.246e-03  2.153e-03   0.579 0.562863
x.18 -2.662e-03  8.977e-04  -2.966 0.003022 **
x.19  1.236e-03  2.937e-04   4.208 2.58e-05 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.005083 on 21806 degrees of freedom
Multiple R-squared:  0.01625,    Adjusted R-squared:  0.01539
F-statistic: 18.96 on 19 and 21806 DF,  p-value: < 2.2e-16
```

其中 lm 是 R 语言进行线性回归的命令。

另外，我们可以看到拟合优度 R^2 的数值是 0.016 25，而调整后的 R^2 是 0.015 39，这两个数值看起来都不是特别高。一般来说，非金融市场类的统计问题一般都有很高的 R^2，比如 80% 以上，但是在金融市场数据方面，一般来说拟合程度都不会特别高。而且，从模型选择的角度而言，拟合程度的具体数值并不重要，我们更看重模型间拟合程度的排序关系，选择拟合程度较好的即可。

需要注意的是，由于 y 的取样是带有重复的，比如从 0 时刻取 16 根 K 线到第 16 时刻，从 1 时刻取到第 17 时刻，中间有 15 个区间有重合，因此各样本之间不是独立的。从统计学角度来说，这种情况下估算出来的系数值也是有意义的，毕竟最小二乘算法并不需要假设样本独立，因此计算出来的拟合优度也是正确的，但 t 值和 p 值等依赖统计分布样本独立的指标计算结果则不一定

正确。

例如，传统来说，对应因子后面有“*”符号的表示显著，“*”的数目越多表示结果越显著。我们可以看到 x8，x16，x19 都有“***”的符号，表示这几个因子结果非常显著，另外还有不少带“**”和“*”的因子，也有部分因子并不显著。但由于样本不独立，这些计算结果未必正确。

很多统计书会采用各种变量选择方法，如前向选择等、后向选择，利用 AIC、BIC 等指标，但那些更多是传统的统计学理论，往往对数据分布有一定假设。既然我们现在的数据不满足独立性假设，那么按这些统计理论选择的模型也未必正确。

下面章节我们看看样本外的情况。

4.1.4 样本外的情况

我们把计算的系数代入样本外的数据，很容易计算出样本外的预测值。

```
> pred <- predict(fit, newdata=test.mat) ## 测试集预测值
> R2(pred, test.mat$y) ## 样本外R平方
[1] -0.002399677
```

其中，R^2 是 caret 包里面的函数，但最新的版本似乎删除了，大家可以自己补上：

```
> R2<- function(pred, obs, formula = "corr", na.rm = FALSE) { ## R平方的
函数
        n <- sum(complete.cases(pred))
        switch(formula,
                                corr = cor(obs, pred, use = ifelse(na.rm,
"complete.obs", "everything"))^2,
                            traditional = 1 - (sum((obs-pred)^2, na.rm =
na.rm)/((n-1)*var(obs, na.rm = na.rm))))
      }
```

这里计算出的拟合优度的值是个负数。很多读者第一次见或许会觉得奇怪，因为按常理来说，拟合优度应该是一个非负数。对样本内来说，线性回归拟合的几何含义是把 y 垂直映射到自变量 X 所在的线性空间，类似于一个直角三角形，原来的 y 是斜边，它的长度大于任意一条直角边，拟合优度也会是正数。但对于样本外来说，线性空间没变，但现在相当于 y 可以随便映射到 X 空间的任何一个点，其长度可以很长，甚至比 y 还长，这样计算下来的拟合优度自然

是负数。另外，从拟合的角度来看，样本外的数据可以跟样本内的数据不一样，我们无法保证样本内拟合的模型对样本外还具有预测能力。

一般来说，直接用平均值作为预测值的拟合优度是零，拟合优度为负数说明预测能力还不如直接用平均值来预测，显然这种结果是不能令人满意的。到目前为止，我们也无法知道模型是过度拟合还是欠拟合而导致在样本外效果不佳，因此还需要做其他的分析。

4.1.5　关于 t-value 等统计量

在做进一步分析之前，我们先来看看 *t* 值、*p* 值等统计量。一般来说，预测模型在统计学和机器学习领域都有出现，很多模型如支持向量机、决策树等，既可以看成属于统计学的领域，也可以看成属于机器学习的领域。传统来说，统计学的参数估计、假设检验等跟“预测”并没有太直接的关系，或者说 98% 的统计学都与预测无关。统计学传统研究的一般是小样本的情况，需要对样本做一些假设，设计一些统计量，然后用统计量的数值对照理论上的概率分布，从而得出是否要拒绝原假设的结论。在线性回归里，原假设的意思是认为系数是零，即系数对应的因子没有作用。如果拒绝原假设，则认为因子有预测能力。这里我们主要关注 *t* 值、*p* 值、Z-score 等。

然而，前文也提到，这些数值的计算过于依赖样本的独立性以及各种概率分布的假设，金融数据一般不能严格满足这些条件。另外，从预测的角度出发，更多的是希望样本外的误差越小越好，或者同理，样本外的拟合优度越大越好，而这些是不依赖样本独立性和数据概率分布的。因此，在预测问题上，也不必太拘泥于使用 t 值等统计理论。

下面章节我们就来看看机器学习对待样本外效果不佳的处理办法。

4.2　带约束的线性回归

样本外效果不佳的可能原因有两种：一是模型欠拟合，即模型不够复杂，而数据太复杂，拟合效果不好；二是模型过拟合，即模型把很多噪音也拟合了

进去，而这些在样本外的数据并不会有所体现，因此在样本外就表现不佳了。为此，我们需要一个验证集来检验。

4.2.1 验证集的使用方法

机器学习一般使用验证集来作为过渡，对模型进行微调。在机器学习中有正则化（regularization）的概念，在统计学里面一般称之为收缩性（shrinkage）。其实很好理解，线性回归计算系数本质上是用最小二乘法来计算，它计算出的系数是无偏估计，对样本内来说是最优的，但这样计算出来的系数方差比较大，可能过度拟合了样本内的数据。加入 shrinkage，也就是对系数进行限制，使它无法完全拟合样本内的数据，这样避免了噪音数据的影响，计算的系数对样本内数据或许是有偏估计，但由于计算系数时方差较小，更加稳定，或许在样本外会有更好的表现。这本质上是对偏差和方差的一种平衡（bias-variacne balance）。

一般来说一个模型越复杂，它的 bias 越小，预测值的 variance 越大；反之，一个模型越简单，它的 bias 越大，预测值的 variance 越小。这里需要一个平衡。如果样本内拟合效果很好，但样本外拟合效果很差，那么很可能就是模型过于复杂导致的，这时候需要的是弱化模型，而不是使用更复杂的非线性模型甚至深度学习模型来解决。一般来说，金融中低频数据的预测问题都是由过度拟合造成的，因为数据量比较小，样本内很容易拟合出赚钱的曲线，但样本外却不赚钱。而对于金融高频数据，由于交易手续费和冲击成本跟频率无关，而高频交易持仓时间更短，这么短时间的波动要覆盖高昂的成本是比较困难的，因此需要模型复杂一些，预测值的方差大一些，这样的话才能覆盖成本。

下面章节我们来看两种常见的 shrinkage 方法。

4.2.2 Ridge 模型

一般的线性回归的求解本质上是一个优化问题，求解最小化均方误差，即求 β_1，…，β_p，使下列式子的值最小：

$$\mathrm{RSS}=\sum_{i=1}^{n}\left(y_i-\beta_0-\sum_{j=1}^{\mathrm{p}}\beta_j X_{ij}\right)^2$$

Ridge Regression（岭回归）本质上也是求解的一个优化问题，但它的表达式加入了对系数的限制：

$$\sum_{i=1}^{n}\left(y_i-\beta_0-\sum_{j=1}^{p}\beta_j x_{ij}\right)^2+\lambda\sum_{j=1}^{p}\beta_j^2=\mathrm{RSS}+\lambda\sum_{j=1}^{p}\beta_j^2$$

其中 $\lambda \geqslant 0$ 是一个调节参数。第二项 $\lambda\sum_{j=1}^{p}\beta_j^2$ 是对系数的惩罚项，若系数取值幅度太大会影响拟合的效果。当 $\lambda=0$ 时，没有惩罚项，就是一般的线性回归最小二乘；当 λ 逐渐变大时，要求系数的幅度变小，对取值大的系数有“惩罚”的作用，防止过度拟合到部分因子；如果 λ 趋向于无穷，那么系数变成零，则用均值来做预测。下面我们来看看 ridge regression 的应用效果。

我们的预测集依然是 7、8、9 合约，而现在引入了验证集，因此训练集变成 1、2、3 合约，验证集是 4、5、6 合约。事实上，只要预测集不改变，训练集和验证集无论怎么调整都是允许的。对于金融这种时间序列，只要训练集、验证集在前面，预测集在后面，都是符合要求的。程序如下：

```
> train.range <- 1:3 ## 训练集为1-3合约
> valid.range <- 4:6 ## 验证集为4-6合约
> test.range <- 7:9 ##  测试集为7-9合约
> valid.contracts <- all.contracts[valid.range] ## 验证合约
> train.contracts <- all.contracts[train.range] ## 训练合约
> test.contracts <- all.contracts[test.range] ## 测试合约
> train.mat <- prepare.data(model$signals, model$y, product, train.
contracts, over.night=-1)
## 训练矩阵
> valid.mat <- prepare.data(model$signals, model$y, product, valid.
contracts, over.night=-1)
## 验证矩阵
> test.mat <- prepare.data(model$signals, model$y, product, test.
contracts, over.night=-1)
## 测试矩阵
> dim(train.mat) ## 训练数据规模
[1] 10485    20
> dim(valid.mat) ## 验证数据规模
[1] 11340    20
> dim(test.mat) ## 测试数据规模
[1] 12756    20
> x <- as.matrix(train.mat[,1:(ncol(train.mat)-1)]) ## 训练数据转成矩阵
> x.valid <- as.matrix(valid.mat[,1:(ncol(valid.mat)-1)]) ## 验证数据转成
矩阵
> n.sample <- nrow(x) ## 训练集样本数目
> x.train <- x
> x.test <- as.matrix(test.mat[,1:(ncol(test.mat)-1)]) ## 测试数据转成矩阵
```

```
> n.signals <- ncol(x) ## 因子数目
> signals <- names(train.mat)[1:n.signals] ## 因子名称
```

这时训练集变成 10 485 个样本，验证集是 11 340，测试集还是 12 756，因子和因变量均没有改变。我们现在来应用 ridge regression：

```
> library(glmnet) ## 调用glmnet包，里面有ridge, lasso和elastic net
> n.coef <- 100 ## 用组参数，这其实是一维的优化，只有1个参数
> grid=10^seq(-4,3,length=n.coef) ## 参数取值
> fit.ridge <- glmnet(x,train.mat$y,intercept=FALSE, alpha=0, lambda =
grid) ## ridge模型
> coef.mat <- coef(fit.ridge)[-1,] ## 找出各个模型的系数
> oos.mat <- rep(0,n.coef) ## 模型验证的结果
> for (i in 1:n.coef) { ## 遍历所有模型
+   cur.coef <- coef.mat[,i] ## 取出模型的系数
+   pred <- x.valid%*%cur.coef ## 计算预测值
+   oos.mat[i] <- R2(pred,valid.mat$y, "traditional") ## 计算R平方
+ }
> plot(oos.mat, type="l", ylab="R2", main="ridge") ## 画出验证集表现
```

在这里我们调用了 glmnet 这个程序包，里面的 glmnet() 命令有 ridge regression 和 lasso 的功能，其中的参数 alpha=0 就是 ridge regression，alpha=1 就是 lasso，如果 0<alpha<1 那么就称为 elastic net，属于 ridge 和 lasso 的混合。在这里我们设 alpha=0，使用 ridge regression 功能。

另外，验证集本质上是优化的取值，为此我们使用了：

```
> grid=10^seq(-4,3,length=n.coef)
```

命令来给 λ 取一系列的参数，在这里，越小表示约束越大，系数取值越小，越大表示约束越小，越接近普通的线性回归。最后，我们使用了：

```
> plot(oos.mat, type="l", ylab="R2", main="ridge")
```

来画出不同的模型在验证集的表现，如图 4-2 所示。

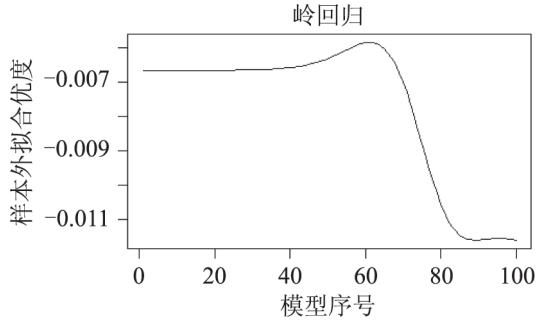


图 4-2　ridge 模型参数优化

可以明显看到，一开始对系数约束很多，系数取值基本上为零，没有改变。随着条件的放宽，各系数取值逐渐增大，预测效果也慢慢发生变化，直到达到最大值。随后，由于系数过度拟合训练样本，在验证集上表现为逐渐下降。我们发现拟合优度 R^2 依然为负数，但实际上我们更关心的是各模型 R^2 的相对大小，而不是绝对大小。

我们从验证集取得最优的 λ 之后，可以用训练集和验证集一起重新拟合一遍，然后用最优的去控制，得到最优的模型：

```
> best <- which.max(oos.mat) ## 找出最优的模型
> best
[1] 61
> train.valid.mat <- rbind(train.mat,valid.mat) ## 训练集和验证集合并
> x.test <- as.matrix(test.mat[,1:(ncol(test.mat)-1)]) ## 构造测试集的矩阵
> x <- as.matrix(train.valid.mat[,1:(ncol(train.valid.mat)-1)]) ## 训练验证集的矩阵
> fit.ridge <- glmnet(x,train.valid.mat$y,intercept=FALSE, alpha=0, lambda = grid)
## ridge回归的模型
> coef.mat <- coef(fit.ridge)[-1,] ## 所有模型的系数
> cur.coef <- coef.mat[,best] ## 最优模型的系数
```

最优模型是第 61 个，也就是图形中最高点的位置，然后我们再看看这个模型在预测集上的表现：

```
> pred <- x.test%*%cur.coef ## 预测集表现
> R2(pred,test.mat$y, "traditional") ## 样本外R平方
[1] -0.001568153
```

比之前的 −0.002 399 677，确实有改进。

另外，我们可以对比一下线性回归与 ridge 回归产生的模型系数：

```
> coef.compare <- as.matrix(cbind(fit$coefficients, cur.coef)) ## 合并两个模型的稀疏
> rownames(coef.compare) <- paste("x",1:19,sep=".") ## 行的名称为因子名
> colnames(coef.compare) <- c("ols","ridge") ## 列的名称是模型名
> coef.compare ## 对比结果
               ols          ridge
x.1  -4.359220e-04  6.267822e-05
x.2  -5.328895e-06  1.321396e-05
x.3  -7.059022e-04 -3.914736e-05
x.4   1.150248e-03  4.485924e-05
x.5   2.227387e-03  1.734975e-04
x.6   8.517942e-05  5.225817e-05
x.7  -5.024898e-04  2.968960e-05
x.8  -1.274319e-03 -3.778020e-05
x.9  -6.205350e-03  1.477475e-04
x.10 -1.086805e-03  4.865833e-05
x.11 -3.040320e-05  1.523727e-06
```

```
x.12  1.858969e-02  6.084253e-04
x.13 -4.232807e-02  1.890575e-03
x.14  3.109798e-03  1.425716e-04
x.15  2.599714e-04 -5.994934e-05
x.16  1.606271e-02  5.090600e-04
x.17  1.245871e-03  9.550450e-05
x.18 -2.662491e-03  6.547399e-05
x.19  1.236075e-03  1.104729e-04
```

Ols（ordinary linear square）是普通最小二乘，ridge 表示 ridge 回归，我们可以看出 ridge 回归产生的系数在取值幅度上明显小于最小二乘产生的系数，这也是收缩的含义。

岭回归（ridge regression）可以起到缩小系数取值幅度避免过度拟合的效果，但是却没有选择因子的功能，我们可以看到每个因子的系数都不为零，如果因子数目过多，仅仅缩小因子取值幅度或许还不能减小过度拟合。一般统计学术界的稀疏 （bet on sparsity）更多指的是减少参数的数目，也就是把参数的系数变成零，因此 ridge 回归不能满足这个条件。对于很多人工智能的研究，其实并没有过于区分 Ll-norm 和 L2-norm 带来的影响，那类方法往往对其他一些减少过度拟合的手段更为依赖，比如 dropout 等，这里我们暂时不讨论。L1-norm 指的是用绝对值之和来限制系数，L2-norm 指的是用平方和来限制系数，用绝对值之和的话可以把系数收缩到零，平方和就不可以。

岭回归在计算上的一个好处是它只是在求逆矩阵里面加了一项，因此还是矩阵求逆，计算起来非常方便，不像 lasso，要用专门的优化算法去搜索。而且，正是由于只涉及矩阵运算，ridge 对增加和减少样本的处理很方便，对于金融领域需要滚动优化的项目其实用 ridge 回归还是很有优势的，它可以高效地加入新的样本并且利用旧的计算结果来更新，且不需要重新计算。很多时候，金融统计建模对速度要求还是挺高的，运算速度快的话可以更高效地验证模型，更容易发现问题并且改正。如果计算一遍需要几个小时甚至几天，那么研究效率就非常低了。

我们下一节将讨论另一种 shrinkage 的方法——lasso 模型。

4.2.3 Lasso 模型

Lasso 模型是对 ridge 模型的稍微改动：

$$\sum_{i=1}^{n}\left(y_i-\beta_0-\sum_{j=1}^{p}\beta_j x_{ij}\right)^2+\lambda\sum_{j=1}^{p}|\beta_j|=\text{RSS}+\lambda\sum_{j=1}^{p}|\beta_j|,$$

原来的系数平方项变成了绝对值，也就是 L2-norm 变成了 L1-norm，这表面上只是一个小变化，但实际带来的影响是非常大的。

一个最大的改变在于 lasso 具有变量选择的功能，也就是说会把其中一些变量的系数收缩到零，因此这个变量就等于删除了。有兴趣的读者可以参考本章小结推荐的相关书籍，里面有详细的介绍。

具体来说，比如有 3 个相关性很强的因子，如果使用 ridge 模型，那么它会把系数平分给这 3 个因子，并且都保留下来；但如果使用 lasso 模型，它会只保留一个因子，而删除其他两个。

我们可以来看使用 lasso 模型的情况：

```
> grid=10^seq(-10,10,length=n.coef) ## 参数取值网格
> x <- as.matrix(train.mat[,1:(ncol(train.mat)-1)]) ## 训练矩阵
> x.new <- as.matrix(train.valid.mat[,1:(ncol(train.mat)-1)]) ## 训练与
验证矩阵
> fit.lasso <- glmnet(x,train.mat$y,intercept=FALSE, lambda = grid) ##
lasso模型
> coef.mat <- coef(fit.lasso)[-1,] ## 系数矩阵
> oos.mat <- rep(0,n.coef) ## 验证集表现
> for (i in 1:n.coef) { ## 遍历所有参数
+   cur.coef <- coef.mat[,i] ## 读取当前系数
+   pred <- x.valid%*%cur.coef ## 验证集的预测值
+   oos.mat[i] <- R2(pred,valid.mat$y, "traditional") ## 计算R平方
+ }
> plot(oos.mat, type="l") ## 画出验证集R平方
> best <- which.max(oos.mat) ## 找出最优模型
> best
[1] 68
> x <- as.matrix(train.valid.mat[,1:(ncol(train.valid.mat)-1)]) ## 训练
与验证矩阵
> fit.lasso <- glmnet(x,train.valid.mat$y,intercept=FALSE, lambda =
grid) ##
> coef.mat <- coef(fit.lasso)[-1,] ## 系数矩阵
> pred.test <- x.test %*% coef.mat[,best] ## 测试集预测值
> R2(pred.test, test.mat$y, "traditional") ## 样本外R平方
[1] -0.001875301
```

拟合优度是 −0.001 875 301，比 ridge 模型的结果略低，但比最小二乘的结果好。我们来看 lasso 模型拟合出来的因子系数：

```
> coef.compare <- as.matrix(cbind(fit$coefficients, coef.mat[,best])) ##
系数对比
> rownames(coef.compare) <- paste("x",1:19,sep=".") ## 行名称改成因子名
> colnames(coef.compare) <- c("ols","lasso") ## 列名称改成模型名
```

```
> coef.compare
               ols          lasso
x.1  -4.359220e-04  0.000000e+00
x.2  -5.328895e-06  0.000000e+00
x.3  -7.059022e-04  0.000000e+00
x.4   1.150248e-03  0.000000e+00
x.5   2.227387e-03  0.000000e+00
x.6   8.517942e-05  0.000000e+00
x.7  -5.024898e-04  0.000000e+00
x.8  -1.274319e-03  0.000000e+00
x.9  -6.205350e-03  0.000000e+00
x.10 -1.086805e-03  0.000000e+00
x.11 -3.040320e-05  0.000000e+00
x.12  1.858969e-02  0.000000e+00
x.13 -4.232807e-02  0.000000e+00
x.14  3.109798e-03  0.000000e+00
x.15  2.599714e-04 -2.359142e-05
x.16  1.606271e-02  2.988264e-03
x.17  1.245871e-03  0.000000e+00
x.18 -2.662491e-03  0.000000e+00
x.19  1.236075e-03  0.000000e+00
```

只有 x.15 和 x.16 的系数非零，由此可见只选出了两个因子，起到了筛选因子的作用。

因此，lasso 模型可以自动删除那些表现不大好的因子。很多公司研究策略时往往有着大量的量化研究员去研究新的因子，因子的数目虽然非常庞大，但很多质量一般，虽然可以用一些简单的相关性等方法过滤掉实在太差的因子，但在建模的时候，如果能进一步精简就更好了。金融建模最怕的是过度拟合训练样本，这样往往会给因子很高的权重，预测值的幅度会偏大，然而样本外依旧进行较为频繁的交易，容易导致亏钱。特别是一些基于第三方平台的规则型策略，使用了极低的交易成本进行测试，而且基本上都是全样本进行优化，过度拟合非常严重，交易次数还多，这种策略在样本外或实盘中大概率会亏钱。

4.3 模型选择

至于选取哪一种模型，需要具体问题具体分析。20 世纪八九十年代很多人运用神经网络模型预测股票市场价格变化，但收效甚微，后来产生的支持向量机、决策树等模型成为主流的机器学习模型。我们先来分析一下金融数据的特

征，以及这两类模型的特征，然后再进行下一步分析。

4.3.1　金融数据特征

样本内因变量 y 的概率密度函数，如图 4-3 所示。

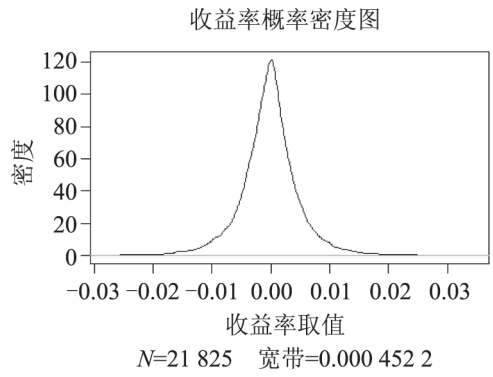


图 4-3　y 的概率密度函数

由图 4-3 可见在零值附近的概率很大，具有尖峰特征。QQ 图则如图 4-4 所示。

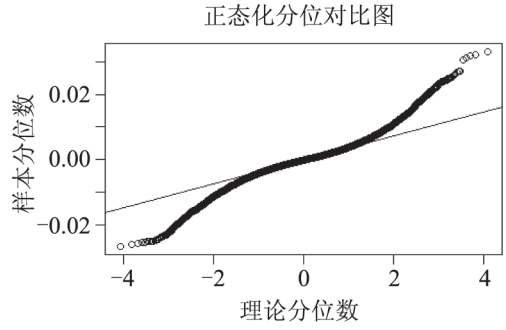


图 4-4　QQ 图理论分位数

正负两倍的分位已经非常远离正态分布了，因此具有“厚尾”特征。这类尖峰厚尾特征的数据不大符合正态分布。代码如下：

```
> plot(density(train.valid.mat$y), main="y") ## 画出y的概率密度函数
> qqnorm(train.valid.mat$y) ## 画出y的QQ图
> qqline(train.valid.mat$y) ## 加入理论上的QQ线
```

y 的分布具有尖峰厚尾的特点，那么拟合之后的残差是否是正态分布呢？为此，我们可以对残差进行类似的分析。这里残差指的是样本内拟合的结果，这种拟合效果最好，因为是最小二乘拟合。

```
plot(density(fit$residuals), main="residuals") ## 画出残差的概率密度函数
qqnorm(fit$residuals) ## 画出残差的QQ图
qqline(fit$residuals)## 加入理论上的QQ线
```

对于残差的概率密度函数如图 4-5 所示。

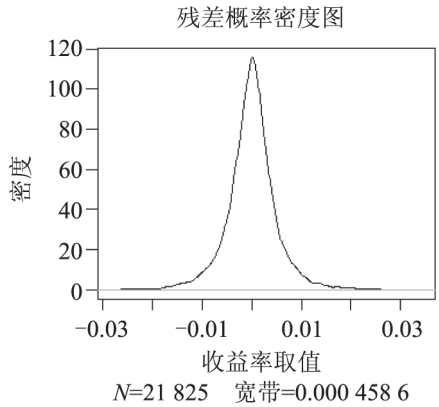


图 4-5 残差的概率密度分布图

由图 4-5 可见，残差也具有尖峰厚尾的特征。由于我们的因子并不能很好地拟合 y 的这个特性，所以残差依旧保留了下来。这在 QQ 图中或许可以更加清晰一些，如图 4-6 所示。

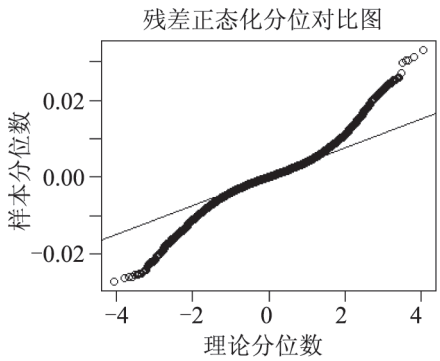


图 4-6 残差的 QQ 图

由图 4-6 可见，残差的分布也不是正态的，也具有尖峰厚尾特征。其实这

也不难理解，因为绝大多数的 y 的取值都是在零附近，对价格平均而言变化就是接近零的，并不存在先验的涨或跌的预判。至于厚尾性，这是由于期货合约波动较大，更容易产生极端的价格变化，而这类变化也是最难拟合的，拟合的误差也比较大，因此厚尾特性在残差中依然保留。但这并不代表实盘中就难以赚钱。事实上，波动大的时候往往是同一个方向发生变化，比如价格变化 2%，我们只预测出 0.2%，但方向没有错，因此并不会亏钱。

4.3.2　贝叶斯观点

刚才提到的 ridge 和 lasso 两种模型，我们是从一般统计分析的角度出发建模的，目的是缩小系数求解的范围来避免过度拟合。如果从另外一种角度来看，如从贝叶斯统计的角度出发，则可以得到另外一种视角。

贝叶斯统计会对系数假设一个先验分布，例如对 ridge regression 而言，对应假设系数的先验分布符合正态分布；对 lasso 而言，假设系数的先验分布符合双指数分布。双指数分布的尖峰特征使其具有筛选因子的功能。

从现代统计学的角度看，更倾向于具有稀疏性（sparsity）的模型，特别是对于高噪音的数据，如果因子数量过多，每个因子的系数值都很难准确估计，还不如把有限的数据，用于估计少数几个因子系数的值，或许更为准确。

4.3.3　数理模型在金融中的应用

现在市面上最时髦的是深度学习模型，比如卷积神经网络、长短时记忆模型、深度增强学习等。然而，在做量化金融的研究时，应该尽量选用一些含义清晰的模型，而不是那种含义不明的黑盒模型。特别是在中低频的研究时，一般把时间花在收集更多数据上会比构造更复杂的模型更有意义。

一些传统的数理模型，如时间序列、随机过程等对研究金融预测问题或许更有效一些。比如“*Empirical Asset Pricing: The Cross Section of Stock Returns*”一书列举了关于股票预测的各种因子，里面也用到了很多计量经济学的知识，并对异方差等进行了处理。这些都是传统数理金融的内容，对金融研究更为有用，而这些书籍和研究报告采用的语言和方法跟计算机流行的机器学

习是非常不一样的，更适合具有数学、统计背景的人学习。另外还有一些用随机过程来研究价格变化的知识也可以学习一下，在统计套利中或许更为有用。

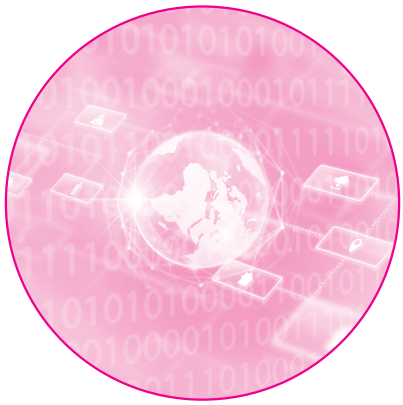
4.4 本章小结

本章介绍了用于研究金融数据的几个预测模型，还介绍了如何避免过度拟合的方法，并运用例子分析了 ridge 和 lasso 两种模型。关于线性回归、ridge、lasso 等内容，有兴趣的读者可以参考“*An Introduction to Statistical Learning with Applications in R*”一书，其中 ridge 和 lasso 在第六章。更深入的讨论可以在“*Elements of Statistical Learning*”一书中找到，里面有详细的理论推导供读者参考学习。

因子和预测模型固然重要，但也只是整个量化交易系统的一部分，如果没有其他部分的配合，也是难以持续盈利的。很多时候，交易模型、投资组合优化也同样非常重要，同样的模型，不同的交易方法和资金配置方法，可以得到截然不同的结果。前面几章主要都是对统计模型层面的讨论，无论是金融交易还是其他领域的数据，其实都是相同的。接下来的几章则是金融交易特有的内容。下面一章我们将介绍如何从预测模型过渡到真实的交易信号，并绘制回测的资金曲线，这样一来我们将会对研究结果有更直观的感受。

第五章

复杂统计模型与机器学习



前面一章介绍了基础的统计模型，主要包括线性模型以及带约束项的线性模型，这一章主要介绍一些更复杂的模型，比如决策树、随机森林等。这类模型是多重可加模型（Multiple Addictive Model），本质上虽然还是线性模型，但每个因子的结构更为复杂，并不是普通的函数生成的，很多因子都有着复杂的树型结构。

此外，我们将研究跨品种的因子。比如螺纹和热卷，它们的相关性很强，所以用热卷数据生成的因子对螺纹也会有一定的预测能力，但是其生成的因子跟用螺纹自身生成的因子相关性并不会很高。

最后，我们将讨论高频数据建模的问题。在第三章简单讨论了lasso用于高频数据的建模，在此，我们将进一步讨论如何运用gbm和随机森林来研究商品高频数据产生的策略。

5.1 复杂统计模型

这里的复杂统计模型主要是指 gradient boosting machine （gbm）和随机森林两种模型。这两种模型都是基于回归树（regression tree）的建模方法。两者在实现和思想上略有差别，下面章节会详细描述。

5.1.1 Gradient Boosting Machine（GBM）

gbm 模型是斯坦福大学统计系教授 Jerome H. Friedman 于 2001 年提出的，当时的论文名称是“*Greedy Function Approximation*：*A Gradient Boosting Machine*”，发表在著名的统计学期刊“*The Annals of Statistics*”上面。其中 boosting 的意思是增强算法，至于对哪方面增强，不同的模型则有不同的处理方法。这里的 gradient boosting 则是对 gradient（梯度）进行增强，这里梯度是指损失函数的导数。

在回归模型中，一般使用最小二乘法进行系数拟合。比如损失函数为：

$$\text{Loss}=\frac{1}{2}\left(y-\hat{y}\right)^2$$

让它对 y 求导得到：

$$\frac{\partial \text{Loss}}{\partial y}=y-\hat{y}=\text{ò}$$

因此，导数恰好就是回归模型的残差。对导数进行增强，其本质是对残差进行增强，即每次构造一个新的因子，使残差减小得更快。

有时候会听到 stochastic gradient（随机梯度）这个词，这其实指的是一种数值计算的技巧。当样本量非常大的时候，每次计算梯度耗时就会很长，此时只需要随机选择一个样本计算其梯度，这样可以在一定程度上减少计算时间。当然，如果是全部样本一起计算梯度，那么可以保证每次迭代之后损失函数是下降的；但如果是随机选择一个样本，则损失函数有可能上升，但总体而言，如果样本分布比较平稳，随机梯度算法还是可以在很大程度上减少计算时间，提高参数收敛的速度。

下面我们回到 gbm 模型。一般来说，gbm 的每个因子都是二叉树结构。

比如我们先输入一些基础因子，gbm 会自动利用这些基础因子合成树型因子，然后把树型因子作为变量放入回归模型。因此，gbm 本质上是一个线性可加模型。

从表面上看，gbm 是一个 forward stepwise regression（前向逐步回归）模型，但本质上它是一个 forward stagewise regression（前向分段回归）模型，区别在于它并不是把树型变量的最佳系数整个放入模型中，而是乘以了一个 learning rate（学习速率），起到了 slow learning（慢速学习）的效果，避免了过度拟合。而且它也有类似 lasso 的性质，有人研究过 gbm 与 lasso 的系数演变轨迹，非常的类似，也有人给出了数学上的证明，详情可以参考“*Elements of Statistical Learning second edition*”一书，第一版出版时还没有证明，第二版出版时才有，这也是统计学界的一个比较漂亮的成果。在此我借用了他们一幅图来说明，如图 5-1 所示。

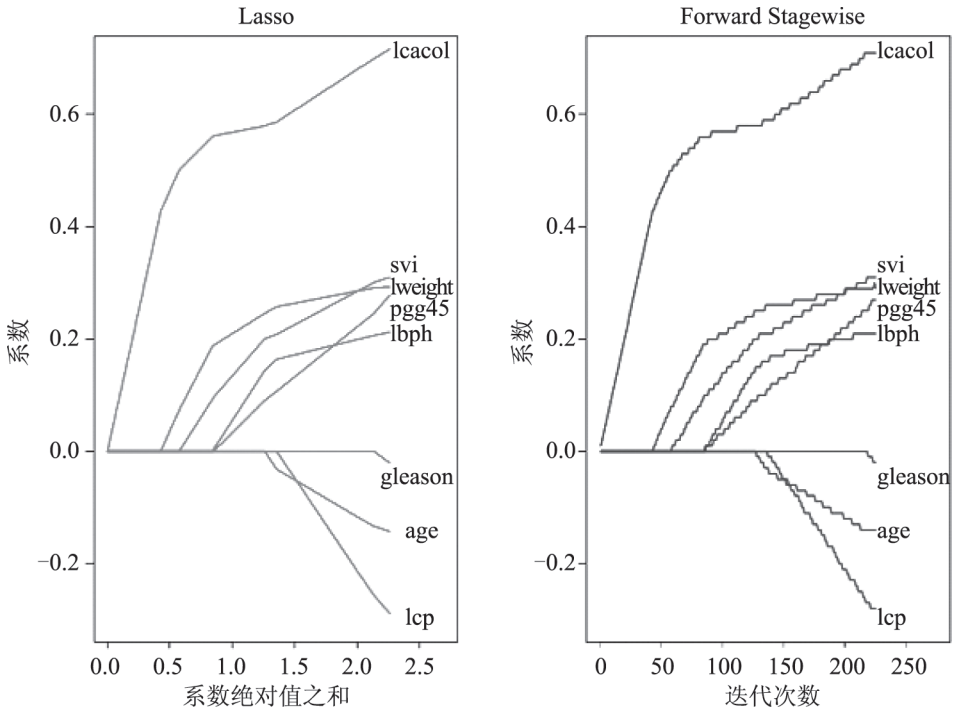


图 5-1　Lasso 与 gbm 系数演变

下面我们可以用 gbm 模型研究一下螺纹中低频策略。

我们先把数据准备好，分成训练集、验证集和测试集三部分：

```
> train.range <- 1:4 ## 训练集
> valid.range <- 5 ## 验证集
> test.range <- 6:9 ## 测试集
> train.contracts <- all.contracts[train.range] ## 训练合约
> valid.contracts <- all.contracts[valid.range] ## 验证合约
> test.contracts <- all.contracts[test.range] ## 测试合约
> train.mat <- prepare.data(model$signals, model$y, product, train.
contracts, over.night=-1)
## 训练矩阵
> dim(train.mat) ## 训练矩阵规模
[1] 14445    20
> valid.mat <- prepare.data(model$signals, model$y, product, valid.
contracts, over.night=-1)
## 验证矩阵
> test.mat <- prepare.data(model$signals, model$y, product, test.
contracts, over.night=-1)
## 测试矩阵
> colnames(train.mat) <- c(paste("x",1:19,sep="."),"y") ## 训练矩阵列名称
> colnames(valid.mat) <- c(paste("x",1:19,sep="."),"y") ## 验证矩阵列名称
> colnames(test.mat) <- c(paste("x",1:19,sep="."),"y") ## 测试矩阵列名称
```

在第 4 章里面的模型中，训练集是 1、2、3，验证集是 4、5、6，测试集是 7、8、9。由于 gbm 是比普通的 lasso 更复杂的模型，因此我们希望训练集更大一些，这样的话参数才能更好地拟合。为此，我们这里的训练集是前 4 个，验证集是第 5 个，为了让这个复杂的模型有更多测试数据，测试集是 6 ～ 9。然后我们进行建模：

```
> library(gbm) ## 调用gbm库
> n.signals <- length(model$signals) ## 因子数目
> set.seed(100) ## 设置随机种子
> n.tree <- 300 ## 树的数目
> depth <- 2 ## 树的深度
> gbm.model <- gbm(y～., data=train.mat, shrinkage = 0.01,
+                  interaction.depth = depth, distribution="gaussian",n.
trees=n.tree, verbose=FALSE) ## gbm建模
```

这里调用的是 R 语言里面的 gbm 包。现在更为流行的是一个称为 xgboost 的包，但 gbm 包使用时间更长，所以我们这里选择了 gbm。另外，由于 gbm 使用了 stochastic gradient 这种随机算法，因此每次运行的结果会不一样，为了使结果更稳定，我们使用了 set.seed（100）这个命令，使结果具有可重复性。

gbm 模型需要优化的参数主要有两个：树的深度和数目，即程序中的 depth 和 n.tree。其实这两个参数是相辅相成的：树的深度越大，数目越多，模

型就越复杂，样本内拟合的效果就越好，同时过度拟合的风险也就越大。因此，不妨固定一个参数，然后调节另一个参数。一个常见的做法是只适用深度为 2 的树，然后只改变树的数目（n.tree），这样只需要调节一个参数即可。我们可以看看这个模型的拟合效果，如图 5-2 所示。

```
r2 <- rep(0,n.tree) ## 各个模型的R平方
> for (i in 1:n.tree) { ## 遍历所有模型
+     pred <- predict.gbm(gbm.model,newdata=valid.mat[,1:n.signals],n.
trees = i) ## 预测值
+   r2[i] <- R2(pred, valid.mat$y,"traditional") ## 计算R平方
+ }
> plot(r2, type="l", main="gbm validation", xlab="n.tree") ## 画出R平方的图
```

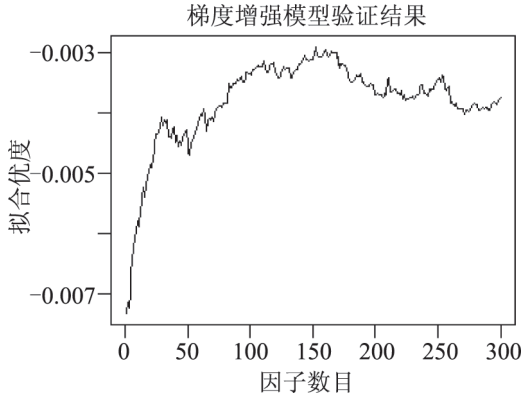


图 5-2　gbm 训练 / 验证结果

由图 5-2 可见在中间某个点取得最大值。我们找出这个点，然后看看在预测集上的效果：

```
> best <- which.max(r2) ## 找出最佳模型
> best ## 最佳模型
[1] 153
> test.pred <- predict.gbm(gbm.model,newdata=test.mat[,1:n.signals],n.
trees = best) ## 预测值
> R2(test.pred, test.mat$y, "traditional") ## 样本外R平方
[1] 0.0006139437
```

在预测集上的效果比较好。我们也可以考察一下在测试集上的最佳拟合效果是怎样的，然后看看我们选取的最优值跟实际最优值的对比，如图 5-3 所示。

```
> plot(test.r2, type="l", main="test set", xlab="n.tree") ## 测试集拟合
> which.max(test.r2) ## 测试集最佳结果
[1] 110
> max(test.r2) ## 最佳结果R平方
[1] 0.001164149
```

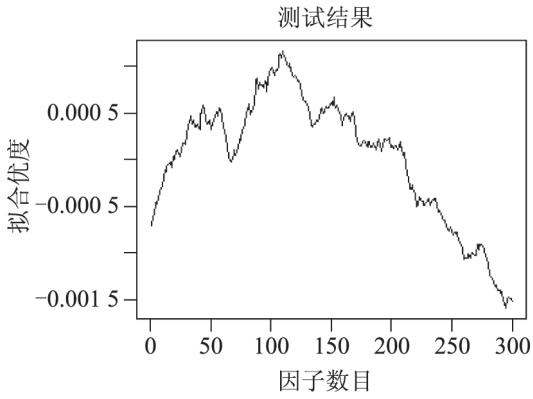


图 5-3　gbm 测试集表现

由图 5-3 可见，在 best=110 之后预测效果持续下降，选取的 153 其实并不是最好，但表现还算不错。一般来说，所选取的模型不一定是验证集最佳的，可能会选择一个次优模型，来换取在测试集上表现会更好一些。选取的规则多种多样，要根据实际应用场景来定。

除了 gbm 外，还有一种很常见的复杂预测模型，也是基于树型结构的，就是下一节介绍的随机森林模型。

5.1.2　随机森林

随机森林是一种拟合效果强大但使用起来十分简单的模型，它需要调整的参数不多。跟 gbm 不同的是，随机森林每棵树的深度并不是固定的。随机森林需要调整的参数也往往只有两个：树的数目和最大的节点数，其中最大的节点数决定了树的深度。随机森林的一个好处是一般不需要人工调整参数，模型自己选择的默认参数已经可以达到比较好的效果，所以被称为“best off-the-shelf model”。

与 gbm 类似，随机森林也是基于树型结构，因此对变量的取值并不敏感，无须进行标准化、去除极端值数据等预处理，而且针对分类和回归问题都适用。

与 gbm 不同的是，随机森林模型对树的数目这个参数并不敏感，因此调整这个参数的意义并不是很大，一般而言，验证集调整的是树的最大节点这

个参数。由于也只需要调整一个参数，因此参数优化不需要网格搜索（grid search），同时可以避免过度拟合。

我们先来看一下不调整参数模型默认参数的情况：

```
> rf.model <- randomForest(y~., data=train.mat) ## 构建随机森林模型
> pred <- predict(rf.model, newdata=test.mat[,1:n.signals]) ## 计算预测值
> R2(pred, test.mat$y, "traditional") ## 样本外R平方
[1] -0.06474626
```

可见样本外的拟合优度只有 -0.064 7，并不是很高。我们再来看看参数优化的结果，如图 5-4 所示。

```
n.node <- 10 ## 节点数目
r2 <- rep(0,n.node) ## R平方
for (i in 1:10) { ## 遍历所有情况
  rf.model <- randomForest(y~., data=train.mat, maxnodes=i*20) ## 计算随机森林模型
  pred <- predict(rf.model, newdata=valid.mat[,1:n.signals]) ## 计算预测值
  r2[i] <- R2(pred, valid.mat$y, "traditional") ## 计算R平方
  cat(i,r2[i],"\n")
}
> which.max(r2) ## 最优模型
[1] 4
plot(1:10*20, r2, type="l", main="random forest optimization", xlab="maxnode", ylab="r2")
## 画出各个模型的表现
```

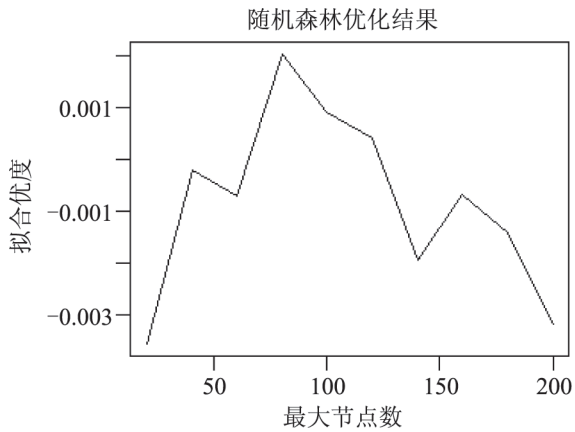


图 5-4　随机森林参数优化

这是在验证集上的结果，这里最大节点数从 20 到 200 进行网格优化，找出中间的最大值作为正则化的结果，这里最大是第 4 个，即 80 个节点，我们用它来作为最终模型预测样本外的情况：

```
> rf.model <- randomForest(y~., data=train.mat, maxnodes=80) ## 最佳模型
```

```
> pred <- predict(rf.model, newdata=test.mat[,1:n.signals]) ## 最佳模型预测值
> R2(pred, test.mat$y, "traditional") ## 最佳模型R平方
[1] -0.01152484
```

最佳结果是 -0.01152，比之前模型默认的参数好很多，之前 gbm 模型在同样数据集的结果是 0.0006139437，因此 gbm 模型要好得多。另外，对比 gbm 和随机森林，gbm 的优点在于：

- 模型不必重复计算。由于gbm正则化优化的是树的数目，我们只需先一次性生成数目较大的树，然后对树的数目进行优化即可，不必重复计算。但对于随机森林，它对树的数目并不敏感，我们优化的是节点的样本数，这就需要重新生成模型了，计算量大大提高。
- 预测效果更好。在这个问题上，gbm的预测效果要远好于随机森林。一般来说，gbm是比随机森林更强大的模型，随机森林的优点是不怎么需要调参数，坏处是模型的能力并不是很强，适合更简单一些的数据集。
- 模型结构简单。gbm每棵树的深度都是固定的，这对于把模型结果转成C++等其他语言来说会更加方便一些，因为参数都是固定格式的系数，可以用相同大小的矩阵表示，当然也可以全部放进一个矩阵里面。但随机森林每棵树的大小不一致，如果要重现模型的话结果就不大方便了。

综上所述，gbm 应该是更适合金融建模的复杂模型。而且它自带的 lasso 功能也可以很大程度上避免过度拟合。对于分笔数据，训练集的大小远超 5 分钟数据，我们有足够的数据量来训练模型。另外，对于金融交易这种高频策略，我们也需要更复杂的模型来捕捉更多微小的波动，从而才能在短短的持仓时间内超过覆盖交易成本。

5.2 跨品种因子

针对研究策略来说，模型固然重要，但很多时候信息比模型更重要。在同样的信息下，复杂的模型可以更充分地利用信息，但如果可以获得更多的信息，即便模型并不复杂，却往往可以有更好的效果。毕竟只是在行情数据的情况下，

从线性回归到梯度增强，其改进也并不会是飞跃性的。下面章节，我们将研究如何利用相关品种来构建预测因子的问题。

5.2.1　相关品种构建因子

我们可以研究一下利用相关性高的品种来构建预测因子，看看能否提高预测能力。比如螺纹钢与热压卷板 hc 的相关性非常高，我们可以尝试利用 hc 来构建因子。

首先来看 hc 的数据：

```
> strat.product <- "hc" ## 相关品种
> strat.contracts <- get.dates(strat.product) ## 相关品种合约
> strat.contracts ## 相关品种合约列表
[1] "hc1501.RData" "hc1505.RData" "hc1510.RData" "hc1601.RData"
"hc1605.RData" "hc1610.RData" "hc1701.RData"
[8] "hc1705.RData"
```

它只有 8 个合约，而且最开始的是 1501，比螺纹钢少很多。而且合约换月的时间未必一致。我们首先要把两个品种的数据对齐，然后才能进行处理。

```
> get.product.date.time <- function(contracts) { ## 获得品种合约的日期和时间
+   date <- c() ## 日期列表
+   time <- c() ## 时间列表
+   for (contract in contracts) { ## 遍历所有合约
+     load(contract) ## 调用合约
+     date <- c(date, data$date[data$continuous]) ## 合并日期
+     time <- c(time, data$time[data$continuous]) ## 合并时间
+   }
+   date.time <- get.date.time(date,time) ## 转换日期和时间
+   return(date.time)
+ }
```

这是提取日期和时间的辅助函数。

```
product.date.time <- get.product.date.time(product.contracts) ## 获得交易品种的日期时间
strat.date.time <- get.product.date.time(strat.contracts) ## 获得辅助品种的日期时间
> length(product.date.time) ## 交易品种时间长度
[1] 74565
> length(strat.date.time) ## 辅助品种时间长度
[1] 47250
```

由此可见两者长度不一样。我们以后上市的 hc 为准进行处理：

```
> if (length(product.date.time)>length(strat.date.time)) { ## 交易品种比辅助品种长
+   product.index <- match(strat.date.time, product.date.time) ## 交易时间匹配
```

```
+   bad <- which(is.na(product.index)) ## 处理一些特殊情况
+   product.index[bad] <- product.index[bad-1]+1
+   aa <- match(product.date.time, strat.date.time)
+   bad <- which(is.na(aa))
+   if (bad[1]==1) bad <- bad[-1]
+   aa[bad] <- aa[bad-1]+1
+   strat.index <- aa[!is.na(aa)]
+ } else { ## 交易品种比辅助品种短
+   strat.index <- match(product.date.time, strat.date.time) ## 交易时间
匹配
+   bad <- which(is.na(strat.index)) ## 处理一些特殊情况
+   strat.index[bad] <- strat.index[bad-1]+1
+   product.index <- 1:length(product.date.time)
+ }
>
> product.data <- prepare.data(signals, y.str, product,product.
contracts)[product.index,]
## 交易品种指标数据准备
> strat.data <- prepare.data(signals, y.str, strat.product, strat.
contracts)[strat.index,]
## 辅助品种指标数据准备
> dim(product.data) ## 交易品种数据规模
[1] 47250    20
> dim(strat.data) ## 辅助品种数据规模
[1] 47250    20
```

这样就得到了长度一样的数据。因为中间一些时间标记有点问题，所以专门写了程序进行处理，实际中的对齐并不这么复杂。事实上，如果是夜盘交易时间段，一般只需要用晚上市的品种的交易时间即可。但由于本人之前的数据存在一些小问题，所以才需要做专门的处理，只要数据是完备的，时间自然是对齐的，这里只是为保险起见。然后进行数据的合并：

```
n.signal <- ncol(product.data)-1 ## 因子的数目
colnames(strat.data) <- paste(strat.product,colnames(strat.
data),sep=".")
 ## 新品种因子名称重新命名
all.data <- cbind(product.data[,1:n.signal], strat.data[,1:n.signal])
## 合并两个品种的数据
all.data$y <- product.data$y ## 因变量用交易品种的因变量
all.data <- clean(all.data) ## 清洗数据
n.data <- nrow(all.data) ## 样本总数
dim(all.data) ## 数据规模
[1] 47250 39
```

原来每个品种有 19 个因子，现在合并起来有 38 个，另外一个是因变量。我们这里做的是螺纹钢，因此只需要螺纹钢的因变量。

```
> n.bar <-nrow(all.data) ## 样本总数
> train.range <- 1:10000 ## 训练范围
> valid.range <- 10001:20000 ## 验证范围
> test.range <- setdiff(1:n.bar, c(train.range, valid.range)) ## 测试范围
```

```
> train.mat <- all.data[train.range,] ## 训练样本
> valid.mat <- all.data[valid.range,] ## 验证样本
> test.mat <- all.data[test.range,] ## 测试样本
> grid=10^seq(-2,-6,length=n.coef) ## 参数矩阵
> x <- as.matrix(train.mat[,1:(ncol(train.mat)-1)]) ## 训练矩阵
> x.valid <- as.matrix(valid.mat[,1:(ncol(train.mat)-1)]) ## 验证矩阵
> x.test <- as.matrix(test.mat[,1:(ncol(train.mat)-1)]) ## 测试矩阵
> fit.lasso <- glmnet(x,train.mat$y,intercept=FALSE, lambda = grid) ##
lasso模型
> coef.mat <- coef(fit.lasso)[-1,] ## 模型系数矩阵
> oos.mat <- rep(0,n.coef) ## 样本外结果
> for (i in 1:n.coef) { ## 遍历全部模型
+   cur.coef <- coef.mat[,i] ## 模型系数
+   pred <- x.valid%*%cur.coef ## 预测值
+   oos.mat[i] <- R2(pred,valid.mat$y, "traditional") ## 样本外R平方
+ }
> plot(oos.mat, type="l") ## 画出样本外表现
> best <- which.max(oos.mat) ## 找出最好的模型
> best
[1] 41
```

这里选取 1 ： 10 000 作为训练集，10 001 ： 20 000 作为验证集，之后的作为测试集，一共有 39 个因子，训练出 100 个模型，最佳模型是第 41 个。其中各个模型在验证集的拟合优度，如图 5-5 所示。

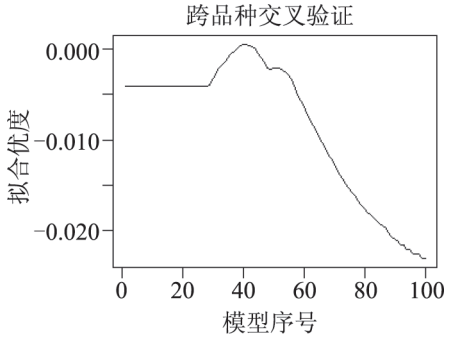


图 5-5　跨品种验证结果

然后把最优模型用在测试集上：

```
> pred.test <- x.test %*% coef.mat[,best] ## 样本外预测值
> R2(pred.test, test.mat$y, "traditional") ## 样本外R平方
[1] -0.002754658
```

样本外拟合优度是 −0.002 754，我们可以对比一下不加 hc 因子的情况：

```
> train.mat <- all.data[train.range,c(1:n.signal,39)] ## 不要hc的训练集
> valid.mat <- all.data[valid.range,c(1:n.signal,39)] ## 不要hc的验证集
> test.mat <- all.data[test.range,c(1:n.signal,39)] ## 不要hc的测试集
> grid=10^seq(-2,-6,length=n.coef) ## 参数网格
```

```
> x <- as.matrix(train.mat[,1:(ncol(train.mat)-1)]) ## 不要hc的训练矩阵
> x.valid <- as.matrix(valid.mat[,1:(ncol(train.mat)-1)]) ## 不要hc的验证矩阵
> x.test <- as.matrix(test.mat[,1:(ncol(train.mat)-1)]) ## 不要hc的测试矩阵
> fit.lasso <- glmnet(x,train.mat$y,intercept=FALSE, lambda = grid) ## 拟合lasso模型
> coef.mat <- coef(fit.lasso)[-1,] ## 模型系数矩阵
> oos.mat <- rep(0,n.coef) ## 验证集表现
> for (i in 1:n.coef) { ## 遍历所有模型
+   cur.coef <- coef.mat[,i] ## 模型系数
+   pred <- x.valid%*%cur.coef ## 验证集预测值
+   oos.mat[i] <- R2(pred,valid.mat$y, "traditional") ## 验证集R平方
+ }
> plot(oos.mat, type="l") ## 画出验证集的R平方
> best <- which.max(oos.mat) ## 最优模型
> best
[1] 58
> pred.test <- x.test %*% coef.mat[,best] ## 样本外预测值
> R2(pred.test, test.mat$y, "traditional") ## 样本外表现
[1] -0.006316881
```

最优模型是 58，样本外拟合优度是 −0.006 32，由此可见结果比加了 hc 的因子要差许多。另外，验证集的结果如图 5-6 所示。

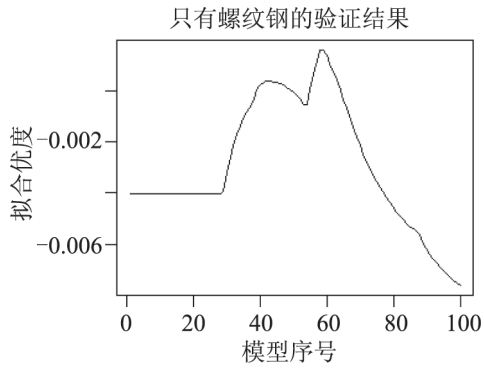


图 5-6　只有螺纹钢的验证结果

我们可以看出加了 hc 之后模型确实有所改进。其实这也不难理解，一般而言，螺纹钢和热卷本质上都是钢材，只不过形状一个是直的一个是卷的，但材料都一样，因此价格是高度相关，做跨品种套利的话手数配比也是 1 ： 1。实际交易过程中，每个品种都是独立交易的，各自都有各自的限价指令队列，因此没办法保证价格是完全同步变化的。如果是一个基本面的信息量丰富的消息导致价格变化，就很可能会使两者同步变化，这是属于价格变化的“信号”部分，模型应该尽量去捕捉。

相反，如果仅仅是个别操盘手无意或者蓄意依靠短时资金优势制造的小行情，往往只在一个品种上发生，另外一个品种缺乏类似的刺激，那么这类价格变化是很难持续下去的，很可能很快价格就会修复，这些属于“噪音”部分，需要避免。

如果仅有螺纹钢的数据，而不知道其他品种的价格变化，就没法很好地捕捉这类联动的行情；但如果加入热卷的数据，则可以通过热卷的技术指标，获得热卷的价格变化，从而更好地预测螺纹钢的价格变化，更好地区分信号和噪音。

5.2.2　建模的注意事项

很多人或许是从机器学习等其他领域转来研究金融的，要知道金融的数据跟其他领域的数据有很多不一样的地方。

比如人工智能应用最广的计算机视觉，如简单地对图像进行识别和分类。Google Brain著名的应用是看一幅图中有没有猫。当然，计算机并没有猫的概念，它需要从训练集中学习大量的图片，有的有猫，有的没有猫，需要人工标记好。然后再从验证集和测试集中进行检查。

人工智能问题有几大特征，如下所述。

- 人类的误差很低。如果评价人类对于判断一幅图中有没有猫的能力，误差基本上可以认为约等于零，普通人都能轻松做到，毕竟视觉系统经历了数百万年进化，已经非常厉害了。但对计算机来说，或许这并不容易。比如猫有不同的颜色，用RGB三色系统来表示，对于128×128的图片，也需要128×128×3长度的向量才能表示。计算机在这方面的误差或许会高一些，但一般也只是1%或是0.5%。很多人工智能研究的问题都是不断提高计算机的能力，使之达到人类的水平。
- 样本间分布很一致。如果都是相机拍的照片，那么照片的分布是非常一致的；如果都是互联网的照片，那么也很一致。如果训练集是互联网的照片，而验证集和测试集是相机拍的照片，或许会有一点问题。但一般来说只是在训练到验证之间会有比较大的下降表现，而验证集和测试集的误差是很小的。因此，这类模型调整参数的话可以有比较好的效果。如果训练集和人类水平相差很大，那么就增加模型复杂程

度；如果训练集和验证集相差很大，那么就加大正则化减少过度拟合；如果验证集和测试集相差很大，那么就可能是数据取的不大好，须重新划分数据。

- 数据信噪比高。图像数据的噪音一般指的是图像边缘会有些模糊，或者照片拍摄的时候有雾影响。图像处理有各种各样的去噪方法，也称为滤波，包括各种高通滤波、低通滤波等，针对不同噪音有不同的滤波，并且还有小波去噪等高级方法。整体而言，图像数据的信噪比是非常高的，数据质量较高，建模就有保障。

以上就是人工智能问题的基本特点，它们一般适合用在比较复杂的模型，比如神经网络以及最近改头换面的深度学习。它们这些领域当然也有很多自己特定的研究问题，比如软件跟硬件结合，各类提高模型计算速度的技巧，以及计算机视觉特有的 dropout 等正则化方法等，这些在传统的统计学里面都不存在。但事实上很多其他领域的问题并不具备这些特征，它们或许更适合用在传统的统计学方法。

以生物统计在制药方面，或者医学领域看哪些因素对疾病有影响为例。由于病人的样本数量不是很多，一般只有几十人，但潜在的致病因素非常多，可能有几十个甚至上百个，这种参数数目 p 远超样本数目 n 的所谓 p>>n 的高维统计问题，就需要用到稀疏建模（sparsity）来解决。当然，人工智能问题虽然也可以用稀疏性这些方法，但并不会起到什么大的作用，对很多生物统计建模来说，稀疏性建模却能起到决定性作用。

还有就是人工智能问题的时间特征。比如制药需要针对一些细菌和病毒，而这些细菌和病毒是会进化的，那么数据样本的分布就会很不稳定，统计模型也要跟着改变。但人工智能问题在这方面不会这么敏感。比如人脸识别的门禁系统，人脸一般需要几万年才会有大的改变，因此机器在算法上的调整不需要太频繁。

另外就是数据的噪音方面。金融时间序列具有高噪音的特征，毕竟大家本质上都想通过预测价格低买高卖来赚钱，使价格变化非常快，很难利用过去的信息来预测未来的价格变化，因此过去的很多信息都沦为噪音，不具有预测性。而人工智能领域这方面的问题则少很多，比如人脸识别，虽然也有人会故意伪装自己来达到某种目的，但绝大多数正常情况下这是不大可能的，而金融数据