0.0101D0TDKDQI0ro0o10100030010T⁰101的t10T00016107C10

tor8161btor6T010rn0efpo orarerG61900Eniaroro

IO50000

C0100rGiorvntoroiT0r0t0Tortoro10002a4oE0 oL01197C6t⁰

1010rOH01970KD0DNrorotat0

**第17章** **数据科学实践**



本章将面向量化交易应用，从数据采集、模型训练、预测、评价到可视化等，带领读 者实现数据处理各个关键环节，打通整个流程。在这个数据科学的实践中，将对结构化的 价格数据和非结构化的新闻/博客数据进行分析，以指导交易决策。

**17.1** **概** **述**

**17.1.1** **算法交易和量化交易**

算法交易和量化交易分别是从英文的Algorithm Trading和 Quantitative Trading 直接 翻译过来的。除此之外，还有自动化交易 (Automated Trading)、程序化交易 (Pro- grammed Trading) 等相似的概念。在这里，不深入区分这些概念的细微差别，我们认为 这些概念的内涵将逐渐趋同。

量化交易是建立交易模型(可以是基于规则的模型，也可以是机器学习模型),利用 计算机分析必要的数据，产生交易信号，自动执行订单，以期获得利润的过程。

量化交易具有若干优势，包括：(1)交易模型体现出很强的一致性。交易模型不像人 类投资者那样具有情感，它仅仅依赖于数据分析结果进行交易决策，因此，它能够保持一 致的交易行为。相反，人类投资者由于受到自己情感的影响，在投资过程中，很难坚持既 定的原则。(2)交易模型可以同时处理很多项金融资产(股票、期货等)。(3)交易模型 进行数据分析和执行订单的速度比人更快。

在这里需要强调的是，量化交易并不是万能的。计算机系统永远无法达到人类投资经 理的市场理解能力、反应能力和心理素质。计算机系统的优势在于它能够不知疲倦地追踪 和分析大量的信息。

金融市场(股票市场、期货市场等)是一个动态变化的复杂系统，进行价格预测是一 件很困难的事情。著名的有效市场理论 (Efficient Market Hypothesis,EMH) 指出，所 有可以获得的信息，立即反映在股票价格的变化中，只有新的信息到达，才能引起股票价



格的新变化，因此，对金融市场的价格运动进行预测几乎是不可能的。

一方面，信息的传播有一定的延迟；另一方面，不同的人对接收到的同一信息会有不 同的解释。很多研究表明， 一些宏观经济变量 (Macroeconomic Variable) 和技术指标 (Technical Indicator) 在一定程度上能够预测股票价格的未来变化。比如，相关研究表 明，南亚地区的四个股票市场的未来收益存在一定的可预测性 (Predictability)。

在实际应用中，世界范围内的各大银行、对冲基金 (Hedge Fund) 广泛采用算法交 易技术和模型。根据调查，仅仅在美国，2000年从华尔街的算法交易系统发起的订单 (Order) 占到全美股票交易量的40%。到2008年，这个数字攀升到60%。到2014年， 这个数字超过了70%。这些交易系统属于各大投资银行、对冲基金、机构投资者等，它们 各展其能，研发了专有的量化交易技术，对于各自的技术细节则秘而不宣。国内的机构投 资者，包括公募基金、私募基金等，也开始采用量化交易系统、技术和策略。

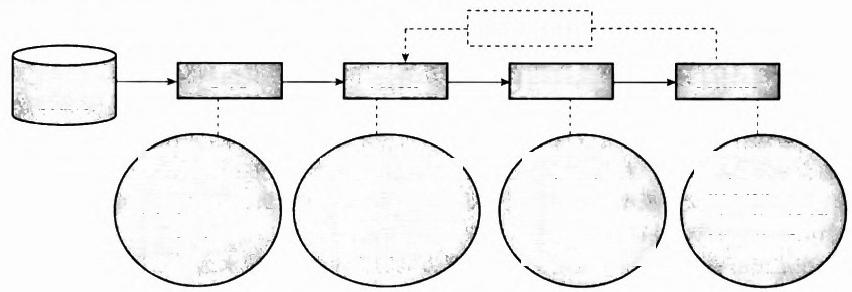
在量化交易方面，最值得关注的公司是文艺复兴科技公司。文艺复兴科技公司是1988 年成立的一家美国私募基金公司，它发行的第一只基金称为大奖章基金 (Medallion) 。 该 公司的两位创始人，也是产品经理，分别是西蒙斯和埃克斯，他们分别于1967年和1976 年获得数学界的最高荣誉——美国数学学会5年一度的伟布伦奖，这正是大奖章基金名称 的由来。自1988年以来，大奖章基金的年均投资回报率超过30%,比投资大师索罗斯和 股神巴菲特的投资回报率都要高很多。

人们采用两种分析策略来预测股票价格的变化： 一种是基本分析 (Fundamental A- nalysis); 另一种是技术分析 (Technical Analysis)。基本分析是通过分析影响股票内在价 值的各种经济因素，对股票价格做出预测， 一般用于中长期预测。技术分析派则相信历史 总会重演，他们使用作图工具、指标计算和数据分析工具，寻找可能盈利的价格变化模式 (Pattern) 。 技术分析一般用于中短期价格预测。

**17.1.2** **量化交易系统的四大模块**

一个量化交易系统包含四个主要的模块，分别是：(1)数据收集模块；(2)模型与数 据分析模块；(3)订单执行模块；(4)监控模块(见图17-1)。

Self adapting



**Model**

**Source**

Statistical method Data mining

Machine learning Social network

Fuzzy logic

Frequency

Buy/sell Size

Time

Holding period

Objective

function

Performance vs. benchmarks

Historical

Market data News

Company Other

Execution

Monitor

Data

**Data**

**图17-** **1** **量化交易系统的概念模型**

数据收集模块，收集必要的数据，以进行分析，帮助决策，即生成交易信号。可以收 集的数据包括历史价格和交易量数据、新闻、博客/微博、公司的季报/年报、国内宏观经 济数据、国际宏观经济数据等。其中，历史价格和交易量数据，可以从财经网站比如Ya-

hoo Finance获得，实时的价格和交易量数据可以从交易所获得，新闻、博客/微博数据可 以从互联网上获取，公司的季报/年报、国内宏观经济数据、国际宏观经济数据可以从权 威的官方网站获取。

模型与数据分析模块利用交易模型，分析相关的数据.生成交易信号 (Trading Sig- nal)。主要的分析方法包括统计分析、数据挖掘与机器学习方法。近年来，大数据技术和 深度学习技术应用到量化投资领域，取得了良好的效果。其中， Binatix 公司是把深度学 习技术运用到量化交易中的一家创业公司，根据其管理层的说法，他们已经开始盈利了 (见图17-2)。

**Introducing Binatix:A Deep Learning Trading Firm That's**

**Already Profitable**

**Tha startup s gKECS sa,appiying the AJ tachnique has given them anedge In imesting**

n li M gu91



**图17-2** **Binatix公司利用深度学习技术实现量化交易**

资料来源： <https://www.recode.net/2014/9/10/11630724/introducing-binatix-a-deep-learning-trading-firm-thats>- already.

订单执行模块负责把交易模型生成的交易信号发送到交易所，真正地执行交易订单 (Order Execution), 返回订单执行结果。订单的执行需要考虑市场冲击 (Market Im- pact) 、滑点 (Slippage) 等因素的影响。可以通过把大的订单分解成一系列小订单来减少 市场冲击。考虑到滑点问题(订单执行时，价格已经在不断变化),目的是希望按照预期 的价格拥有金融资产头寸 (Position)。

监控模块负责监控系统运行。具体的功能包括进行投资组合的管理、风险管理以及对 模型进行持续的优化和改进等。其中，风险管理部分需要考虑适时地止损 (Stop Loss) 或者获利了结 (Take Profit)。

在上述几大模块中，最重要的模块是数据分析和交易信号生成，本章后续内容将重点 讲述相关的数据分析技术。

**17.1.3** **交易策略的研发、测试、上线过程**

交易策略的研发、测试和上线过程大致分为以下几个步骤，分别是：(1)交易策略研 发。在这个阶段，程序员编写投资策略代码。这些代码可以是对投资专家的投资思想的固 化，也可以是基于机器学习模型的策略的实现。基于机器学习的交易策略，充分利用大数 据实现模型的训练，使得模型可以适应不同的市场状况.做出比较准确的预测。(2)历史 回测 (Back Testing)。交易策略开发完成之后，首先要在历史数据上进行回测，并计算性

能指标.根据性能指标评价模型的优劣。(3)实盘模拟 (Paper Trading)。如果对模型的 历史回测得到的结果比较好，也就是交易模型在历史数据上的表现不错，可以考虑把模型 应用到实时的交易所价格数据上.但是订单并没有路由 (Route) 到交易所而是模拟执行， 也就是仅仅记录做了什么操作，盈利还是亏损。实盘数据是正在发生的交易以及价格的变 化，是交易模型没有见过的。模型的泛化能力决定了其性能好坏。(4)实盘交易 (Live Trading) 。 如果实盘模拟的结果显示交易策略的泛化能力还不错.就可以对交易模型进行 生产部署，在实际交易中使用该交易模型。实盘交易与实盘模拟的区别在于实盘交易的订 单直接路由到交易所执行。

在历史回测、实盘模拟、实盘交易等步骤中，如果发现交易策略的性能不是很理 想，需要回头去对交易模型进行调整和优化。对交易模型进行调整和优化的方法包括改 变模型参数、使用新的数据和指标、使用新的数据分析方法或者对若干分析方法进行综 合 等 。

在实盘交易过程中，需要对交易模型进行持续的监控，因为交易模型进行了真正的买 卖操作，稍有不慎便可能造成不应有的损失。使用了一段时间以后，有可能现有的模型不 能适应新的市场情况的变化，需要放弃该模型，研发新的模型。

**17.1.4** **量化交易系统的评价指标**

对于交易模型的评价，人们最感兴趣的是其年化利润率 (Annualized Profit Ratio, 也 称收益率)。如果一个交易模型不能在无风险的定期存款以外创造额外的收益，它是没有 用的。假设我们投资的初始资金为10000美元， 一年以后资产价值(包括现金和股票) 为11000美元，那么利润率为10%。如果投资周期不是一年.那么我们得到的利润率需 要转化成年化利润率，其计算公式为：

年化收益率=(投资收益/本金)!(投资天数/365)×100%

人们往往追逐更高的利润率而忽视了其他方面。我们认为利润率不是越高越好，差不 多就行，交易模型的优劣还需要看其他指标。除了年化利润率，还有两个重要的性能指 标，即最大回撤 (Max Drawdown,MDD) 和夏普指数 (Sharpe Ratio)。

最大回撤指标评价一个交易模型及其投资组合的风险度，最大回撤越小越好。最大回 撤的计算公式为MDD=(A-B)/A。 式中，A 为最大下跌之前的资产最大净值 (Peak Asset Value before Largest Drop);B为资产净值创出新高之前的最低的资产净值 (Low- est Asset Value before New High Established)。比如，一个投资组合的开始净值为10000 美元，这个净值经过一系列股票交易以后发生变化，第一天为17000美元，第二天为 8000美元，第三天为13000美元，第四天为7000美元，第五天为19000美元，那么最 大回撤为(17000-7000)/17000=58.8%。最高净值19000美元没有用来计算最大回 撤，因为这里的回撤是从最高点17000美元开始的。

夏普指数是由 William F.Sharpe发明的一个指标，他用标准差 (Standard Deviation) 以及超额收益 (Excess Return) 来计算每单位风险获得的收益，这里的风险指的是波动 性。假设投资组合的年化收益率为12%,波动性为10%,无风险的定期存款利率为5%, 那么夏普指数为(0.12-0.05)/0.1=70%。投资模型的夏普指数越高.显示投资模型越 稳健，收益越好。



**17.2** **工具和平台介绍**

**17.2.1 Zipline 函数库介绍**

Zipline 是 Quantopian 公司提供的开源 Python 软件包，用于开发量化交易程序。 Quantopian 自身的交易云平台就是使用Zipline 开发的。Zipline 是一个事件驱动的软件系 统，它支持交易策略的历史回测 (Back Testing) 和实盘交易 (Live Trading) 。Zipline库 提供了常用的技术指标的计算函数，包括移动平均指标等，在用户编写的交易策略中，可 以很方便地调用这些函数计算价格指标。Zipline的历史价格数据输入和性能指标输出，采 用的是Pandas 的 DataFrame 数据结构。

Zipline 的接口简单，易于理解，容易使用，同时给予用户尽可能灵活的定制能力。用 户可以使用包括scipy,statsmodels,sklearn,matplotlib 等统计与机器学习库、数据可视 化库，实现交易系统的数据分析功能，并且对交易结果进行可视化。用户可以使用两种方 式安装Zipline, 具体如下。

在 Windows 平台上，从开始菜单，打开 “Anaconda2(64bit)” 程序组的 “Command Prompt” 命令，打开命令行窗口，使用pip install zipline命令，安装Zipline (假设用户已 经安装Zipline 依赖的其他第三方库)。另一种方法是使用Conda Package Manager软件包 管理器进行安装，在命令行窗口，使用conda install-c Quantopian zipline 命令，安装Zi- pline 。Anaconda2 除了基本的Python 开发环境以外，已经集成了conda。

**使用Zipline 开发交易策略，其程序结构很简单。下面展示一个最简单的交易策略。**

**对于每个** **Zipline 程序，用户至少需要定义两个函数，分别是** **initialize(context) 和** **handle\_data(context,data) 。** **在交易策略开始执行之前，** **Zipline 调用initialize函数，参** **数是** **context 对象。在** **initialize 函数中，context 对象可以对交易策略用到的一些数据结构** **进行初始化。context 对象包含运行交易策略需要的上下文信息，包括杠杆率、交易成本、** **滑点率、可以使用的资金量、当前金融资产头寸等。**

Z**ipline运行一个交易策略时，它以事件驱动的方式，迭代式(一次次地)调用handle**\_ **data** **函数。context** **对象作为handle\_data函数的一个参数，在历次迭代式调用handle\_data** **过程中，在上次调用和本次调用以及下次调用间传递必要的信息。**

**data** **对象则把当前交易策略交易的各个金融资产(股票)的历史价格信息保留起来**， 让**handle\_data 函数可以访问到。其中，data 对象包含了当前** **Trading Bar,Trading Bar 把当前时段(1天、1小时等)的最高价** **(high)、** **最低价** **(low)、** **开盘价** **(open)、** **收盘** **价** **(close) (这些价格简称** **OHLC) 和交易量数据组织起来。**

**Zipline的常用函数在zipline.api 模块中进行定义。在本实例中，我们使用了order** (a**rg1,arg2)** **函数，它带两个参数，** **argl** **表示要交易的证券** **(security),arg2** **表示交易** **的数量。如果交易数量是正数，order 函数执行买入操作(做多，** **Long), 如果交易数量是** **负数，order 函数执行卖出操作(做空，Short) 。** **在本实例中，我们在每一次迭代调用** **handle\_data 的过程中，买入Apple 的10份** **(Shares) 股票。需要注意的是，这个实例的**

**目的是展示** **Zipline 交易策略的框架，不分青红皂白地买人，有可能造成损失。**

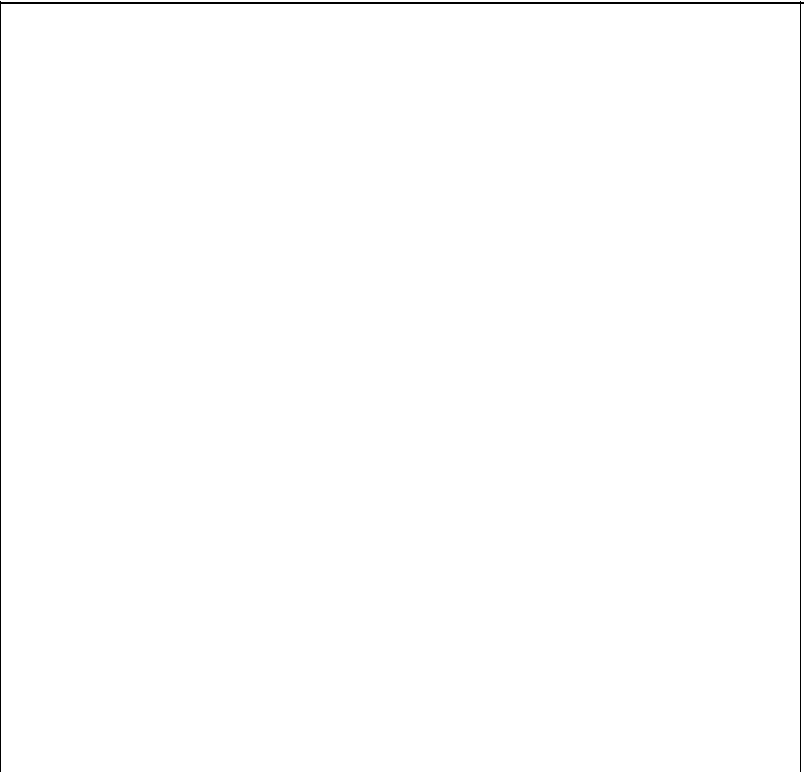
**在这个实例中，使用record方法，在每一次** **handle\_data** **函数的调用中，把某个变量** **名和变量值保存起来。当交易策略执行完毕后，可以查看在算法执行过程中记录的变量的**

值，可以追踪到算法的执行情况。

**为了执行该交易策略，需要从** **Yahoo Finance 上下载一些价格数据，可以通过调用** lo**ad\_bars\_from\_yahoo 方法来获取某只股票的历史价格。下载的价格数据，以** **Pandas** 的 DataFrame 数据结构进行组织。

最后，调用TradingAlgorithm 函数，创建一个交易策略对象，TradingAlgorithm 的参数 包括initialize方法和handle\_data 方法。然后把下载的数据作为测试数据，运行交易策略对 象(调用其run 方法)。这个run 方法，首先调用initialize方法对交易策略进行初始化，然后 把历史价格数据，以数据流的方式，1天的价格 (OHLC) 接着1天的价格 (OHLC) 的方 式，递交给handle\_data 函数执行。在本实例中，我们每次买人10份AAPL 股票。

从这个实例，我们可以看出 Zipline 为用户约定了几个主要的函数接口，用户只要完 成这几个接口，就实现了一个交易策略。Zipline提供的TradingAlgorithm 对象，则接受 用户的函数定义，构造 TradingAlgorithm 对象。TradingAlgorithm 的 run 方法，则把价 格数据以数据流的方式，馈入用户定义的方法，实现交易策略的运行和测试。

#run in spyder

#a simple trading strategy import pytz

from datetime import datetime

from zipline.api import order,symbol,record

from zipline.algorithm import TradingAlgorithm

from zipline.utils.factory import load\_bars\_from\_yahoo

def initialize(context):

pass

def handle\_data(context,data):

order(symbol('AAPL'),10)

record(AAPL=data.current(symbol('AAPL'),'price'))

#(1)从Yahoo!Finance 装载价格数据

start=datetime(2011,1,1,0,0,0,0,pytz.utc).date()

end =datetime(2012,1.1,0,0,0,0,pytz.utc).date()

data=load\_bars\_from\_yahoo(stocks=['AAPL'],start=start,end=end)

print data['AAPL'].head()

print 'data downloaded

#(2)运行交易策略

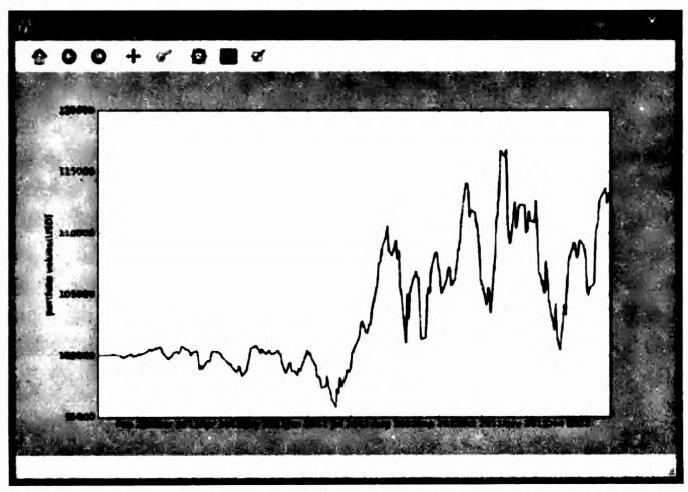
algo\_obj=TradingAlgorithm(initialize =initialize,handle\_data=handle\_

data)

|  |
| --- |
| results =algo\_obj.run(data)  print 'algorithm strategy runned  #(3)显示结果  import matplotlib.pyplot as plt plt.plot(results.portfolio\_value)  plt.ylabel('portfolio values(USD)') plt.show() |

可以打开Spyder 编程环境，新建一个文件，把上述代码拷贝进去，然后运行它。

这段程序的运行结果如图17-3所示，这张图展示了这个交易策略运行过程中的投资 组合价值 (Portfolio Value) 的变化。



**图17-3** **simple 交易策略投资组合价值的变化**

**17.2.2 Quantopian介绍**

**1.** **平台**

算法交易平台提供商Quantopian 于2011年成立于波士顿。 Quantopian 首次在云计算 平台上提供算法交易服务，用户通过浏览器使用这些服务，他们可以基于Zipline 库编写 交易策略，然后用Quantopian 数据库里的历史数据对算法进行回测，这些都是免费的。 目 前 ，Quantopian 支持分钟级的交易频率。当算法在回测的过程中取得满意的效果，用户 就可以通过Quantopian 平台进行实盘交易。

Quantopian 通过浏览器提供了集成开发环境 (Interactive Development Environment, IDE) 。用户可以从社区里拷贝现有的算法进行改进，或者从头开始写自己的算法。算法编 写完成后，马上就可以在Quantopian 平台上，使用2002年以来、分钟级的美国股票价格

数据以及基本面数据 (Fundamental Data), 对算法进行回测，用户的算法受版权保护。

在 Quantopian 平台上，用户可以携自己的交易策略参加比赛，在实盘模拟中一较高 下。Quantopian 平台支持 Interactive Brokers(IB) 和 Robinhood 两家经纪公司的实盘交 易账户，用户可以在它们那里开户.经过配置后，就可以进行实盘交易。

Quantopian 网站维护了一个社区，用户可以通过社区寻求帮助、分享想法、讨论代码 以及分享数据。通过社区.用户能够互相学习，甚至协作开发交易策略。

**2.** **数据**

为了对交易策略进行回测，需要历史价格数据。目前， Quantopian 提供了自2002年 以来的分钟级美国股票市场的历史价格数据以及基本面数据 (Fundamental Data)。这个 数据集每晚更新。分钟级的 Price Bar 包含该分钟时间段内的最高价、最低价、开盘价、 收盘价以及交易量数据。

Quantopian 维护的数据集包括已经退市的公司的价格数据。如果数据集不全，用户的 交易策略就不能得到很好的回测。由于倒闭的公司的价格数据和基本面数据并没有拿来对 交易策略进行回测，交易策略是偏向幸存者的 (Survivor Bias)。

关于5000家公司的基本面数据，由 Morningstar提供，Quantopian 用户可以免费使 用。基本面数据集提供了超过600个指标 (Metric), 用户可以在交易策略的回测中使用。

Quantopian还把一些合作伙伴提供的数据集提供给用户使用，有些数据集是免费的，有 些数据集需要按月进行订购。这些数据集包括VIX数据集、新闻情感数据 (News Sentiment)、 财报发布日程表 (Earnings Calendar, 关于季度收益情况公告及发布日期和时间的数据集)。 VIX数据集是芝加哥期权交易所 (Chicago Board(ptions Exchange,CBOE) 发布的关于标普 500指数期权 (S&.P500 Index Options) 的隐含波动率 (Implied Volatility) 指标数据。

对于实盘模拟 (Paper Trading) 和实盘交易 (Live Trading) 来讲，Quantopian 从 Nanex 公司的NxCore 服务获得实时交易数据 (Real Time Trade)。这些数据组装成分钟 级的 Price Bar, 然后馈人交易策略，驱动其执行。实盘模拟有15分钟的延迟，实盘交易 则实时处理，没有延迟。

**3.** **盈利模式**

Quantopian公司的目标是所有的人，不管他是什么背景，有什么经验，来自哪里，都 可以凭本事一试身手。Quantopian 提供工具(编程环境、函数库)、数据和资金，用户只 需关心交易策略的研发。

Quantopian 的收入来源主要有如下几种方式，包括：(1)如果用户使用Quantopian 实现实盘交易(真实交易),Quantopian 在这个环节收取一定的费用。(2) Quantopian 交 易平台吸引了大量优秀的人才来编写交易策略。平台的用户数已经突破10000,这些用户 来自180多个国家，他们有的是学术机构的研究人员，有的是工业界的数据科学家或者开 发人员，他们正在努力开发赚钱的投资算法，目前已经开发出40多万种算法。在这些算 法中，根据Quantopian 平台的历史回测、实盘模拟、实盘交易等数据， Quantopian 公司 将了解到哪些算法是真正赚钱的。Quantopian 的投资人将和这些算法的作者洽谈，获得他 们的授权。经过对他们的算法进行进一步的优化，用来管理投资基金。算法的作者可以从 投资收益中，根据他们提供的投资策略的表现获得报酬。

**4.** **交易模型的开发与回测**

本章后面将要介绍的交易策略将不在本地运行，而是充分利用Quantopian 云平台进 行开发和测试。



**17.3** **基于规则的交易策略**

本章中，我们将研究两种类型的交易策略，分别是基于规则的交易策略和基于机器学 习的交易策略。基于规则的交易策略，把交易规则直接写成代码，根据历史价格上计算的 指标和一定的规则，判断买入和卖出的时机。

**17.3.1** **移动平均交叉交易策略**

移动平均交叉交易策略的核心思想是从历史价格中计算两个移动平均，其中一个的天数 较小(比如10天，称为短移动平均),一个的天数较大(比如30天，称为长移动平均)。当 短移动平均大于长移动平均，也就是短移动平均在长移动平均之上，我们认为价格的总体趋 势是向上的，于是买入股票。如果短移动平均小于长移动平均，我们认为价格的总体趋势是 向下的，于是卖出股票。下面是实际代码，关键代码的后边增加了必要的注释。

#run in quantopian

#a simple dual moving average crossing strategy def initialize(context):

context.i=0

schedule\_function(trade,date\_rules.every\_day(), time\_rules.market\_close(minutes =60))

def trade(context,data):

#def handle\_data(context,data):

#Skip first 30 days to get full windows

context.i+=1

#第30天及其以后，才能计算有效的30天的移动平均 if context.i<30:

return

# 对Google 股票进行操作

stock=symbol('GOOG')

#if get\_open\_orders():return

#计算10天的移动平均

#计算30天的移动平均

mavg\_fast=data.history(stock,'price',10,'ld').mean()

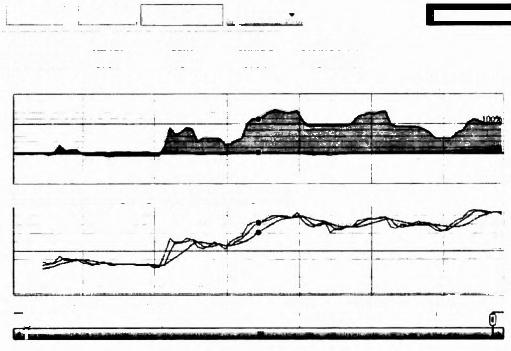
mavg\_slow=data.history(stock,'price', 30,'1d').mean()

#Trading logic

#是否可以进行交易，参考When to use data.can\_trade

|  |
| --- |
| #<https://www.quantopian.com/posts/when-to-use-data-dot-can-trade> if data.can\_trade(stock):  #if(True):  if mavg\_fast>mavg\_slow:  #order\_target\_percent(stock,10.0)  #order\_target orders as many shares as needed to #achieve the desired number of shares  #使用剩余的现金，买入股票  order\_target(stock,100) elif mavg\_fast<mavg\_slow:  #order\_target\_percent(stock,0.0)  #把所有的股票卖出  order\_target(stock,0)  #Save values for later inspection  #记录每个时间点(每天)的收盘价、10天移动平均、30天移动平均  record(GOOG=data[stock].price,  short\_mavg=mavg\_fast,  long\_mavg=mavg\_slow) |

在2015年1月1日到2016年9月14日的Google 公司的股票价格上做历史回测，结 果如图17-4所示。该交易模型在回测中的利润率是101.48%,最大回撤是一43.94%, 夏普指数是0.83。对于这个结果，我们不应该过于乐观。一个交易模型是否足够好，需要 进行更加彻底的测试(其他时间段、其他股票等)。



1 >

RETURNS

101.48%

115.93%■Benchmerk(SM 1.85%

-100%

■long\_mavg 680.42 ■shortmevg724.79

600

400

Ap 2015 215 Od2015 Aan2016 Ap2016

u₁2015 Jan 2016

Dmzo15

to 09/42016510000

ALPHA BETA

0.67 1.91

US Equdes

SHARPE

0.83

DRAWDOWN

-43.94%

址 2 0 %

ul 2016

Woek of Nov 9.2015

01/012015

■GOOG728.75

■Algorithm

**图** **1** **7** **-** **4** **移动平均交叉交易策略的历史回测结果**

**17.3.2** **均值回归交易策略**

在量化交易领域，均值回归 (Mean Reversion) 模型假设股票价格有向均值运动的趋 势，也就是股票价格离开均值太远，就很有可能回到均值附近。



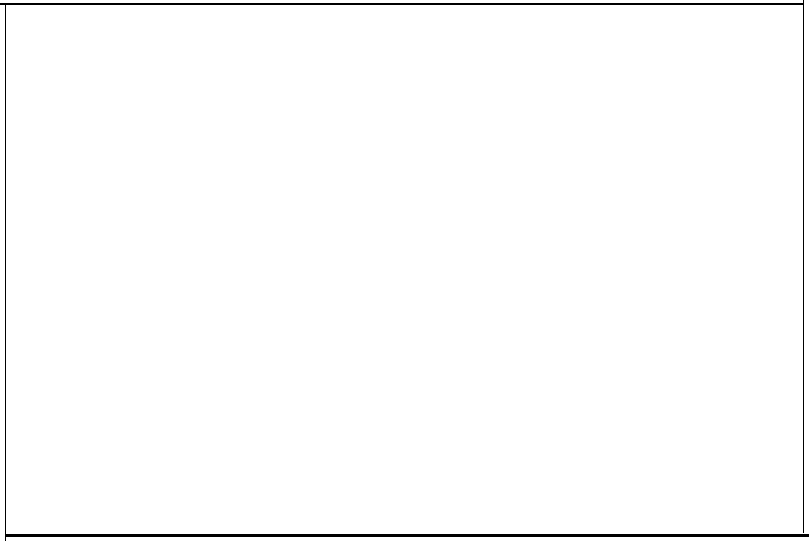
在这个交易策略里①,每个星期一对交易量较大的股票进行排序，排序的标准是最近 5天的收益 (Return) 。 对于倒数20%的股票(过去5天的收益最差),买进；对于Top 20%的股票(过去5天的收益最好),卖出。该交易策略假设，上周表现最好 (Top-Per- forming) 的股票，本周的表现将变差，或者相反。

**17.4** **基于机器学习的交易策略**

**17.4.1 基于Random Forest 分类器的交易策略(分类)**

该交易策略的执行机制是，在每周末交易市场关闭之前5分钟，重新对投资组合进行 再平衡 (Rebalance) 。 再平衡首先计算股票最近11天，每相邻两天之间的价格差，并进 行标准化，然后用这些差值作为输入数据，预测接下来的一天的价格是上涨还是下跌。如 果预测上涨，则买入；如果预测下跌，则卖出。

模型是在交易策略执行的过程中不断被更新的数据训练的，这些数据保留在X 队列和 y 队列中.X 队列保存输入向量，即某个11天的时间窗口内，相邻两天之间的价格差形成 的向量的单位向量，y 队列保存对应的接下来1天的涨跌情况。X 队列和y 队列的容量都是 500,当有新的训练样本 (X,y) 放入队列里，队列原有的最老的样本将被剔除掉。具 体代码如下，重要语句的后边加了必要的注释。



#run in quantopian

#Use the previous 10 bars'movements to predict the next movement.

#Use a random forest classifier.More here:<http://scikit-learn.org/sta>- ble/user\_guide.html

from sklearn.ensemble import RandomForestClassifier

from collections import deque import numpy as np

def initialize(context):

井对波音公司股票进行交易

context.security=sid(698)

#用前10个price bar 进行预测

context.window\_length=10

# 用Random Forest分类器进行预测

context.classifier=RandomForestClassifier()



①[https://www.quantopian.com/posts/sample mean-reversion-1.](https://www.quantopian.com/posts/samplemean-reversion-1.)

#deques are lists with a maximum length where old entries are shifted out #Stores recent prices

context.recent\_prices =deque(maxlen=context.window\_length+2) #自变量或者输人变量(Independent,or input variables)

context.X=deque(maxlen=500)

#因变量或者输出变量(Dependent,or output variable)

context.Y=deque(maxlen=500)

context.prediction =0#Stores most recent prediction

#week\_end 表示每个周末，可以改成every\_day, 表示每一天

#在每个周末，股票市场关闭前5分钟，进行投资组合再平衡

schedule\_function(rebalance,date\_rules.week\_end().time\_rules.market\_ close(minutes=5))

#在每个周末，股票市场关闭时，记录必要信息

schedule\_function(record\_vars,date\_rules.week\_end(),time\_ rules.market\_close())

def rebalance(context,data):

#更新最近的prices

context.recent\_prices.append(data.current(context.security,'price'))

#如果已经收集足够的价格数据

if len(context.recent\_prices)==context.window\_length+2:

#根据前后价格的变化，建立0/1列表

#1表示价格上升，0表示价格下降(相对于前一个price bar) changes =np.diff(context.recent\_prices)>0

#插入 X

context.X.append(changes[:-1])

#插入Y,X 和Y构成一个新的训练样本

context.Y.append(changes[-1])

#当样本数量大于100时，才开始训练模型

if len(context.Y)≥100:

#训练模型

context.classifier.fit(context.X,context.Y)

#预测明天的涨跌

context.prediction=context.classifier.predict(changes[1:]) 井如果prediction ==1,用可用的现金买入股票

#如果prediction ==0,把所有的股票卖出

#如果prediction 在0到1之间，

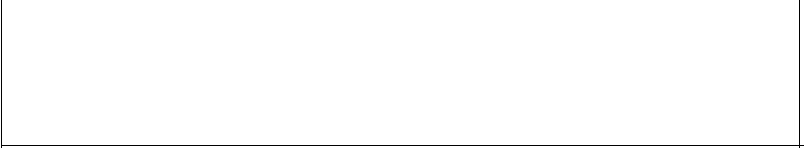
#则在现金和股票之间进行一定比例的配置



order\_target\_percent(context.security,context.prediction)

def record\_vars(context,data):

#记录模型的预测结果

record(prediction =int(context prediction))

.

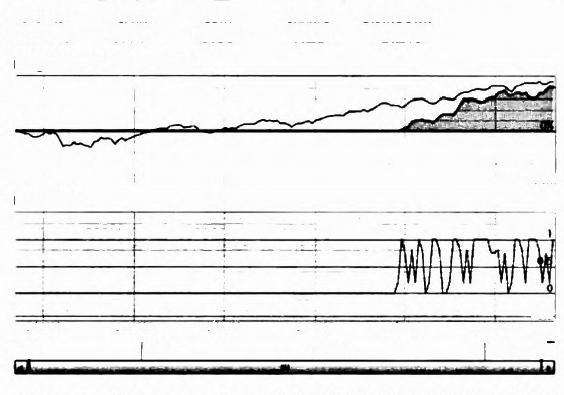
在2011年1月5日到2014年1月5日的波音公司 (Boeing) 的股票价格上做历史回 测，结果如图17- 5所示。该交易模型在回测中的利润率是40 .89%,最大回撤是 一8.2%,夏普指数是1.29。



**05/01/2011 to 05/01/2014 $50000**



kLnutLa



RETURNS

40.89%

**■Algorithm**

-50%

**■prediction**

0.5

JM 2011

了

**图17-5** **基于Random Forest分类器的交易策略的历史回测结果**

ALPHA BETA

0.11 **0.09**

**■Benchmark(SPY)**

M2013 声201J

2013

**DRAWDOWN**

**-8.2%**

**SHARPE**

**1.29**

2014

2014

Jan 2012

u 2012

2012

0

**17.4.2 基** **于SVM 回归模型的交易策略(回归)**

这个交易策略使用SVM 回归模型进行价格预测，然后做交易决策。它使用微软、 Yahoo、Intel的价格数据作为输入 (Input), 预测Apple 的股票价格。

这个交易策略同时使用MACD(Moving Average Crossover Divergence) 指标，辅助 进行交易决策，产生交易信号 (Buy/Sell Signal)。

在股票价格的技术分析中， MACD 指标 (Moving Average Convergence Divergence) 经常被用到。MACD 线用12天的 EMA(Exponential Moving Average) 减去26天的 EMA 绘制出来。MACD 9天的EMA 线用作一个信号线 (Signal Line), 用来确定价格是 否发生反转 (Reversal)。

投资者一般寻找如下的投资机会，即信号线交叉 (Signal Line Crossovers)。当 MACD 指标从下往上和信号线交叉，这是一个牛市信号线交叉，适宜买人；当MACD 指 标从上往下和信号线交叉，这是一个熊市信号线交叉，适宜卖出。

这个交易策略，每天只在市场开盘时进行一次交易， 一天中剩下的时间就不再交易 了。如果馈入这个交易策略的是分钟级的价格数据，那么需要注意这个约束。在程序的处 理逻辑中，利用context.date 来保存当前的日期，实现日期检查。

|  |
| --- |
| #run in quantopian  import talib#technical analysis library #该函数库提供了主要技术指标的计算函数 import numpy as np  import pandas as pd  # 从sklearn 库导人svm 模型 from sklearn import svm  #Setup our variables  def initialize(context):  # 对Apple 股票进行交易  context.sec =symbol('AAPL')  #辅助进行价格预测的股票(价格)  context.auxsec =symbols(MSFT",'YHOO','INTC')  context.pct\_per\_stock=0.5  #Create a variable to track the date change  #这个变量用于追踪日期是否已经发生变化，也就是进入下一天  context.date =None  def handle\_data(context,data):  sec =context.sec  todays\_date =get\_datetime().date()  #Do nothing unless the date has changed and its a new day. #每天只做一次交易，然后如日期仍然是今天，什么也不做  if todays\_date ==context.date:  return  #Set the new date  context.date=todays\_date  #Load historical data for the stocks  #今天以及今天以前40天的4个公司的历史价格  prices =history(40,'ld','price')  #今天以及今天以前200天的4个公司的历史价格  hist =history(200,'1d,'price')  #在价格数据上计算MACD 信号(指标),  #传递三个参数：fast period,slow period,and the signal.  井macd是一个Series,  #可以通过sids (不同股票的id) 来提取4个公司的MACD指标 |

|  |
| --- |
| macd=prices.apply(MACD,fastperiod=12,slowperiod=26,signalperiod= 9)  #训练SVM回归模型，并且用之进行预测  X=hist[context.auxsec].values[:-1,]  y=hist[context.sec].values[:-1,]  newx=hist[context.auxsec].values[-1,]  predicted=ml(X,y,newx)  current\_position =context.portfolio.positions[sec].amount #当MACD信号是负的(看跌),预测价格小于当前价格的1.1倍，  #并且我们拥有股票头寸，则卖出股票  if macd[sec]<0 and predicted<1.1\*data[sec].price and current\_posi- tion>0:  order\_target(sec,0)  #当MACD信号是正的(看涨),预测价格大于当前价格的0.9倍，  #并且股票头寸是0,则买入股票  elif macd[sec]>0 and predicted>0.9\*data[sec].price and current\_po- sition ==0:  order\_target\_percent(sec,context.pct\_per\_stock)  #记录必要信息，MACD/price/predicted  record(macd=macd[sec],  real=data[sec].price,  pred=predicted)  #定义MACD函数  def MACD(prices,fastperiod=12,slowperiod=26,signalperiod=9):  "  Function to return the difference between the most recent MACD value and MACD signal.Positive values are long  position entry signals  optional args:  fastperiod=12  slowperiod=26  signalperiod=9  Returns:macd-signal  "  macd,signal,hist=talib.MACD(prices,  fastperiod=fastperiod, |

又

slowperiod=slowperiod,

signalperiod=signalperiod)

#MACD与信号线的交叉

return macd[-1]-signal[-1]

# 定 义ml 函 数

def ml(X,y,newx):

II!

A demo SVM regression

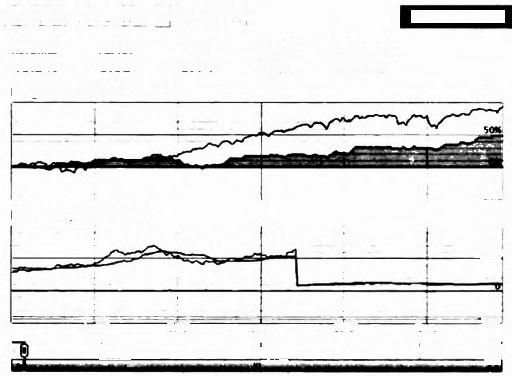
Il

clf=svm.SVR()

clf.fit(X,y)

return clf.predict(newx)[0]

在2011年1月4日到2016年11月24日的上述公司的股票价格上做历史回测，结果 如图17-6所示。该交易模型在回测中的利润率是48.3%,最大回撤是一 17.7%,夏普指 数是0.85。



**01/04/2011 to 11/242016 $1000000**

RETURNS ALPHA BETA

48.3% 0.05 0.14

**■Algonithm ■Benchmark(SPY)**

50%

■real ■pred ■macd

500

-500

2012 2013

2012

DRAWDOWN

-17.7%

018

2016

SHARPE

0.85

5, 1>

冥

2014

7013

**图17-6** **基于SVM** **回归模型的交易策略的历史回测结果**

**17.5** **关于若干专题的讨论**

**17.5.1** **股票价格预测的基础数据**

在量化交易中，进行股票价格预测可以使用的数据包括：

(1)股票的历史价格数据、历史交易量数据，以及在这些数据之上计算的指标，比如 MACD 指标、RSI 指标等。



我们可以设计数据挖掘和机器学习模型，从某只股票的历史价格变化中学习到价格变 化模式，进而做出预测，这种模型可以称为股票内模型 (Intro Stock Model)。我们也可 以把某些股票价格及其指标变化情况作为自变量，某只股票的价格变化作为因变量，建立 预测模型，这种模型可以称为股票间模型 (Inter Stock Model)。

价格数据的时间粒度，可以是天，也可以是小时级、分钟级，甚至秒级的数据，细粒度 的大量的股票价格数据，其数据规模很大，可以利用计算机强大的计算能力进行处理。

(2)对某个上市公司的基本面数据(比如上市公司的财务报表)的分析结果，包括其 资产负债表、现金流量表、利润表等报表的关键指标及其变化情况，比如利润率、近几年 来利润率的变化情况等。

(3)关于公司、行业、整个经济形势的新闻、博客，对这些数据进行分析，了解投资 者的情感倾向，可以用于对交易模型产生的买/卖信号进行确认，或者把情感分析结果作 为模型的自变量，输入模型，直接参与买卖信号的生成。

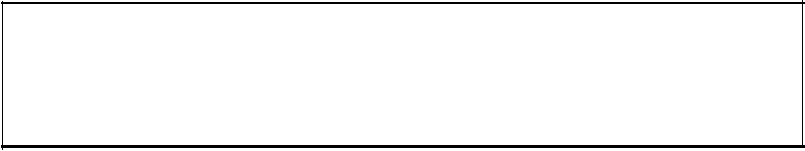
**17.5.2 特征选择** **(Feature Selection)**

在股票涨跌预测、价格预测中，我们可以使用各种各样的数据和指标。这些指标有的 具有较强的预测能力，有些则预测的能力较弱。通过对这些特征进行适当选择，既可以保 持预测准确度，又可以通过对高维数据进行降维处理，提高机器学习和预测的速度。 Sklearn 机器学习库的sklearn.feature\_selection 模块①,提供了相关的类，实现特征选 择/降维处理。

**1. 剔除低方差的特征** **(Removing Features with Low Variance)**

VarianceThreshold 是进行特征选择的最基本的方法，它把所有方差小于某个阈值的 特征全部剔除掉。在所有样本里，取值都相同的特征 (Feature), 其方差为0,没有区分 度，当然都被剔除。

下面的实例，在样本里各个特征都是布尔特征(只能取值0或者1)。布尔特征是伯努 力随机变量，其方差的计算公式为Var[X]=p(1—p), 于是我们可以使用0.8\*(1一 0.8)作为阈值，把样本里超过80%都是0(或者都是1)的特征剔除掉(第一个特征被剔 除了)。



from sklearn.feature\_selection import VarianceThreshold

X=[[0.0,1].[0,1,0].[1.0,0],[0,1.1],[0,1,0],[0,1,1]] sel=VarianceThreshold(threshold=(.8\*(1-.8)))

sel.fit\_transform(X)

**2. 单变量特征选择** **(Univariate Feature Selection)**

单变量特征选择，通过单变量统计检验 (Statistical Test) 来选择最好的特征。具体 包括：(1) SelectKBest 把获得最高得分 (Scoring) 的 K 个 (K Best) 特征保留下来，其 他特征全部剔除。(2) SelectPercentile 把获得最高得分 (Scoring) 的一定百分比 (Per- centile) 的特征保留下来，其他特征全部剔除。(3)对每个特征进行单变量统计检验，Se-

①<http://scikit-learn.org/stable/modules/feature_selection.html.>



lectFpr 可以设定一个假阳性比率 (False Positive Rate,FPR),SelectFdr 可以设定一个伪 发现率 (False Discovery Rate,FDR),SelectFwe 可以设定一个多重比较谬误 (Family Wise Error Rate,FWE) 。(4)GenericUnivariateSelect, 允许用户对特征选择策略进行定制，包括 定制评分函数，以及定制评分模式，比如Percentile,K Best,FPR,FDR以 及FWE 等。

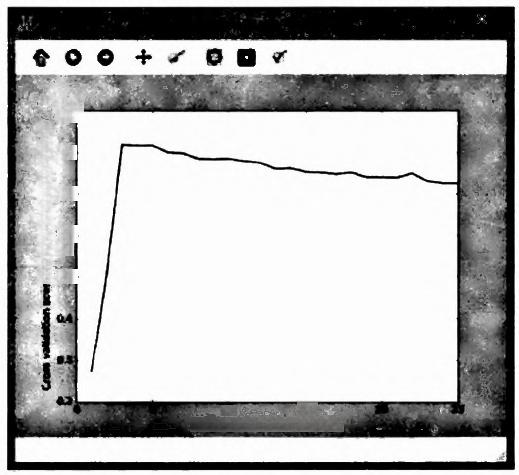
下面的示例代码，对样本进行卡方x² 检验，保留2个最好的特征。

|  |
| --- |
| from sklearn.datasets import load\_iris  from sklearn.feature\_selection import SelectKBest  from sklearn.feature\_selection import chi2  iris =load\_iris() X,y=iris.data,iris.target print X.shape  X\_new=SelectKBest(chi2,k=2).fit\_transform(X,y) print X\_new.shape |

**3. 递归特征消除** **(Recursive Feature Elimination,RFE)**

如果已经有一个预测器，给各个特征赋予权重(比如线性模型中各个变量的系数)。 REF的原理是，通过在不断缩小的特征集合上进行比较(计算一个得分，这个得分可以 是分类器的正确率),对特征进行选择。首先，预测器在所有的特征上进行训练，每个特 征被赋予一个权重。接着，具有最小权重的特征，从特征集合中裁剪掉。在经过裁剪的特 征集合上递归执行这个过程，直到最后获得指定数量的特征为止。

下面的实例，对特征集进行递归特征消除，通过交叉验证自动选择优化的 (Optimal) 特征个数。其中，RFECV通过交叉验证，实现递归特征消除，寻找最优化的特征个数。 图17-7显示了不同的特征个数及其对应的交叉验证分数(预测器的正确率)。

0

( 子 百 4

A7

B

**图17-7** **特征个数vs. 交叉验证分数**



|  |
| --- |
| #如果下面的代码运行出错，请运行conda update scikit-learn 命令更新sci- kit-learn  import matplotlib.pyplot as plt  from sklearn.svm import SVC  from sklearn.model\_selection import StratifiedKFold from sklearn.feature\_selection import RFECV  from sklearn.datasets import make\_classification  #建立一个分类数据集，使用3个提供有用信息的(informative) 特征  X,y=make\_classification(n\_samples =1000,n\_features =25,n\_informative= 3,  n\_redundant =2,n\_repeated=0,n\_classes =8, n\_clusters\_per\_class =1.random\_state =0)  #创建一个RFE 对象，计算交叉验证分数 sVc=SVC(kernel="linear")  #“准确率”得分和正确分类的数量成正比  rfecv=RFECV(estimator=svc,step=1,cv=StratifiedKFold(2), scoring='accuracy')  rfecv.fit(X,y)  print("Optimal number of features:%d"%rfecv.n\_features\_)  #绘制特征个数与交叉验证分数的对应关系 plt.figure()  plt.xlabel("Number of features selected")  plt.ylabel("Cross validation score(nb of correct classifications)")  plt.plot(range(1,len(rfecv.grid\_scores\_)+1),rfecv.grid\_scores\_)  plt.show() |

这段代码的输出是“Optimal number of features:3”,即3是最优化的特征个数。

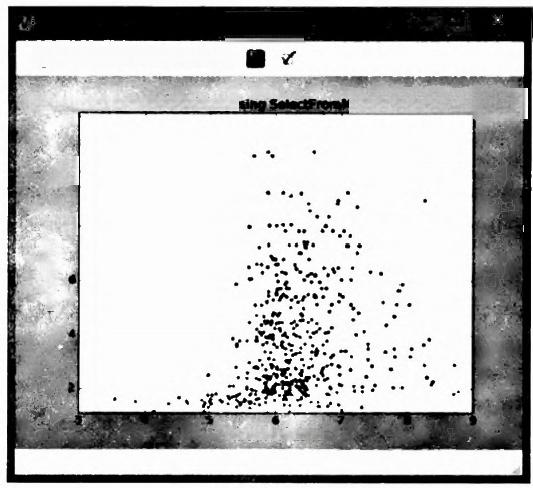
4. 使用SelectFromModel 进行特征选择

SelectFromModel是一个元转换器 (Meta-Transformer), 它可以和任意训练以后具 有coef\_ 属性或者feature\_importances\_ 属性的预测器结合，进行特征选择。

不重要的特征，即特征的 coef\_ 属性值或者 feature\_importances\_ 属性值低于某个 阈值(在模型训练和拟合过程中，获得各个特征的重要性评价),被剔除。用户除了设置 具体的阈值之外，也可以通过设定一个字符串参数来设定阈值，比如 “mean”“median” 等，分别表示重要性 (feature\_importances\_) 或者系数 (coef\_) 的均值、中位数， “0.1\*mean” 表示均值的0.1倍等。下面的实例，从 Boston 数据集中选择两个最重要的 特征，而且无须用户事先指定一个阈值。具体代码如下。图17-8显示了代码执行以后绘 制的两个最重要的特征的散点图。



h



食 ◎ + 0 I

Frotur secdedGron BsIenu 3F

10

Fiotun mmb1

*lalg 0.750*

2 4 u 2

**图17-8** **特征1和特征2的散点图**

|  |
| --- |
| import matplotlib.pyplot as plt import numpy as np  from sklearn.datasets import load\_boston  from sklearn.feature\_selection import SelectFromModel from sklearn.linear\_model import LassoCV  #装载Boston数据集  boston =load\_boston()  X,y=boston['data'],boston['target']  #We use the base estimator LassoCV since the L1 norm promotes sparsity  of features clf=LassoCV()  #最小的阈值为0.25  sfm=SelectFromModel(clf,threshold=0.25)  sfm.fit(X,y)  n\_features =sfm.transform(X).shape[1]  #不断设置阈值，直到选择出的特征数量为2  #对meta Transformer进行训练(拟合到数据上)  while n\_features>2:  sfm.threshold+=0.1 |



X\_transform=sfm.transform(X)

n\_features=X\_transform.shape[1]

井绘制选择出来的两个特征的散点图

plt.title("Features selected from Boston using SelectFromModel with" "threshold %0.3f."%sfm.threshold)

feature1=X\_transform[:,0]

feature2=X\_transform[:,1]

plt.plot(feature1,feature2,'r.')

plt.xlabel("Feature number 1")

plt.ylabel("Feature number 2")

plt.ylim([np.min(feature2),np.max(feature2)])

plt.show()

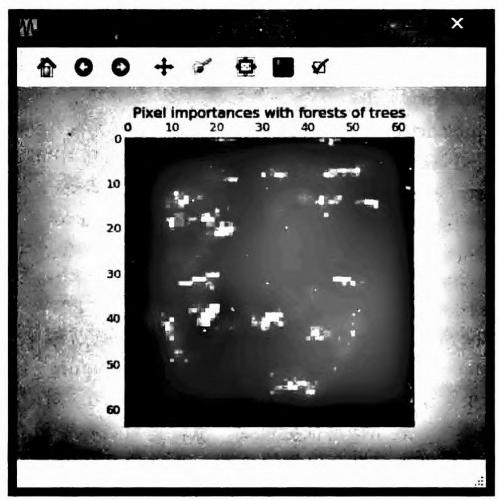
基于决策树的预测器 (Tree-BasedEstimator), 在训练的过程中，可以计算出每个特 征的重要性 (importance) 。 这些重要性参数存放在模型的feature\_importances\_ 属性里， 每个值都是0或者一个大于0的小数，累加起来的总和是1.0。对应某个特征的feature\_ importances\_属性值越大，表示该特征更重要，也就是对目标变量有更强的解释(预测) 作用，于是该特征对于模型来讲具有更大的贡献。通过和SelectFromModel结合，可以把 重要的特征选出来，把不重要的特征剔除掉。下面给出几个实例。

第一个实例，对Iris 数据集进行特征选择，从4个特征中选出2个重要的特征。具体 代码如下：

|  |
| --- |
| from sklearn.ensemble import ExtraTreesClassifier  from sklearn.datasets import load\_iris  from sklearn.feature\_selection import SelectFromModel  #装载数据集  iris=load\_iris()  X,y=iris.data,iris.target print X.shape  井训练模型  clf=ExtraTreesClassifier()  clf=clf.fit(X,y)  print clf.feature\_importances\_  #进行特征选择  model =SelectFromModel(clf,prefit=True)  X\_new =model.transform(X) print X\_new.shape |

第二个实例，使用决策树森林 (Forest of Trees),评估在图像分类任务中 (Image

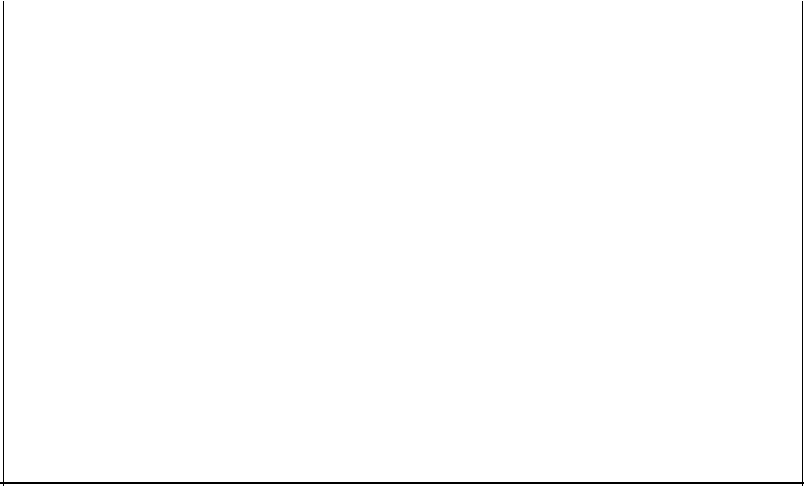
Classification Task),一 幅图像(人脸， Face) 的各个像素的重要性。颜色越红(越热), 表明该像素越重要，如图17-9所示。该实例展示了如何利用多个CPU Core进行并行处 理。具体代码如下：



**图17-9** **决策树森林模型下的像素重要性**

|  |
| --- |
| from time import time  import matplotlib.pyplot as plt  from sklearn.datasets import fetch\_olivetti\_faces from sklearn.ensemble import ExtraTreesClassifier  # 对Forest Model进行并行训练时，使用的CPU Cores**数量** n\_jobs =1  #装载 faces 数据集  data=fetch\_olivetti\_faces()  X=data.images.reshape((len(data.images),-1)) y=data.target  mask=y<5# 限制为5个类别  X=X[mask] Y=y[mask]  # 创 建Forest Model,计算像素的重要性(像素即特征)  print("Fitting ExtraTreesClassifier on faces data with %d cores…"%n\_ jobs) |





t0=time()# 计时开始

forest =ExtraTreesClassifier(n\_estimators=1000, max\_features =128,

n\_jobs=n\_jobs, random\_state=0)

forest.fit(X,y)

print("done in %0.3fs"%(time()-t0))#计时结束

importances =forest.feature\_importances\_

importances=importances.reshape(data.images[0].shape)# 转换为2维数组， 其shape 和 data.images[0] 的 shape 一样，也就是把像素重要性按照矩阵排列

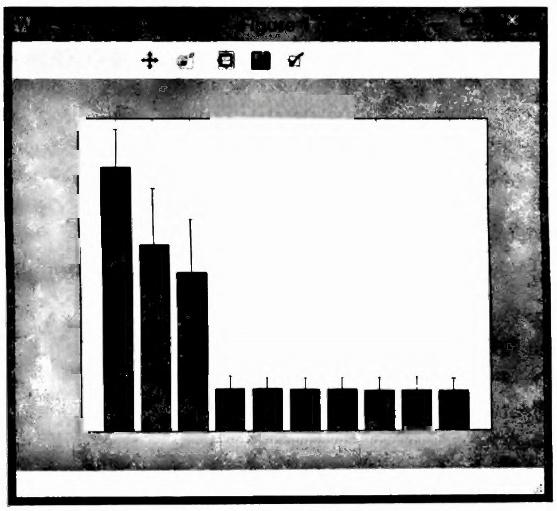
并绘制像素的重要性

plt.matshow(importances,cmap=plt.cm.hot)

plt.title("Pixel importances with forests of trees")

plt.show()

第三个实例，使用Forest of Trees模型，在一个人工的分类任务中评价特征的重要 性。图17-10显示了各个特征的相对重要性，其中柱状表示某个特征的重要性取值(在 所有的决策树中取重要性的平均值),每个柱状上的线条，表示在各个决策树之间该特征 重要性的变化情况(标准差)。由于构造分类任务时指定3个特征为提供有用信息的 (in- formative) 特征，正如预料的一样，从图17-10可以看出，有3个特征是比较重要的， 其他特征则没有那么重要。具体代码如下：

仓

Fgoture importances

0.3广

0.30

0.26

*a20*

015

0.10

D05

3 6 B 7 4 5

*coo* 1 2

**图** **1** **7** **-** **1** **0** **特征重要性排序**

|  |
| --- |
| import numpy as np  import matplotlib.pyplot as plt  from sklearn.datasets import make\_classification from sklearn.ensemble import ExtraTreesClassifier  #建立一个分类数据集，使用3个提供有用信息的(informative)特征 X,y=make\_classification(n\_samples=1000,  n\_features=10,  n\_informative=3, n\_redundant=0,  n\_repeated=0,  n\_classes=2,  random\_state=0, shuffle=False)  #创建一个Forest Model,计算各个特征的重要性  forest=ExtraTreesClassifier(n\_estimators=250, random\_state=0)  forest.fit(X,y)  importances =forest.feature\_importances\_  std=np.std([tree.feature\_importances\_for tree in forest.estimators\_], axis=0)  indices =np.argsort(importances)[::-1]  #输出重要性列表，从大到小的各个特征的下标为[0123586794] print indices  #把特征排序结果显示出来  print("Feature ranking:")  for f in range(X.shape[1]):  print("%d.feature %d(%f)"%(f+1,indices[f],importances[indices  [f]]))  #绘制Forest Model的特征重要性 plt.figure()  plt.title("Feature importances")  plt.bar(range(X.shape[1]),importances[indices], color="r",yerr=std[indices],align="center")  plt.xticks(range(X.shape[1]),indices)  plt.xlim([-1,X.shape[1]])  plt.show() |

17.5.3 模型的组合 (Model Combination)

不同的预测器具有不同的预测能力。 一些预测器在某些市场条件下具有好的预测能 力，在另外一些市场条件下则表现不好。还有一些预测器，则有可能与其形成互补关系， 在不同的市场状况下有相反的表现。提高机器学习模型的预测能力， 一种有效的办法是把 多个模型组合起来，构成一个元模型。下面给出两个实例。

第一个实例，利用投票分类器 (VotingClassifier), 把基于决策树的分类器 (Deci- sionTreeClassifier) 、 基于K 近邻的分类器 (KneighborsClassifier) 、 基于支持向量机的分 类器 (SVC) 结合起来，它们的权重分别是2,1,2。

最后，针对Iris 数据集的两个特征 (Feature), 对上述分类器的分类边界进行绘制， 以比较不同分类器的分类效果。具体代码如下：

from itertools import product

import numpy as np

import matplotlib.pyplot as plt

from sklearn import datasets

from sklearn.tree import DecisionTreeClassifier

from sklearn.neighbors import KNeighborsClassifier from sklearn.svm import SVC

from sklearn.ensemble import VotingClassifier

#装载示例数据

iris=datasets.load\_iris()

X=iris.data[:,[0,2]]# 取第0列、第2列作为输入特征 y=iris.target

#训练分类器

clf1=DecisionTreeClassifier(max\_depth=4)

clf2=KNeighborsClassifier(n\_neighbors =7)

clf3=SVC(kernel='rbf',probability=True)

eclf=VotingClassifier(estimators=[('dt',clf1),('knn',clf2), ('svc',clf3)],voting='soft',weights=[2,1,2])

clf1.fit(X,y)

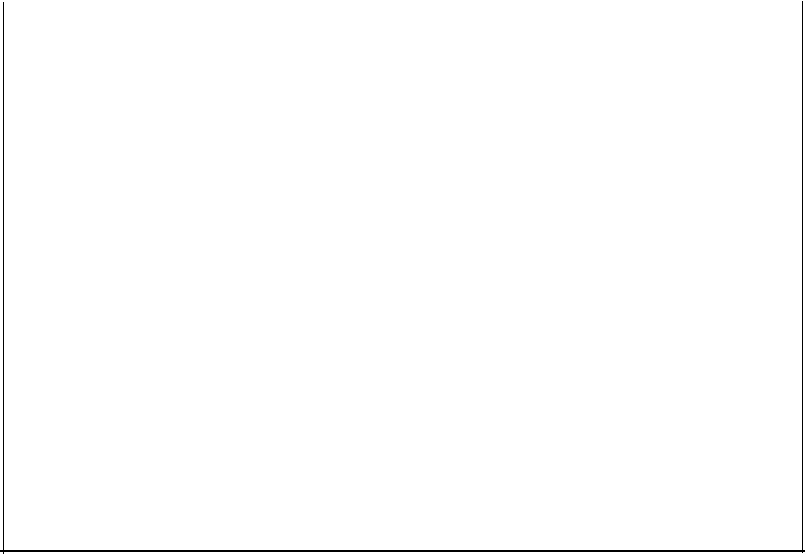
clf2.fit(X,y)

clf3.fit(X,y)

eclf.fit(X,y)

#绘制4个分类器的分类边界

x\_min,x\_max=X[:,0].min()-1,X[:,0].max()+1



Y\_min,y\_max=X[:,1].min()-1,X[:,1].max()+1

xx,yY=np.meshgrid(np.arange(x\_min,x\_max,0.1), np.arange(y\_min,y\_max,0.1))

**f,axarr=plt.subplots(2,2,sharex=** 'col',sharey='row',figsize=(10,8))

for idx,clf,tt in zip(product([0,1],[0,1]), [clf1,clf2,clf3,eclf],

['Decision Tree(depth=4)','KNN(k=7)', 'Kernel SVM','Soft Voting']):

Z=clf.predict(np.c\_[xx.ravel(),yy.ravel()])

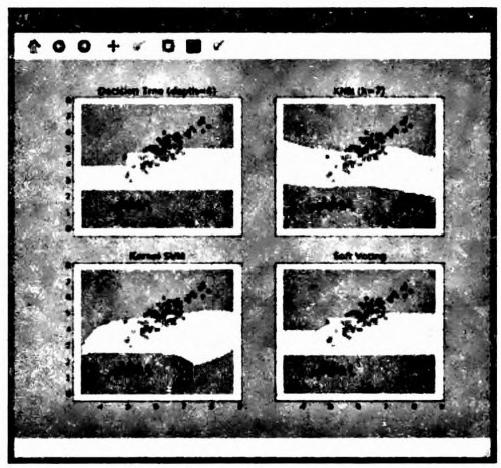
Z=Z.reshape(xx.shape)

axarr[idx[0],idx[1]].contourf(xx,yy,Z,alpha=0.4)

axarr[idx[0],idx[1]].scatter(X[:,0],X[:,1],c=y,alpha=0.8) axarr[idx[0],idx[1]].set\_title(tt)

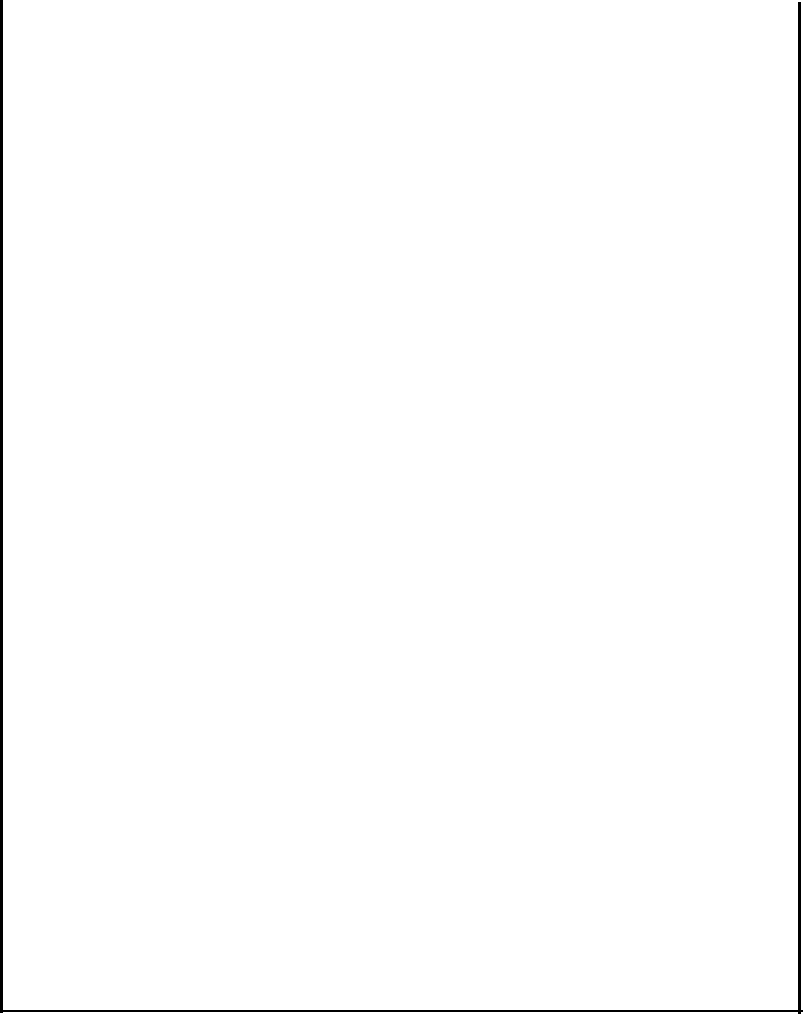
plt.show()

该实例的执行结果如图17-11所示。由该图可以看出，通过投票方式把几个分类器 结合起来，分类的准确性有了一定程度的提高。



**图17-11** **VotingClassifier 的分类边界**

第二个实例，利用AdaBoost 算法对决策树进行提升，使用的数据集是1维的带少量 高斯噪声 (Gaussian noise) 的正弦曲线。经过299次提升 (Boost) 的300个决策树回归 的结果和单一的决策树回归的结果进行了比较。从图17-12中可以看出，随着提升数量 的提高，回归的结果和原始数据集的拟合效果越来越好。该实例的代码如下：

import numpy as np

import matplotlib.pyplot as plt

from sklearn.tree import DecisionTreeRegressor from sklearn.ensemble import AdaBoostRegressor

井创建数据集

rng =np.random.RandomState(1)

X=np.linspace(0,6,100)[:,np.newaxis]

y=np.sin(X).ravel()+np.sin(6\*X).ravel()+rng.normal(0,0.1,X.shape [0])

#拟合回归模型

regr\_1=DecisionTreeRegressor(max\_depth=4)

regr\_2=AdaBoostRegressor(DecisionTreeRegressor(max\_depth=4), n\_estimators =300,random\_state =rng)

regr\_1.fit(X,y)

regr\_2.fit(X,y)

#进行预测

Y\_ 1=regr\_ 1.predict(X)

y\_2=regr\_2.predict(X)

#绘制结果 plt.figure()

plt.scatter(X.y,c="k",label="training samples")

plt.plot(X,y\_1,c="g",label="n\_estimators=1",linewidth=2)

plt.plot(X.y\_2,c="r",label="n\_estimators=300",linewidth=2)

plt.xlabel("data")

plt.ylabel("target")

plt.title("Boosted Decision Tree Regression")

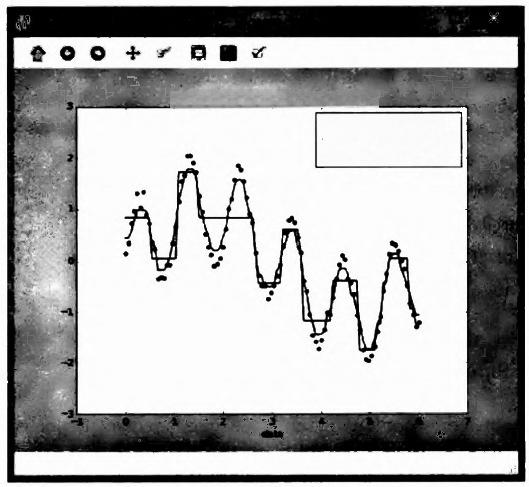
plt.legend()

.

plt show()

机器学习的集成 (Ensemble) 方法，通过把若干预测器 (Estimator) 的预测结果结 合起来，提高模型的泛化能力 (Generalizability) 以及健壮性 (Robustness) 。 集成方法分 为两类，分别是平均方法 (Averaging Method) 和提升方法 (Boosting Method) 。(1) 平 均方法首先独立地创建若干预测器，然后把它们的预测结果进行平均 (Average) 。 一般来 讲，组合预测器比单一的预测器的预测效果要好，因为方差 (Variance) 变小了。主要的 平均方法包括 Bagging,Forest of Randomized Trees 等。(2)在提升方法中，预测器是按 照一定顺序 (Sequentially) 建立的，后续增加的预测器试图减少已经建立的组合预测器的 预测偏差 (Bias), 其目的是把若干弱预测器组合成一个强大的预测器。主要的提升方法





Boocted Decslon Tee oonclan

一 n\_estimators-1

一 n\_estimators=300 … trainng samples

t

**图17-12** **经过AdaBoost 提升的决策树回归** **(Decision Tree Regression**)

包括AdaBoost,Gradient Tree Boosting 。AdaBoost已经在第5章“数据的深度分析”中 进行了介绍，其他方法的更为详细的介绍，请参考相关资料。①

**17.5.4 离线的模型训练** **(Offline Model Training)**

设计好机器学习模型后，首先经过训练、评估，在其性能指标达到一定水平的基础 上，我们才会把机器学习模型用于实际的股票价格预测。机器学习模型的训练是一个非常 耗时的过程，特别是深度学习模型，需要大量的数据对其进行训练，才能使其学习到数据 中体现的非线性关系。

在本章中，基于机器学习的交易策略，对机器学习模型的训练是在交易决策过程中进 行的。由于训练特别耗费时间， 一方面，我们不宜用大量的历史数据对模型进行训练，以 致模型的准确度受到影响；另一方面，这些交易策略不适合交易频率较高的应用场合，因 为训练本身就已经耗费了大量时间，来不及做出及时的预测。

我们可以采用离线的方式对模型进行训练，然后把模型保存起来。在交易策略执行 时，无须再对模型进行训练，而是直接装载和使用模型，根据新到来的数据，进行股票价 格预测和买卖信号生成。这样，完成交易决策的时间就大大缩短。

在这种模式下，需要持续地监控模型预测的准确度，当模型的预测准确度已经明显下 降，换句话说，模型已经不能适应新的市场状况时，我们需要用新的数据(可以和已有的 数据一起)训练一个新的模型，这个训练过程也是以离线的方式进行的，比如利用下午市 场关闭以后第二天开盘之前的时间，对模型进行训练和更新。开盘以后，交易策略将装载 使用新的训练好的模型。



①<http://scikit-learn.org/stable/modules/ensemble.html.>



**17.5.5 使用大数据处理引擎处理大数据** **(Using of Big Data Engine)**

本章给出的交易策略处理的数据量都比较小。在实际应用中，为了提高模型的预测准 确度，需要使用不同类型的数据，加大数据量来训练模型。数据规模一旦增长，一台机器 就很难胜任处理的任务。在存储方面，需要依赖大数据平台进行数据的存储，在模型训练 方面，则需要分布式算法进行模型的训练，以加快训练的速度。对于深度学习算法的训 练，为了进一步提高训练速度，一般还需要依赖于GPU (图形处理器)对训练进行加速。

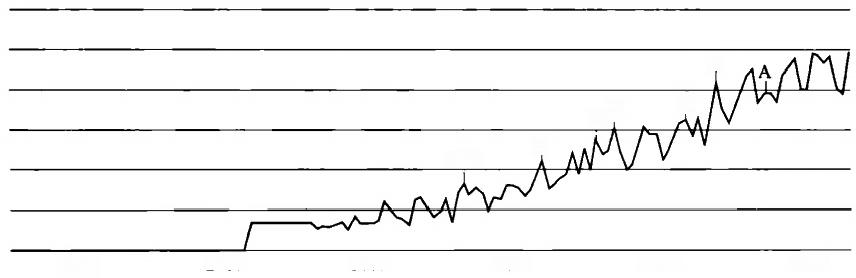
面向大数据处理的平台和工具包括 Hadoop 平台、Spark 平台、MongoDB 文档数据库 等。大量的数据可以保存在这些平台上，比如我们可以把价格数据放在 HDFS 上，把新 闻、博客等数据放在MongoDB 数据库上。

这些平台和工具，都提供了各种编程语言的接口 (Application Programming Inter- face,API), 包括Python 语言的接口，方便用户用Python 语言编写程序.调用这些平台 的大数据分析处理能力。(1) Pydoop 是 Hadoop 平台的Python 编程接口函数库，用户可 以使用Python 语言直接编写 MapReduce 应用程序。Pydoop 的功能包括 HDFS 分布式文 件系统编程接口、MapReduce 计算模型编程接口、序列化和反序列化 (Avro(de)serial- ization) 编程接口等。(2) PySpark 是 Spark 平台的Python 编程接口函数库。用户编写 Python 程序，通过PySpark 提供的函数，调用Spark 平台的数据处理功能，完成大数据 的分析处理。(3) PyMongo 是对MongoDB 数据库进行编程的Python 函数库，方便用户 编写 Python 程序，存取和处理MongoDB 数据库里的数据。

**17.5.6 在交易模型中集成情感分析结果** **(Integrating Sentiment Analysis into**

**Trading Model)**

根据Google Trends(Google 搜索引擎的搜索关键字及其趋势),自2007年以来， Sentiment Analysis (即情感分析)获得人们的持续关注(见图17-13)。情感是人们表达 出来的对于某个事物的态度。



B ED C

F

G

2013 2015

2007 2009 2011

2005

**图17-13** **Google** **Trends上2007—2015年人们对情感分析的关注度的变化**

情感分析也可以应用到量化交易中。在量化交易中，情感分析可以用来给出买入或者 卖出的方向性信号。传统的做法是，当有关某公司的正面消息发布，就买入该公司的股 票，当有关某公司的负面消息发布，就卖出该公司的股票。这样做的假设是，当有关某公

司的正面消息发布，它将吸引投资者买入股票，从而推高股票价格；当有关某公司的负面 消息发布，它将引起投资者的恐慌心理，导致他们倾向于抛出该公司股票，从而拉低股票 价 格 。

对于普通投资者来讲，只要他认真阅读一篇文章，就能理解文章所传达的态度。但 是，如果投资组合里包含了10只、100只，甚至上千只股票呢?追踪关于这些公司的新 闻、博客等信息，并及时进行判读，了解每篇文章所传递的态度，是相当困难的。正常的 成年人阅读文章的速度大概是每分钟300个单词，读一篇600个单词的文章大概需要2分 钟。在2分钟之内，新闻网站、社交媒体网站上产生的关于某个投资组合里10个公司的 文章可能达到上百篇，谈论这些公司的博客则可能达到10000篇。我们必须寻求计算机 的帮助，才能及时对这些新闻和博客进行情感分析。在人类投资者完成一篇文章的判读的 时间里，计算机已经完成了上百万篇文章的判读，确定每篇文章所传递的态度或者情感， 并且给出一个分数，分析的粒度甚至可以做到段落级别、语句级别。

在量化交易领域，很多投资基金把新闻、博客数据和其他数据源(价格数据、交易量 数据、基本面数据等)结合起来，把情感分析的结果和其他数据分析的结果结合起来，进 而做出交易决策，这已经形成了一个显著的趋势。Derwent Capital Markets①是伦敦的一 家对冲基金，公司的 CE() 是 Paul Hawtin。他带领团队研发了一款Twitter 监控程序，每 天处理3.4亿条Tweet, 评估公众的集体情绪 (Collective Mood), 据此进行交易。这个 程序计算出一个全球情绪评分 (Global Sentiment Score), 取值范围为1～50,根据人们在 社交媒体的在线对话中表现出来的悲观和乐观情绪计算出来。Derwent Capital Markets 在 此基础上，买卖上百万美元的股票。当人们高兴时， 一般做多(买进),当人们的焦虑开 始上扬时， 一般做空(卖出)。2012年第一季度，该交易模型获得了超过7%的收益。作 为先行者，Derwent Capital Markets的业绩令人瞩目。值得注意的是，当越来越多的投资 者掌握和使用类似的技术，这样的交易模型还能否有利可图?

有些创业公司则把情感分析包装成数据产品 (Data Product) 销售给投资者。至于如 何在投资模型里使用这些情感分析的结果，仁者见仁、智者见智，完全取决于各个投资者 自己的见解。比如 Accern 公司②,它实时监控2000万个互联网上的网站(包括新闻 (News) 、 博客 (Blogs) 、 社交媒体 (Social Media)、上市公司年报 (SEC Filings) 等), 利用人工智能技术，对来自这些网站的信息进行分析(包括情感分析),根据投资者的订 阅要求，把关于某些公司的关键信息以及分析结果发布给投资者，帮助投资者(在投资模 型里)据此进行投资决策。

进行情感分析最简单的办法是使用一个情感词典，标定文章中的正面情感词 (Posi- tive Word)、负面情感词 (Negative Word), 正面情感词的得分是+1.0,负面情感词的得 分是-1.0,然后累加正面情感得分和负面情感得分，两者之比就是最终的情感得分。为 了方便比较，一般把情感得分 (Sentiment Score) 规范化到[ - 1 . 0,+1 . 0]范围之内。 关于情感分析的更为详细的介绍，请参考第7章“文本分析”。



① <https://www.washingtonpost.com/business/economy/big-data-from-social-media-elsewhere-online-take-trend>- watching-to-new-level/2012/06/06/gJQArWWpJV\_story.html?utm\_term=.6cb95cbe27b8.

②<https://www.accern.com/.>



**17.6** **思考题**

**(1)量化交易概念。**

**(2)量化交易系统的四大模块。**

**(3)交易策略的研发、测试、上线过程。**

**(4)量化交易系统的评价指标。**

**(5)Zipline 库和** **Quantopian 平台介绍。**

**(6)基于规则的交易策略。**

**(7)基于机器学习的交易策略。**

**(8)交易模型可以使用的数据及其特点。**

**(9)交易模型的特征选择。**

**(10)使用模型组合提高交易模型预测的准确率。**