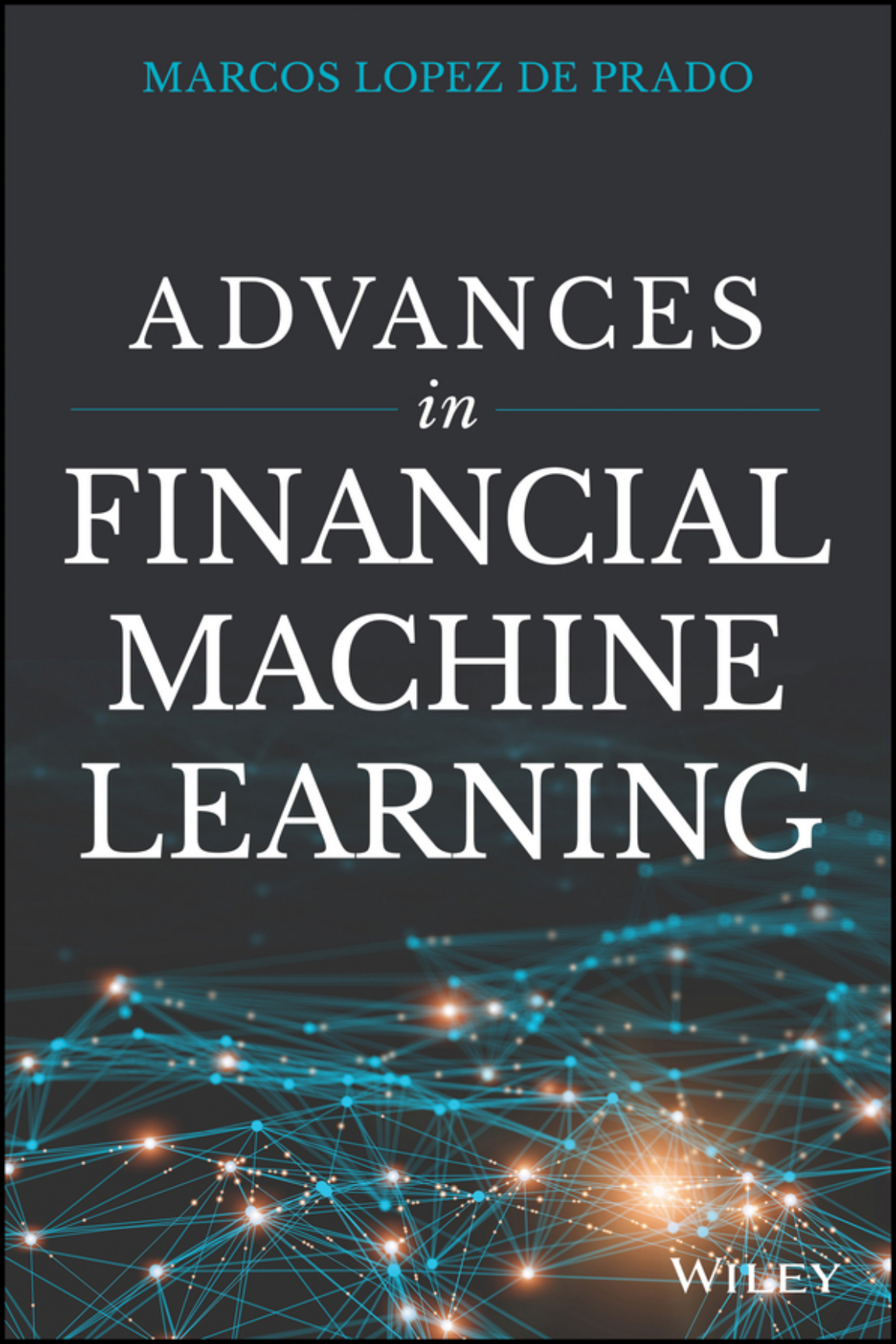
MARCOS LOPEZ DE PRADO

ADVANCES *in* FINANCIAL MACHINE LEARNING



WILEY

Praise for *Advances in Financial Machine Learning*

In his new book *Advances in Financial Machine Learning*, noted financial scholar Marcos López de Prado strikes a well-aimed karate chop at the naive and often statistically overfit techniques that are so prevalent in the financial world today. He points out that not only are business-as-usual approaches largely impotent in today's high-tech finance, but in many cases they are actually prone to lose money. But López de Prado does more than just expose the mathematical and statistical sins of the finance world. Instead, he offers a technically sound roadmap for finance professionals to join the wave of machine learning. What is particularly refreshing is the author's empirical approach—his focus is on real-world data analysis, not on purely theoretical methods that may look pretty on paper but which, in many cases, are largely ineffective in practice. The book is geared to finance professionals who are already familiar with statistical data analysis techniques, but it is well worth the effort for those who want to do real state-of-the-art work in the field."

Dr. David H. Bailey, former Complex Systems Lead,
Lawrence Berkeley National Laboratory. Co-discoverer of the
BBP spigot algorithm

"Finance has evolved from a compendium of heuristics based on historical financial statements to a highly sophisticated scientific discipline relying on computer farms to analyze massive data streams in real time. The recent highly impressive advances in machine learning (ML) are fraught with both promise and peril when applied to modern finance. While finance offers up the nonlinearities and large data sets upon which ML thrives, it also offers up noisy data and the human element which presently lie beyond the scope of standard ML techniques. To err is human, but if you really want to f**k things up, use a computer. Against this background, Dr. López de Prado has written the first comprehensive book describing the application of modern ML to financial modeling. The book blends the latest technological developments in ML with critical life lessons learned from the author's decades of financial experience in leading academic and industrial institutions. I highly recommend this exciting book to both prospective students of financial ML and the professors and supervisors who teach and guide them."

Prof. Peter Carr, Chair of the Finance and Risk Engineering
Department, NYU Tandon School of Engineering

"Marcos is a visionary who works tirelessly to advance the finance field. His writing is comprehensive and masterfully connects the theory to the application. It is not often you find a book that can cross that divide. This book is an essential read for both practitioners and technologists working on solutions for the investment community."

Landon Downs, President and Cofounder, 1QBit

"Academics who want to understand modern investment management need to read this book. In it, Marcos López de Prado explains how portfolio managers use machine learning to derive, test, and employ trading strategies. He does this from a very unusual combination of an academic perspective and extensive experience in industry, allowing him to both explain in detail what happens in industry and to explain

how it works. I suspect that some readers will find parts of the book that they do not understand or that they disagree with, but everyone interested in understanding the application of machine learning to finance will benefit from reading this book."

Prof. David Easley, Cornell University. Chair of the NASDAQ-OMX Economic Advisory Board

"For many decades, finance has relied on overly simplistic statistical techniques to identify patterns in data. Machine learning promises to change that by allowing researchers to use modern nonlinear and highly dimensional techniques, similar to those used in scientific fields like DNA analysis and astrophysics. At the same time, applying those machine learning algorithms to model financial problems would be dangerous. Financial problems require very distinct machine learning solutions. Dr. López de Prado's book is the first one to characterize what makes standard machine learning tools fail when applied to the field of finance, and the first one to provide practical solutions to unique challenges faced by asset managers. Everyone who wants to understand the future of finance should read this book."

Prof. Frank Fabozzi, EDHEC Business School. Editor of *The Journal of Portfolio Management*

"This is a welcome departure from the knowledge hoarding that plagues quantitative finance. López de Prado defines for all readers the next era of finance: industrial scale scientific research powered by machines."

John Fawcett, Founder and CEO, Quantopian

"Marcos has assembled in one place an invaluable set of lessons and techniques for practitioners seeking to deploy machine learning techniques in finance. If machine learning is a new and potentially powerful weapon in the arsenal of quantitative finance, Marcos's insightful book is laden with useful advice to help keep a curious practitioner from going down any number of blind alleys, or shooting oneself in the foot."

Ross Garon, Head of Cubist Systematic Strategies. Managing Director, Point72 Asset Management

"The first wave of quantitative innovation in finance was led by Markowitz optimization. Machine Learning is the second wave, and it will touch every aspect of finance. López de Prado's *Advances in Financial Machine Learning* is essential for readers who want to be ahead of the technology rather than being replaced by it."

Prof. Campbell Harvey, Duke University. Former President of the American Finance Association

"How does one make sense of todays' financial markets in which complex algorithms route orders, financial data is voluminous, and trading speeds are measured in nanoseconds? In this important book, Marcos López de Prado sets out a new paradigm for investment management built on machine learning. Far from being a "black box" technique, this book clearly explains the tools and process of financial

machine learning. For academics and practitioners alike, this book fills an important gap in our understanding of investment management in the machine age."

Prof. Maureen O'Hara, Cornell University. Former President of the American Finance Association

"Marcos López de Prado has produced an extremely timely and important book on machine learning. The author's academic and professional first-rate credentials shine through the pages of this book—indeed, I could think of few, if any, authors better suited to explaining both the theoretical and the practical aspects of this new and (for most) unfamiliar subject. Both novices and experienced professionals will find insightful ideas, and will understand how the subject can be applied in novel and useful ways. The Python code will give the novice readers a running start and will allow them to gain quickly a hands-on appreciation of the subject. Destined to become a classic in this rapidly burgeoning field."

Prof. Riccardo Rebonato, EDHEC Business School. Former Global Head of Rates and FX Analytics at PIMCO

"A tour de force on practical aspects of machine learning in finance, brimming with ideas on how to employ cutting-edge techniques, such as fractional differentiation and quantum computers, to gain insight and competitive advantage. A useful volume for finance and machine learning practitioners alike."

Dr. Collin P. Williams, Head of Research, D-Wave Systems

Advances in Financial Machine Learning

Advances in Financial Machine Learning

MARCOS LÓPEZ DE PRADO

WILEY

Cover image: © Erikona/Getty Images
Cover design: Wiley

Published by John Wiley & Sons, Inc., Hoboken, New Jersey.

Published simultaneously in Canada.

The views expressed in this book are the author's and do not necessarily reflect those of the organizations he is affiliated with.

ISBN 978-1-119-48208-6 (Hardcover)
ISBN 978-1-119-48211-6 (ePDF)
ISBN 978-1-119-48210-9 (ePub)

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

Dedicated to the memory of my coauthor and friend,
Professor Jonathan M. Borwein, FRSC, FAAAS,
FBAS, FAustMS, FAA, FAMS, FRSNSW
(1951–2016)

There are very few things which we know, which are not capable of being reduced to a mathematical reasoning. And when they cannot, it's a sign our knowledge of them is very small and confused. Where a mathematical reasoning can be had, it's as great a folly to make use of any other, as to grope for a thing in the dark, when you have a candle standing by you.

—*Of the Laws of Chance*, Preface (1692)
John Arbuthnot (1667–1735)

Contents

About the Author

Marcos López de Prado manages several multibillion-dollar funds for institutional investors using machine learning algorithms. Over the past 20 years, his work has combined advanced mathematics with supercomputing technologies to deliver billions of dollars in net profits for investors and firms. A proponent of research by collaboration, Marcos has published with more than 30 leading academics, resulting in some of the most-read papers in finance.

Since 2010, Marcos has also been a Research Fellow at Lawrence Berkeley National Laboratory (U.S. Department of Energy's Office of Science), where he conducts research focused on the mathematics of large-scale financial problems and high-performance computing at the Computational Research department. For the past seven years he has lectured at Cornell University, where he currently teaches a graduate course in financial big data and machine learning in the Operations Research department.

Marcos is the recipient of the 1999 National Award for Academic Excellence, which the government of Spain bestows upon the best graduate student nationally. He earned a PhD in financial economics (2003) and a second PhD in mathematical finance (2011) from Universidad Complutense de Madrid. Between his two doctorates, Marcos was a postdoctoral research fellow of RCC at Harvard University for three years, during which he published more than a dozen articles in JCR-indexed scientific journals. Marcos has an Erdős #2 and an Einstein #4, according to the American Mathematical Society.

Preamble

CHAPTER 1

Financial Machine Learning as a Distinct Subject

1.1 MOTIVATION

Machine learning (ML) is changing virtually every aspect of our lives. Today ML algorithms accomplish tasks that until recently only expert humans could perform. As it relates to finance, this is the most exciting time to adopt a disruptive technology that will transform how everyone invests for generations. This book explains scientifically sound ML tools that have worked for me over the course of two decades, and have helped me to manage large pools of funds for some of the most demanding institutional investors.

Books about investments largely fall in one of two categories. On one hand we find books written by authors who have not practiced what they teach. They contain extremely elegant mathematics that describes a world that does not exist. Just because a theorem is true in a logical sense does not mean it is true in a physical sense. On the other hand we find books written by authors who offer explanations absent of any rigorous academic theory. They misuse mathematical tools to describe actual observations. Their models are overfit and fail when implemented. Academic investigation and publication are divorced from practical application to financial markets, and many applications in the trading/investment world are not grounded in proper science.

A first motivation for writing this book is to cross the proverbial divide that separates academia and the industry. I have been on both sides of the rift, and I understand how difficult it is to cross it and how easy it is to get entrenched on one side. Virtue is in the balance. This book will not advocate a theory merely because of its mathematical beauty, and will not propose a solution just because it appears to work. My goal is to transmit the kind of knowledge that only comes from experience, formalized in a rigorous manner.

A second motivation is inspired by the desire that finance serves a purpose. Over the years some of my articles, published in academic journals and newspapers, have expressed my displeasure with the current role that finance plays in our society. Investors are lured to gamble their wealth on wild hunches originated by charlatans and encouraged by mass media. One day in the near future, ML will dominate finance, science will curtail guessing, and investing will not mean gambling. I would like the reader to play a part in that revolution.

A third motivation is that many investors fail to grasp the complexity of ML applications to investments. This seems to be particularly true for discretionary firms moving into the "quantamental" space. I am afraid their high expectations will not be met, not because ML failed, but because they used ML incorrectly. Over the coming years, many firms will invest with off-the-shelf ML algorithms, directly imported from academia or Silicon Valley, and my forecast is that they will lose money (to better ML solutions). Beating the wisdom of the crowds is harder than recognizing faces or driving cars. With this book my hope is that you will learn how to solve some of the challenges that make finance a particularly difficult playground for ML, like backtest overfitting. Financial ML is a subject in its own right, related to but separate from standard ML, and this book unravels it for you.

1.2 THE MAIN REASON FINANCIAL MACHINE LEARNING PROJECTS USUALLY FAIL

The rate of failure in quantitative finance is high, particularly so in financial ML. The few who succeed amass a large amount of assets and deliver consistently exceptional performance to their investors. However, that is a rare outcome, for reasons explained in this book. Over the past two decades, I have seen many faces come and go, firms started and shut down. In my experience, there is one critical mistake that underlies all those failures.

1.2.1 The Sisyphus Paradigm

Discretionary portfolio managers (PMs) make investment decisions that do not follow a particular theory or rationale (if there were one, they would be systematic PMs). They consume raw news and analyses, but mostly rely on their judgment or intuition. They may rationalize those decisions based on some story, but there is always a story for every decision. Because nobody fully understands the logic behind their bets, investment firms ask them to work independently from one another, in silos, to ensure diversification. If you have ever attended a meeting of discretionary PMs, you probably noticed how long and aimless they can be. Each attendee seems obsessed about one particular piece of anecdotal information, and giant argumentative leaps are made without fact-based, empirical evidence. This does not mean that discretionary PMs cannot be successful. On the contrary, a few of them are. The point is, they cannot naturally work as a team. Bring 50 discretionary PMs together, and they

will influence one another until eventually you are paying 50 salaries for the work of one. Thus it makes sense for them to work in silos so they interact as little as possible.

Wherever I have seen that formula applied to quantitative or ML projects, it has led to disaster. The boardroom's mentality is, let us do with quants what has worked with discretionary PMs. Let us hire 50 PhDs and demand that each of them produce an investment strategy within six months. This approach always backfires, because each PhD will frantically search for investment opportunities and eventually settle for (1) a false positive that looks great in an overfit backtest or (2) standard factor investing, which is an overcrowded strategy with a low Sharpe ratio, but at least has academic support. Both outcomes will disappoint the investment board, and the project will be cancelled. Even if 5 of those PhDs identified a true discovery, the profits would not suffice to cover for the expenses of 50, so those 5 will relocate somewhere else, searching for a proper reward.

1.2.2 The Meta-Strategy Paradigm

If you have been asked to develop ML strategies on your own, the odds are stacked against you. It takes almost as much effort to produce one true investment strategy as to produce a hundred, and the complexities are overwhelming: data curation and processing, HPC infrastructure, software development, feature analysis, execution simulators, backtesting, etc. Even if the firm provides you with shared services in those areas, you are like a worker at a BMW factory who has been asked to build an entire car by using all the workshops around you. One week you need to be a master welder, another week an electrician, another week a mechanical engineer, another week a painter . . . You will try, fail, and circle back to welding. How does that make sense?

Every successful quantitative firm I am aware of applies the meta-strategy paradigm (López de Prado [2014]). Accordingly, this book was written as a research manual for teams, not for individuals. Through its chapters you will learn how to set up a research factory, as well as the various stations of the assembly line. The role of each quant is to specialize in a particular task, to become the best there is at it, while having a holistic view of the entire process. This book outlines the factory plan, where teamwork yields discoveries at a predictable rate, with no reliance on lucky strikes. This is how Berkeley Lab and other U.S. National Laboratories routinely make scientific discoveries, such as adding 16 elements to the periodic table, or laying out the groundwork for MRIs and PET scans.[1] No particular individual is responsible for these discoveries, as they are the outcome of team efforts where everyone contributes. Of course, setting up these financial laboratories takes time, and requires people who know what they are doing and have done it before. But what do you think has a higher chance of success, this proven paradigm of organized collaboration or the Sisyphean alternative of having every single quant rolling their immense boulder up the mountain?

[1] Berkeley Lab, http://www.lbl.gov/about.

1.3 BOOK STRUCTURE

This book disentangles a web of interconnected topics and presents them in an ordered fashion. Each chapter assumes that you have read the previous ones. Part 1 will help you structure your financial data in a way that is amenable to ML algorithms. Part 2 discusses how to do research with ML algorithms on that data. Here the emphasis is on doing research and making an actual discovery through a scientific process, as opposed to searching aimlessly until some serendipitous (likely false) result pops up. Part 3 explains how to backtest your discovery and evaluate the probability that it is false.

These three parts give an overview of the entire process, from data analysis to model research to discovery evaluation. With that knowledge, Part 4 goes back to the data and explains innovative ways to extract informative features. Finally, much of this work requires a lot of computational power, so Part 5 wraps up the book with some useful HPC recipes.

1.3.1 Structure by Production Chain

Mining gold or silver was a relatively straightforward endeavor during the 16th and 17th centuries. In less than a hundred years, the Spanish treasure fleet quadrupled the amount of precious metals in circulation throughout Europe. Those times are long gone, and today prospectors must deploy complex industrial methods to extract microscopic bullion particles out of tons of earth. That does not mean that gold production is at historical lows. On the contrary, nowadays miners extract 2,500 metric tons of microscopic gold every year, compared to the average annual 1.54 metric tons taken by the Spanish conquistadors throughout the entire 16th century![2] Visible gold is an infinitesimal portion of the overall amount of gold on Earth. *El Dorado* was always there . . . if only Pizarro could have exchanged the sword for a microscope.

The discovery of investment strategies has undergone a similar evolution. If a decade ago it was relatively common for an individual to discover macroscopic alpha (i.e., using simple mathematical tools like econometrics), currently the chances of that happening are quickly converging to zero. Individuals searching nowadays for macroscopic alpha, regardless of their experience or knowledge, are fighting overwhelming odds. The only true alpha left is microscopic, and finding it requires capital-intensive industrial methods. Just like with gold, microscopic alpha does not mean smaller overall profits. Microscopic alpha today is much more abundant than macroscopic alpha has ever been in history. There is a lot of money to be made, but you will need to use heavy ML tools.

Let us review some of the stations involved in the chain of production within a modern asset manager.

---

[2] http://www.numbersleuth.org/worlds-gold/.

1.3.1.1 Data Curators

This is the station responsible for collecting, cleaning, indexing, storing, adjusting, and delivering all data to the production chain. The values could be tabulated or hierarchical, aligned or misaligned, historical or real-time feeds, etc. Team members are experts in market microstructure and data protocols such as FIX. They must develop the data handlers needed to understand the context in which that data arises. For example, was a quote cancelled and replaced at a different level, or cancelled without replacement? Each asset class has its own nuances. For instance, bonds are routinely exchanged or recalled; stocks are subjected to splits, reverse-splits, voting rights, etc.; futures and options must be rolled; currencies are not traded in a centralized order book. The degree of specialization involved in this station is beyond the scope of this book, and Chapter 1 will discuss only a few aspects of data curation.

1.3.1.2 Feature Analysts

This is the station responsible for transforming raw data into informative signals. These informative signals have some predictive power over financial variables. Team members are experts in information theory, signal extraction and processing, visualization, labeling, weighting, classifiers, and feature importance techniques. For example, feature analysts may discover that the probability of a sell-off is particularly high when: (1) quoted offers are cancelled-replaced with market sell orders, and (2) quoted buy orders are cancelled-replaced with limit buy orders deeper in the book. Such a finding is not an investment strategy on its own, and can be used in alternative ways: execution, monitoring of liquidity risk, market making, position taking, etc. A common error is to believe that feature analysts develop strategies. Instead, feature analysts collect and catalogue libraries of findings that can be useful to a multiplicity of stations. Chapters 2–9 and 17–19 are dedicated to this all-important station.

1.3.1.3 Strategists

In this station, informative features are transformed into actual investment algorithms. A strategist will parse through the libraries of features looking for ideas to develop an investment strategy. These features were discovered by different analysts studying a wide range of instruments and asset classes. The goal of the strategist is to make sense of all these observations and to formulate a general theory that explains them. Therefore, the strategy is merely the experiment designed to test the validity of this theory. Team members are data scientists with a deep knowledge of financial markets and the economy. Remember, the theory needs to explain a large collection of important features. In particular, a theory must identify the economic mechanism that causes an agent to lose money to us. Is it a behavioral bias? Asymmetric information? Regulatory constraints? Features may be discovered by a black box, but the strategy is developed in a white box. Gluing together a number of catalogued features does not constitute a theory. Once a strategy is finalized, the strategists will prepare code that utilizes the full algorithm and submit that prototype to the backtesting team described below. Chapters 10 and 16 are dedicated to this station, with the understanding that it would be unreasonable for a book to reveal specific investment strategies.

1.3.1.4 Backtesters

This station assesses the profitability of an investment strategy under various scenarios. One of the scenarios of interest is how the strategy would perform if history repeated itself. However, the historical path is merely one of the possible outcomes of a stochastic process, and not necessarily the most likely going forward. Alternative scenarios must be evaluated, consistent with the knowledge of the weaknesses and strengths of a proposed strategy. Team members are data scientists with a deep understanding of empirical and experimental techniques. A good backtester incorporates in his analysis meta-information regarding how the strategy came about. In particular, his analysis must evaluate the probability of backtest overfitting by taking into account the number of trials it took to distill the strategy. The results of this evaluation will not be reused by other stations, for reasons that will become apparent in Chapter 11. Instead, backtest results are communicated to management and not shared with anyone else. Chapters 11–16 discuss the analyses carried out by this station.

1.3.1.5 Deployment Team

The deployment team is tasked with integrating the strategy code into the production line. Some components may be reused by multiple strategies, especially when they share common features. Team members are algorithm specialists and hardcore mathematical programmers. Part of their job is to ensure that the deployed solution is logically identical to the prototype they received. It is also the deployment team's responsibility to optimize the implementation sufficiently, such that production latency is minimized. As production calculations often are time sensitive, this team will rely heavily on process schedulers, automation servers (Jenkins), vectorization, multithreading, multiprocessing, graphics processing unit (GPU-NVIDIA), distributed computing (Hadoop), high-performance computing (Slurm), and parallel computing techniques in general. Chapters 20–22 touch on various aspects interesting to this station, as they relate to financial ML.

1.3.1.6 Portfolio Oversight

Once a strategy is deployed, it follows a *cursus honorum*, which entails the following stages or lifecycle:

1. **Embargo:** Initially, the strategy is run on data observed after the end date of the backtest. Such a period may have been reserved by the backtesters, or it may be the result of implementation delays. If embargoed performance is consistent with backtest results, the strategy is promoted to the next stage.
2. **Paper trading:** At this point, the strategy is run on a live, real-time feed. In this way, performance will account for data parsing latencies, calculation latencies, execution delays, and other time lapses between observation and positioning. Paper trading will take place for as long as it is needed to gather enough evidence that the strategy performs as expected.
3. **Graduation:** At this stage, the strategy manages a real position, whether in isolation or as part of an ensemble. Performance is evaluated precisely, including attributed risk, returns, and costs.

4. **Re-allocation:** Based on the production performance, the allocation to graduated strategies is re-assessed frequently and automatically in the context of a diversified portfolio. In general, a strategy's allocation follows a concave function. The initial allocation (at graduation) is small. As time passes, and the strategy performs as expected, the allocation is increased. Over time, performance decays, and allocations become gradually smaller.
5. **Decommission**: Eventually, all strategies are discontinued. This happens when they perform below expectations for a sufficiently extended period of time to conclude that the supporting theory is no longer backed by empirical evidence.

In general, it is preferable to release new variations of a strategy and run them in parallel with old versions. Each version will go through the above lifecycle, and old strategies will receive smaller allocations as a matter of diversification, while taking into account the degree of confidence derived from their longer track record.

1.3.2 Structure by Strategy Component

Many investment managers believe that the secret to riches is to implement an extremely complex ML algorithm. They are setting themselves up for a disappointment. If it was as easy as coding a state-of-the art classifier, most people in Silicon Valley would be billionaires. A successful investment strategy is the result of multiple factors. Table 1.1 summarizes what chapters will help you address each of the challenges involved in developing a successful investment strategy.

Throughout the book, you will find many references to journal articles I have published over the years. Rather than repeating myself, I will often refer you to one of them, where you will find a detailed analysis of the subject at hand. All of my cited papers can be downloaded for free, in pre-print format, from my website: www.QuantResearch.org.

1.3.2.1 Data

- Problem: Garbage in, garbage out.
- Solution: Work with unique, hard-to-manipulate data. If you are the only user of this data, whatever its value, it is all for you.
- How:
  - Chapter 2: Structure your data correctly.
  - Chapter 3: Produce informative labels.
  - Chapters 4 and 5: Model non-IID series properly.
  - Chapters 17–19: Find predictive features.

1.3.2.2 Software

- Problem: A specialized task requires customized tools.
- Solution: Develop your own classes. Using popular libraries means more competitors tapping the same well.

TABLE 1.1 Overview of the Challenges Addressed by Every Chapter

Part	Chapter	Fin. data	Software	Hardware	Math	Meta-Strat	Overfitting
1	2	X	X				
1	3	X	X				
1	4	X	X				
1	5	X	X		X		
2	6		X				
2	7		X			X	X
2	8		X			X	
2	9		X			X	
3	10		X			X	
3	11		X		X		X
3	12		X		X		X
3	13		X		X		X
3	14		X		X		X
3	15		X		X		X
3	16		X		X	X	X
4	17	X	X		X		
4	18	X	X		X		
4	19	X	X				
5	20		X	X	X		
5	21		X	X	X		
5	22		X	X	X		

- How:
  - Chapters 2–22: Throughout the book, for each chapter, we develop our own functions. For your particular problems, you will have to do the same, following the examples in the book.

1.3.2.3 Hardware

- Problem: ML involves some of the most computationally intensive tasks in all of mathematics.
- Solution: Become an HPC expert. If possible, partner with a National Laboratory to build a supercomputer.
- How:
  - Chapters 20 and 22: Learn how to think in terms of multiprocessing architectures. Whenever you code a library, structure it in such a way that functions can be called in parallel. You will find plenty of examples in the book.
  - Chapter 21: Develop algorithms for quantum computers.

1.3.2.4 Math

- Problem: Mathematical proofs can take years, decades, and centuries. No investor will wait that long.

- Solution: Use experimental math. Solve hard, intractable problems, not by proof but by experiment. For example, Bailey, Borwein, and Plouffe [1997] found a spigot algorithm for π (pi) without proof, against the prior perception that such mathematical finding would not be possible.
- How:
  - Chapter 5: Familiarize yourself with memory-preserving data transformations.
  - Chapters 11–15: There are experimental methods to assess the value of your strategy, with greater reliability than a historical simulation.
  - Chapter 16: An algorithm that is optimal in-sample can perform poorly out-of-sample. There is no mathematical proof for investment success. Rely on experimental methods to lead your research.
  - Chapters 17 and 18: Apply methods to detect structural breaks, and quantify the amount of information carried by financial series.
  - Chapter 20: Learn queuing methods for distributed computing so that you can break apart complex tasks and speed up calculations.
  - Chapter 21: Become familiar with discrete methods, used among others by quantum computers, to solve intractable problems.

1.3.2.5 Meta-Strategies

- Problem: Amateurs develop individual strategies, believing that there is such a thing as a magical formula for riches. In contrast, professionals develop methods to mass-produce strategies. The money is not in making a car, it is in making a car factory.
- Solution: Think like a business. Your goal is to run a research lab like a factory, where true discoveries are not born out of inspiration, but out of methodic hard work. That was the philosophy of physicist Ernest Lawrence, the founder of the first U.S. National Laboratory.
- How:
  - Chapters 7–9: Build a research process that identifies features relevant across asset classes, while dealing with multi-collinearity of financial features.
  - Chapter 10: Combine multiple predictions into a single bet.
  - Chapter 16: Allocate funds to strategies using a robust method that performs well out-of-sample.

1.3.2.6 Overfitting

- Problem: Standard cross-validation methods fail in finance. Most discoveries in finance are false, due to multiple testing and selection bias.
- Solution:
  - Whatever you do, always ask yourself in what way you may be overfitting. Be skeptical about your own work, and constantly challenge yourself to prove that you are adding value.

  - Overfitting is unethical. It leads to promising outcomes that cannot be delivered. When done knowingly, overfitting is outright scientific fraud. The fact that many academics do it does not make it right: They are not risking anyone's wealth, not even theirs.
  - It is also a waste of your time, resources, and opportunities. Besides, the industry only pays for out-of-sample returns. You will only succeed *after* you have created substantial wealth for your investors.
- How:
  - Chapters 11–15: There are three backtesting paradigms, of which historical simulation is only one. Each backtest is always overfit to some extent, and it is critical to learn to quantify by how much.
  - Chapter 16: Learn robust techniques for asset allocation that do not overfit in-sample signals at the expense of out-of-sample performance.

1.3.3 Structure by Common Pitfall

Despite its many advantages, ML is no panacea. The flexibility and power of ML techniques have a dark side. When misused, ML algorithms will confuse statistical flukes with patterns. This fact, combined with the low signal-to-noise ratio that characterizes finance, all but ensures that careless users will produce false discoveries at an ever-greater speed. This book exposes some of the most pervasive errors made by ML experts when they apply their techniques on financial datasets. Some of these pitfalls are listed in Table 1.2, with solutions that are explained in the indicated chapters.

1.4 TARGET AUDIENCE

This book presents advanced ML methods specifically designed to address the challenges posed by financial datasets. By "advanced" I do not mean extremely difficult to grasp, or explaining the latest reincarnation of deep, recurrent, or convolutional neural networks. Instead, the book answers questions that senior researchers, who have experience applying ML algorithms to financial problems, will recognize as critical. If you are new to ML, and you do not have experience working with complex algorithms, this book may not be for you (yet). Unless you have confronted in practice the problems discussed in these chapters, you may have difficulty understanding the utility of solving them. Before reading this book, you may want to study several excellent introductory ML books published in recent years. I have listed a few of them in the references section.

The core audience of this book is investment professionals with a strong ML background. My goals are that you monetize what you learn in this book, help us modernize finance, and deliver actual value for investors.

This book also targets data scientists who have successfully implemented ML algorithms in a variety of fields outside finance. If you have worked at Google and have applied deep neural networks to face recognition, but things do not seem to

TABLE 1.2 Common Pitfalls in Financial ML

#	Category	Pitfall	Solution	Chapter
1	Epistemological	The Sisyphus paradigm	The meta-strategy paradigm	1
2	Epistemological	Research through backtesting	Feature importance analysis	8
3	Data processing	Chronological sampling	The volume clock	2
4	Data processing	Integer differentiation	Fractional differentiation	5
5	Classification	Fixed-time horizon labeling	The triple-barrier method	3
6	Classification	Learning side and size simultaneously	Meta-labeling	3
7	Classification	Weighting of non-IID samples	Uniqueness weighting; sequential bootstrapping	4
8	Evaluation	Cross-validation leakage	Purging and embargoing	7, 9
9	Evaluation	Walk-forward (historical) backtesting	Combinatorial purged cross-validation	11, 12
10	Evaluation	Backtest overfitting	Backtesting on synthetic data; the deflated Sharpe ratio	10–16

work so well when you run your algorithms on financial data, this book will help you. Sometimes you may not understand the financial rationale behind some structures (e.g., meta-labeling, the triple-barrier method, fracdiff), but bear with me: Once you have managed an investment portfolio long enough, the rules of the game will become clearer to you, along with the meaning of these chapters.

1.5 REQUISITES

Investment management is one of the most multi-disciplinary areas of research, and this book reflects that fact. Understanding the various sections requires a practical knowledge of ML, market microstructure, portfolio management, mathematical finance, statistics, econometrics, linear algebra, convex optimization, discrete math, signal processing, information theory, object-oriented programming, parallel processing, and supercomputing.

Python has become the *de facto* standard language for ML, and I have to assume that you are an experienced developer. You must be familiar with scikit-learn (sklearn), pandas, numpy, scipy, multiprocessing, matplotlib and a few other libraries.

Code snippets invoke functions from these libraries using their conventional prefix, pd for pandas, np for numpy, mpl for matplotlib, etc. There are numerous books on each of these libraries, and you cannot know enough about the specifics of each one. Throughout the book we will discuss some issues with their implementation, including unresolved bugs to keep in mind.

1.6 FAQs

How can ML algorithms be useful in finance?

Many financial operations require making decisions based on pre-defined rules, like option pricing, algorithmic execution, or risk monitoring. This is where the bulk of automation has taken place so far, transforming the financial markets into ultra-fast, hyper-connected networks for exchanging information. In performing these tasks, machines were asked to follow the rules as fast as possible. High-frequency trading is a prime example. See Easley, López de Prado, and O'Hara [2013] for a detailed treatment of the subject.

The algorithmization of finance is unstoppable. Between June 12, 1968, and December 31, 1968, the NYSE was closed every Wednesday, so that back office could catch up with paperwork. Can you imagine that? We live in a different world today, and in 10 years things will be even better. Because the next wave of automation does not involve following rules, but making judgment calls. As emotional beings, subject to fears, hopes, and agendas, humans are not particularly good at making fact-based decisions, particularly when those decisions involve conflicts of interest. In those situations, investors are better served when a machine makes the calls, based on facts learned from hard data. This not only applies to investment strategy development, but to virtually every area of financial advice: granting a loan, rating a bond, classifying a company, recruiting talent, predicting earnings, forecasting inflation, etc. Furthermore, machines will comply with the law, always, when programmed to do so. If a dubious decision is made, investors can go back to the logs and understand exactly what happened. It is much easier to improve an algorithmic investment process than one relying entirely on humans.

How can ML algorithms beat humans at investing?

Do you remember when people were certain that computers would never beat humans at chess? Or *Jeopardy!*? Poker? Go? Millions of years of evolution (a genetic algorithm) have fine-tuned our ape brains to survive in a hostile 3-dimensional world where the laws of nature are static. Now, when it comes to identifying subtle patterns in a high-dimensional world, where the rules of the game change every day, all that fine-tuning turns out to be detrimental. An ML algorithm can spot patterns in a 100-dimensional world as easily as in our familiar 3-dimensional one. And while we all laugh when we see an algorithm make a silly mistake, keep in mind, algorithms have been around only a fraction of our millions of years. Every day they get better at this, we do not. Humans are slow learners, which puts us at a disadvantage in a fast-changing world like finance.

Does that mean that there is no space left for human investors?

Not at all. No human is better at chess than a computer. And no computer is better at chess than a human supported by a computer. Discretionary PMs are at a disadvantage when betting against an ML algorithm, but it is possible that the best results are achieved by combining discretionary PMs with ML algorithms. This is what has come to be known as the "quantamental" way. Throughout the book you will find techniques that can be used by quantamental teams, that is, methods that allow you to combine human guesses (inspired by fundamental variables) with mathematical forecasts. In particular, Chapter 3 introduces a new technique called meta-labeling, which allows you to add an ML layer on top of a discretionary one.

How does financial ML differ from econometrics?

Econometrics is the application of classical statistical methods to economic and financial series. The essential tool of econometrics is multivariate linear regression, an 18th-century technology that was already mastered by Gauss before 1794 (Stigler [1981]). Standard econometric models do not learn. It is hard to believe that something as complex as 21st-century finance could be grasped by something as simple as inverting a covariance matrix.

Every empirical science must build theories based on observation. If the statistical toolbox used to model these observations is linear regression, the researcher will fail to recognize the complexity of the data, and the theories will be awfully simplistic, useless. I have no doubt in my mind, econometrics is a primary reason economics and finance have not experienced meaningful progress over the past 70 years (Calkin and López de Prado [2014a, 2014b]).

For centuries, medieval astronomers made observations and developed theories about celestial mechanics. These theories never considered non-circular orbits, because they were deemed unholy and beneath God's plan. The prediction errors were so gross, that ever more complex theories had to be devised to account for them. It was not until Kepler had the temerity to consider non-circular (elliptical) orbits that all of the sudden a much simpler general model was able to predict the position of the planets with astonishing accuracy. What if astronomers had never considered non-circular orbits? Well . . . what if economists finally started to consider non-linear functions? Where is our Kepler? Finance does not have a *Principia* because no Kepler means no Newton.

Financial ML methods do not replace theory. They guide it. An ML algorithm learns patterns in a high-dimensional space without being specifically directed. Once we understand what features are predictive of a phenomenon, we can build a theoretical explanation, which can be tested on an independent dataset. Students of economics and finance would do well enrolling in ML courses, rather than econometrics. Econometrics may be good enough to succeed in financial academia (for now), but succeeding in business requires ML.

What do you say to people who dismiss ML algorithms as black boxes?

If you are reading this book, chances are ML algorithms are white boxes to you. They are transparent, well-defined, crystal-clear, pattern-recognition functions. Most

people do not have your knowledge, and to them ML is like a magician's box: "Where did that rabbit come from? How are you tricking us, witch?" People mistrust what they do not understand. Their prejudices are rooted in ignorance, for which the Socratic remedy is simple: education. Besides, some of us enjoy using our brains, even though neuroscientists still have not figured out exactly how they work (a black box in itself).

From time to time you will encounter Luddites, who are beyond redemption. Ned Ludd was a weaver from Leicester, England, who in 1779 smashed two knitting frames in an outrage. With the advent of the industrial revolution, mobs infuriated by mechanization sabotaged and destroyed all machinery they could find. Textile workers ruined so much industrial equipment that Parliament had to pass laws making "machine breaking" a capital crime. Between 1811 and 1816, large parts of England were in open rebellion, to the point that there were more British troops fighting Luddites than there were fighting Napoleon on the Iberian Peninsula. The Luddite rebellion ended with brutal suppression through military force. Let us hope that the black box movement does not come to that.

Why don't you discuss specific ML algorithms?

The book is agnostic with regards to the particular ML algorithm you choose. Whether you use convolutional neural networks, AdaBoost, RFs, SVMs, and so on, there are many shared generic problems you will face: data structuring, labeling, weighting, stationary transformations, cross-validation, feature selection, feature importance, overfitting, backtesting, etc. In the context of financial modeling, answering these questions is non-trivial, and framework-specific approaches need to be developed. That is the focus of this book.

What other books do you recommend on this subject?

To my knowledge, this is the first book to provide a complete and systematic treatment of ML methods specific for finance: starting with a chapter dedicated to financial data structures, another chapter for labeling of financial series, another for sample weighting, time series differentiation, . . . all the way to a full part devoted to the proper backtesting of investment strategies. To be sure, there are a handful of prior publications (mostly journal articles) that have applied standard ML to financial series, but that is not what this book offers. My goal has been to address the unique nuisances that make financial ML modeling particularly challenging. Like any new subject, it is fast evolving, and the book will be updated as major advances take place. Please contact me at mldp@quantresearch.org if there is any particular topic you would like to see treated in future editions. I will gladly add those chapters, while acknowledging the names of those readers who suggested them.

I do not understand some of the sections and chapters. What should I do?

My advice is that you start by reading the references listed at the end of the chapter. When I wrote the book, I had to assume the reader was familiar with the existing literature, or this book would lose its focus. If after reading those references the sections still do not make sense, the likely reason is that they are related to a problem well understood by investment professionals (even if there is no mention of it in the

literature). For example, Chapter 2 will discuss effective methods to adjust futures prices for the roll, a problem known to most practitioners, even though it is rarely addressed in textbooks. I would encourage you to attend one of my regular seminars, and ask me your question at the end of my talk.

Why is the book so fixated on backtest overfitting?

There are two reasons. First, backtest overfitting is arguably the most important open problem in all of mathematical finance. It is our equivalent to "P versus NP" in computer science. If there was a precise method to prevent backtest overfitting, we would be able to take backtests to the bank. A backtest would be almost as good as cash, rather than a sales pitch. Hedge funds would allocate funds to portfolio managers with confidence. Investors would risk less, and would be willing to pay higher fees. Regulators would grant licenses to hedge fund managers on the basis of reliable evidence of skill and knowledge, leaving no space for charlatans. In my opinion, an investments book that does not address this issue is not worth your time. Why would you read a book that deals with CAPM, APT, asset allocation techniques, risk management, etc. when the empirical results that support those arguments were selected without determining their false discovery probabilities?

The second reason is that ML is a great weapon in your research arsenal, and a dangerous one to be sure. If backtest overfitting is an issue in econometric analysis, the flexibility of ML makes it a constant threat to your work. This is particularly the case in finance, because our datasets are shorter, with lower signal-to-noise ratio, and we do not have laboratories where we can conduct experiments while controlling for all environmental variables (López de Prado [2015]). An ML book that does not tackle these concerns can be more detrimental than beneficial to your career.

What is the mathematical nomenclature of the book?

When I started to write this book, I thought about assigning one symbol to each mathematical variable or function through all the chapters. That would work well if this book dealt with a single subject, like stochastic optimal control. However this book deals with a wide range of mathematical subjects, each with its own conventions. Readers would find it harder to consult references unless I also followed literature standards, which means that sometimes we must re-use symbols. To prevent any confusion, every chapter explains the nomenclature as it is being used. Most of the math is accompanied by a code snippet, so in case of doubt, please always follow the code.

Who wrote Chapter 22?

A popular perception is that ML is a new fascinating technology invented or perfected at IBM, Google, Facebook, Amazon, Netflix, Tesla, etc. It is true that technology firms have become heavy users of ML, especially in recent years. Those firms sponsored some of the most publicized recent ML achievements (like *Jeopardy!* or Go), which may have reinforced that perception.

However, the reader may be surprised to learn that, in fact, U.S. National Laboratories are among the research centers with the longest track record and experience

in using ML. These centers utilized ML before it was cool, and they applied it successfully for many decades to produce astounding scientific discoveries. If predicting what movies Netflix should recommend you to watch next is a worthy endeavor, so it is to understand the rate of expansion of the universe, or forecasting what coastlines will be most impacted by global warming, or preventing a cataclysmic failure of our national power grid. These are just some of the amazing questions that institutions like Berkeley Lab work on every day, quietly but tirelessly, with the help of ML.

In Chapter 22, Drs. Horst Simon and Kesheng Wu offer the perspective of a deputy director and a project leader at a major U.S. National Laboratory specializing in large-scale scientific research involving big data, high-performance computing, and ML. Unlike traditional university settings, National Laboratories achieve scientific breakthroughs by putting together interdisciplinary teams that follow well-devised procedures, with strong division of labor and responsibilities. That kind of research model by production chain was born at Berkeley Lab almost 90 years ago and inspired the meta-strategy paradigm explained in Sections 1.2.2 and 1.3.1.

1.7 ACKNOWLEDGMENTS

Dr. Horst Simon, who is the deputy director of Lawrence Berkeley National Laboratory, accepted to co-author Chapter 22 with Dr. Kesheng Wu, who leads several projects at Berkeley Lab and the National Energy Research Scientific Computing Center (NERSC).[3] ML requires extreme amounts of computing power, and my research would not have been possible without their generous support and guidance. In that chapter, Horst and Kesheng explain how Berkeley Lab satisfies the supercomputing needs of researchers worldwide, and the instrumental role played by ML and big data in today's scientific breakthroughs.

Prof. Riccardo Rebonato was the first to read this manuscript and encouraged me to publish it. My many conversations with Prof. Frank Fabozzi on these topics were instrumental in shaping the book in its current form. Very few people in academia have Frank's and Riccardo's industry experience, and very few people in the industry have Riccardo's and Frank's academic pedigree.

Over the past two decades, I have published nearly a hundred works on this book's subject, including journal articles, books, chapters, lectures, source code, etc. In my latest count, these works were co-authored with more than 30 leading experts in this field, including Prof. David H. Bailey (15 articles), Prof. David Easley (8 articles), Prof. Maureen O'Hara (8 articles), and Prof. Jonathan M. Borwein (6 articles). This book is to a great extent also theirs, for it would not have been possible without their support, insights, and continuous exchange of ideas over the years. It would take too long to give them proper credit, so instead I have published the following link where you can find our collective effort: http://www.quantresearch.org/Co-authors.htm.

Last but not least, I would like to thank some of my research team members for proofreading the book and helping me produce some of the figures: Diego Aparicio,

[3] http://www.nersc.gov/about.

Dr. Lee Cohn, Dr. Michael Lewis, Dr. Michael Lock, Dr. Yaxiong Zeng, and Dr. Zhibai Zhang.

EXERCISES

1.1 Are you aware of firms that have attempted to transition from discretionary investments to ML-led investments, or blending them into what they call "quantamental" funds?

(a) Have they succeeded?

(b) What are the cultural difficulties involved in this transition?

1.2 What is the most important open problem in mathematical finance? If this problem was resolved, how could:

(a) regulators use it to grant investment management licenses?

(b) investors use it to allocate funds?

(c) firms use it to reward researchers?

1.3 According to *Institutional Investor,* only 17% of hedge fund assets are managed by quantitative firms. That is about $500 billion allocated in total across all quantitative funds as of June 2017, compared to $386 billion a year earlier. What do you think is driving this massive reallocation of assets?

1.4 According to *Institutional Investor*'s Rich List, how many quantitative investment firms are placed within the top 10 most profitable firms? How does that compare to the proportion of assets managed by quantitative funds?

1.5 What is the key difference between econometric methods and ML? How would economics and finance benefit from updating their statistical toolkit?

1.6 Science has a very minimal understanding of how the human brain (or any brain) works. In this sense, the brain is an absolute black box. What do you think causes critics of financial ML to disregard it as a black box, while embracing discretionary investing?

1.7 You read a journal article that describes an investment strategy. In a backtest, it achieves an annualized Sharpe ratio in excess of 2, with a confidence level of 95%. Using their dataset, you are able to reproduce their result in an independent backtest. Why is this discovery likely to be false?

1.8 Investment advisors are plagued with conflicts of interest while making decisions on behalf of their investors.

(a) ML algorithms can manage investments without conflict of interests. Why?

(b) Suppose that an ML algorithm makes a decision that leads to a loss. The algorithm did what it was programmed to do, and the investor agreed to the terms of the program, as verified by forensic examination of the computer logs. In what sense is this situation better for the investor, compared to a loss caused by a discretionary PM's poor judgment? What is the investor's recourse in each instance?

(c) Would it make sense for financial advisors to benchmark their decisions against the decisions made by such neutral agents?

REFERENCES

Bailey, D., P. Borwein, and S. Plouffe (1997): "On the rapid computation of various polylogarithmic constants." *Mathematics of Computation*, Vol. 66, No. 218, pp. 903–913.

Calkin, N. and M. López de Prado (2014a): "Stochastic flow diagrams." *Algorithmic Finance*, Vol. 3, No. 1, pp. 21–42.

Calkin, N. and M. López de Prado (2014b): "The topology of macro financial flows: An application of stochastic flow diagrams." *Algorithmic Finance*, Vol. 3, No. 1, pp. 43–85.

Easley, D., M. López de Prado, and M. O'Hara (2013): *High-Frequency Trading*, 1st ed. Risk Books.

López de Prado, M. (2014): "Quantitative meta-strategies." *Practical Applications, Institutional Investor Journals*, Vol. 2, No. 3, pp. 1–3.

López de Prado, M. (2015): "The Future of Empirical Finance." *Journal of Portfolio Management*, Vol. 41, No. 4, pp. 140–144.

Stigler, Stephen M. (1981): "Gauss and the invention of least squares." *Annals of Statistics*, Vol. 9, No. 3, pp. 465–474.

BIBLIOGRAPHY

Abu-Mostafa, Y., M. Magdon-Ismail, and H. Lin (2012): *Learning from Data*, 1st ed. AMLBook.

Akansu, A., S. Kulkarni, and D. Malioutov (2016): *Financial Signal Processing and Machine Learning*, 1st ed. John Wiley & Sons-IEEE Press.

Aronson, D. and T. Masters (2013): *Statistically Sound Machine Learning for Algorithmic Trading of Financial Instruments: Developing Predictive-Model-Based Trading Systems Using TSSB*, 1st ed. CreateSpace Independent Publishing Platform.

Boyarshinov, V. (2012): *Machine Learning in Computational Finance: Practical Algorithms for Building Artificial Intelligence Applications*, 1st ed. LAP LAMBERT Academic Publishing.

Cerniglia, J., F. Fabozzi, and P. Kolm (2016): "Best practices in research for quantitative equity strategies." *Journal of Portfolio Management*, Vol. 42, No. 5, pp. 135–143.

Chan, E. (2017): *Machine Trading: Deploying Computer Algorithms to Conquer the Markets*, 1st ed. John Wiley & Sons.

Gareth, J., D. Witten, T. Hastie, and R. Tibshirani (2013): *An Introduction to Statistical Learning: with Applications in R*, 1st ed. Springer.

Geron, A. (2017): *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*, 1st ed. O'Reilly Media.

Gyorfi, L., G. Ottucsak, and H. Walk (2012): *Machine Learning for Financial Engineering*, 1st ed. Imperial College Press.

Hackeling, G. (2014): *Mastering Machine Learning with Scikit-Learn*, 1st ed. Packt Publishing.

Hastie, T., R. Tibshirani, and J. Friedman (2016): *The Elements of Statistical Learning*, 2nd ed. Springer-Verlag.

Hauck, T. (2014): *Scikit-Learn Cookbook*, 1st ed. Packt Publishing.

McNelis, P. (2005): *Neural Networks in Finance*, 1st ed. Academic Press.

Raschka, S. (2015): *Python Machine Learning*, 1st ed. Packt Publishing.

PART 1

Data Analysis

CHAPTER 2

Financial Data Structures

2.1 MOTIVATION

In this chapter we will learn how to work with unstructured financial data, and from that to derive a structured dataset amenable to ML algorithms. In general, you do not want to consume someone else's processed dataset, as the likely outcome will be that you discover what someone else already knows or will figure out soon. Ideally your starting point is a collection of unstructured, raw data that you are going to process in a way that will lead to informative features.

2.2 ESSENTIAL TYPES OF FINANCIAL DATA

Financial data comes in many shapes and forms. Table 2.1 shows the four essential types of financial data, ordered from left to right in terms of increasing diversity. Next, we will discuss their different natures and applications.

2.2.1 Fundamental Data

Fundamental data encompasses information that can be found in regulatory filings and business analytics. It is mostly accounting data, reported quarterly. A particular aspect of this data is that it is reported with a lapse. You must confirm exactly when each data point was released, so that your analysis uses that information only after it was publicly available. A common beginner's error is to assume that this data was published at the end of the reporting period. That is never the case.

For example, fundamental data published by Bloomberg is indexed by the last date included in the report, which precedes the date of the release (often by 1.5 months). In other words, Bloomberg is assigning those values to a date when they were not known. You could not believe how many papers are published every year using misaligned

TABLE 2.1 The Four Essential Types of Financial Data

Fundamental Data	Market Data	Analytics	Alternative Data
• Assets • Liabilities • Sales • Costs/earnings • Macro variables • . . .	• Price/yield/implied volatility • Volume • Dividend/coupons • Open interest • Quotes/cancellations • Aggressor side • . . .	• Analyst recommendations • Credit ratings • Earnings expectations • News sentiment • . . .	• Satellite/CCTV images • Google searches • Twitter/chats • Metadata • . . .

fundamental data, especially in the factor-investing literature. Once you align the data correctly, a substantial number of findings in those papers cannot be reproduced.

A second aspect of fundamental data is that it is often backfilled or reinstated. "Backfilling" means that missing data is assigned a value, even if those values were unknown at that time. A "reinstated value" is a corrected value that amends an incorrect initial release. A company may issue multiple corrections for a past quarter's results long after the first publication, and data vendors may overwrite the initial values with their corrections. The problem is, the corrected values were not known on that first release date. Some data vendors circumvent this problem by storing multiple release dates and values for each variable. For example, we typically have three values for a single quarterly GDP release: the original released value and two monthly revisions. Still, it is very common to find studies that use the final released value and assign it to the time of the first release, or even to the last day in the reporting period. We will revisit this mistake, and its implications, when we discuss backtesting errors in Chapter 11.

Fundamental data is extremely regularized and low frequency. Being so accessible to the marketplace, it is rather unlikely that there is much value left to be exploited. Still, it may be useful in combination with other data types.

2.2.2 Market Data

Market data includes all trading activity that takes place in an exchange (like CME) or trading venue (like MarketAxess). Ideally, your data provider has given you a raw feed, with all sorts of unstructured information, like FIX messages that allow you to fully reconstruct the trading book, or the full collection of BWIC (bids wanted in competition) responses. Every market participant leaves a characteristic footprint in the trading records, and with enough patience, you will find a way to anticipate a competitor's next move. For example, TWAP algorithms leave a very particular footprint that is used by predatory algorithms to front-run their end-of-day trading (usually hedging) activity (Easley, López de Prado, and O'Hara [2011]). Human GUI traders often trade in round lots, and you can use this fact to estimate what percentage of the volume is coming from them at a given point in time, then associate it with a particular market behavior.

One appealing aspect of FIX data is that it is not trivial to process, unlike fundamental data. It is also very abundant, with over 10 TB being generated on a daily basis. That makes it a more interesting dataset for strategy research.

2.2.3 Analytics

You could think of analytics as derivative data, based on an original source, which could be fundamental, market, alternative, or even a collection of other analytics. What characterizes analytics is not the content of the information, but that it is not readily available from an original source, and that it has been processed for you in a particular way. Investment banks and research firms sell valuable information that results from in-depth analyses of companies' business models, activities, competition, outlook, etc. Some specialized firms sell statistics derived from alternative data, for example, the sentiment extracted from news reports and social media.

A positive aspect of analytics is that the signal has been extracted for you from a raw source. The negative aspects are that analytics may be costly, the methodology used in their production may be biased or opaque, and you will not be the sole consumer.

2.2.4 Alternative Data

Kolanovic and Krishnamachari [2017] differentiate among alternative data produced by individuals (social media, news, web searches, etc.), business processes (transactions, corporate data, government agencies, etc.), and sensors (satellites, geolocation, weather, CCTV, etc.). Some popular satellite image or video feeds include monitoring of tankers, tunnel traffic activity, or parking lot occupancies.

What truly characterizes alternative data is that it is primary information, that is, information that has not made it to the other sources. Before Exxon Mobile reported increased earnings, before its market price shot up, before analysts wrote their commentary of their latest filings, before all of that, there were movements of tankers and drillers and pipeline traffic. They happened months before those activities were reflected in the other data types. Two problematic aspects of alternative data are their cost and privacy concerns. All that spy craft is expensive, and the surveilled company may object, not to mention bystanders.

Alternative data offers the opportunity to work with truly unique, hard-to-process datasets. Remember, data that is hard to store, manipulate, and operate is always the most promising. You will recognize that a dataset *may be* useful if it annoys your data infrastructure team. Perhaps your competitors did not try to use it for logistic reasons, gave up midway, or processed it incorrectly.

2.3 BARS

In order to apply ML algorithms on your unstructured data, we need to parse it, extract valuable information from it, and store those extractions in a regularized format. Most ML algorithms assume a table representation of the extracted data.

Finance practitioners often refer to those tables' rows as "bars." We can distinguish between two categories of bar methods: (1) standard bar methods, which are common in the literature, and (2) more advanced, information-driven methods, which sophisticated practitioners use although they cannot be found (yet) in journal articles. In this section, we will discuss how to form those bars.

2.3.1 Standard Bars

Some bar construction methods are very popular in the financial industry, to the point that most data vendors' APIs offer several of them. The purpose of these methods is to transform a series of observations that arrive at irregular frequency (often referred to as "inhomogeneous series") into a homogeneous series derived from regular sampling.

2.3.1.1 Time Bars

Time bars are obtained by sampling information at fixed time intervals, e.g., once every minute. The information collected usually includes:

- Timestamp
- Volume-weighted average price (VWAP)
- Open (i.e., first) price
- Close (i.e., last) price
- High price
- Low price
- Volume traded, etc.

Although time bars are perhaps the most popular among practitioners and academics, they should be avoided for two reasons. First, markets do not process information at a constant time interval. The hour following the open is much more active than the hour around noon (or the hour around midnight in the case of futures). As biological beings, it makes sense for humans to organize their day according to the sunlight cycle. But today's markets are operated by algorithms that trade with loose human supervision, for which CPU processing cycles are much more relevant than chronological intervals (Easley, López de Prado, and O'Hara [2011]). This means that time bars oversample information during low-activity periods and undersample information during high-activity periods. Second, time-sampled series often exhibit poor statistical properties, like serial correlation, heteroscedasticity, and non-normality of returns (Easley, López de Prado, and O'Hara [2012]). GARCH models were developed, in part, to deal with the heteroscedasticity associated with incorrect sampling. As we will see next, forming bars as a subordinated process of trading activity avoids this problem in the first place.

2.3.1.2 Tick Bars

The idea behind tick bars is straightforward: The sample variables listed earlier (timestamp, VWAP, open price, etc.) will be extracted each time a pre-defined number

of transactions takes place, e.g., 1,000 ticks. This allows us to synchronize sampling with a proxy of information arrival (the speed at which ticks are originated).

Mandelbrot and Taylor [1967] were among the first to realize that sampling as a function of the number of transactions exhibited desirable statistical properties: "Price changes over a fixed number of transactions may have a Gaussian distribution. Price changes over a fixed time period may follow a stable Paretian distribution, whose variance is infinite. Since the number of transactions in any time period is random, the above statements are not necessarily in disagreement."

Ever since Mandelbrot and Taylor's paper, multiple studies have confirmed that sampling as a function of trading activity allows us to achieve returns closer to IID Normal (see Ané and Geman [2000]). This is important, because many statistical methods rely on the assumption that observations are drawn from an IID Gaussian process. Intuitively, we can only draw inference from a random variable that is invariant, and tick bars allow for better inference than time bars.

When constructing tick bars, you need to be aware of outliers. Many exchanges carry out an auction at the open and an auction at the close. This means that for a period of time, the order book accumulates bids and offers without matching them. When the auction concludes, a large trade is published at the clearing price, for an outsized amount. This auction trade could be the equivalent of thousands of ticks, even though it is reported as one tick.

2.3.1.3 Volume Bars

One problem with tick bars is that order fragmentation introduces some arbitrariness in the number of ticks. For example, suppose that there is one order sitting on the offer, for a size of 10. If we buy 10 lots, our one order will be recorded as one tick. If instead on the offer there are 10 orders of size 1, our one buy will be recorded as 10 separate transactions. In addition, matching engine protocols can further split one fill into multiple artificial partial fills, as a matter of operational convenience.

Volume bars circumvent that problem by sampling every time a pre-defined amount of the security's units (shares, futures contracts, etc.) have been exchanged. For example, we could sample prices every time a futures contract exchanges 1,000 units, regardless of the number of ticks involved.

It is hard to imagine these days, but back in the 1960s vendors rarely published volume data, as customers were mostly concerned with tick prices. After volume started to be reported as well, Clark [1973] realized that sampling returns by volume achieved even better statistical properties (i.e., closer to an IID Gaussian distribution) than sampling by tick bars. Another reason to prefer volume bars over time bars or tick bars is that several market microstructure theories study the interaction between prices and volume. Sampling as a function of one of these variables is a convenient artifact for these analyses, as we will find out in Chapter 19.

2.3.1.4 Dollar Bars

Dollar bars are formed by sampling an observation every time a pre-defined market value is exchanged. Of course, the reference to dollars is meant to apply to the

currency in which the security is denominated, but nobody refers to euro bars, pound bars, or yen bars (although gold bars would make for a fun pun).

Let me illustrate the rationale behind dollar bars with a couple of examples. First, suppose that we wish to analyze a stock that has exhibited an appreciation of 100% over a certain period of time. Selling $1,000 worth of that stock at the end of the period requires trading half the number of shares it took to buy $1,000 worth of that stock at the beginning. In other words, the number of shares traded is a function of the actual value exchanged. Therefore, it makes sense sampling bars in terms of dollar value exchanged, rather than ticks or volume, particularly when the analysis involves significant price fluctuations. This point can be verified empirically. If you compute tick bars and volume bars on E-mini S&P 500 futures for a given bar size, the number of bars per day will vary wildly over the years. That range and speed of variation will be reduced once you compute the number of dollar bars per day over the years, for a constant bar size. Figure 2.1 plots the exponentially weighted average number of bars per day when we apply a fixed bar size on tick, volume, and dollar sampling methods.

A second argument that makes dollar bars more interesting than time, tick, or volume bars is that the number of outstanding shares often changes multiple times over the course of a security's life, as a result of corporate actions. Even after adjusting for splits and reverse splits, there are other actions that will impact the amount of ticks and volumes, like issuing new shares or buying back existing shares (a very common practice since the Great Recession of 2008). Dollar bars tend to be robust in the face of those actions. Still, you may want to sample dollar bars where the size of the bar is not kept constant over time. Instead, the bar size could be adjusted dynamically as a function of the free-floating market capitalization of a company (in the case of stocks), or the outstanding amount of issued debt (in the case of fixed-income securities).

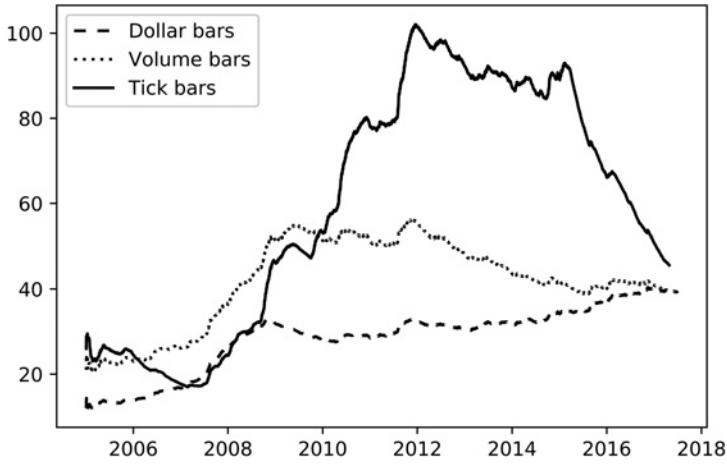


FIGURE 2.1 Average daily frequency of tick, volume, and dollar bars

2.3.2 Information-Driven Bars

The purpose of information-driven bars is to sample more frequently when new information arrives to the market. In this context, the word "information" is used in a market microstructural sense. As we will see in Chapter 19, market microstructure theories confer special importance to the persistence of imbalanced signed volumes, as that phenomenon is associated with the presence of informed traders. By synchronizing sampling with the arrival of informed traders, we may be able to make decisions before prices reach a new equilibrium level. In this section we will explore how to use various indices of information arrival to sample bars.

2.3.2.1 Tick Imbalance Bars

Consider a sequence of ticks $\{(p_t, v_t)\}_{t=1,\ldots,T}$, where p_t is the price associated with tick t and v_t is the volume associated with tick t. The so-called tick rule defines a sequence $\{b_t\}_{t=1,\ldots,T}$ where

$$b_t = \begin{cases} b_{t-1} & \text{if } \Delta p_t = 0 \\ \dfrac{|\Delta p_t|}{\Delta p_t} & \text{if } \Delta p_t \neq 0 \end{cases}$$

with $b_t \in \{-1, 1\}$, and the boundary condition b_0 is set to match the terminal value b_T from the immediately preceding bar. The idea behind tick imbalance bars (TIBs) is to sample bars whenever tick imbalances exceed our expectations. We wish to determine the tick index, T, such that the accumulation of signed ticks (signed according to the tick rule) exceeds a given threshold. Next, let us discuss the procedure to determine T.

First, we define the tick imbalance at time T as

$$\theta_T = \sum_{t=1}^{T} b_t$$

Second, we compute the expected value of θ_T at the beginning of the bar, $\mathrm{E}_0[\theta_T] = \mathrm{E}_0[T](\mathrm{P}[b_t = 1] - \mathrm{P}[b_t = -1])$, where $\mathrm{E}_0[T]$ is the expected size of the tick bar, $\mathrm{P}[b_t = 1]$ is the unconditional probability that a tick is classified as a buy, and $\mathrm{P}[b_t = -1]$ is the unconditional probability that a tick is classified as a sell. Since $\mathrm{P}[b_t = 1] + \mathrm{P}[b_t = -1] = 1$, then $\mathrm{E}_0[\theta_T] = \mathrm{E}_0[T](2\mathrm{P}[b_t = 1] - 1)$. In practice, we can estimate $\mathrm{E}_0[T]$ as an exponentially weighted moving average of T values from prior bars, and $(2\mathrm{P}[b_t = 1] - 1)$ as an exponentially weighted moving average of b_t values from prior bars.

Third, we define a tick imbalance bar (TIB) as a T^*-contiguous subset of ticks such that the following condition is met:

$$T^* = \arg\min_T \left\{ |\theta_T| \geq \mathrm{E}_0[T] \left| 2\mathrm{P}\left[b_t = 1\right] - 1 \right| \right\}$$

where the size of the expected imbalance is implied by $|2\mathrm{P}[b_t = 1] - 1|$. When θ_T is more imbalanced than expected, a low T will satisfy these conditions. Accordingly, TIBs are produced more frequently under the presence of informed trading (asymmetric information that triggers one-side trading). In fact, we can understand TIBs as buckets of trades containing equal amounts of information (regardless of the volumes, prices, or ticks traded).

2.3.2.2 Volume/Dollar Imbalance Bars

The idea behind volume imbalance bars (VIBs) and dollar imbalance bars (DIBs) is to extend the concept of tick imbalance bars (TIBs). We would like to sample bars when volume or dollar imbalances diverge from our expectations. Based on the same notions of tick rule and boundary condition b_0 as we discussed for TIBs, we will define a procedure to determine the index of the next sample, T.

First, we define the imbalance at time T as

$$\theta_T = \sum_{t=1}^{T} b_t v_t$$

where v_t may represent either the number of securities traded (VIB) or the dollar amount exchanged (DIB). Your choice of v_t is what determines whether you are sampling according to the former or the latter.

Second, we compute the expected value of θ_T at the beginning of the bar

$$\mathrm{E}_0[\theta_T] = \mathrm{E}_0\left[\sum_{t|b_t=1}^{T} v_t\right] - \mathrm{E}_0\left[\sum_{t|b_t=-1}^{T} v_t\right] = \mathrm{E}_0[T](\mathrm{P}[b_t = 1]\mathrm{E}_0[v_t|b_t = 1] - \mathrm{P}[b_t = -1]\mathrm{E}_0[v_t|b_t = -1])$$

Let us denote $v^+ = \mathrm{P}[b_t = 1]\mathrm{E}_0[v_t|b_t = 1]$, $v^- = \mathrm{P}[b_t = -1]\mathrm{E}_0[v_t|b_t = -1]$, so that $\mathrm{E}_0[T]^{-1}E_0[\sum_t v_t] = \mathrm{E}_0[v_t] = v^+ + v^-$. You can think of v^+ and v^- as decomposing the initial expectation of v_t into the component contributed by buys and the component contributed by sells. Then

$$\mathrm{E}_0[\theta_T] = \mathrm{E}_0[T](v^+ - v^-) = \mathrm{E}_0[T](2v^+ - \mathrm{E}_0[v_t])$$

In practice, we can estimate $\mathrm{E}_0[T]$ as an exponentially weighted moving average of T values from prior bars, and $(2v^+ - \mathrm{E}_0[v_t])$ as an exponentially weighted moving average of $b_t v_t$ values from prior bars.

Third, we define VIB or DIB as a T^*-contiguous subset of ticks such that the following condition is met:

$$T^* = \arg\min_T \{|\theta_T| \geq \mathrm{E}_0[T]|2v^+ - \mathrm{E}_0[v_t]|\}$$

where the size of the expected imbalance is implied by $|2v^+ - \mathrm{E}_0[v_t]|$. When θ_T is more imbalanced than expected, a low T will satisfy these conditions. This is the information-based analogue of volume and dollar bars, and like its predecessors, it addresses the same concerns regarding tick fragmentation and outliers. Furthermore, it also addresses the issue of corporate actions, because the above procedure does not rely on a constant bar size. Instead, the bar size is adjusted dynamically.

2.3.2.3 Tick Runs Bars

TIBs, VIBs, and DIBs monitor order flow imbalance, as measured in terms of ticks, volumes, and dollar values exchanged. Large traders will sweep the order book, use iceberg orders, or slice a parent order into multiple children, all of which leave a trace of runs in the $\{b_t\}_{t=1,\ldots,T}$ sequence. For this reason, it can be useful to monitor the *sequence* of buys in the overall volume, and take samples when that sequence diverges from our expectations.

First, we define the length of the current run as

$$\theta_T = \max\left\{\sum_{t|b_t=1}^{T} b_t, -\sum_{t|b_t=-1}^{T} b_t\right\}$$

Second, we compute the expected value of θ_T at the beginning of the bar

$$\mathrm{E}_0[\theta_T] = \mathrm{E}_0[T]\max\{\mathrm{P}[b_t = 1], 1 - \mathrm{P}[b_t = 1]\}$$

In practice, we can estimate $\mathrm{E}_0[T]$ as an exponentially weighted moving average of T values from prior bars, and $\mathrm{P}[b_t = 1]$ as an exponentially weighted moving average of the proportion of buy ticks from prior bars.

Third, we define a tick runs bar (TRB) as a T^*-contiguous subset of ticks such that the following condition is met:

$$T^* = \arg\min_T\{\theta_T \geq \mathrm{E}_0[T]\max\{\mathrm{P}[b_t = 1],1 - \mathrm{P}[b_t = 1]\}\}$$

where the expected count of ticks from runs is implied by $\max\{\mathrm{P}[b_t = 1], 1 - \mathrm{P}[b_t = -1]\}$. When θ_T exhibits more runs than expected, a low T will satisfy these conditions. Note that in this definition of runs we allow for sequence breaks. That is, instead of measuring the length of the longest sequence, we count the number of ticks of each side, without offsetting them (no imbalance). In the context of forming bars, this turns out to be a more useful definition than measuring sequence lengths.

2.3.2.4 Volume/Dollar Runs Bars

Volume runs bars (VRBs) and dollar runs bars (DRBs) extend the above definition of runs to volumes and dollars exchanged, respectively. The intuition is that we wish to sample bars whenever the volumes or dollars traded by one side exceed our expectation for a bar. Following our customary nomenclature for the tick rule, we need to determine the index T of the last observation in the bar.

First, we define the volumes or dollars associated with a run as

$$\theta_T = \max\left\{\sum_{t|b_t=1}^{T} b_t v_t, -\sum_{t|b_t=-1}^{T} b_t v_t\right\}$$

where v_t may represent either number of securities traded (VRB) or dollar amount exchanged (DRB). Your choice of v_t is what determines whether you are sampling according to the former or the latter.

Second, we compute the expected value of θ_T at the beginning of the bar,

$$\mathrm{E}_0[\theta_T] = \mathrm{E}_0[T]\max\{\mathrm{P}[b_t = 1]\mathrm{E}_0[v_t|b_t = 1], (1 - \mathrm{P}[b_t = 1])\mathrm{E}_0[v_t|b_t = -1]\}$$

In practice, we can estimate $\mathrm{E}_0[T]$ as an exponentially weighted moving average of T values from prior bars, $\mathrm{P}[b_t = 1]$ as an exponentially weighted moving average of the proportion of buy ticks from prior bars, $\mathrm{E}_0[v_t|b_t = 1]$ as an exponentially weighted moving average of the buy volumes from prior bars, and $\mathrm{E}_0[v_t|b_t = -1]$ as an exponentially weighted moving average of the sell volumes from prior bars.

Third, we define a volume runs bar (VRB) as a T^*-contiguous subset of ticks such that the following condition is met:

$$\begin{aligned} T^* = \arg\min_T \{\theta_T \geq \mathrm{E}_0[T]\max\{\mathrm{P}[b_t = 1]\mathrm{E}_0[v_t|b_t = 1], \\ (1 - \mathrm{P}[b_t = 1])\mathrm{E}_0[v_t|b_t = -1]\}\} \end{aligned}$$

where the expected volume from runs is implied by $\max\{\mathrm{P}[b_t = 1]\mathrm{E}_0[v_t|b_t = 1], (1 - \mathrm{P}[b_t = 1])\mathrm{E}_0[v_t|b_t = -1]\}$. When θ_T exhibits more runs than expected, or the volume from runs is greater than expected, a low T will satisfy these conditions.

2.4 DEALING WITH MULTI-PRODUCT SERIES

Sometimes we are interested in modelling a time series of instruments, where the weights need to be dynamically adjusted over time. Other times we must deal with products that pay irregular coupons or dividends, or that are subject to corporate actions. Events that alter the nature of the time series under study need to be treated properly, or we will inadvertently introduce a structural break that will mislead our research efforts (more on this in Chapter 17). This problem appears in many guises: when we model spreads with changing weights, or baskets of securities where dividends/coupons must be reinvested, or baskets that must be rebalanced, or when an index's constituents are changed, or when we must replace an expired/matured contract/security with another, etc.

Futures are a case in point. In my experience, people struggle unnecessarily when manipulating futures, mainly because they do not know how to handle the roll well. The same can be said of strategies based on spreads of futures, or baskets of stocks or bonds. In the next section, I'll show you one way to model a basket of securities as if it was a single cash product. I call it the "ETF trick" because the goal is to transform any complex multi-product dataset into a single dataset that resembles a total-return ETF. Why is this useful? Because your code can always assume that you only trade cashlike products (non-expiring cash instruments), regardless of the complexity and composition of the underlying series.

2.4.1 The ETF Trick

Suppose we wish to develop a strategy that trades a spread of futures. A few nuisances arise from dealing with a spread rather than an outright instrument. First, the spread is characterized by a vector of weights that changes over time. As a result, the spread itself may converge even if prices do not change. When that happens, a model trading that series will be misled to believe that PnL (the net mark-to-market value of profits and losses) has resulted from that weight-induced convergence. Second, spreads can acquire negative values, because they do not represent a price. This can often be problematic, as most models assume positive prices. Third, trading times will not align exactly for all constituents, so the spread is not always tradeable at the last levels published, or with zero latency risk. Also, execution costs must be considered, like crossing the bid-ask spread.

One way to avoid these issues is to produce a time series that reflects the value of $1 invested in a spread. Changes in the series will reflect changes in PnL, the series will be strictly positive (at worst, infinitesimal), and the implementation shortfall will be taken into account. This will be the series used to model, generate signals, and trade, as if it were an ETF.

Suppose that we are given a history of bars, as derived from any of the methods explained in Section 2.3. These bars contain the following columns:

- $o_{i,t}$ is the raw open price of instrument $i = 1, \ldots, I$ at bar $t = 1, \ldots, T$.
- $p_{i,t}$ is the raw close price of instrument $i = 1, \ldots, I$ at bar $t = 1, \ldots, T$.
- $\varphi_{i,t}$ is the USD value of one point of instrument $i = 1, \ldots, I$ at bar $t = 1, \ldots, T$. This includes foreign exchange rate.
- $v_{i,t}$ is the volume of instrument $i = 1, \ldots, I$ at bar $t = 1, \ldots, T$.
- $d_{i,t}$ is the carry, dividend, or coupon paid by instrument i at bar t. This variable can also be used to charge margin costs, or costs of funding.

where all instruments $i = 1, \ldots, I$ were tradeable at bar $t = 1, \ldots, T$. In other words, even if some instruments were not tradeable over the entirety of the time interval $[t-1, t]$, at least they were tradeable at times $t-1$ and t (markets were open and able to execute orders at those instants). For a basket of futures characterized by an

allocations vector ω_t rebalanced (or rolled) on bars $B \subseteq \{1, \ldots, T\}$, the \$1 investment value $\{K_t\}$ is derived as

$$h_{i,t} = \begin{cases} \dfrac{\omega_{i,t} K_t}{o_{i,t+1} \varphi_{i,t} \sum_{i=1}^{I} |\omega_{i,t}|} & \text{if } t \in B \\ h_{i,t-1} & \text{otherwise} \end{cases}$$

$$\delta_{i,t} = \begin{cases} p_{i,t} - o_{i,t} & \text{if } (t-1) \in B \\ \Delta p_{i,t} & \text{otherwise} \end{cases}$$

$$K_t = K_{t-1} + \sum_{i=1}^{I} h_{i,t-1} \varphi_{i,t} \left(\delta_{i,t} + d_{i,t} \right)$$

and $K_0 = 1$ in the initial AUM. Variable $h_{i,t}$ represents the holdings (number of securities or contracts) of instrument i at time t. Variable $\delta_{i,t}$ is the change of market value between $t-1$ and t for instrument i. Note that profits or losses are being reinvested whenever $t \in B$, hence preventing the negative prices. Dividends $d_{i,t}$ are already embedded in K_t, so there is no need for the strategy to know about them. The purpose of $\omega_{i,t} \left(\sum_{i=1}^{I} |\omega_{i,t}| \right)^{-1}$ in $h_{i,t}$ is to de-lever the allocations. For series of futures, we may not know $p_{i,t}$ of the new contract at a roll time t, so we use $o_{i,t+1}$ as the closest in time.

Let τ_i be the transaction cost associated with trading \$1 of instrument i, e.g., $\tau_i = 1E-4$ (one basis point). There are three additional variables that the strategy needs to know for every observed bar t:

1. **Rebalance costs:** The variable cost $\{c_t\}$ associated with the allocation rebalance is $c_t = \sum_{i=1}^{I} (|h_{i,t-1}| p_{i,t} + |h_{i,t}| o_{i,t+1}) \varphi_{i,t} \tau_i$, $\forall t \in B$. We do not embed c_t in K_t, or shorting the spread will generate fictitious profits when the allocation is rebalanced. In your code, you can treat $\{c_t\}$ as a (negative) dividend.
2. **Bid-ask spread:** The cost $\{\tilde{c}_t\}$ of buying or selling one unit of this virtual ETF is $\tilde{c}_t = \sum_{i=1}^{I} |h_{i,t-1}| p_{i,t} \varphi_{i,t} \tau_i$. When a unit is bought or sold, the strategy must charge this cost $\tilde{c}_t$, which is the equivalent to crossing the bid-ask spread of this virtual ETF.
3. **Volume:** The volume traded $\{v_t\}$ is determined by the least active member in the basket. Let $v_{i,t}$ be the volume traded by instrument i over bar t. The number of tradeable basket units is $v_t = \min_i \left\{ \frac{v_{i,t}}{|h_{i,t-1}|} \right\}$.

Transaction costs functions are not necessarily linear, and those non-linear costs can be simulated by the strategy based on the above information. Thanks to the ETF trick, we can model a basket of futures (or a single futures) as if it was a single non-expiring cash product.

2.4.2 PCA Weights

The interested reader will find many practical ways of computing hedging weights in López de Prado and Leinweber [2012] and Bailey and López de Prado [2012]. For the sake of completeness, let us review one way to derive the vector $\{\omega_t\}$ used in the previous section. Consider an IID multivariate Gaussian process characterized by a vector of means μ, of size *Nx1*, and a covariance matrix V, of size *NxN*. This stochastic process describes an invariant random variable, like the returns of stocks, the changes in yield of bonds, or changes in options' volatilities, for a portfolio of N instruments. We would like to compute the vector of allocations ω that conforms to a particular distribution of risks across V's principal components.

First, we perform a spectral decomposition, $VW = W\Lambda$, where the columns in W are reordered so that the elements of Λ's diagonal are sorted in descending order. Second, given a vector of allocations ω, we can compute the portfolio's risk as $\sigma^2 = \omega' V\omega = \omega' W\Lambda W'\omega = \beta'\Lambda\beta = (\Lambda^{1/2}\beta)'(\Lambda^{1/2}\beta)$, where β represents the projection of ω on the orthogonal basis. Third, Λ is a diagonal matrix, thus $\sigma^2 = \sum_{n=1}^{N} \beta_n^2 \Lambda_{n,n}$, and the risk attributed to the nth component is $R_n = \beta_n^2 \Lambda_{n,n}\sigma^{-2} = [W'\omega]_n^2 \Lambda_{n,n}\sigma^{-2}$, with $R'1_N = 1$, and 1_N is a vector of N ones. You can interpret $\{R_n\}_{n=1,\ldots,N}$ as the distribution of risks across the orthogonal components.

Fourth, we would like to compute the vector ω that delivers a user-defined risk distribution R. From earlier steps, $\beta = \left\{\sigma \sqrt{\frac{R_n}{\Lambda_{n,n}}}\right\}_{n=1,\ldots,N}$, which represents the allocation in the new (orthogonal) basis. Fifth, the allocation in the old basis is given by $\omega = W\beta$. Re-scaling ω merely re-scales σ, hence keeping the risk distribution constant. Figure 2.2 illustrates the contribution to risk per principal component for an

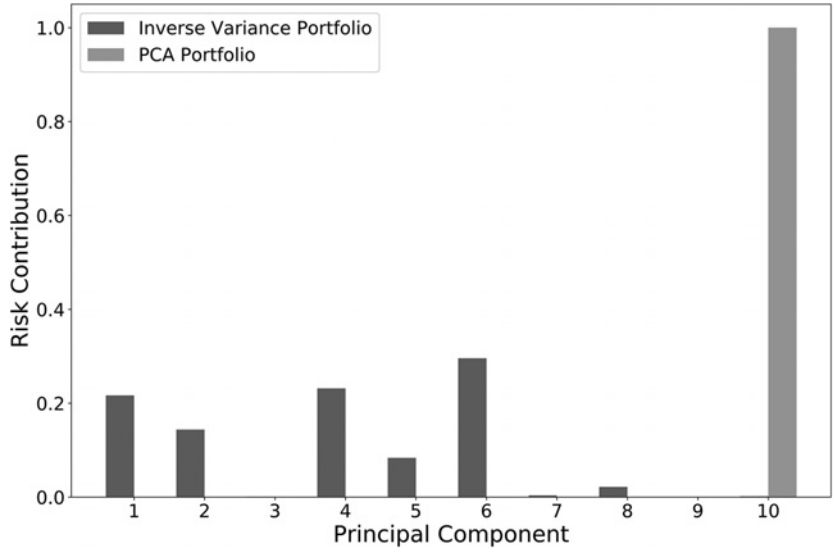


FIGURE 2.2 Contribution to risk per principal component

inverse variance allocation. Almost all principal components contribute risk, including those with highest variance (components 1 and 2). In contrast, for the PCA portfolio, only the component with lowest variance contributes risk.

Snippet 2.1 implements this method, where the user-defined risk distribution R is passed through argument `riskDist` (optional None). If `riskDist` is None, the code will assume all risk must be allocated to the principal component with smallest eigenvalue, and the weights will be the last eigenvector re-scaled to match σ (`riskTarget`).

SNIPPET 2.1 PCA WEIGHTS FROM A RISK DISTRIBUTION R

```
def pcaWeights(cov,riskDist=None,riskTarget=1.):
    # Following the riskAlloc distribution, match riskTarget
    eVal,eVec=np.linalg.eigh(cov) # must be Hermitian
    indices=eVal.argsort()[::-1] # arguments for sorting eVal desc
    eVal,eVec=eVal[indices],eVec[:,indices]
    if riskDist is None:
        riskDist=np.zeros(cov.shape[0])
        riskDist[-1]=1.
    loads=riskTarget*(riskDist/eVal)**.5
    wghts=np.dot(eVec,np.reshape(loads,(-1,1)))
    #ctr=(loads/riskTarget)**2*eVal # verify riskDist
    return wghts
```

2.4.3 Single Future Roll

The ETF trick can handle the rolls of a single futures contract, as a particular case of a 1-legged spread. However, when dealing with a single futures contract, an equivalent and more direct approach is to form a time series of cumulative roll gaps, and detract that gaps series from the price series. Snippet 2.2 shows a possible implementation of this logic, using a series of tick bars downloaded from Bloomberg and stored in a HDF5 table. The meaning of the Bloomberg fields is as follows:

- `FUT_CUR_GEN_TICKER`: It identifies the contract associated with that price. Its value changes with every roll.
- `PX_OPEN`: The open price associated with that bar.
- `PX_LAST`: The close price associated with the bar.
- `VWAP`: The volume-weighted average price associated with that bar.

The argument `matchEnd` in function `rollGaps` determines whether the futures series should be rolled forward (`matchEnd=False`) or backward (`matchEnd=True`). In a forward roll, the price at the start of the rolled series

matches the price at the start of the raw series. In a backward roll, the price at the end of the rolled series matches the price at the end of the raw series.

SNIPPET 2.2 FORM A GAPS SERIES, DETRACT IT FROM PRICES

```
def getRolledSeries(pathIn,key):
    series=pd.read_hdf(pathIn,key='bars/ES_10k')
    series['Time']=pd.to_datetime(series['Time'],format='%Y%m%d%H%M%S%f')
    series=series.set_index('Time')
    gaps=rollGaps(series)
    for fld in ['Close','VWAP']:series[fld]-=gaps
    return series
#------------------------------------------
def rollGaps(series,dictio={'Instrument':'FUT_CUR_GEN_TICKER','Open':'PX_OPEN', \
    'Close':'PX_LAST'},matchEnd=True):
    # Compute gaps at each roll, between previous close and next open
    rollDates=series[dictio['Instrument']].drop_duplicates(keep='first').index
    gaps=series[dictio['Close']]*0
    iloc=list(series.index)
    iloc=[iloc.index(i)-1 for i in rollDates] # index of days prior to roll
    gaps.loc[rollDates[1:]]=series[dictio['Open']].loc[rollDates[1:]]- \
        series[dictio['Close']].iloc[iloc[1:]].values
    gaps=gaps.cumsum()
    if matchEnd:gaps-=gaps.iloc[-1] # roll backward
    return gaps
```

Rolled prices are used for simulating PnL and portfolio mark-to-market values. However, raw prices should still be used to size positions and determine capital consumption. Keep in mind, rolled prices can indeed become negative, particularly in futures contracts that sold off while in contango. To see this, run Snippet 2.2 on a series of Cotton #2 futures or Natural Gas futures.

In general, we wish to work with non-negative rolled series, in which case we can derive the price series of a $1 investment as follows: (1) Compute a time series of rolled futures prices, (2) compute the return (r) as rolled price change divided by the previous raw price, and (3) form a price series using those returns (i.e., `(1+r).cumprod()`). Snippet 2.3 illustrates this logic.

SNIPPET 2.3 NON-NEGATIVE ROLLED PRICE SERIES

```
raw=pd.read_csv(filePath,index_col=0,parse_dates=True)
gaps=rollGaps(raw,dictio={'Instrument':'Symbol','Open':'Open','Close':'Close'})
rolled=raw.copy(deep=True)
for fld in ['Open','Close']:rolled[fld]-=gaps
rolled['Returns']=rolled['Close'].diff()/raw['Close'].shift(1)
rolled['rPrices']=(1+rolled['Returns']).cumprod()
```

2.5 SAMPLING FEATURES

So far we have learned how to produce a continuous, homogeneous, and structured dataset from a collection of unstructured financial data. Although you could attempt to apply an ML algorithm on such a dataset, in general that would not be a good idea, for a couple of reasons. First, several ML algorithms do not scale well with sample size (e.g., SVMs). Second, ML algorithms achieve highest accuracy when they attempt to learn from relevant examples. Suppose that you wish to predict whether the next 5% absolute return will be positive (a 5% rally) or negative (a 5% sell-off). At any random time, the accuracy of such a prediction will be low. However, if we ask a classifier to predict the sign of the next 5% absolute return after certain catalytic conditions, we are more likely to find informative features that will help us achieve a more accurate prediction. In this section we discuss ways of sampling bars to produce a features matrix with relevant training examples.

2.5.1 Sampling for Reduction

As we have mentioned earlier, one reason for sampling features from a structured dataset is to reduce the amount of data used to fit the ML algorithm. This operation is also referred to as *downsampling*. This is often done by either sequential sampling at a constant step size (linspace sampling), or by sampling randomly using a uniform distribution (uniform sampling).

The main advantage of linspace sampling is its simplicity. The disadvantages are that the step size is arbitrary, and that outcomes may vary depending on the seed bar. Uniform sampling addresses these concerns by drawing samples uniformly across the entire set of bars. Still, both methods suffer the criticism that the sample does not necessarily contain the subset of most relevant observations in terms of their predictive power or informational content.

2.5.2 Event-Based Sampling

Portfolio managers typically place a bet after some event takes place, such as a structural break (Chapter 17), an extracted signal (Chapter 18), or microstructural phenomena (Chapter 19). These events could be associated with the release of some macroeconomic statistics, a spike in volatility, a significant departure in a spread away from its equilibrium level, etc. We can characterize an event as significant, and let the ML algorithm learn whether there is an accurate prediction function under those circumstances. Perhaps the answer is no, in which case we would redefine what constitutes an event, or try again with alternative features. For illustration purposes, let us discuss one useful event-based sampling method.

2.5.2.1 The CUSUM Filter

The CUSUM filter is a quality-control method, designed to detect a shift in the mean value of a measured quantity away from a target value. Consider IID

observations $\{y_t\}_{t=1,\ldots,T}$ arising from a locally stationary process. We define the cumulative sums

$$S_t = \max\left\{0, S_{t-1} + y_t - \mathrm{E}_{t-1}\left[y_t\right]\right\}$$

with boundary condition $S_0 = 0$. This procedure would recommend an action at the first t satisfying $S_t \geq h$, for some threshold h (the filter size). Note that $S_t = 0$ whenever $y_t \leq \mathrm{E}_{t-1}[y_t] - S_{t-1}$. This zero floor means that we will skip some downward deviations that otherwise would make S_t negative. The reason is, the filter is set up to identify a sequence of upside divergences from any reset level zero. In particular, the threshold is activated when

$$S_t \geq h \Leftrightarrow \exists \tau \in [1, t] \left| \sum_{i=\tau}^{t} \left(y_i - \mathrm{E}_{i-1}\left[y_t\right]\right) \geq h \right.$$

This concept of run-ups can be extended to include run-downs, giving us a symmetric CUSUM filter:

$$S_t^+ = \max\left\{0, S_{t-1}^+ + y_t - \mathrm{E}_{t-1}\left[y_t\right]\right\},\ S_0^+ = 0$$

$$S_t^- = \min\left\{0, S_{t-1}^- + y_t - \mathrm{E}_{t-1}\left[y_t\right]\right\},\ S_0^- = 0$$

$$S_t = \max\left\{S_t^+, -S_t^-\right\}$$

Lam and Yam [1997] propose an investment strategy whereby alternating buy-sell signals are generated when an absolute return h is observed relative to a prior high or low. Those authors demonstrate that such strategy is equivalent to the so-called "filter trading strategy" studied by Fama and Blume [1966]. Our use of the CUSUM filter is different: We will sample a bar t if and only if $S_t \geq h$, at which point S_t is reset. Snippet 2.4 shows an implementation of the symmetric CUSUM filter, where $\mathrm{E}_{t-1}[y_t] = y_{t-1}$.

SNIPPET 2.4 THE SYMMETRIC CUSUM FILTER

```
def getTEvents(gRaw,h):
    tEvents,sPos,sNeg=[],0,0
    diff=gRaw.diff()
    for i in diff.index[1:]:
        sPos,sNeg=max(0,sPos+diff.loc[i]),min(0,sNeg+diff.loc[i])
        if sNeg<-h:
            sNeg=0;tEvents.append(i)
        elif sPos>h:
            sPos=0;tEvents.append(i)
    return pd.DatetimeIndex(tEvents)
```

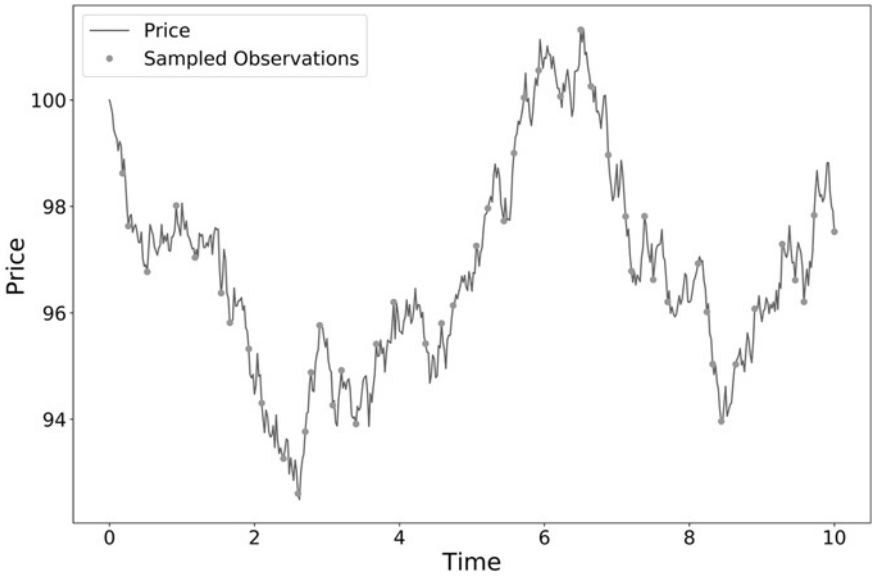


FIGURE 2.3 CUSUM sampling of a price series

The function `getTEvents` receives two arguments: the raw time series we wish to filter (`gRaw`) and the threshold, `h`. One practical aspect that makes CUSUM filters appealing is that multiple events are not triggered by `gRaw` hovering around a threshold level, which is a flaw suffered by popular market signals such as Bollinger bands. It will require a full run of length `h` for `gRaw` to trigger an event. Figure 2.3 illustrates the samples taken by a CUSUM filter on a price series.

Variable S_t could be based on any of the features we will discuss in Chapters 17–19, like structural break statistics, entropy, or market microstructure measurements. For example, we could declare an event whenever SADF departs sufficiently from a previous reset level (to be defined in Chapter 17). Once we have obtained this subset of event-driven bars, we will let the ML algorithm determine whether the occurrence of such events constitutes actionable intelligence.

EXERCISES

2.1 On a series of E-mini S&P 500 futures tick data:

- **(a)** Form tick, volume, and dollar bars. Use the ETF trick to deal with the roll.
- **(b)** Count the number of bars produced by tick, volume, and dollar bars on a weekly basis. Plot a time series of that bar count. What bar type produces the most stable weekly count? Why?
- **(c)** Compute the serial correlation of returns for the three bar types. What bar method has the lowest serial correlation?

(d) Partition the bar series into monthly subsets. Compute the variance of returns for every subset of every bar type. Compute the variance of those variances. What method exhibits the smallest variance of variances?

(e) Apply the Jarque-Bera normality test on returns from the three bar types. What method achieves the lowest test statistic?

2.2 On a series of E-mini S&P 500 futures tick data, compute dollar bars and dollar imbalance bars. What bar type exhibits greater serial correlation? Why?

2.3 On dollar bar series of E-mini S&P 500 futures and Eurostoxx 50 futures:

(a) Apply Section 2.4.2 to compute the $\{\hat{\omega}_t\}$ vector used by the ETF trick. (Hint: You will need FX values for EUR/USD at the roll dates.)

(b) Derive the time series of the S&P 500/Eurostoxx 50 spread.

(c) Confirm that the series is stationary, with an ADF test.

2.4 Form E-mini S&P 500 futures dollar bars:

(a) Compute Bollinger bands of width 5% around a rolling moving average. Count how many times prices cross the bands out (from within the bands to outside the bands).

(b) Now sample those bars using a CUSUM filter, where $\{y_t\}$ are returns and $h = 0.05$. How many samples do you get?

(c) Compute the rolling standard deviation of the two-sampled series. Which one is least heteroscedastic? What is the reason for these results?

2.5 Using the bars from exercise 4:

(a) Sample bars using the CUSUM filter, where $\{y_t\}$ are absolute returns and $h = 0.05$.

(b) Compute the rolling standard deviation of the sampled bars.

(c) Compare this result with the results from exercise 4. What procedure delivered the least heteroscedastic sample? Why?

REFERENCES

Ané, T. and H. Geman (2000): "Order flow, transaction clock and normality of asset returns." *Journal of Finance*, Vol. 55, pp. 2259–2284.

Bailey, David H., and M. López de Prado (2012): "Balanced baskets: A new approach to trading and hedging risks." *Journal of Investment Strategies (Risk Journals)*, Vol. 1, No. 4 (Fall), pp. 21–62.

Clark, P. K. (1973): "A subordinated stochastic process model with finite variance for speculative prices." *Econometrica*, Vol. 41, pp. 135–155.

Easley, D., M. López de Prado, and M. O'Hara (2011): "The volume clock: Insights into the high frequency paradigm." *Journal of Portfolio Management*, Vol. 37, No. 2, pp. 118–128.

Easley, D., M. López de Prado, and M. O'Hara (2012): "Flow toxicity and liquidity in a high frequency world." *Review of Financial Studies*, Vol. 25, No. 5, pp. 1457–1493.

Fama, E. and M. Blume (1966): "Filter rules and stock market trading." *Journal of Business*, Vol. 40, pp. 226–241.

Kolanovic, M. and R. Krishnamachari (2017): "Big data and AI strategies: Machine learning and alternative data approach to investing." White paper, JP Morgan, Quantitative and Derivatives Strategy. May 18.

Lam, K. and H. Yam (1997): "CUSUM techniques for technical trading in financial markets." *Financial Engineering and the Japanese Markets*, Vol. 4, pp. 257–274.

López de Prado, M. and D. Leinweber (2012): "Advances in cointegration and subset correlation hedging methods." *Journal of Investment Strategies (Risk Journals)*, Vol. 1, No. 2 (Spring), pp. 67–115.

Mandelbrot, B. and M. Taylor (1967): "On the distribution of stock price differences." *Operations Research*, Vol. 15, No. 5, pp. 1057–1062.

CHAPTER 3

Labeling

3.1 MOTIVATION

In Chapter 2 we discussed how to produce a matrix X of financial features out of an unstructured dataset. Unsupervised learning algorithms can learn the patterns from that matrix X, for example whether it contains hierarchical clusters. On the other hand, supervised learning algorithms require that the rows in X are associated with an array of labels or values y, so that those labels or values can be predicted on unseen features samples. In this chapter we will discuss ways to label financial data.

3.2 THE FIXED-TIME HORIZON METHOD

As it relates to finance, virtually all ML papers label observations using the fixed-time horizon method. This method can be described as follows. Consider a features matrix X with I rows, $\{X_i\}_{i=1,\ldots,I}$, drawn from some bars with index $t = 1, \ldots, T$, where $I \leq T$. Chapter 2, Section 2.5 discussed sampling methods that produce the set of features $\{X_i\}_{i=1,\ldots,I}$. An observation X_i is assigned a label $y_i \in \{-1, 0, 1\}$,

$$y_i = \begin{cases} -1 & \text{if } r_{t_{i,0},t_{i,0}+h} < -\tau \\ 0 & \text{if } |r_{t_{i,0},t_{i,0}+h}| \leq \tau \\ 1 & \text{if } r_{t_{i,0},t_{i,0}+h} > \tau \end{cases}$$

where τ is a pre-defined constant threshold, $t_{i,0}$ is the index of the bar immediately after X_i takes place, $t_{i,0} + h$ is the index of the h-th bar after $t_{i,0}$, and $r_{t_{i,0},t_{i,0}+h}$ is the price return over a bar horizon h,

$$r_{t_{i,0},t_{i,0}+h} = \frac{p_{t_{i,0}+h}}{p_{t_{i,0}}} - 1$$

Because the literature almost always works with time bars, h implies a fixed-time horizon. The bibliography section lists multiple ML studies, of which Dixon et al. [2016] is a recent example of this labeling method. Despite its popularity, there are several reasons to avoid this approach in most cases. First, as we saw in Chapter 2, time bars do not exhibit good statistical properties. Second, the same threshold τ is applied regardless of the observed volatility. Suppose that $\tau = 1E-2$, where sometimes we label an observation as $y_i = 1$ subject to a realized bar volatility of $\sigma_{t_{i,0}} = 1E-4$ (e.g., during the night session), and sometimes $\sigma_{t_{i,0}} = 1E-2$ (e.g., around the open). The large majority of labels will be 0, even if return $r_{t_{i,0},t_{i,0}+h}$ was predictable and statistically significant.

In other words, it is a very common error to label observations according to a fixed threshold on time bars. Here are a couple of better alternatives. First, label per a varying threshold $\sigma_{t_{i,0}}$, estimated using a rolling exponentially weighted standard deviation of returns. Second, use volume or dollar bars, as their volatilities are much closer to constant (homoscedasticity). But even these two improvements miss a key flaw of the fixed-time horizon method: the path followed by prices. Every investment strategy has stop-loss limits, whether they are self-imposed by the portfolio manager, enforced by the risk department, or triggered by a margin call. It is simply unrealistic to build a strategy that profits from positions that would have been stopped-out by the exchange. That virtually no publication accounts for that when labeling observations tells you something about the current state of the investment literature.

3.3 COMPUTING DYNAMIC THRESHOLDS

As argued in the previous section, in practice we want to set profit taking and stop-loss limits that are a function of the risks involved in a bet. Otherwise, sometimes we will be aiming too high ($\tau \gg \sigma_{t_{i,0}}$), and sometimes too low ($\tau \ll \sigma_{t_{i,0}}$), considering the prevailing volatility.

Snippet 3.1 computes the daily volatility at intraday estimation points, applying a span of `span0` days to an exponentially weighted moving standard deviation. See the pandas documentation for details on the `pandas.Series.ewm` function.

SNIPPET 3.1 DAILY VOLATILITY ESTIMATES

```
def getDailyVol(close,span0=100):
    # daily vol, reindexed to close
    df0=close.index.searchsorted(close.index-pd.Timedelta(days=1))
    df0=df0[df0>0]
    df0=pd.Series(close.index[df0-1], index=close.index[close.shape[0]-df0.shape[0]:])
    df0=close.loc[df0.index]/close.loc[df0.values].values-1 # daily returns
    df0=df0.ewm(span=span0).std()
    return df0
```

We can use the output of this function to set default profit taking and stop-loss limits throughout the rest of this chapter.

3.4 THE TRIPLE-BARRIER METHOD

Here I will introduce an alternative labeling method that I have not found in the literature. If you are an investment professional, I think you will agree that it makes more sense. I call it the triple-barrier method because it labels an observation according to the first barrier touched out of three barriers. First, we set two horizontal barriers and one vertical barrier. The two horizontal barriers are defined by profit-taking and stop-loss limits, which are a dynamic function of estimated volatility (whether realized or implied). The third barrier is defined in terms of number of bars elapsed since the position was taken (an expiration limit). If the upper barrier is touched first, we label the observation as a 1. If the lower barrier is touched first, we label the observation as a -1. If the vertical barrier is touched first, we have two choices: the sign of the return, or a 0. I personally prefer the former as a matter of realizing a profit or loss within limits, but you should explore whether a 0 works better in your particular problems.

You may have noticed that the triple-barrier method is path-dependent. In order to label an observation, we must take into account the entire path spanning $[t_{i,0}, t_{i,0} + h]$, where h defines the vertical barrier (the expiration limit). We will denote $t_{i,1}$ the time of the first barrier touch, and the return associated with the observed feature is $r_{t_{i,0},t_{i,1}}$. For the sake of clarity, $t_{i,1} \leq t_{i,0} + h$ and the horizontal barriers are not necessarily symmetric.

Snippet 3.2 implements the triple-barrier method. The function receives four arguments:

- `close`: A pandas series of prices.
- `events`: A pandas dataframe, with columns,
  - `t1`: The timestamp of vertical barrier. When the value is `np.nan`, there will not be a vertical barrier.
  - `trgt`: The unit width of the horizontal barriers.
- `ptSl`: A list of two non-negative float values:
  - `ptSl[0]`: The factor that multiplies `trgt` to set the width of the upper barrier. If 0, there will not be an upper barrier.
  - `ptSl[1]`: The factor that multiplies `trgt` to set the width of the lower barrier. If 0, there will not be a lower barrier.
- `molecule`: A list with the subset of event indices that will be processed by a single thread. Its use will become clear later on in the chapter.

SNIPPET 3.2 TRIPLE-BARRIER LABELING METHOD

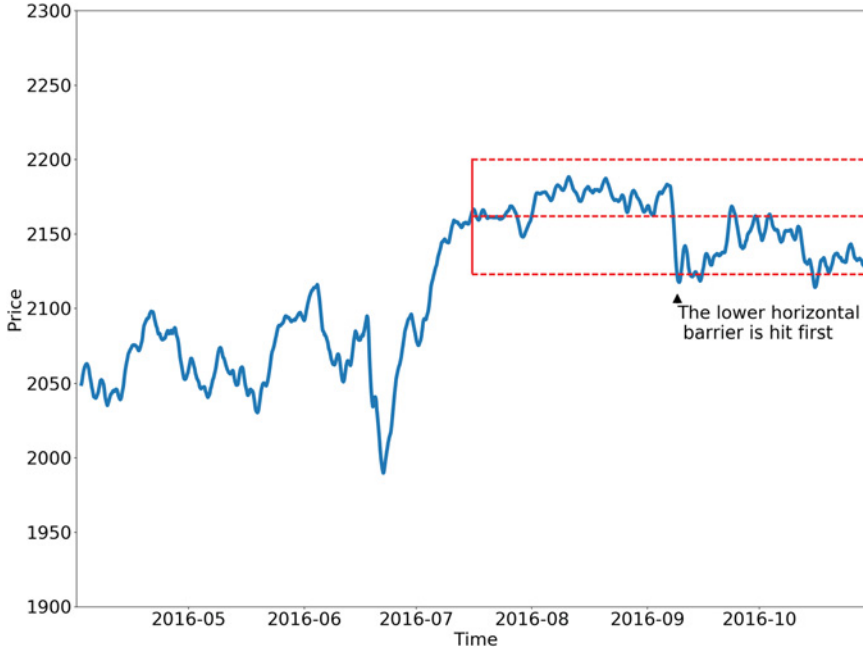
```
def applyPtSlOnT1(close,events,ptSl,molecule):
    # apply stop loss/profit taking, if it takes place before t1 (end of event)
    events_=events.loc[molecule]
    out=events_[['t1']].copy(deep=True)
```

```
if ptSl[0]>0:pt=ptSl[0]*events_['trgt']
else:pt=pd.Series(index=events.index) # NaNs
if ptSl[1]>0:sl=-ptSl[1]*events_['trgt']
else:sl=pd.Series(index=events.index) # NaNs
for loc,t1 in events_['t1'].fillna(close.index[-1]).iteritems():
    df0=close[loc:t1] # path prices
    df0=(df0/close[loc]-1)*events_.at[loc,'side'] # path returns
    out.loc[loc,'sl']=df0[df0<sl[loc]].index.min() # earliest stop loss.
    out.loc[loc,'pt']=df0[df0>pt[loc]].index.min() # earliest profit taking.
return out
```

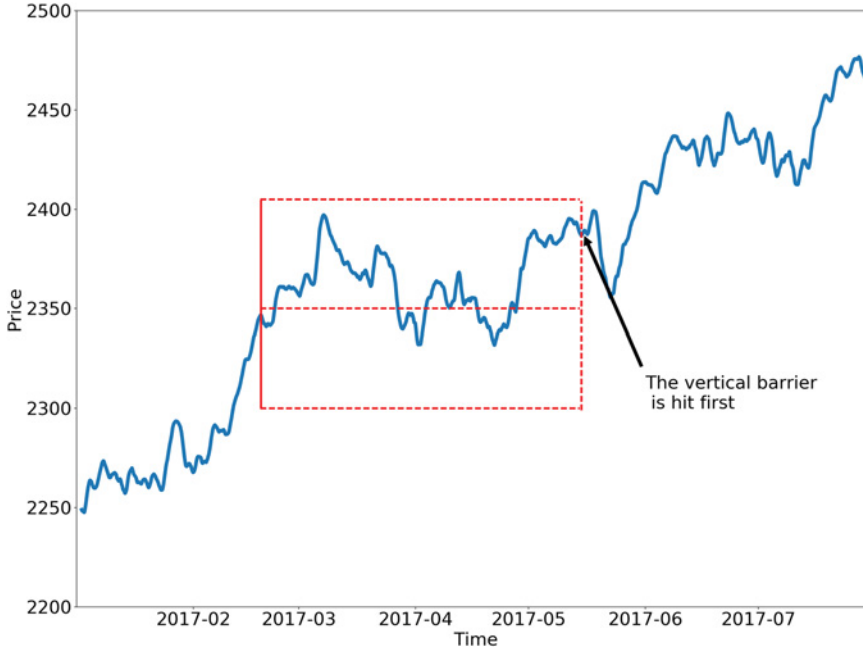
The output from this function is a pandas dataframe containing the timestamps (if any) at which each barrier was touched. As you can see from the previous description, the method considers the possibility that each of the three barriers may be disabled. Let us denote a barrier configuration by the triplet [`pt,sl,t1`], where a 0 means that the barrier is inactive and a 1 means that the barrier is active. The possible eight configurations are:

- Three useful configurations:
  - [1,1,1]: This is the standard setup, where we define three barrier exit conditions. We would like to realize a profit, but we have a maximum tolerance for losses and a holding period.
  - [0,1,1]: In this setup, we would like to exit after a number of bars, unless we are stopped-out.
  - [1,1,0]: Here we would like to take a profit as long as we are not stopped-out. This is somewhat unrealistic in that we are willing to hold the position for as long as it takes.
- Three less realistic configurations:
  - [0,0,1]: This is equivalent to the fixed-time horizon method. It may still be useful when applied to volume-, dollar-, or information-driven bars, and multiple forecasts are updated within the horizon.
  - [1,0,1]: A position is held until a profit is made or the maximum holding period is exceeded, without regard for the intermediate unrealized losses.
  - [1,0,0]: A position is held until a profit is made. It could mean being locked on a losing position for years.
- Two illogical configurations:
  - [0,1,0]: This is an aimless configuration, where we hold a position until we are stopped-out.
  - [0,0,0]: There are no barriers. The position is locked forever, and no label is generated.

Figure 3.1 shows two alternative configurations of the triple-barrier method. On the left, the configuration is [1,1,0], where the first barrier touched is the lower horizontal one. On the right, the configuration is [1,1,1], where the first barrier touched is the vertical one.



(a)



(b)

FIGURE 3.1 Two alternative configurations of the triple-barrier method

3.5 LEARNING SIDE AND SIZE

In this section we will discuss how to label examples so that an ML algorithm can learn both the side and the size of a bet. We are interested in learning the side of a bet when we do not have an underlying model to set the sign of our position (long or short). Under such circumstance, we cannot differentiate between a profit-taking barrier and a stop-loss barrier, since that requires knowledge of the side. Learning the side implies that either there are no horizontal barriers or that the horizontal barriers must be symmetric.

Snippet 3.3 implements the function `getEvents`, which finds the time of the first barrier touch. The function receives the following arguments:

- `close`: A pandas series of prices.
- `tEvents`: The pandas timeindex containing the timestamps that will seed every triple barrier. These are the timestamps selected by the sampling procedures discussed in Chapter 2, Section 2.5.
- `ptSl`: A non-negative float that sets the width of the two barriers. A 0 value means that the respective horizontal barrier (profit taking and/or stop loss) will be disabled.
- `t1`: A pandas series with the timestamps of the vertical barriers. We pass a `False` when we want to disable vertical barriers.
- `trgt`: A pandas series of targets, expressed in terms of absolute returns.
- `minRet`: The minimum target return required for running a triple barrier search.
- `numThreads`: The number of threads concurrently used by the function.

SNIPPET 3.3 GETTING THE TIME OF FIRST TOUCH

```
def getEvents(close,tEvents,ptSl,trgt,minRet,numThreads,t1=False):
    #1) get target
    trgt=trgt.loc[tEvents]
    trgt=trgt[trgt>minRet] # minRet
    #2) get t1 (max holding period)
    if t1 is False:t1=pd.Series(pd.NaT,index=tEvents)
    #3) form events object, apply stop loss on t1
    side_=pd.Series(1.,index=trgt.index)
    events=pd.concat({'t1':t1,'trgt':trgt,'side':side_}, \
        axis=1).dropna(subset=['trgt'])
    df0=mpPandasObj(func=applyPtSlOnT1,pdObj=('molecule',events.index), \
        numThreads=numThreads,close=close,events=events,ptSl=[ptSl,ptSl])
    events['t1']=df0.dropna(how='all').min(axis=1) # pd.min ignores nan
    events=events.drop('side',axis=1)
    return events
```

Suppose that $I = 1E6$ and $h = 1E3$, then the number of conditions to evaluate is up to one billion on a single instrument. Many ML tasks are computationally

expensive unless you are familiar with multi-threading, and this is one of them. Here is where parallel computing comes into play. Chapter 20 discusses a few multiprocessing functions that we will use throughout the book.

Function `mpPandasObj` calls a multiprocessing engine, which is explained in depth in Chapter 20. For the moment, you simply need to know that this function will execute `applyPtSlOnT1` in parallel. Function `applyPtSlOnT1` returns the timestamps at which each barrier is touched (if any). Then, the time of the first touch is the earliest time among the three returned by `applyPtSlOnT1`. Because we must learn the side of the bet, we have passed `ptSl = [ptSl,ptSl]` as argument, and we arbitrarily set the side to be always long (the horizontal barriers are symmetric, so the side is irrelevant to determining the time of the first touch). The output from this function is a pandas dataframe with columns:

- `t1`: The timestamp at which the first barrier is touched.
- `trgt`: The target that was used to generate the horizontal barriers.

Snippet 3.4 shows one way to define a vertical barrier. For each index in `tEvents`, it finds the timestamp of the next price bar at or immediately after a number of days `numDays`. This vertical barrier can be passed as optional argument `t1` in `getEvents`.

SNIPPET 3.4 ADDING A VERTICAL BARRIER

```
t1=close.index.searchsorted(tEvents+pd.Timedelta(days=numDays))
t1=t1[t1<close.shape[0]]
t1=pd.Series(close.index[t1],index=tEvents[:t1.shape[0]]) # NaNs at end
```

Finally, we can label the observations using the `getBins` function defined in Snippet 3.5. The arguments are the `events` dataframe we just discussed, and the `close` pandas series of prices. The output is a dataframe with columns:

- `ret`: The return realized at the time of the first touched barrier.
- `bin`: The label, $\{-1, 0, 1\}$, as a function of the sign of the outcome. The function can be easily adjusted to label as 0 those events when the vertical barrier was touched first, which we leave as an exercise.

SNIPPET 3.5 LABELING FOR SIDE AND SIZE

```
def getBins(events,close):
    #1) prices aligned with events
    events_=events.dropna(subset=['t1'])
    px=events_.index.union(events_['t1'].values).drop_duplicates()
    px=close.reindex(px,method='bfill')
```

```
    #2) create out object
    out=pd.DataFrame(index=events_.index)
    out['ret']=px.loc[events_['t1'].values].values/px.loc[events_.index]-1
    out['bin']=np.sign(out['ret'])
    return out
```

3.6 META-LABELING

Suppose that you have a model for setting the side of the bet (long or short). You just need to learn the size of that bet, which includes the possibility of no bet at all (zero size). This is a situation that practitioners face regularly. We often know whether we want to buy or sell a product, and the only remaining question is how much money we should risk in such a bet. We do not want the ML algorithm to learn the side, just to tell us what is the appropriate size. At this point, it probably does not surprise you to hear that no book or paper has so far discussed this common problem. Thankfully, that misery ends here. I call this problem meta-labeling because we want to build a secondary ML model that learns how to use a primary exogenous model.

Rather than writing an entirely new `getEvents` function, we will make some adjustments to the previous code, in order to handle meta-labeling. First, we accept a new `side` optional argument (with default `None`), which contains the side of our bets as decided by the primary model. When `side` is not `None`, the function understands that meta-labeling is in play. Second, because now we know the side, we can effectively discriminate between profit taking and stop loss. The horizontal barriers do not need to be symmetric, as in Section 3.5. Argument `ptSl` is a list of two non-negative float values, where `ptSl[0]` is the factor that multiplies `trgt` to set the width of the upper barrier, and `ptSl[1]` is the factor that multiplies `trgt` to set the width of the lower barrier. When either is 0, the respective barrier is disabled. Snippet 3.6 implements these enhancements.

SNIPPET 3.6 EXPANDING `getEvents` TO INCORPORATE META-LABELING

```
def getEvents(close,tEvents,ptSl,trgt,minRet,numThreads,t1=False,side=None):
    #1) get target
    trgt=trgt.loc[tEvents]
    trgt=trgt[trgt>minRet] # minRet
    #2) get t1 (max holding period)
    if t1 is False:t1=pd.Series(pd.NaT,index=tEvents)
    #3) form events object, apply stop loss on t1
    if side is None:side_,ptSl_=pd.Series(1.,index=trgt.index),[ptSl[0],ptSl[0]]
    else:side_,ptSl_=side.loc[trgt.index],ptSl[:2]
    events=pd.concat({'t1':t1,'trgt':trgt,'side':side_}, \
        axis=1).dropna(subset=['trgt'])
    df0=mpPandasObj(func=applyPtSlOnT1,pdObj=('molecule',events.index), \
        numThreads=numThreads,close=inst['Close'],events=events,ptSl=ptSl_)
```

```
    events['t1']=df0.dropna(how='all').min(axis=1) # pd.min ignores nan
    if side is None:events=events.drop('side',axis=1)
    return events
```

Likewise, we need to expand the `getBins` function, so that it handles meta-labeling. Snippet 3.7 implements the necessary changes.

SNIPPET 3.7 EXPANDING `getBins` TO INCORPORATE META-LABELING

```
def getBins(events,close):
    '''
    Compute event's outcome (including side information, if provided).
    events is a DataFrame where:
    —events.index is event's starttime
    —events['t1'] is event's endtime
    —events['trgt'] is event's target
    —events['side'] (optional) implies the algo's position side
    Case 1: ('side' not in events): bin in (-1,1) <—label by price action
    Case 2: ('side' in events): bin in (0,1) <—label by pnl (meta-labeling)
    '''
    #1) prices aligned with events
    events_=events.dropna(subset=['t1'])
    px=events_.index.union(events_['t1'].values).drop_duplicates()
    px=close.reindex(px,method='bfill')
    #2) create out object
    out=pd.DataFrame(index=events_.index)
    out['ret']=px.loc[events_['t1'].values].values/px.loc[events_.index]-1
    if 'side' in events_:out['ret']*=events_['side'] # meta-labeling
    out['bin']=np.sign(out['ret'])
    if 'side' in events_:out.loc[out['ret']<=0,'bin']=0 # meta-labeling
    return out
```

Now the possible values for labels in `out['bin']` are $\{0,1\}$, as opposed to the previous feasible values $\{-1,0,1\}$. The ML algorithm will be trained to decide whether to take the bet or pass, a purely binary prediction. When the predicted label is 1, we can use the probability of this secondary prediction to derive the size of the bet, where the side (sign) of the position has been set by the primary model.

3.7 HOW TO USE META-LABELING

Binary classification problems present a trade-off between type-I errors (false positives) and type-II errors (false negatives). In general, increasing the true positive rate of a binary classifier will tend to increase its false positive rate. The receiver operating

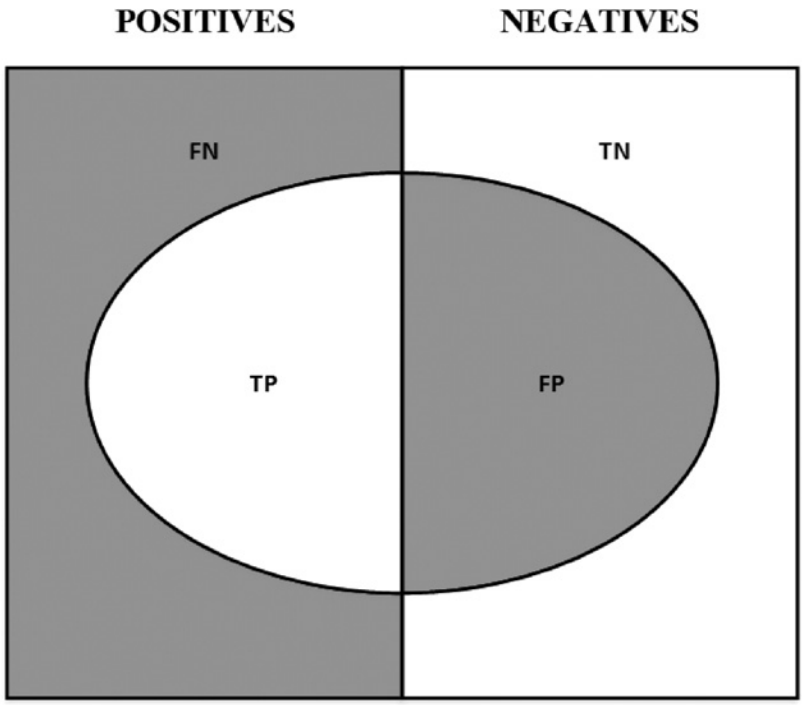


FIGURE 3.2 A visualization of the "confusion matrix"

characteristic (ROC) curve of a binary classifier measures the cost of increasing the true positive rate, in terms of accepting higher false positive rates.

Figure 3.2 illustrates the so-called "confusion matrix." On a set of observations, there are items that exhibit a condition (positives, left rectangle), and items that do not exhibit a condition (negative, right rectangle). A binary classifier predicts that some items exhibit the condition (ellipse), where the TP area contains the true positives and the TN area contains the true negatives. This leads to two kinds of errors: false positives (FP) and false negatives (FN). "Precision" is the ratio between the TP area and the area in the ellipse. "Recall" is the ratio between the TP area and the area in the left rectangle. This notion of recall (aka true positive rate) is in the context of classification problems, the analogous to "power" in the context of hypothesis testing. "Accuracy" is the sum of the TP and TN areas divided by the overall set of items (square). In general, decreasing the FP area comes at a cost of increasing the FN area, because higher precision typically means fewer calls, hence lower recall. Still, there is some combination of precision and recall that maximizes the overall efficiency of the classifier. The F1-score measures the efficiency of a classifier as the harmonic average between precision and recall (more on this in Chapter 14).

Meta-labeling is particularly helpful when you want to achieve higher F1-scores. First, we build a model that achieves high recall, even if the precision is not particularly high. Second, we correct for the low precision by applying meta-labeling to the positives predicted by the primary model.

Meta-labeling will increase your F1-score by filtering out the false positives, where the majority of positives have already been identified by the primary model. Stated differently, the role of the secondary ML algorithm is to determine whether a positive from the primary (exogenous) model is true or false. It is *not* its purpose to come up with a betting opportunity. Its purpose is to determine whether we should act or pass on the opportunity that has been presented.

Meta-labeling is a very powerful tool to have in your arsenal, for four additional reasons. First, ML algorithms are often criticized as black boxes (see Chapter 1). Meta-labeling allows you to build an ML system on top of a white box (like a fundamental model founded on economic theory). This ability to transform a fundamental model into an ML model should make meta-labeling particularly useful to "quantamental" firms. Second, the effects of overfitting are limited when you apply meta-labeling, because ML will not decide the side of your bet, only the size. Third, by decoupling the side prediction from the size prediction, meta-labeling enables sophisticated strategy structures. For instance, consider that the features driving a rally may differ from the features driving a sell-off. In that case, you may want to develop an ML strategy exclusively for long positions, based on the buy recommendations of a primary model, and an ML strategy exclusively for short positions, based on the sell recommendations of an entirely different primary model. Fourth, achieving high accuracy on small bets and low accuracy on large bets will ruin you. As important as identifying good opportunities is to size them properly, so it makes sense to develop an ML algorithm solely focused on getting that critical decision (sizing) right. We will retake this fourth point in Chapter 10. In my experience, meta-labeling ML models can deliver more robust and reliable outcomes than standard labeling models.

3.8 THE QUANTAMENTAL WAY

You may have read in the press that many hedge funds are embracing the quantamental approach. A simple Google search will show reports that many hedge funds, including some of the most traditional ones, are investing tens of millions of dollars in technologies designed to combine human expertise with quantitative methods. It turns out, meta-labeling is exactly what these people have been waiting for. Let us see why.

Suppose that you have a series of features that you believe can forecast some prices, you just do not know how. Since you do not have a model to determine the side of each bet, you need to learn both side and size. You apply what you have learned in Section 3.5, and produce some labels based on the triple-barrier method with symmetric horizontal barriers. Now you are ready to fit your algorithm on a training set, and evaluate the accuracy of your forecasts on a testing set. Alternatively, you could do the following:

1. Use your forecasts from the primary model, and generate meta-labels. Remember, horizontal barriers do not need to be symmetric in this case.
2. Fit your model again on the same training set, but this time using the meta-labels you just generated.
3. Combine the "sides" from the first ML model with the "sizes" from the second ML model.

You can always add a meta-labeling layer to any primary model, whether that is an ML algorithm, an econometric equation, a technical trading rule, a fundamental analysis, etc. That includes forecasts generated by a human, solely based on his intuition. In that case, meta-labeling will help us figure out when we should pursue or dismiss a discretionary PM's call. The features used by such meta-labeling ML algorithm could range from market information to biometric statistics to psychological assessments. For example, the meta-labeling ML algorithm could find that discretionary PMs tend to make particularly good calls when there is a structural break (Chapter 17), as they may be quicker to grasp a change in the market regime. Conversely, it may find that PMs under stress, as evidenced by fewer hours of sleep, fatigue, change in weight, etc. tend to make inaccurate predictions.[1] Many professions require regular psychological exams, and an ML meta-labeling algorithm may find that those scores are also relevant to assess our current degree of confidence on a PM's predictions. Perhaps none of these factors affect discretionary PMs, and their brains operate independently from their emotional being, like cold calculating machines. My guess is that this is not the case, and therefore meta-labeling should become an essential ML technique for every discretionary hedge fund. In the near future, every discretionary hedge fund will become a quantamental firm, and meta-labeling offers them a clear path to make that transition.

3.9 DROPPING UNNECESSARY LABELS

Some ML classifiers do not perform well when classes are too imbalanced. In those circumstances, it is preferable to drop extremely rare labels and focus on the more common outcomes. Snippet 3.8 presents a procedure that recursively drops observations associated with extremely rare labels. Function `dropLabels` recursively eliminates those observations associated with classes that appear less than a fraction `minPct` of cases, unless there are only two classes left.

SNIPPET 3.8 DROPPING UNDER-POPULATED LABELS

```
def dropLabels(events,minPtc=.05):
    # apply weights, drop labels with insufficient examples
    while True:
        df0=events['bin'].value_counts(normalize=True)
        if df0.min()>minPct or df0.shape[0]<3:break
        print 'dropped label',df0.argmin(),df0.min()
        events=events[events['bin']!=df0.argmin()]
    return events
```

[1] You are probably aware of at least one large hedge fund that monitors the emotional state of their research analysts on a daily basis.

Incidentally, another reason you may want to drop unnecessary labels is this known sklearn bug: https://github.com/scikit-learn/scikit-learn/issues/8566. This sort of bug is a consequence of very fundamental assumptions in sklearn implementation, and resolving them is far from trivial. In this particular instance, the error stems from sklearn's decision to operate with standard numpy arrays rather than structured arrays or pandas objects. It is unlikely that there will be a fix by the time you are reading this chapter, or in the near future. In later chapters, we will study ways to circumvent these sorts of implementation errors, by building your own classes and expanding sklearn's functionality.

EXERCISES

3.1 Form dollar bars for E-mini S&P 500 futures:

- **(a)** Apply a symmetric CUSUM filter (Chapter 2, Section 2.5.2.1) where the threshold is the standard deviation of daily returns (Snippet 3.1).
- **(b)** Use Snippet 3.4 on a pandas series `t1`, where `numDays = 1`.
- **(c)** On those sampled features, apply the triple-barrier method, where `ptSl = [1,1]` and `t1` is the series you created in point 1.b.
- **(d)** Apply `getBins` to generate the labels.

3.2 From exercise 1, use Snippet 3.8 to drop rare labels.

3.3 Adjust the `getBins` function (Snippet 3.5) to return a 0 whenever the vertical barrier is the one touched first.

3.4 Develop a trend-following strategy based on a popular technical analysis statistic (e.g., crossing moving averages). For each observation, the model suggests a side, but not a size of the bet.

- **(a)** Derive meta-labels for `ptSl = [1,2]` and `t1` where `numDays = 1`. Use as `trgt` the daily standard deviation as computed by Snippet 3.1.
- **(b)** Train a random forest to decide whether to trade or not. Note: The decision is whether to trade or not, $\{0,1\}$, since the underlying model (the crossing moving average) has decided the side, $\{-1,1\}$.

3.5 Develop a mean-reverting strategy based on Bollinger bands. For each observation, the model suggests a side, but not a size of the bet.

- **(a)** Derive meta-labels for `ptSl = [0,2]` and `t1` where `numDays = 1`. Use as `trgt` the daily standard deviation as computed by Snippet 3.1.
- **(b)** Train a random forest to decide whether to trade or not. Use as features: volatility, serial correlation, and the crossing moving averages from exercise 2.
- **(c)** What is the accuracy of predictions from the primary model (i.e., if the secondary model does not filter the bets)? What are the precision, recall, and F1-scores?
- **(d)** What is the accuracy of predictions from the secondary model? What are the precision, recall, and F1-scores?

BIBLIOGRAPHY

Ahmed, N., A. Atiya, N. Gayar, and H. El-Shishiny (2010): "An empirical comparison of machine learning models for time series forecasting." *Econometric Reviews*, Vol. 29, No. 5–6, pp. 594–621.

Ballings, M., D. van den Poel, N. Hespeels, and R. Gryp (2015): "Evaluating multiple classifiers for stock price direction prediction." *Expert Systems with Applications*, Vol. 42, No. 20, pp. 7046–7056.

Bontempi, G., S. Taieb, and Y. Le Borgne (2012): "Machine learning strategies for time series forecasting." *Lecture Notes in Business Information Processing*, Vol. 138, No. 1, pp. 62–77.

Booth, A., E. Gerding and F. McGroarty (2014): "Automated trading with performance weighted random forests and seasonality." *Expert Systems with Applications*, Vol. 41, No. 8, pp. 3651–3661.

Cao, L. and F. Tay (2001): "Financial forecasting using support vector machines." *Neural Computing & Applications*, Vol. 10, No. 2, pp. 184–192.

Cao, L., F. Tay and F. Hock (2003): "Support vector machine with adaptive parameters in financial time series forecasting." *IEEE Transactions on Neural Networks*, Vol. 14, No. 6, pp. 1506–1518.

Cervelló-Royo, R., F. Guijarro, and K. Michniuk (2015): "Stock market trading rule based on pattern recognition and technical analysis: Forecasting the DJIA index with intraday data." *Expert Systems with Applications*, Vol. 42, No. 14, pp. 5963–5975.

Chang, P., C. Fan and J. Lin (2011): "Trend discovery in financial time series data using a case-based fuzzy decision tree." *Expert Systems with Applications*, Vol. 38, No. 5, pp. 6070–6080.

Kuan, C. and L. Tung (1995): "Forecasting exchange rates using feedforward and recurrent neural networks." *Journal of Applied Econometrics*, Vol. 10, No. 4, pp. 347–364.

Creamer, G. and Y. Freund (2007): "A boosting approach for automated trading." *Journal of Trading*, Vol. 2, No. 3, pp. 84–96.

Creamer, G. and Y. Freund (2010): "Automated trading with boosting and expert weighting." *Quantitative Finance*, Vol. 10, No. 4, pp. 401–420.

Creamer, G., Y. Ren, Y. Sakamoto, and J. Nickerson (2016): "A textual analysis algorithm for the equity market: The European case." *Journal of Investing*, Vol. 25, No. 3, pp. 105–116.

Dixon, M., D. Klabjan, and J. Bang (2016): "Classification-based financial markets prediction using deep neural networks." *Algorithmic Finance*, forthcoming (2017). Available at SSRN: https://ssrn.com/abstract=2756331.

Dunis, C., and M. Williams (2002): "Modelling and trading the euro/US dollar exchange rate: Do neural network models perform better?" *Journal of Derivatives & Hedge Funds*, Vol. 8, No. 3, pp. 211–239.

Feuerriegel, S. and H. Prendinger (2016): "News-based trading strategies." *Decision Support Systems*, Vol. 90, pp. 65–74.

Hsu, S., J. Hsieh, T. Chih, and K. Hsu (2009): "A two-stage architecture for stock price forecasting by integrating self-organizing map and support vector regression." *Expert Systems with Applications*, Vol. 36, No. 4, pp. 7947–7951.

Huang, W., Y. Nakamori, and S. Wang (2005): "Forecasting stock market movement direction with support vector machine." *Computers & Operations Research*, Vol. 32, No. 10, pp. 2513–2522.

Kara, Y., M. Boyacioglu, and O. Baykan (2011): "Predicting direction of stock price index movement using artificial neural networks and support vector machines: The sample of the Istanbul Stock Exchange." *Expert Systems with Applications*, Vol. 38, No. 5, pp. 5311–5319.

Kim, K. (2003): "Financial time series forecasting using support vector machines." *Neurocomputing*, Vol. 55, No. 1, pp. 307–319.

Krauss, C., X. Do, and N. Huck (2017): "Deep neural networks, gradient-boosted trees, random forests: Statistical arbitrage on the S&P 500." *European Journal of Operational Research*, Vol. 259, No. 2, pp. 689–702.

Laborda, R. and J. Laborda (2017): "Can tree-structured classifiers add value to the investor?" *Finance Research Letters*, Vol. 22 (August), pp. 211–226.

Nakamura, E. (2005): "Inflation forecasting using a neural network." *Economics Letters*, Vol. 86, No. 3, pp. 373–378.

Olson, D. and C. Mossman (2003): "Neural network forecasts of Canadian stock returns using accounting ratios." *International Journal of Forecasting*, Vol. 19, No. 3, pp. 453–465.

Patel, J., S. Sha, P. Thakkar, and K. Kotecha (2015): "Predicting stock and stock price index movement using trend deterministic data preparation and machine learning techniques." *Expert Systems with Applications*, Vol. 42, No. 1, pp. 259–268.

Patel, J., S. Sha, P. Thakkar, and K. Kotecha (2015): "Predicting stock market index using fusion of machine learning techniques." *Expert Systems with Applications*, Vol. 42, No. 4, pp. 2162–2172.

Qin, Q., Q. Wang, J. Li, and S. Shuzhi (2013): "Linear and nonlinear trading models with gradient boosted random forests and application to Singapore Stock Market." *Journal of Intelligent Learning Systems and Applications*, Vol. 5, No. 1, pp. 1–10.

Sorensen, E., K. Miller, and C. Ooi (2000): "The decision tree approach to stock selection." *Journal of Portfolio Management*, Vol. 27, No. 1, pp. 42–52.

Theofilatos, K., S. Likothanassis, and A. Karathanasopoulos (2012): "Modeling and trading the EUR/USD exchange rate using machine learning techniques." *Engineering, Technology & Applied Science Research*, Vol. 2, No. 5, pp. 269–272.

Trafalis, T. and H. Ince (2000): "Support vector machine for regression and applications to financial forecasting." *Neural Networks*, Vol. 6, No. 1, pp. 348–353.

Trippi, R. and D. DeSieno (1992): "Trading equity index futures with a neural network." *Journal of Portfolio Management*, Vol. 19, No. 1, pp. 27–33.

Tsai, C. and S. Wang (2009): "Stock price forecasting by hybrid machine learning techniques." *Proceedings of the International Multi-Conference of Engineers and Computer Scientists*, Vol. 1, No. 1, pp. 755–760.

Tsai, C., Y. Lin, D. Yen, and Y. Chen (2011): "Predicting stock returns by classifier ensembles." *Applied Soft Computing*, Vol. 11, No. 2, pp. 2452–2459.

Wang, J. and S. Chan (2006): "Stock market trading rule discovery using two-layer bias decision tree." *Expert Systems with Applications*, Vol. 30, No. 4, pp. 605–611.

Wang, Q., J. Li, Q. Qin, and S. Ge (2011): "Linear, adaptive and nonlinear trading models for Singapore Stock Market with random forests." Proceedings of the 9th IEEE International Conference on Control and Automation, pp. 726–731.

Wei, P. and N. Wang (2016): "Wikipedia and stock return: Wikipedia usage pattern helps to predict the individual stock movement." Proceedings of the 25th International Conference Companion on World Wide Web, Vol. 1, pp. 591–594.

Żbikowski, K. (2015): "Using volume weighted support vector machines with walk forward testing and feature selection for the purpose of creating stock trading strategy." *Expert Systems with Applications*, Vol. 42, No. 4, pp. 1797–1805.

Zhang, G., B. Patuwo, and M. Hu (1998): "Forecasting with artificial neural networks: The state of the art." *International Journal of Forecasting*, Vol. 14, No. 1, pp. 35–62.

Zhu, M., D. Philpotts and M. Stevenson (2012): "The benefits of tree-based models for stock selection." *Journal of Asset Management*, Vol. 13, No. 6, pp. 437–448.

Zhu, M., D. Philpotts, R. Sparks, and J. Stevenson, Maxwell (2011): "A hybrid approach to combining CART and logistic regression for stock ranking." *Journal of Portfolio Management*, Vol. 38, No. 1, pp. 100–109.

CHAPTER 4

Sample Weights

4.1 MOTIVATION

Chapter 3 presented several new methods for labeling financial observations. We introduced two novel concepts, the triple-barrier method and meta-labeling, and explained how they are useful in financial applications, including quantamental investment strategies. In this chapter you will learn how to use sample weights to address another problem ubiquitous in financial applications, namely that observations are not generated by independent and identically distributed (IID) processes. Most of the ML literature is based on the IID assumption, and one reason many ML applications fail in finance is because those assumptions are unrealistic in the case of financial time series.

4.2 OVERLAPPING OUTCOMES

In Chapter 3 we assigned a label y_i to an observed feature X_i, where y_i was a function of price bars that occurred over an interval $\left[t_{i,0}, t_{i,1}\right]$. When $t_{i,1} > t_{j,0}$ and $i < j$, then y_i and y_j will both depend on a common return $r_{t_{j,0},\min\left\{t_{i,1},t_{j,1}\right\}}$, that is, the return over the interval $\left[t_{j,0}, \min\left\{t_{i,1}, t_{j,1}\right\}\right]$. The implication is that the series of labels, $\left\{y_i\right\}_{i=1,\ldots,I}$, are not IID whenever there is an overlap between any two consecutive outcomes, $\exists i \left| t_{i,1} > t_{i+1,0} \right.$.

Suppose that we circumvent this problem by restricting the bet horizon to $t_{i,1} \leq t_{i+1,0}$. In this case there is no overlap, because every feature outcome is determined before or at the onset of the next observed feature. That would lead to coarse models where the features' sampling frequency would be limited by the horizon used to determine the outcome. On one hand, if we wished to investigate outcomes that lasted a month, features would have to be sampled with a frequency up to monthly. On the other hand, if we increased the sampling frequency to let's say daily, we would be

forced to reduce the outcome's horizon to one day. Furthermore, if we wished to apply a path-dependent labeling technique, like the triple-barrier method, the sampling frequency would be subordinated to the first barrier's touch. No matter what you do, restricting the outcome's horizon to eliminate overlaps is a terrible solution. We must allow $t_{i,1} > t_{i+1,0}$, which brings us back to the problem of overlapping outcomes described earlier.

This situation is characteristic of financial applications. Most non-financial ML researchers can assume that observations are drawn from IID processes. For example, you can obtain blood samples from a large number of patients, and measure their cholesterol. Of course, various underlying common factors will shift the mean and standard deviation of the cholesterol distribution, but the samples are still independent: There is one observation per subject. Suppose you take those blood samples, and someone in your laboratory spills blood from each tube into the following nine tubes to their right. That is, tube 10 contains blood for patient 10, but also blood from patients 1 through 9. Tube 11 contains blood from patient 11, but also blood from patients 2 through 10, and so on. Now you need to determine the features predictive of high cholesterol (diet, exercise, age, etc.), without knowing for sure the cholesterol level of each patient. That is the equivalent challenge that we face in financial ML, with the additional handicap that the spillage pattern is non-deterministic and unknown. Finance is not a plug-and-play subject as it relates to ML applications. Anyone who tells you otherwise will waste your time and money.

There are several ways to attack the problem of non-IID labels, and in this chapter we will address it by designing sampling and weighting schemes that correct for the undue influence of overlapping outcomes.

4.3 NUMBER OF CONCURRENT LABELS

Two labels y_i and y_j are concurrent at t when both are a function of at least one common return, $r_{t-1,t} = \frac{p_t}{p_{t-1}} - 1$. The overlap does not need to be perfect, in the sense of both labels spanning the same time interval. In this section we are going to compute the number of labels that are a function of a given return, $r_{t-1,t}$. First, for each time point $t = 1, \ldots, T$, we form a binary array, $\left\{1_{t,i}\right\}_{i=1,\ldots,I}$, where $1_{t,i} \in \{0, 1\}$. Variable $1_{t,i} = 1$ if and only if $\left[t_{i,0}, t_{i,1}\right]$ overlaps with $[t-1, t]$ and $1_{t,i} = 0$ otherwise. Recall that the labels' spans $\left\{\left[t_{i,0}, t_{i,1}\right]\right\}_{i=1,\ldots,I}$ are defined by the `t1` object introduced in Chapter 3. Second, we compute the number of labels concurrent at t, $c_t = \sum_{i=1}^{I} 1_{t,i}$. Snippet 4.1 illustrates an implementation of this logic.

SNIPPET 4.1 ESTIMATING THE UNIQUENESS OF A LABEL

```
def mpNumCoEvents(closeIdx,t1,molecule):
    '''
    Compute the number of concurrent events per bar.
    +molecule[0] is the date of the first event on which the weight will be computed
    +molecule[-1] is the date of the last event on which the weight will be computed
```

```
    Any event that starts before t1[molecule].max() impacts the count.
    '''
    #1) find events that span the period [molecule[0],molecule[-1]]
    t1=t1.fillna(closeIdx[-1]) # unclosed events still must impact other weights
    t1=t1[t1>=molecule[0]] # events that end at or after molecule[0]
    t1=t1.loc[:t1[molecule].max()] # events that start at or before t1[molecule].max()
    #2) count events spanning a bar
    iloc=closeIdx.searchsorted(np.array([t1.index[0],t1.max()]))
    count=pd.Series(0,index=closeIdx[iloc[0]:iloc[1]+1])
    for tIn,tOut in t1.iteritems():count.loc[tIn:tOut]+=1.
    return count.loc[molecule[0]:t1[molecule].max()]
```

4.4 AVERAGE UNIQUENESS OF A LABEL

In this section we are going to estimate a label's uniqueness (non-overlap) as its average uniqueness over its lifespan. First, the uniqueness of a label i at time t is $u_{t,i} = 1_{t,i}c_t^{-1}$. Second, the average uniqueness of label i is the average $u_{t,i}$ over the label's lifespan, $\bar{u}_i = \left(\sum_{t=1}^{T} u_{t,i}\right)\left(\sum_{t=1}^{T} 1_{t,i}\right)^{-1}$. This average uniqueness can also be interpreted as the reciprocal of the harmonic average of c_t over the event's lifespan. Figure 4.1 plots the histogram of uniqueness values derived from an object `t1`. Snippet 4.2 implements this calculation.

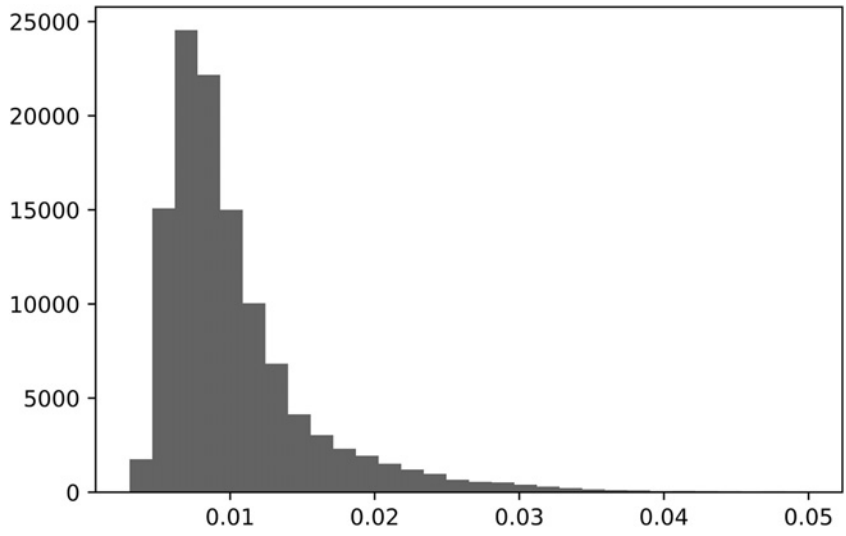


FIGURE 4.1 Histogram of uniqueness values

SNIPPET 4.2 ESTIMATING THE AVERAGE UNIQUENESS OF A LABEL

```
def mpSampleTW(t1,numCoEvents,molecule):
    # Derive average uniqueness over the event's lifespan
    wght=pd.Series(index=molecule)
    for tIn,tOut in t1.loc[wght.index].iteritems():
        wght.loc[tIn]=(1./numCoEvents.loc[tIn:tOut]).mean()
    return wght
#-----------------------------------------------
numCoEvents=mpPandasObj(mpNumCoEvents,('molecule',events.index),numThreads, \
        closeIdx=close.index,t1=events['t1'])
numCoEvents=numCoEvents.loc[~numCoEvents.index.duplicated(keep='last')]
numCoEvents=numCoEvents.reindex(close.index).fillna(0)
out['tW']=mpPandasObj(mpSampleTW,('molecule',events.index),numThreads, \
        t1=events['t1'],numCoEvents=numCoEvents)
```

Note that we are making use again of the function `mpPandasObj`, which speeds up calculations via multiprocessing (see Chapter 20). Computing the average uniqueness associated with label i, $\bar{u}_i$, requires information that is not available until a future time, `events['t1']`. This is not a problem, because $\{\bar{u}_i\}_{i=1,\ldots,I}$ are used on the training set in combination with label information, and not on the testing set. These $\{\bar{u}_i\}_{i=1,\ldots,I}$ are not used for forecasting the label, hence there is no information leakage. This procedure allows us to assign a uniqueness score between 0 and 1 for each observed feature, in terms of non-overlapping outcomes.

4.5 BAGGING CLASSIFIERS AND UNIQUENESS

The probability of not selecting a particular item i after I draws with replacement on a set of I items is $(1 - I^{-1})^I$. As the sample size grows, that probability converges to the asymptotic value $\lim_{I\to\infty}(1 - I^{-1})^I = e^{-1}$. That means that the number of unique observations drawn is expected to be $(1 - e^{-1}) \approx \frac{2}{3}$.

Suppose that the maximum number of non-overlapping outcomes is $K \leq I$. Following the same argument, the probability of not selecting a particular item i after I draws with replacement on a set of I items is $(1 - K^{-1})^I$. As the sample size grows, that probability can be approximated as $(1 - I^{-1})^{I\frac{K}{I}} \approx e^{-\frac{K}{I}}$. That means that the number of unique observations drawn is expected to be $1 - e^{-\frac{K}{I}} \leq 1 - e^{-1}$. The implication is that incorrectly assuming IID draws leads to oversampling.

When sampling with replacement (bootstrap) on observations with $I^{-1}\sum_{i=1}^{I}\bar{u}_i \ll 1$, it becomes increasingly likely that in-bag observations will be (1) redundant to each other, and (2) very similar to out-of-bag observations.

Redundancy of draws makes the bootstrap inefficient (see Chapter 6). For example, in the case of a random forest, all trees in the forest will essentially be very similar copies of a single overfit decision tree. And because the random sampling makes out-of-bag examples very similar to the in-bag ones, out-of-bag accuracy will be grossly inflated. We will address this second issue in Chapter 7, when we study cross-validation under non-IID observations. For the moment, let us concentrate on the first issue, namely bagging under observations where $I^{-1}\sum_{i=1}^{I}\bar{u}_i \ll 1$.

A first solution is to drop overlapping outcomes before performing the bootstrap. Because overlaps are not perfect, dropping an observation just because there is a partial overlap will result in an extreme loss of information. I do not advise you to follow this solution.

A second and better solution is to utilize the average uniqueness, $I^{-1}\sum_{i=1}^{I}\bar{u}_i$, to reduce the undue influence of outcomes that contain redundant information. Accordingly, we could sample only a fraction `out['tW'].mean()` of the observations, or a small multiple of that. In sklearn, the `sklearn.ensemble.BaggingClassifier` class accepts an argument `max_samples`, which can be set to `max_samples=out['tW'].mean()`. In this way, we enforce that the in-bag observations are not sampled at a frequency much higher than their uniqueness. Random forests do not offer that `max_samples` functionality, however, a solution is to bag a large number of decision trees. We will discuss this solution further in Chapter 6.

4.5.1 Sequential Bootstrap

A third and better solution is to perform a sequential bootstrap, where draws are made according to a changing probability that controls for redundancy. Rao et al. [1997] propose sequential resampling with replacement until K distinct original observations appear. Although interesting, their scheme does not fully apply to our financial problem. In the following sections we introduce an alternative method that addresses directly the problem of overlapping outcomes.

First, an observation X_i is drawn from a uniform distribution, $i \sim U[1, I]$, that is, the probability of drawing any particular value i is originally $\delta_i^{(1)} = I^{-1}$. For the second draw, we wish to reduce the probability of drawing an observation X_j with a highly overlapping outcome. Remember, a bootstrap allows sampling with repetition, so it is still possible to draw X_i again, but we wish to reduce its likelihood, since there is an overlap (in fact, a perfect overlap) between X_i and itself. Let us denote as φ the sequence of draws so far, which may include repetitions. Until now, we know that $\varphi^{(1)} = \{i\}$. The uniqueness of j at time t is $u_{t,j}^{(2)} = 1_{t,j}\left(1 + \sum_{k\in\varphi^{(1)}} 1_{t,k}\right)^{-1}$, as that is the uniqueness that results from adding alternative j's to the existing sequence of draws $\varphi^{(1)}$. The average uniqueness of j is the average $u_{t,j}^{(2)}$ over j's lifespan,

$\bar{u}_j^{(2)} = \left(\sum_{t=1}^{T} u_{t,j}\right)\left(\sum_{t=1}^{T} 1_{t,j}\right)^{-1}$. We can now make a second draw based on the updated probabilities $\left\{\delta_j^{(2)}\right\}_{j=1,\ldots,I}$,

$$\delta_j^{(2)} = \bar{u}_j^{(2)}\left(\sum_{k=1}^{I} \bar{u}_k^{(2)}\right)^{-1}$$

where $\left\{\delta_j^{(2)}\right\}_{j=1,\ldots,I}$ are scaled to add up to 1, $\sum_{j=1}^{I} \delta_j^{(2)} = 1$. We can now do a second draw, update $\varphi^{(2)}$ and re-evaluate $\left\{\delta_j^{(3)}\right\}_{j=1,\ldots,I}$. The process is repeated until I draws have taken place. This sequential bootstrap scheme has the advantage that overlaps (even repetitions) are still possible, but decreasingly likely. The sequential bootstrap sample will be much closer to IID than samples drawn from the standard bootstrap method. This can be verified by measuring an increase in $I^{-1}\sum_{i=1}^{I} \bar{u}_i$, relative to the standard bootstrap method.

4.5.2 Implementation of Sequential Bootstrap

Snippet 4.3 derives an indicator matrix from two arguments: the index of bars (`barIx`), and the pandas Series `t1`, which we used multiple times in Chapter 3. As a reminder, `t1` is defined by an index containing the time at which the features are observed, and a values array containing the time at which the label is determined. The output of this function is a binary matrix indicating what (price) bars influence the label for each observation.

SNIPPET 4.3 BUILD AN INDICATOR MATRIX

```
import pandas as pd,numpy as np
#-------------------------------------------------
def getIndMatrix(barIx,t1):
    # Get indicator matrix
    indM=pd.DataFrame(0,index=barIx,columns=range(t1.shape[0]))
    for i,(t0,t1) in enumerate(t1.iteritems()):indM.loc[t0:t1,i]=1.
    return indM
```

Snippet 4.4 returns the average uniqueness of each observed feature. The input is the indicator matrix built by `getIndMatrix`.

SNIPPET 4.4 COMPUTE AVERAGE UNIQUENESS

```
def getAvgUniqueness(indM):
    # Average uniqueness from indicator matrix
    c=indM.sum(axis=1) # concurrency
    u=indM.div(c,axis=0) # uniqueness
    avgU=u[u>0].mean() # average uniqueness
    return avgU
```

Snippet 4.5 gives us the index of the features sampled by sequential bootstrap. The inputs are the indicator matrix (`indM`) and an optional sample length (`sLength`), with a default value of as many draws as rows in `indM`.

SNIPPET 4.5 RETURN SAMPLE FROM SEQUENTIAL BOOTSTRAP

```
def seqBootstrap(indM,sLength=None):
    # Generate a sample via sequential bootstrap
    if sLength is None:sLength=indM.shape[1]
    phi=[]
    while len(phi)<sLength:
        avgU=pd.Series()
        for i in indM:
            indM_=indM[phi+[i]] # reduce indM
            avgU.loc[i]=getAvgUniqueness(indM_).iloc[-1]
        prob=avgU/avgU.sum() # draw prob
        phi+=[np.random.choice(indM.columns,p=prob)]
    return phi
```

4.5.3 A Numerical Example

Consider a set of labels $\{y_i\}_{i=1,2,3}$, where label y_1 is a function of return $r_{0,3}$, label y_2 is a function of return $r_{2,4}$ and label y_3 is a function of return $r_{4,6}$. The outcomes' overlaps are characterized by this indicator matrix $\{1_{t,i}\}$,

$$\{1_{t,i}\} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

The procedure starts with $\varphi^{(0)} = \emptyset$, and a uniform distribution of probability, $\delta_i = \frac{1}{3}$, $\forall i = 1, 2, 3$. Suppose that we randomly draw a number from $\{1, 2, 3\}$, and 2 is selected. Before we make a second draw on $\{1, 2, 3\}$ (remember, a bootstrap samples with repetition), we need to adjust the probabilities. The set of observations drawn so far is $\varphi^{(1)} = \{2\}$. The average uniqueness for the first feature is $\bar{u}_1^{(2)} = \left(1 + 1 + \frac{1}{2}\right)\frac{1}{3} = \frac{5}{6} < 1$, and for the second feature is $\bar{u}_2^{(2)} = \left(\frac{1}{2} + \frac{1}{2}\right)\frac{1}{2} = \frac{1}{2} < 1$. The probabilities for the second draw are $\delta^{(2)} = \left\{\frac{5}{14}, \frac{3}{14}, \frac{6}{14}\right\}$. Two points are worth mentioning: (1) The lowest probability goes to the feature that was picked in the first draw, as that would exhibit the highest overlap; and (2) among the two possible draws outside $\varphi^{(1)}$, the greater probability goes to $\delta_3^{(2)}$, as that is the label with no overlap to $\varphi^{(1)}$. Suppose that the second draw selects number 3. We leave as an exercise the update of the probabilities $\delta^{(3)}$ for the third and final draw. Snippet 4.6 runs a sequential bootstrap on the $\{1_{t,i}\}$ indicator matrix in this example.

SNIPPET 4.6 EXAMPLE OF SEQUENTIAL BOOTSTRAP

```
def main():
    t1=pd.Series([2,3,5],index=[0,2,4]) # t0,t1 for each feature obs
    barIx=range(t1.max()+1) # index of bars
    indM=getIndMatrix(barIx,t1)
    phi=np.random.choice(indM.columns,size=indM.shape[1])
    print phi
    print 'Standard uniqueness:',getAvgUniqueness(indM[phi]).mean()
    phi=seqBootstrap(indM)
    print phi
    print 'Sequential uniqueness:',getAvgUniqueness(indM[phi]).mean()
    return
```

4.5.4 Monte Carlo Experiments

We can evaluate the efficiency of the sequential bootstrap algorithm through experimental methods. Snippet 4.7 lists the function that generates a random `t1` series for a number of observations `numObs` (I). Each observation is made at a random number, drawn from a uniform distribution, with boundaries 0 and `numBars`, where `numBars` is the number of bars (T). The number of bars spanned by the observation is determined by drawing a random number from a uniform distribution with boundaries 0 and `maxH`.

SNIPPET 4.7 GENERATING A RANDOM T1 SERIES

```
def getRndT1(numObs,numBars,maxH):
    # random t1 Series
    t1=pd.Series()
```

```
    for i in xrange(numObs):
        ix=np.random.randint(0,numBars)
        val=ix+np.random.randint(1,maxH)
        t1.loc[ix]=val
    return t1.sort_index()
```

Snippet 4.8 takes that random `t1` series, and derives the implied indicator matrix, `indM`. This matrix is then subjected to two procedures. In the first one, we derive the average uniqueness from a standard bootstrap (random sampling with replacement). In the second one, we derive the average uniqueness by applying our sequential bootstrap algorithm. Results are reported as a dictionary.

SNIPPET 4.8 UNIQUENESS FROM STANDARD AND SEQUENTIAL BOOTSTRAPS

```
def auxMC(numObs,numBars,maxH):
    # Parallelized auxiliary function
    t1=getRndT1(numObs,numBars,maxH)
    barIx=range(t1.max()+1)
    indM=getIndMatrix(barIx,t1)
    phi=np.random.choice(indM.columns,size=indM.shape[1])
    stdU=getAvgUniqueness(indM[phi]).mean()
    phi=seqBootstrap(indM)
    seqU=getAvgUniqueness(indM[phi]).mean()
    return {'stdU':stdU,'seqU':seqU}
```

These operations have to be repeated over a large number of iterations. Snippet 4.9 implements this Monte Carlo using the multiprocessing techniques discussed in Chapter 20. For example, it will take about 6 hours for a 24-cores server to carry out a Monte Carlo of 1E6 iterations, where `numObs=10`, `numBars=100`, and `maxH=5`. Without parallelization, a similar Monte Carlo experiment would have taken about 6 days.

SNIPPET 4.9 MULTI-THREADED MONTE CARLO

```
import pandas as pd,numpy as np
from mpEngine import processJobs,processJobs_
#-------------------------------------
def mainMC(numObs=10,numBars=100,maxH=5,numIters=1E6,numThreads=24):
    # Monte Carlo experiments
    jobs=[]
    for i in xrange(int(numIters)):
        job={'func':auxMC,'numObs':numObs,'numBars':numBars,'maxH':maxH}
        jobs.append(job)
```

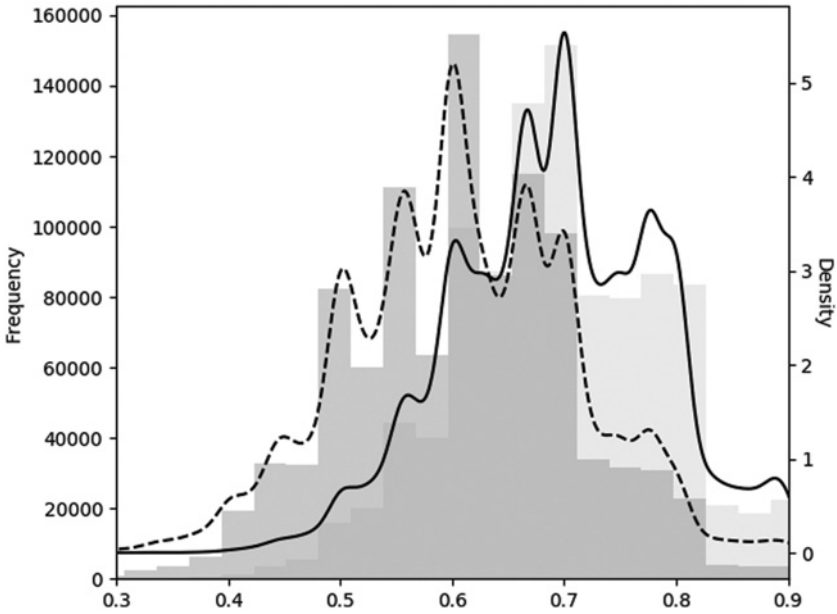


FIGURE 4.2 Monte Carlo experiment of standard vs. sequential bootstraps

```
if numThreads==1:out=processJobs_(jobs)
else:out=processJobs(jobs,numThreads=numThreads)
print pd.DataFrame(out).describe()
return
```

Figure 4.2 plots the histogram of the uniqueness from standard bootstrapped samples (left) and the sequentially bootstrapped samples (right). The median of the average uniqueness for the standard method is 0.6, and the median of the average uniqueness for the sequential method is 0.7. An ANOVA test on the difference of means returns a vanishingly small probability. Statistically speaking, samples from the sequential bootstrap method have an expected uniqueness that exceeds that of the standard bootstrap method, at any reasonable confidence level.

4.6 RETURN ATTRIBUTION

In the previous section we learned a method to bootstrap samples closer to IID. In this section we will introduce a method to weight those samples for the purpose of training an ML algorithm. Highly overlapping outcomes would have disproportionate weights if considered equal to non-overlapping outcomes. At the same time, labels associated with large absolute returns should be given more importance than labels

with negligible absolute returns. In short, we need to weight observations by some function of both uniqueness and absolute return.

When labels are a function of the return sign ($\{-1, 1\}$ for standard labeling or $\{0, 1\}$ for meta-labeling), the sample weights can be defined in terms of the sum of the attributed returns over the event's lifespan, $\left[t_{i,0}, t_{i,1}\right]$,

$$\tilde{w}_i = \left| \sum_{t=t_{i,0}}^{t_{i,1}} \frac{r_{t-1,t}}{c_t} \right|$$

$$w_i = \tilde{w}_i I \left(\sum_{j=1}^{I} \tilde{w}_j \right)^{-1}$$

hence $\sum_{i=1}^{I} w_i = I$. We have scaled these weights to add up to I, since libraries (including sklearn) usually define algorithmic parameters assuming a default weight of 1.

The rationale for this method is that we wish to weight an observation as a function of the absolute log returns that can be attributed uniquely to it. However, this method will not work if there is a "neutral" (return below threshold) case. For that case, lower returns should be assigned higher weights, not the reciprocal. The "neutral" case is unnecessary, as it can be implied by a "−1" or "1" prediction with low confidence. This is one of several reasons I would generally advise you to drop "neutral" cases. Snippet 4.10 implements this method.

SNIPPET 4.10 DETERMINATION OF SAMPLE WEIGHT BY ABSOLUTE RETURN ATTRIBUTION

```
def mpSampleW(t1,numCoEvents,close,molecule):
    # Derive sample weight by return attribution
    ret=np.log(close).diff() # log-returns, so that they are additive
    wght=pd.Series(index=molecule)
    for tIn,tOut in t1.loc[wght.index].iteritems():
        wght.loc[tIn]=(ret.loc[tIn:tOut]/numCoEvents.loc[tIn:tOut]).sum()
    return wght.abs()
#————————————————————————————————————
out['w']=mpPandasObj(mpSampleW,('molecule',events.index),numThreads, \
        t1=events['t1'],numCoEvents=numCoEvents,close=close)
out['w']*=out.shape[0]/out['w'].sum()
```

4.7 TIME DECAY

Markets are adaptive systems (Lo [2017]). As markets evolve, older examples are less relevant than the newer ones. Consequently, we would typically like sample weights to decay as new observations arrive. Let $d[x] \geq 0, \forall x \in \left[0, \sum_{i=1}^{I} \bar{u}_i\right]$ be the time-decay factors that will multiply the sample weights derived in the previous section. The final weight has no decay, $d\left[\sum_{i=1}^{I} \bar{u}_i\right] = 1$, and all other weights will be adjusted relative to that. Let $c \in (-1, 1]$ be a user-defined parameter that determines the decay function as follows: For $c \in [0, 1]$, then $d[1] = c$, with linear decay; for $c \in (-1, 0)$, then $d\left[-c\sum_{i=1}^{I} \bar{u}_i\right] = 0$, with linear decay between $\left[-c\sum_{i=1}^{I} \bar{u}_i, \sum_{i=1}^{I} \bar{u}_i\right]$ and $d[x] = 0 \; \forall x \leq -c\sum_{i=1}^{I} \bar{u}_i$. For a linear piecewise function $d = \max\{0, a + bx\}$, such requirements are met by the following boundary conditions:

1. $d = a + b\sum_{i=1}^{I} \bar{u}_i = 1 \Rightarrow a = 1 - b\sum_{i=1}^{I} \bar{u}_i$.
2. Contingent on c:
   - **(a)** $d = a + b0 = c \Rightarrow b = (1 - c)\left(\sum_{i=1}^{I} \bar{u}_i\right)^{-1}, \forall c \in [0, 1]$
   - **(b)** $d = a - bc\sum_{i=1}^{I} \bar{u}_i = 0 \Rightarrow b = \left[(c + 1)\sum_{i=1}^{I} \bar{u}_i\right]^{-1}, \forall c \in (-1, 0)$

Snippet 4.11 implements this form of time-decay factors. Note that time is not meant to be chronological. In this implementation, decay takes place according to cumulative uniqueness, $x \in \left[0, \sum_{i=1}^{I} \bar{u}_i\right]$, because a chronological decay would reduce weights too fast in the presence of redundant observations.

SNIPPET 4.11 IMPLEMENTATION OF TIME-DECAY FACTORS

```
def getTimeDecay(tW,clfLastW=1.):
    # apply piecewise-linear decay to observed uniqueness (tW)
    # newest observation gets weight=1, oldest observation gets weight=clfLastW
    clfW=tW.sort_index().cumsum()
    if clfLastW>=0:slope=(1.-clfLastW)/clfW.iloc[-1]
    else:slope=1./((clfLastW+1)*clfW.iloc[-1])
    const=1.-slope*clfW.iloc[-1]
    clfW=const+slope*clfW
    clfW[clfW<0]=0
    print const,slope
    return clfW
```

It is worth discussing a few interesting cases:

- $c = 1$ means that there is no time decay.
- $0 < c < 1$ means that weights decay linearly over time, but every observation still receives a strictly positive weight, regardless of how old.

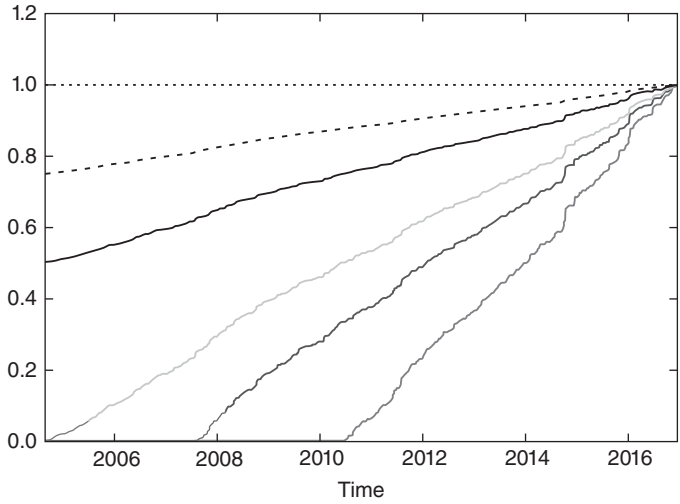


FIGURE 4.3 Piecewise-linear time-decay factors

- $c = 0$ means that weights converge linearly to zero, as they become older.
- $c < 0$ means that the oldest portion cT of the observations receive zero weight (i.e., they are erased from memory).

Figure 4.3 shows the decayed weights, `out['w']*df`, after applying the decay factors for $c \in \{1, .75, .5, 0, -.25, -.5\}$. Although not necessarily practical, the procedure allows the possibility of generating weights that increase as they get older, by setting $c > 1$.

4.8 CLASS WEIGHTS

In addition to sample weights, it is often useful to apply class weights. Class weights are weights that correct for underrepresented labels. This is particularly critical in classification problems where the most important classes have rare occurrences (King and Zeng [2001]). For example, suppose that you wish to predict liquidity crisis, like the flash crash of May 6, 2010. These events are rare relative to the millions of observations that take place in between them. Unless we assign higher weights to the samples associated with those rare labels, the ML algorithm will maximize the accuracy of the most common labels, and flash crashes will be deemed to be outliers rather than rare events.

ML libraries typically implement functionality to handle class weights. For example, sklearn penalizes errors in samples of `class[j]`, *j=1,…,J*, with weighting `class_weight[j]` rather than 1. Accordingly, higher class weights on label *j* will force the algorithm to achieve higher accuracy on *j*. When class weights do not add up to *J*, the effect is equivalent to changing the regularization parameter of the classifier.

In financial applications, the standard labels of a classification algorithm are $\{-1, 1\}$, where the zero (or neutral) case will be implied by a prediction with probability only slightly above 0.5 and below some neutral threshold. There is no reason for favoring accuracy of one class over the other, and as such a good default is to assign `class_weight='balanced'`. This choice re-weights observations so as to simulate that all classes appeared with equal frequency. In the context of bagging classifiers, you may want to consider the argument `class_weight='balanced_subsample'`, which means that `class_weight='balanced'` will be applied to the in-bag bootstrapped samples, rather than to the entire dataset. For full details, it is helpful to read the source code implementing `class_weight` in sklearn. Please also be aware of this reported bug: https://github.com/scikit-learn/scikit-learn/issues/4324.

EXERCISES

4.1 In Chapter 3, we denoted as `t1` a pandas series of timestamps where the first barrier was touched, and the index was the timestamp of the observation. This was the output of the `getEvents` function.

- **(a)** Compute a `t1` series on dollar bars derived from E-mini S&P 500 futures tick data.
- **(b)** Apply the function `mpNumCoEvents` to compute the number of overlapping outcomes at each point in time.
- **(c)** Plot the time series of the number of concurrent labels on the primary axis, and the time series of exponentially weighted moving standard deviation of returns on the secondary axis.
- **(d)** Produce a scatterplot of the number of concurrent labels (x-axis) and the exponentially weighted moving standard deviation of returns (y-axis). Can you appreciate a relationship?

4.2 Using the function `mpSampleTW`, compute the average uniqueness of each label. What is the first-order serial correlation, AR(1), of this time series? Is it statistically significant? Why?

4.3 Fit a random forest to a financial dataset where $I^{-1}\sum_{i=1}^{I}\bar{u}_i \ll 1$.

- **(a)** What is the mean out-of-bag accuracy?
- **(b)** What is the mean accuracy of k-fold cross-validation (without shuffling) on the same dataset?
- **(c)** Why is out-of-bag accuracy so much higher than cross-validation accuracy? Which one is more correct / less biased? What is the source of this bias?

4.4 Modify the code in Section 4.7 to apply an exponential time-decay factor.

4.5 Consider you have applied meta-labels to events determined by a trend-following model. Suppose that two thirds of the labels are 0 and one third of the labels are 1.

- **(a)** What happens if you fit a classifier without balancing class weights?

(b) A label 1 means a true positive, and a label 0 means a false positive. By applying balanced class weights, we are forcing the classifier to pay more attention to the true positives, and less attention to the false positives. Why does that make sense?

(c) What is the distribution of the predicted labels, before and after applying balanced class weights?

4.6 Update the draw probabilities for the final draw in Section 4.5.3.

4.7 In Section 4.5.3, suppose that number 2 is picked again in the second draw. What would be the updated probabilities for the third draw?

REFERENCES

Rao, C., P. Pathak and V. Koltchinskii (1997): "Bootstrap by sequential resampling." *Journal of Statistical Planning and Inference*, Vol. 64, No. 2, pp. 257–281.

King, G. and L. Zeng (2001): "Logistic Regression in Rare Events Data." Working paper, Harvard University. Available at https://gking.harvard.edu/files/0s.pdf.

Lo, A. (2017): *Adaptive Markets*, 1st ed. Princeton University Press.

BIBLIOGRAPHY

Sample weighting is a common topic in the ML learning literature. However the practical problems discussed in this chapter are characteristic of investment applications, for which the academic literature is extremely scarce. Below are some publications that tangentially touch some of the issues discussed in this chapter.

Efron, B. (1979): "Bootstrap methods: Another look at the jackknife." *Annals of Statistics*, Vol. 7, pp. 1–26.

Efron, B. (1983): "Estimating the error rote of a prediction rule: Improvement on cross-validation." *Journal of the American Statistical Association*, Vol. 78, pp. 316–331.

Bickel, P. and D. Freedman (1981): "Some asymptotic theory for the bootstrap." *Annals of Statistics*, Vol. 9, pp. 1196–1217.

Gine, E. and J. Zinn (1990): "Bootstrapping general empirical measures." *Annals of Probability*, Vol. 18, pp. 851–869.

Hall, P. and E. Mammen (1994): "On general resampling algorithms and their performance in distribution estimation." *Annals of Statistics*, Vol. 24, pp. 2011–2030.

Mitra, S. and P. Pathak (1984): "The nature of simple random sampling." *Annals of Statistics*, Vol. 12, pp. 1536–1542.

Pathak, P. (1964): "Sufficiency in sampling theory." *Annals of Mathematical Statistics*, Vol. 35, pp. 795–808.

Pathak, P. (1964): "On inverse sampling with unequal probabilities." *Biometrika*, Vol. 51, pp. 185–193.

Praestgaard, J. and J. Wellner (1993): "Exchangeably weighted bootstraps of the general empirical process." *Annals of Probability*, Vol. 21, pp. 2053–2086.

Rao, C., P. Pathak and V. Koltchinskii (1997): "Bootstrap by sequential resampling." *Journal of Statistical Planning and Inference*, Vol. 64, No. 2, pp. 257–281.

CHAPTER 5

Fractionally Differentiated Features

5.1 MOTIVATION

It is known that, as a consequence of arbitrage forces, financial series exhibit low signal-to-noise ratios (López de Prado [2015]). To make matters worse, standard stationarity transformations, like integer differentiation, further reduce that signal by removing memory. Price series have memory, because every value is dependent upon a long history of previous levels. In contrast, integer differentiated series, like returns, have a memory cut-off, in the sense that history is disregarded entirely after a finite sample window. Once stationarity transformations have wiped out all memory from the data, statisticians resort to complex mathematical techniques to extract whatever residual signal remains. Not surprisingly, applying these complex techniques on memory-erased series likely leads to false discoveries. In this chapter we introduce a data transformation method that ensures the stationarity of the data while preserving as much memory as possible.

5.2 THE STATIONARITY VS. MEMORY DILEMMA

It is common in finance to find non-stationary time series. What makes these series non-stationary is the presence of memory, i.e., a long history of previous levels that shift the series' mean over time. In order to perform inferential analyses, researchers need to work with invariant processes, such as returns on prices (or changes in log-prices), changes in yield, or changes in volatility. These data transformations make the series stationary, at the expense of removing all memory from the original series (Alexander [2001], chapter 11). Although stationarity is a necessary property for inferential purposes, it is rarely the case in signal processing that we wish all memory to be erased, as that memory is the basis for the model's predictive power. For example, equilibrium (stationary) models need some memory to assess how far the

price process has drifted away from the long-term expected value in order to generate a forecast. The dilemma is that returns are stationary, however memory-less, and prices have memory, however they are non-stationary. The question arises: What is the minimum amount of differentiation that makes a price series stationary while preserving as much memory as possible? Accordingly, we would like to generalize the notion of returns to consider *stationary series where not all memory is erased.* Under this framework, returns are just one kind of (and in most cases suboptimal) price transformation among many other possibilities.

Part of the importance of cointegration methods is their ability to model series with memory. But why would the particular case of zero differentiation deliver best outcomes? Zero differentiation is as arbitrary as 1-step differentiation. There is a wide region between these two extremes (fully differentiated series on one hand, and zero differentiated series on the other) that can be explored through fractional differentiation for the purpose of developing a highly predictive ML model.

Supervised learning algorithms typically require stationary features. The reason is that we need to map a previously unseen (unlabeled) observation to a collection of labeled examples, and infer from them the label of that new observation. If the features are not stationary, we cannot map the new observation to a large number of known examples. But stationarity does not ensure predictive power. Stationarity is a necessary, non-sufficient condition for the high performance of an ML algorithm. The problem is, there is a trade-off between stationarity and memory. We can always make a series more stationary through differentiation, but it will be at the cost of erasing some memory, which will defeat the forecasting purpose of the ML algorithm. In this chapter, we will study one way to resolve this dilemma.

5.3 LITERATURE REVIEW

Virtually all the financial time series literature is based on the premise of making non-stationary series stationary through integer transformation (see Hamilton [1994] for an example). This raises two questions: (1) Why would integer 1 differentiation (like the one used for computing returns on log-prices) be optimal? (2) Is over-differentiation one reason why the literature has been so biased in favor of the efficient markets hypothesis?

The notion of fractional differentiation applied to the predictive time series analysis dates back at least to Hosking [1981]. In that paper, a family of ARIMA processes was generalized by permitting the degree of differencing to take fractional values. This was useful because fractionally differenced processes exhibit long-term persistence and antipersistence, hence enhancing the forecasting power compared to the standard ARIMA approach. In the same paper, Hosking states: "Apart from a passing reference by Granger (1978), fractional differencing does not appear to have been previously mentioned in connection with time series analysis."

After Hosking's paper, the literature on this subject has been surprisingly scarce, adding up to eight journal articles written by only nine authors: Hosking, Johansen, Nielsen, MacKinnon, Jensen, Jones, Popiel, Cavaliere, and Taylor. See the references for details. Most of those papers relate to technical matters, such as fast algorithms for

the calculation of fractional differentiation in continuous stochastic processes (e.g., Jensen and Nielsen [2014]).

Differentiating the stochastic process is a computationally expensive operation. In this chapter we will take a practical, alternative, and novel approach to recover stationarity: We will generalize the difference operator to non-integer steps.

5.4 THE METHOD

Consider the backshift operator, B, applied to a matrix of real-valued features $\{X_t\}$, where $B^k X_t = X_{t-k}$ for any integer $k \geq 0$. For example, $(1-B)^2 = 1 - 2B + B^2$, where $B^2 X_t = X_{t-2}$, so that $(1-B)^2 X_t = X_t - 2X_{t-1} + X_{t-2}$. Note that $(x+y)^n = \sum_{k=0}^{n} \binom{n}{k} x^k y^{n-k} = \sum_{k=0}^{n} \binom{n}{k} x^{n-k} y^k$, for n a positive integer. For a real number d, $(1+x)^d = \sum_{k=0}^{\infty} \binom{d}{k} x^k$, the binomial series.

In a fractional model, the exponent d is allowed to be a real number, with the following formal binomial series expansion:

$$\begin{aligned}(1-B)^d = \sum_{k=0}^{\infty} \binom{d}{k} (-B)^k &= \sum_{k=0}^{\infty} \frac{\prod_{i=0}^{k-1}(d-i)}{k!}(-B)^k \\ &= \sum_{k=0}^{\infty} (-B)^k \prod_{i=0}^{k-1} \frac{d-i}{k-i} \\ &= 1 - dB + \frac{d(d-1)}{2!}B^2 - \frac{d(d-1)(d-2)}{3!}B^3 + \cdots\end{aligned}$$

5.4.1 Long Memory

Let us see how a real (non-integer) positive d preserves memory. This arithmetic series consists of a dot product

$$\tilde{X}_t = \sum_{k=0}^{\infty} \omega_k X_{t-k}$$

with weights ω

$$\omega = \left\{1, -d, \frac{d(d-1)}{2!}, -\frac{d(d-1)(d-2)}{3!}, \ldots, (-1)^k \prod_{i=0}^{k-1} \frac{d-i}{k!}, \ldots\right\}$$

and values X

$$X = \left\{X_t, X_{t-1}, X_{t-2}, X_{t-3}, \ldots, X_{t-k}, \ldots\right\}$$

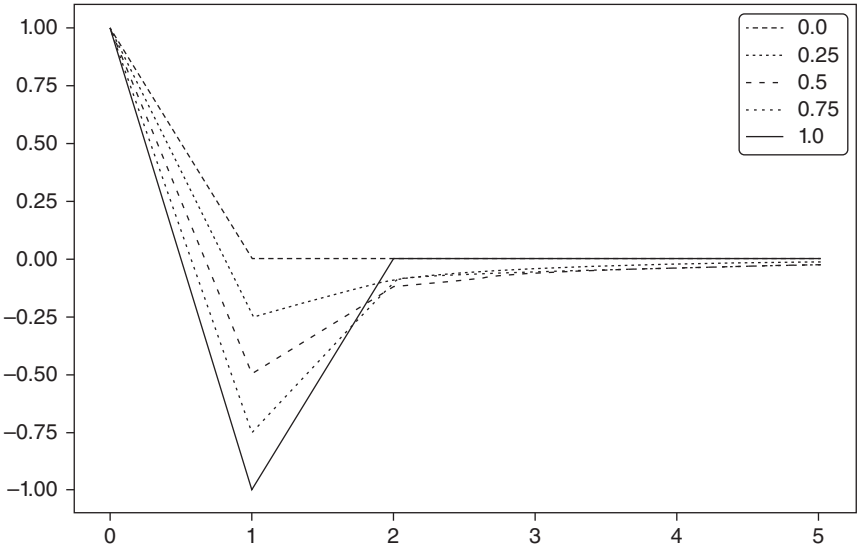


FIGURE 5.1 ω_k (y-axis) as k increases (x-axis). Each line is associated with a particular value of $d \in [0,1]$, in 0.1 increments.

When d is a positive integer number, $\prod_{i=0}^{k-1} \frac{d-i}{k!} = 0, \forall k > d$, and memory beyond that point is cancelled. For example, $d = 1$ is used to compute returns, where $\prod_{i=0}^{k-1} \frac{d-i}{k!} = 0, \forall k > 1$, and $\omega = \{1, -1, 0, 0, \dots\}$.

5.4.2 Iterative Estimation

Looking at the sequence of weights, ω, we can appreciate that for $k = 0, \dots, \infty$, with $\omega_0 = 1$, the weights can be generated iteratively as:

$$\omega_k = -\omega_{k-1} \frac{d - k + 1}{k}$$

Figure 5.1 plots the sequence of weights used to compute each value of the fractionally differentiated series. The legend reports the value of d used to generate each sequence, the x-axis indicates the value of k, and the y-axis shows the value of ω_k. For example, for $d = 0$, all weights are 0 except for $\omega_0 = 1$. That is the case where the differentiated series coincides with the original one. For $d = 1$, all weights are 0 except for $\omega_0 = 1$ and $\omega_1 = -1$. That is the standard first-order integer differentiation, which is used to derive log-price returns. Anywhere in between these two cases, all weights after $\omega_0 = 1$ are negative and greater than -1.

Figure 5.2 plots the sequence of weights where $d \in [1, 2]$, at increments of 0.1. For $d > 1$, we observe $\omega_1 < -1$ and $\omega_k > 0, \ \forall k \geq 2$.

Snippet 5.1 lists the code used to generate these plots.

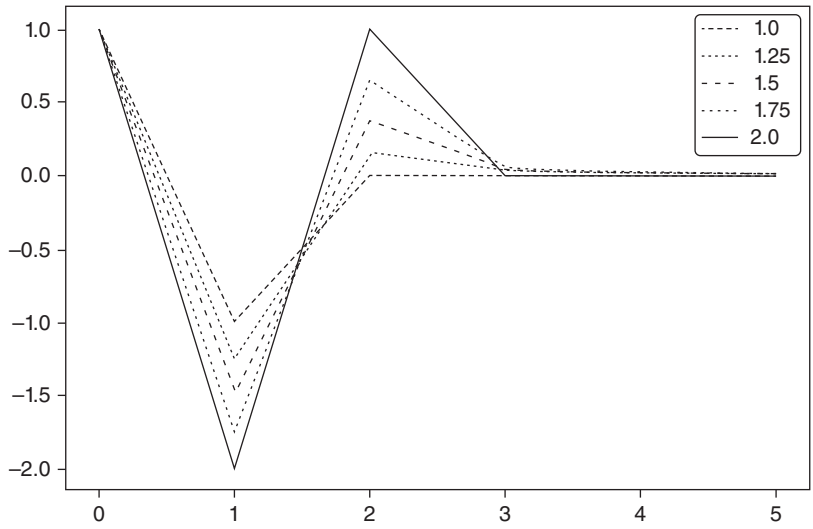


FIGURE 5.2 ω_k (y-axis) as k increases (x-axis). Each line is associated with a particular value of $d \in [1,2]$, in 0.1 increments.

SNIPPET 5.1 WEIGHTING FUNCTION

```
def getWeights(d,size):
    # thres>0 drops insignificant weights
    w=[1.]
    for k in range(1,size):
        w_=-w[-1]/k*(d-k+1)
        w.append(w_)
    w=np.array(w[::-1]).reshape(-1,1)
    return w
#---------------------------------------------------
def plotWeights(dRange,nPlots,size):
    w=pd.DataFrame()
    for d in np.linspace(dRange[0],dRange[1],nPlots):
        w_=getWeights(d,size=size)
        w_=pd.DataFrame(w_,index=range(w_.shape[0])[::-1],columns=[d])
        w=w.join(w_,how='outer')
    ax=w.plot()
    ax.legend(loc='upper left');mpl.show()
    return
#---------------------------------------------------
if __name__=='__main__':
    plotWeights(dRange=[0,1],nPlots=11,size=6)
    plotWeights(dRange=[1,2],nPlots=11,size=6)
```

5.4.3 Convergence

Let us consider the convergence of the weights. From the above result, we can see that for $k > d$, if $\omega_{k-1} \neq 0$, then $\left|\frac{\omega_k}{\omega_{k-1}}\right| = \left|\frac{d-k+1}{k}\right| < 1$, and $\omega_k = 0$ otherwise. Consequently, the weights converge asymptotically to zero, as an infinite product of factors within the unit circle. Also, for a positive d and $k < d+1$, we have $\frac{d-k+1}{k} \geq 0$, which makes the initial weights alternate in sign. For a non-integer d, once $k \geq d+1$, ω_k will be negative if int$[d]$ is even, and positive otherwise. Summarizing, $\lim_{k\to\infty} \omega_k = 0^-$ (converges to zero from the left) when int$[d]$ is even, and $\lim_{k\to\infty} \omega_k = 0^+$ (converges to zero from the right) when Int$[d]$ is odd. In the special case $d \in (0,1)$, this means that $-1 < \omega_k < 0, \forall k > 0$. This alternation of weight signs is necessary to make $\{\tilde{X}_t\}_{t=1,\ldots,T}$ stationary, as memory wanes or is offset over the long run.

5.5 IMPLEMENTATION

In this section we will explore two alternative implementations of fractional differentiation: the standard "expanding window" method, and a new method that I call "fixed-width window fracdiff" (FFD).

5.5.1 Expanding Window

Let us discuss how to fractionally differentiate a (finite) time series in practice. Suppose a time series with T real observations, $\{X_t\},\ t = 1, \ldots, T$. Because of data limitations, the fractionally differentiated value $\tilde{X}_T$ cannot be computed on an infinite series of weights. For instance, the last point $\tilde{X}_T$ will use weights $\{\omega_k\},\ k = 0, \ldots, T-1$, and $\tilde{X}_{T-l}$ will use weights $\{\omega_k\},\ k = 0, \ldots, T-l-1$. This means that the initial points will have a different amount of memory compared to the final points. For each l, we can determine the relative weight-loss, $\lambda_l = \frac{\sum_{j=T-l}^{T} |\omega_j|}{\sum_{i=0}^{T-1} |\omega_i|}$. Given a tolerance level $\tau \in [0,1]$, we can determine the value l^* such that $\lambda_{l^*} \leq \tau$ and $\lambda_{l^*+1} > \tau$. This value l^* corresponds to the first results $\{\tilde{X}_t\}_{t=1,\ldots,l^*}$ where the weight-loss is beyond the acceptable threshold, $\lambda_t > \tau$ (e.g., $\tau = 0.01$).

From our earlier discussion, it is clear that λ_{l^*} depends on the convergence speed of $\{\omega_k\}$, which in turn depends on $d \in [0,1]$. For $d = 1$, $\omega_k = 0,\ \forall k > 1$, and $\lambda_l = 0,\ \forall l > 1$, hence it suffices to drop $\tilde{X}_1$. As $d \to 0^+$, l^* increases, and a larger portion of the initial $\{\tilde{X}_t\}_{t=1,\ldots,l^*}$ needs to be dropped in order to keep the weight-loss $\lambda_{l^*} \leq \tau$. Figure 5.3 plots the E-mini S&P 500 futures trade bars of size 1E4, rolled forward, fractionally differentiated, with parameters $(d = .4, \tau = 1)$ on the top and parameters $(d = .4, \tau = 1E-2)$ on the bottom.

The negative drift in both plots is caused by the negative weights that are added to the initial observations as the window is expanded. When we do not control for weight loss, the negative drift is extreme, to the point that only that trend is visible. The negative drift is somewhat more moderate in the right plot, after controlling

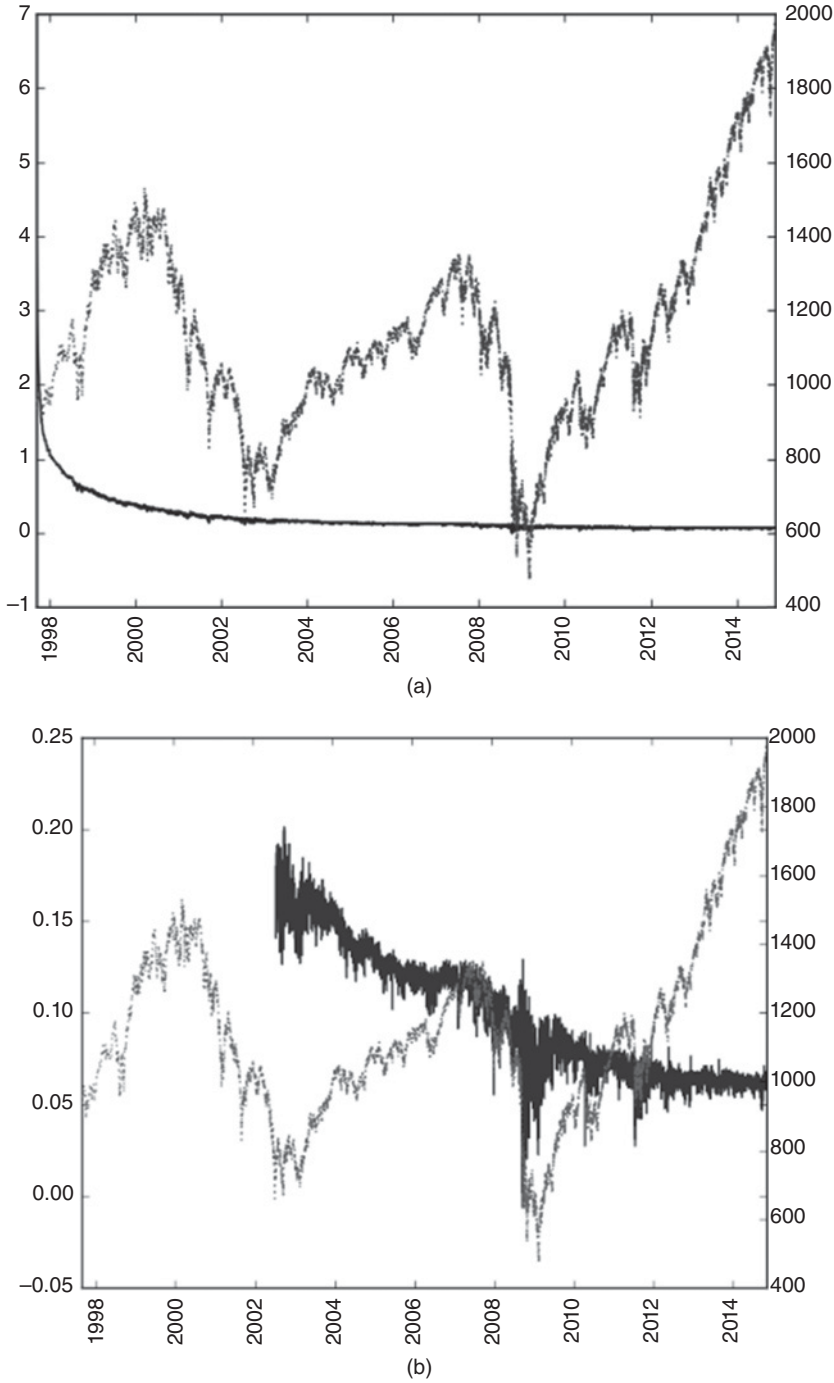


FIGURE 5.3 Fractional differentiation without controlling for weight loss (top plot) and after controlling for weight loss with an expanding window (bottom plot)

for the weight loss, however, it is still substantial, because values $\{\tilde{X}_t\}_{t=l^*+1,\ldots,T}$ are computed on an expanding window. This problem can be corrected by a fixed-width window, implemented in Snippet 5.2.

SNIPPET 5.2 STANDARD FRACDIFF (EXPANDING WINDOW)

```
def fracDiff(series,d,thres=.01):
    '''
    Increasing width window, with treatment of NaNs
    Note 1: For thres=1, nothing is skipped.
    Note 2: d can be any positive fractional, not necessarily bounded [0,1].
    '''
    #1) Compute weights for the longest series
    w=getWeights(d,series.shape[0])
    #2) Determine initial calcs to be skipped based on weight-loss threshold
    w_=np.cumsum(abs(w))
    w_/=w_[-1]
    skip=w_[w_>thres].shape[0]
    #3) Apply weights to values
    df={}
    for name in series.columns:
        seriesF,df_=series[[name]].fillna(method='ffill').dropna(),pd.Series()
        for iloc in range(skip,seriesF.shape[0]):
            loc=seriesF.index[iloc]
            if not np.isfinite(series.loc[loc,name]):continue # exclude NAs
            df_[loc]=np.dot(w[-(iloc+1):,:].T,seriesF.loc[:loc])[0,0]
        df[name]=df_.copy(deep=True)
    df=pd.concat(df,axis=1)
    return df
```

5.5.2 Fixed-Width Window Fracdiff

Alternatively, fractional differentiation can be computed using a fixed-width window, that is, dropping the weights after their modulus ($|\omega_k|$) falls below a given threshold value (τ). This is equivalent to finding the first l^* such that $|\omega_{l^*}| \geq \tau$ and $|\omega_{l^*+1}| \leq \tau$, setting a new variable $\tilde{\omega}_k$

$$\tilde{\omega}_k = \begin{cases} \omega_k & \text{if } k \leq l^* \\ 0 & \text{if } k > l^* \end{cases}$$

and $\tilde{X}_t = \sum_{k=0}^{l^*} \tilde{\omega}_k X_{t-k}$, for $t = T - l^* + 1, \ldots, T$. Figure 5.4 plots E-mini S&P 500 futures trade bars of size 1E4, rolled forward, fractionally differentiated ($d = .4, \tau = 1E-5$).

This procedure has the advantage that the same vector of weights is used across all estimates of $\{\tilde{X}_t\}_{t=l^*,\ldots,T}$, hence avoiding the negative drift caused by an

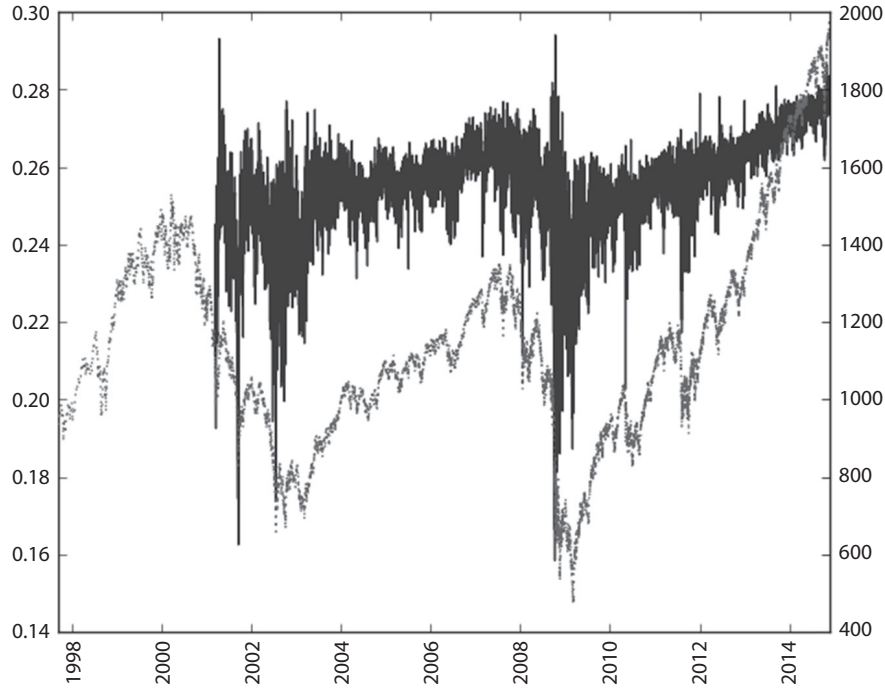


FIGURE 5.4 Fractional differentiation after controlling for weight loss with a fixed-width window

expanding window's added weights. The result is a driftless blend of level plus noise, as expected. The distribution is no longer Gaussian, as a result of the skewness and excess kurtosis that comes with memory, however it is stationary. Snippet 5.3 presents an implementation of this idea.

SNIPPET 5.3 THE NEW FIXED-WIDTH WINDOW FRACDIFF METHOD

```
def fracDiff_FFD(series,d,thres=1e-5):
    '''
    Constant width window (new solution)
    Note 1: thres determines the cut-off weight for the window
    Note 2: d can be any positive fractional, not necessarily bounded [0,1].
    '''
    #1) Compute weights for the longest series
    w=getWeights_FFD(d,thres)
    width=len(w)-1
    #2) Apply weights to values
    df={}
```

```
for name in series.columns:
    seriesF,df_=series[[name]].fillna(method='ffill').dropna(),pd.Series()
    for iloc1 in range(width,seriesF.shape[0]):
        loc0,loc1=seriesF.index[iloc1-width],seriesF.index[iloc1]
        if not np.isfinite(series.loc[loc1,name]):continue # exclude NAs
        df_[loc1]=np.dot(w.T,seriesF.loc[loc0:loc1])[0,0]
    df[name]=df_.copy(deep=True)
df=pd.concat(df,axis=1)
return df
```

5.6 STATIONARITY WITH MAXIMUM MEMORY PRESERVATION

Consider a series $\{X_t\}_{t=1,\ldots,T}$. Applying the fixed-width window fracdiff (FFD) method on this series, we can compute the minimum coefficient d^* such that the resulting fractionally differentiated series $\{\tilde{X}_t\}_{t=l^*,\ldots,T}$ is stationary. This coefficient d^* quantifies the amount of memory that needs to be removed to achieve stationarity. If $\{X_t\}_{t=l^*,\ldots,T}$ is already stationary, then $d^* = 0$. If $\{X_t\}_{t=l^*,\ldots,T}$ contains a unit root, then $d^* < 1$. If $\{X_t\}_{t=l^*,\ldots,T}$ exhibits explosive behavior (like in a bubble), then $d^* > 1$. A case of particular interest is $0 < d^* \ll 1$, when the original series is "mildly non-stationary." In this case, although differentiation is needed, a full integer differentiation removes excessive memory (and predictive power).

Figure 5.5 illustrates this concept. On the right y-axis, it plots the ADF statistic computed on E-mini S&P 500 futures log-prices, rolled forward using the ETF trick

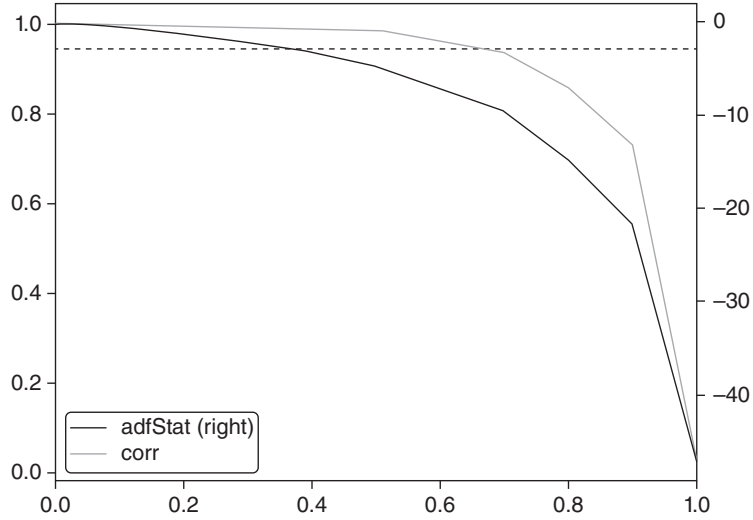


FIGURE 5.5 ADF statistic as a function of d, on E-mini S&P 500 futures log-prices

(see Chapter 2), downsampled to daily frequency, going back to the contract's inception. On the x-axis, it displays the d value used to generate the series on which the ADF statistic was computed. The original series has an ADF statistic of –0.3387, while the returns series has an ADF statistic of –46.9114. At a 95% confidence level, the test's critical value is –2.8623. The ADF statistic crosses that threshold in the vicinity of $d = 0.35$. The left y-axis plots the correlation between the original series ($d = 0$) and the differentiated series at various d values. At $d = 0.35$ the correlation is still very high, at 0.995. This confirms that the procedure introduced in this chapter has been successful in achieving stationarity without giving up too much memory. In contrast, the correlation between the original series and the returns series is only 0.03, hence showing that the standard integer differentiation wipes out the series' memory almost entirely.

Virtually all finance papers attempt to recover stationarity by applying an integer differentiation $d = 1 \gg 0.35$, which means that most studies have over-differentiated the series, that is, they have removed much more memory than was necessary to satisfy standard econometric assumptions. Snippet 5.4 lists the code used to produce these results.

SNIPPET 5.4 FINDING THE MINIMUM *D* VALUE THAT PASSES THE ADF TEST

```
def plotMinFFD():
    from statsmodels.tsa.stattools import adfuller
    path,instName='./','ES1_Index_Method12'
    out=pd.DataFrame(columns=['adfStat','pVal','lags','nObs','95% conf','corr'])
    df0=pd.read_csv(path+instName+'.csv',index_col=0,parse_dates=True)
    for d in np.linspace(0,1,11):
        df1=np.log(df0[['Close']]).resample('1D').last() # downcast to daily obs
        df2=fracDiff_FFD(df1,d,thres=.01)
        corr=np.corrcoef(df1.loc[df2.index,'Close'],df2['Close'])[0,1]
        df2=adfuller(df2['Close'],maxlag=1,regression='c',autolag=None)
        out.loc[d]=list(df2[:4])+[df2[4]['5%']]+[corr] # with critical value
    out.to_csv(path+instName+'_testMinFFD.csv')
    out[['adfStat','corr']].plot(secondary_y='adfStat')
    mpl.axhline(out['95% conf'].mean(),linewidth=1,color='r',linestyle='dotted')
    mpl.savefig(path+instName+'_testMinFFD.png')
    return
```

The example on E-mini futures is by no means an exception. Table 5.1 shows the ADF statistics after applying FFD(d) on various values of d, for 87 of the most liquid futures worldwide. In all cases, the standard $d = 1$ used for computing returns implies over-differentiation. In fact, in all cases stationarity is achieved with $d < 0.6$. In some cases, like orange juice (JO1 Comdty) or live cattle (LC1 Comdty) no differentiation at all was needed.

TABLE 5.1 ADF Statistic on FFD(d) for Some of the Most Liquid Futures Contracts

	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
AD1 Curncy	−1.7253	−1.8665	−2.2801	−2.9743	−3.9590	−5.4450	−7.7387	−10.3412	−15.7255	−22.5170	−43.8281
BO1 Comdty	−0.7039	−1.0021	−1.5848	−2.4038	−3.4284	−4.8916	−7.0604	−9.5089	−14.4065	−20.4393	−38.0683
BP1 Curncy	−1.0573	−1.4963	−2.3223	−3.4641	−4.8976	−6.9157	−9.8833	−13.1575	−19.4238	−26.6320	−43.3284
BTS1 Comdty	−1.7987	−2.1428	−2.7600	−3.7019	−4.8522	−6.2412	−7.8115	−9.4645	−11.0334	−12.4470	−13.6410
BZ1 Index	−1.6569	−1.8766	−2.3948	−3.2145	−4.2821	−5.9431	−8.3329	−10.9046	−15.7006	−20.7224	−29.9510
C 1 Comdty	−1.7870	−2.1273	−2.9539	−4.1642	−5.7307	−7.9577	−11.1798	−14.6946	−20.9925	−27.6602	−39.3576
CC1 Comdty	−2.3743	−2.9503	−4.1694	−5.8997	−8.0868	−10.9871	−14.8206	−18.6154	−24.1738	−29.0285	−34.8580
CD1 Curncy	−1.6304	−2.0557	−2.7284	−3.8380	−5.2341	−7.3172	−10.3738	−13.8263	−20.2897	−27.6242	−43.6794
CF1 Index	−1.5539	−1.9387	−2.7421	−3.9235	−5.5085	−7.7585	−11.0571	−14.6829	−21.4877	−28.9810	−44.5059
CL1 Comdty	−0.3795	−0.7164	−1.3359	−2.2018	−3.2603	−4.7499	−6.9504	−9.4531	−14.4936	−20.8392	−41.1169
CN1 Comdty	−0.8798	−0.8711	−1.1020	−1.4626	−1.9732	−2.7508	−3.9217	−5.2944	−8.4257	−12.7300	−42.1411
CO1 Comdty	−0.5124	−0.8468	−1.4247	−2.2402	−3.2566	−4.7022	−6.8601	−9.2836	−14.1511	−20.2313	−39.2207
CT1 Comdty	−1.7604	−2.0728	−2.7529	−3.7853	−5.1397	−7.1123	−10.0137	−13.1851	−19.0603	−25.4513	−37.5703
DM1 Index	−0.1929	−0.5718	−1.2414	−2.1127	−3.1765	−4.6695	−6.8852	−9.4219	−14.6726	−21.5411	−49.2663
DU1 Comdty	−0.3365	−0.4572	−0.7647	−1.1447	−1.6132	−2.2759	−3.3389	−4.5689	−7.2101	−10.9025	−42.9012
DX1 Curncy	−1.5768	−1.9458	−2.7358	−3.8423	−5.3101	−7.3507	−10.3569	−13.6451	−19.5832	−25.8907	−37.2623
EC1 Comdty	−0.2727	−0.6650	−1.3359	−2.2112	−3.3112	−4.8320	−7.0777	−9.6299	−14.8258	−21.4634	−44.6452
EC1 Curncy	−1.4733	−1.9344	−2.8507	−4.1588	−5.8240	−8.1834	−11.6278	−15.4095	−22.4317	−30.1482	−45.6373
ED1 Comdty	−0.4084	−0.5350	−0.7948	−1.1772	−1.6633	−2.3818	−3.4601	−4.7041	−7.4373	−11.3175	−46.4487
EE1 Curncy	−1.2100	−1.6378	−2.4216	−3.5470	−4.9821	−7.0166	−9.9962	−13.2920	−19.5047	−26.5158	−41.4672
EO1 Comdty	−0.7903	−0.8917	−1.0551	−1.3465	−1.7302	−2.3500	−3.3068	−4.5136	−7.0157	−10.6463	−45.2100

EO1 Index	−0.6561	−1.0567	−1.7409	−2.6774	−3.8543	−5.5096	−7.9133	−10.5674	−15.6442	−21.3066	−35.1397
ER1 Comdty	−0.1970	−0.3442	−0.6334	−1.0363	−1.5327	−2.2378	−3.2819	−4.4647	−7.1031	−10.7389	−40.0407
ES1 Index	−0.3387	−0.7206	−1.3324	−2.2252	−3.2733	−4.7976	−7.0436	−9.6095	−14.8624	−21.6177	−46.9114
FA1 Index	−0.5292	−0.8526	−1.4250	−2.2359	−3.2500	−4.6902	−6.8272	−9.2410	−14.1664	−20.3733	−41.9705
FC1 Comdty	−1.8846	−2.1853	−2.8808	−3.8546	−5.1483	−7.0226	−9.6889	−12.5679	−17.8160	−23.0530	−31.6503
FV1 Comdty	−0.7257	−0.8515	−1.0596	−1.4304	−1.8312	−2.5302	−3.6296	−4.9499	−7.8292	−12.0467	−49.1508
G 1 Comdty	0.2326	0.0026	−0.4686	−1.0590	−1.7453	−2.6761	−4.0336	−5.5624	−8.8575	−13.3277	−42.9177
GC1 Comdty	−2.2221	−2.3544	−2.7467	−3.4140	−4.4861	−6.0632	−8.4803	−11.2152	−16.7111	−23.1750	−39.0715
GX1 Index	−1.5418	−1.7749	−2.4666	−3.4417	−4.7321	−6.6155	−9.3667	−12.5240	−18.6291	−25.8116	−43.3610
HG1 Comdty	−1.7372	−2.1495	−2.8323	−3.9090	−5.3257	−7.3805	−10.4121	−13.7669	−19.8902	−26.5819	−39.3267
HI1 Index	−1.8289	−2.0432	−2.6203	−3.5233	−4.7514	−6.5743	−9.2733	−12.3722	−18.5308	−25.9762	−45.3396
HO1 Comdty	−1.6024	−1.9941	−2.6619	−3.7131	−5.1772	−7.2468	−10.3326	−13.6745	−19.9728	−26.9772	−40.9824
IB1 Index	−2.3912	−2.8254	−3.5813	−4.8774	−6.5884	−9.0665	−12.7381	−16.6706	−23.6752	−30.7986	−43.0687
IK1 Comdty	−1.7373	−2.3000	−2.7764	−3.7101	−4.8686	−6.3504	−8.2195	−9.8636	−11.7882	−13.3983	−14.8391
IR1 Comdty	−2.0622	−2.4188	−3.1736	−4.3178	−5.8119	−7.9816	−11.2102	−14.7956	−21.6158	−29.4555	−46.2683
JA1 Comdty	−2.4701	−2.7292	−3.3925	−4.4658	−5.9236	−8.0270	−11.2082	−14.7198	−21.2681	−28.4380	−42.1937
JB1 Comdty	−0.2081	−0.4319	−0.8490	−1.4289	−2.1160	−3.0932	−4.5740	−6.3061	−9.9454	−15.0151	−47.6037
JE1 Curncy	−0.9268	−1.2078	−1.7565	−2.5398	−3.5545	−5.0270	−7.2096	−9.6808	−14.6271	−20.7168	−37.6954
JG1 Comdty	−1.7468	−1.8071	−2.0654	−2.5447	−3.2237	−4.3418	−6.0690	−8.0537	−12.3908	−18.1881	−44.2884
JO1 Comdty	−3.0052	−3.3099	−4.2639	−5.7291	−7.5686	−10.1683	−13.7068	−17.3054	−22.7853	−27.7011	−33.4658
JY1 Curncy	−1.2616	−1.5891	−2.2042	−3.1407	−4.3715	−6.1600	−8.8261	−11.8449	−17.8275	−25.0700	−44.8394
KC1 Comdty	−0.7786	−1.1172	−1.7723	−2.7185	−3.8875	−5.5651	−8.0217	−10.7422	−15.9423	−21.8651	−35.3354
L 1 Comdty	−0.0805	−0.2228	−0.6144	−1.0751	−1.6335	−2.4186	−3.5676	−4.8749	−7.7528	−11.7669	−44.0349

At a 95% confidence level, the ADF test's critical value is −2.8623. All of the log-price series achieve stationarity at $d < 0.6$, and the great majority are stationary at $d < 0.3$.

5.7 CONCLUSION

To summarize, most econometric analyses follow one of two paradigms:

1. Box-Jenkins: Returns are stationary, however memory-less.
2. Engle-Granger: Log-prices have memory, however they are non-stationary. Cointegration is the trick that makes regression work on non-stationary series, so that memory is preserved. However the number of cointegrated variables is limited, and the cointegrating vectors are notoriously unstable.

In contrast, the FFD approach introduced in this chapter shows that there is no need to give up all of the memory in order to gain stationarity. And there is no need for the cointegration trick as it relates to ML forecasting. Once you become familiar with FFD, it will allow you to achieve stationarity without renouncing to memory (or predictive power).

In practice, I suggest you experiment with the following transformation of your features: First, compute a cumulative sum of the time series. This guarantees that some order of differentiation is needed. Second, compute the FFD(d) series for various $d \in [0, 1]$. Third, determine the minimum d such that the p-value of the ADF statistic on FFD(d) falls below 5%. Fourth, use the FFD(d) series as your predictive feature.

EXERCISES

5.1 Generate a time series from an IID Gaussian random process. This is a memory-less, stationary series:
- **(a)** Compute the ADF statistic on this series. What is the p-value?
- **(b)** Compute the cumulative sum of the observations. This is a non-stationary series without memory.
  - **(i)** What is the order of integration of this cumulative series?
  - **(ii)** Compute the ADF statistic on this series. What is the p-value?
- **(c)** Differentiate the series twice. What is the p-value of this over-differentiated series?

5.2 Generate a time series that follows a sinusoidal function. This is a stationary series with memory.
- **(a)** Compute the ADF statistic on this series. What is the p-value?
- **(b)** Shift every observation by the same positive value. Compute the cumulative sum of the observations. This is a non-stationary series with memory.
  - **(i)** Compute the ADF statistic on this series. What is the p-value?
  - **(ii)** Apply an expanding window fracdiff, with $\tau = 1E-2$. For what minimum d value do you get a p-value below 5%?
  - **(iii)** Apply FFD, with $\tau = 1E-5$. For what minimum d value do you get a p-value below 5%?

5.3 Take the series from exercise 2.b:

(a) Fit the series to a sine function. What is the R-squared?

(b) Apply FFD($d = 1$). Fit the series to a sine function. What is the R-squared?

(c) What value of d maximizes the R-squared of a sinusoidal fit on FFD(d). Why?

5.4 Take the dollar bar series on E-mini S&P 500 futures. Using the code in Snippet 5.3, for some $d \in [0, 2]$, compute `fracDiff_FFD(fracDiff_FFD(series,d),-d)`. What do you get? Why?

5.5 Take the dollar bar series on E-mini S&P 500 futures.

(a) Form a new series as a cumulative sum of log-prices.

(b) Apply FFD, with $\tau = 1E-5$. Determine for what minimum $d \in [0, 2]$ the new series is stationary.

(c) Compute the correlation of the fracdiff series to the original (untransformed) series.

(d) Apply an Engel-Granger cointegration test on the original and fracdiff series. Are they cointegrated? Why?

(e) Apply a Jarque-Bera normality test on the fracdiff series.

5.6 Take the fracdiff series from exercise 5.

(a) Apply a CUSUM filter (Chapter 2), where h is twice the standard deviation of the series.

(b) Use the filtered timestamps to sample a features' matrix. Use as one of the features the fracdiff value.

(c) Form labels using the triple-barrier method, with symmetric horizontal barriers of twice the daily standard deviation, and a vertical barrier of 5 days.

(d) Fit a bagging classifier of decision trees where:

(i) The observed features are bootstrapped using the sequential method from Chapter 4.

(ii) On each bootstrapped sample, sample weights are determined using the techniques from Chapter 4.

REFERENCES

Alexander, C. (2001): *Market Models*, 1st edition. John Wiley & Sons.

Hamilton, J. (1994): *Time Series Analysis*, 1st ed. Princeton University Press.

Hosking, J. (1981): "Fractional differencing." *Biometrika*, Vol. 68, No. 1, pp. 165–176.

Jensen, A. and M. Nielsen (2014): "A fast fractional difference algorithm." *Journal of Time Series Analysis*, Vol. 35, No. 5, pp. 428–436.

López de Prado, M. (2015): "The Future of Empirical Finance." *Journal of Portfolio Management*, Vol. 41, No. 4, pp. 140–144. Available at https://ssrn.com/abstract=2609734.

BIBLIOGRAPHY

Cavaliere, G., M. Nielsen, and A. Taylor (2017): "Quasi-maximum likelihood estimation and bootstrap inference in fractional time series models with heteroskedasticity of unknown form." *Journal of Econometrics*, Vol. 198, No. 1, pp. 165–188.

Johansen, S. and M. Nielsen (2012): "A necessary moment condition for the fractional functional central limit theorem." *Econometric Theory*, Vol. 28, No. 3, pp. 671–679.

Johansen, S. and M. Nielsen (2012): "Likelihood inference for a fractionally cointegrated vector autoregressive model." *Econometrica*, Vol. 80, No. 6, pp. 2267–2732.

Johansen, S. and M. Nielsen (2016): "The role of initial values in conditional sum-of-squares estimation of nonstationary fractional time series models." *Econometric Theory*, Vol. 32, No. 5, pp. 1095–1139.

Jones, M., M. Nielsen and M. Popiel (2015): "A fractionally cointegrated VAR analysis of economic voting and political support." *Canadian Journal of Economics*, Vol. 47, No. 4, pp. 1078–1130.

Mackinnon, J. and M. Nielsen, M. (2014): "Numerical distribution functions of fractional unit root and cointegration tests." *Journal of Applied Econometrics*, Vol. 29, No. 1, pp. 161–171.

PART 2

Modelling

CHAPTER 6

Ensemble Methods

6.1 MOTIVATION

In this chapter we will discuss two of the most popular ML ensemble methods.[1] In the references and footnotes you will find books and articles that introduce these techniques. As everywhere else in this book, the assumption is that you have already used these approaches. The goal of this chapter is to explain what makes them effective, and how to avoid common errors that lead to their misuse in finance.

6.2 THE THREE SOURCES OF ERRORS

ML models generally suffer from three errors:[2]

1. **Bias:** This error is caused by unrealistic assumptions. When bias is high, the ML algorithm has failed to recognize important relations between features and outcomes. In this situation, the algorithm is said to be "underfit."
2. **Variance:** This error is caused by sensitivity to small changes in the training set. When variance is high, the algorithm has overfit the training set, and that is why even minimal changes in the training set can produce wildly different predictions. Rather than modelling the general patterns in the training set, the algorithm has mistaken noise with signal.

[1] For an introduction to ensemble methods, please visit: http://scikit-learn.org/stable/modules/ensemble.html.

[2] I would not typically cite Wikipedia, however, on this subject the user may find some of the illustrations in this article useful: https://en.wikipedia.org/wiki/Bias%E2%80%93variance_tradeoff.

3. **Noise:** This error is caused by the variance of the observed values, like unpredictable changes or measurement errors. This is the irreducible error, which cannot be explained by any model.

Consider a training set of observations $\{x_i\}_{i=1,\ldots,n}$ and real-valued outcomes $\{y_i\}_{i=1,\ldots,n}$. Suppose a function $f[x]$ exists, such that $y = f[x] + \varepsilon$, where ε is white noise with $\mathrm{E}[\varepsilon_i] = 0$ and $\mathrm{E}[\varepsilon_i^2] = \sigma_\varepsilon^2$. We would like to estimate the function $\hat{f}[x]$ that best fits $f[x]$, in the sense of making the variance of the estimation error $\mathrm{E}[(y_i - \hat{f}[x_i])^2]$ minimal (the mean squared error cannot be zero, because of the noise represented by σ_ε^2). This mean-squared error can be decomposed as

$$\mathrm{E}\left[(y_i - \hat{f}[x_i])^2\right] = \left(\underbrace{\mathrm{E}[\hat{f}[x_i] - f[x_i]]}_{bias}\right)^2 + \underbrace{\mathrm{V}[\hat{f}[x_i]]}_{variance} + \underbrace{\sigma_\varepsilon^2}_{noise}$$

An ensemble method is a method that combines a set of weak learners, all based on the same learning algorithm, in order to create a (stronger) learner that performs better than any of the individual ones. Ensemble methods help reduce bias and/or variance.

6.3 BOOTSTRAP AGGREGATION

Bootstrap aggregation (bagging) is an effective way of reducing the variance in forecasts. It works as follows: First, generate N training datasets by random sampling *with replacement.* Second, fit N estimators, one on each training set. These estimators are fit independently from each other, hence the models can be fit in parallel. Third, the ensemble forecast is the *simple* average of the individual forecasts from the N models. In the case of categorical variables, the probability that an observation belongs to a class is given by the proportion of estimators that classify that observation as a member of that class (majority voting). When the base estimator can make forecasts with a prediction probability, the bagging classifier may derive a mean of the probabilities.

If you use sklearn's `BaggingClassifier` class to compute the out-of-bag accuracy, you should be aware of this bug: https://github.com/scikit-learn/scikit-learn/issues/8933. One workaround is to rename the labels in integer sequential order.

6.3.1 Variance Reduction

Bagging's main advantage is that it reduces forecasts' variance, hence helping address overfitting. The variance of the bagged prediction ($\varphi_i[c]$) is a function of the number of bagged estimators (N), the average variance of a single estimator's prediction ($\bar{\sigma}$),

and the average correlation among their forecasts ($\bar{\rho}$):

$$\begin{aligned}
\mathrm{V}\left[\frac{1}{N}\sum_{i=1}^{N}\varphi_i[c]\right] &= \frac{1}{N^2}\sum_{i=1}^{N}\left(\sum_{j=1}^{N}\sigma_{i,j}\right) = \frac{1}{N^2}\sum_{i=1}^{N}\left(\sigma_i^2 + \sum_{j\neq i}^{N}\sigma_i\sigma_j\rho_{i,j}\right) \\
&= \frac{1}{N^2}\sum_{i=1}^{N}\left(\bar{\sigma}^2 + \underbrace{\sum_{j\neq i}^{N}\bar{\sigma}^2\bar{\rho}}_{\substack{=(N-1)\bar{\sigma}^2\bar{\rho} \\ \text{for a fixed } i}}\right) = \frac{\bar{\sigma}^2 + (N-1)\bar{\sigma}^2\bar{\rho}}{N} \\
&= \bar{\sigma}^2\left(\bar{\rho} + \frac{1-\bar{\rho}}{N}\right)
\end{aligned}$$

where $\sigma_{i,j}$ is the covariance of predictions by estimators i,j; $\sum_{i=1}^{N}\bar{\sigma}^2 = \sum_{i=1}^{N}\sigma_i^2 \Leftrightarrow \bar{\sigma}^2 = N^{-1}\sum_{i=1}^{N}\sigma_i^2$; and $\sum_{j\neq i}^{N}\bar{\sigma}^2\bar{\rho} = \sum_{j\neq i}^{N}\sigma_i\sigma_j\rho_{i,j} \Leftrightarrow \bar{\rho} = (\bar{\sigma}^2 N(N-1))^{-1}\sum_{j\neq i}^{N}\sigma_i\sigma_j\rho_{i,j}$.

The equation above shows that bagging is only effective to the extent that $\bar{\rho} < 1$; as $\bar{\rho} \to 1 \Rightarrow \mathrm{V}[\frac{1}{N}\sum_{i=1}^{N}\varphi_i[c]] \to \bar{\sigma}^2$. One of the goals of sequential bootstrapping (Chapter 4) is to produce samples as independent as possible, thereby reducing $\bar{\rho}$, which should lower the variance of bagging classifiers. Figure 6.1 plots the standard deviation of the bagged prediction as a function of $N \in [5, 30]$, $\bar{\rho} \in [0, 1]$ and $\bar{\sigma} = 1$.

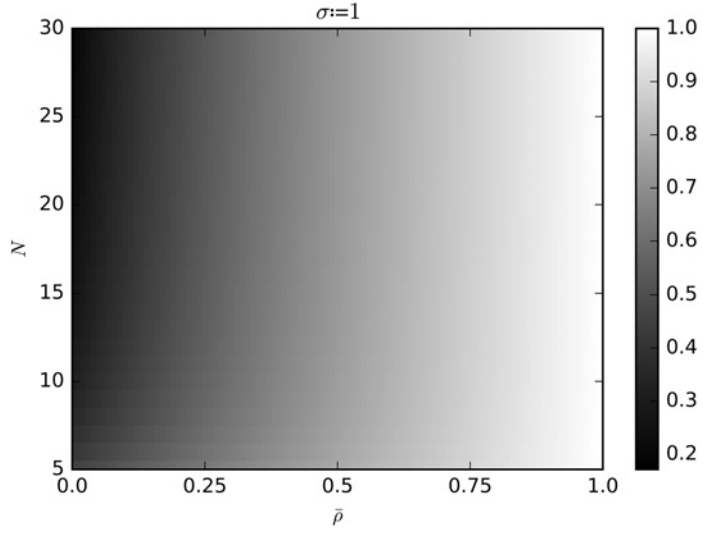


FIGURE 6.1 Standard deviation of the bagged prediction

6.3.2 Improved Accuracy

Consider a bagging classifier that makes a prediction on k classes by majority voting among N independent classifiers. We can label the predictions as {0,1}, where 1 means a correct prediction. The accuracy of a classifier is the probability p of labeling a prediction as 1. On average we will get Np predictions labeled as 1, with variance $Np(1-p)$. Majority voting makes the correct prediction when the most forecasted class is observed. For example, for $N = 10$ and $k = 3$, the bagging classifier made a correct prediction when class A was observed and the cast votes were $[A, B, C] = [4,3,3]$. However, the bagging classifier made an incorrect prediction when class A was observed and the cast votes were $[A, B, C] = [4,1,5]$. A sufficient condition is that the sum of these labels is $X > \frac{N}{2}$. A necessary (non-sufficient) condition is that $X > \frac{N}{k}$, which occurs with probability

$$\mathrm{P}\left[X > \frac{N}{k}\right] = 1 - \mathrm{P}\left[X \leq \frac{N}{k}\right] = 1 - \sum_{i=0}^{\left\lfloor N/k \right\rfloor} \binom{N}{i} p^i (1-p)^{N-i}$$

The implication is that for a sufficiently large N, say $N > p(p - 1/k)^{-2}$, then $p > \frac{1}{k} \Rightarrow \mathrm{P}[X > \frac{N}{k}] > p$, hence the bagging classifier's accuracy exceeds the average accuracy of the individual classifiers. Snippet 6.1 implements this calculation.

SNIPPET 6.1 ACCURACY OF THE BAGGING CLASSIFIER

```
from scipy.misc import comb
N,p,k=100,1./3,3.
p_=0
for i in xrange(0,int(N/k)+1):
    p_+=comb(N,i)*p**i*(1-p)**(N-i)
print p,1-p_
```

This is a strong argument in favor of bagging any classifier in general, when computational requirements permit it. However, unlike boosting, bagging cannot improve the accuracy of poor classifiers: If the individual learners are poor classifiers ($p \ll \frac{1}{k}$), majority voting will still perform poorly (although with lower variance). Figure 6.2 illustrates these facts. Because it is easier to achieve $\bar{\rho} \ll 1$ than $p > \frac{1}{k}$, bagging is more likely to be successful in reducing variance than in reducing bias.

For further analysis on this topic, the reader is directed to Condorcet's Jury Theorem. Although the theorem is derived for the purposes of majority voting in political

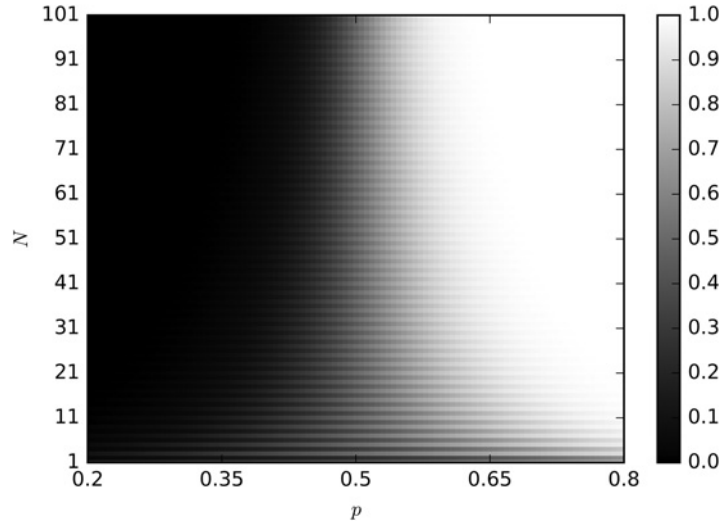


FIGURE 6.2 Accuracy of a bagging classifier as a function of the individual estimator's accuracy (P), the number of estimators (N), and $k = 2$

science, the problem addressed by this theorem shares similarities with the above discussion.

6.3.3 Observation Redundancy

In Chapter 4 we studied one reason why financial observations cannot be assumed to be IID. Redundant observations have two detrimental effects on bagging. First, the samples drawn with replacement are more likely to be virtually identical, even if they do not share the same observations. This makes $\bar{\rho} \approx 1$, and bagging will not reduce variance, regardless of N. For example, if each observation at t is labeled according to the return between t and $t + 100$, we should sample 1% of the observations per bagged estimator, but not more. Chapter 4, Section 4.5 recommended three alternative solutions, one of which consisted of setting `max_samples=out['tW'].mean()` in sklearn's implementation of the bagging classifier class. Another (better) solution was to apply the sequential bootstrap method.

The second detrimental effect from observation redundancy is that out-of-bag accuracy will be inflated. This happens because random sampling with replacement places in the training set samples that are very similar to those out-of-bag. In such a case, a proper stratified k-fold cross-validation without shuffling before partitioning will show a much lower testing-set accuracy than the one estimated out-of-bag. For this reason, it is advisable to set `StratifiedKFold(n_splits=k, shuffle=False)` when using that sklearn class, cross-validate the bagging classifier, and ignore the out-of-bag accuracy results. A low number k is preferred to a high

one, as excessive partitioning would again place in the testing set samples too similar to those used in the training set.

6.4 RANDOM FOREST

Decision trees are known to be prone to overfitting, which increases the variance of the forecasts.[3] In order to address this concern, the random forest (RF) method was designed to produce ensemble forecasts with lower variance.

RF shares some similarities with bagging, in the sense of training independently individual estimators over bootstrapped subsets of the data. The key difference with bagging is that random forests incorporate a second level of randomness: When optimizing each node split, only a random subsample (without replacement) of the attributes will be evaluated, with the purpose of further decorrelating the estimators.

Like bagging, RF reduces forecasts' variance without overfitting (remember, as long as $\bar{\rho} < 1$). A second advantage is that RF evaluates feature importance, which we will discuss in depth in Chapter 8. A third advantage is that RF provides out-of-bag accuracy estimates, however in financial applications they are likely to be inflated (as discussed in Section 6.3.3). But like bagging, RF will not necessarily exhibit lower bias than individual decision trees.

If a large number of samples are redundant (non-IID), overfitting will still take place: Sampling randomly with replacement will build a large number of essentially identical trees ($\bar{\rho} \approx 1$), where each decision tree is overfit (a flaw for which decision trees are notorious). Unlike bagging, RF always fixes the size of the bootstrapped samples to match the size of the training dataset. Let us review ways we can address this RF overfitting problem in sklearn. For illustration purposes, I will refer to sklearn's classes; however, these solutions can be applied to any implementation:

1. Set a parameter `max_features` to a lower value, as a way of forcing discrepancy between trees.
2. Early stopping: Set the regularization parameter `min_weight_fraction_leaf` to a sufficiently large value (e.g., 5%) such that out-of-bag accuracy converges to out-of-sample (k-fold) accuracy.
3. Use `BaggingClassifier` on `DecisionTreeClassifier` where `max_samples` is set to the average uniqueness (`avgU`) between samples.
   (a) `clf=DecisionTreeClassifier(criterion='entropy',max_features='auto',class_weight='balanced')`
   (b) `bc=BaggingClassifier(base_estimator=clf,n_estimators=1000,max_samples=avgU,max_features=1.)`
4. Use `BaggingClassifier` on `RandomForestClassifier` where `max_samples` is set to the average uniqueness (`avgU`) between samples.

[3] For an intuitive explanation of Random Forest, visit the following link: https://quantdare.com/random-forest-many-is-better-than-one/.

(a) `clf=RandomForestClassifier(n_estimators=1,criterion='entropy',bootstrap=False,class_weight='balanced_subsample')`

(b) `bc=BaggingClassifier(base_estimator=clf,n_estimators=1000,max_samples=avgU,max_features=1.)`

5. Modify the RF class to replace standard bootstrapping with sequential bootstrapping.

In summary, Snippet 6.2 demonstrates three alternative ways of setting up an RF, using different classes.

SNIPPET 6.2 THREE WAYS OF SETTING UP AN RF

```
clf0=RandomForestClassifier(n_estimators=1000,class_weight='balanced_subsample',
    criterion='entropy')
clf1=DecisionTreeClassifier(criterion='entropy',max_features='auto',
    class_weight='balanced')
clf1=BaggingClassifier(base_estimator=clf1,n_estimators=1000,max_samples=avgU)
clf2=RandomForestClassifier(n_estimators=1,criterion='entropy',bootstrap=False,
    class_weight='balanced_subsample')
clf2=BaggingClassifier(base_estimator=clf2,n_estimators=1000,max_samples=avgU,
    max_features=1.)
```

When fitting decision trees, a rotation of the features space in a direction that aligns with the axes typically reduces the number of levels needed by the tree. For this reason, I suggest you fit RF on a PCA of the features, as that may speed up calculations and reduce some overfitting (more on this in Chapter 8). Also, as discussed in Chapter 4, Section 4.8, `class_weight='balanced_subsample'` will help you prevent the trees from misclassifying minority classes.

6.5 BOOSTING

Kearns and Valiant [1989] were among the first to ask whether one could combine weak estimators in order to achieve one with high accuracy. Shortly after, Schapire [1990] demonstrated that the answer to that question was affirmative, using the procedure we today call boosting. In general terms, it works as follows: First, generate one training set by random sampling with replacement, according to some sample weights (initialized with uniform weights). Second, fit one estimator using that training set. Third, if the single estimator achieves an accuracy greater than the acceptance threshold (e.g., 50% in a binary classifier, so that it performs better than chance), the estimator is kept, otherwise it is discarded. Fourth, give more weight to misclassified observations, and less weight to correctly classified observations. Fifth, repeat the previous steps until N estimators are produced. Sixth, the ensemble forecast is the *weighted* average of the individual forecasts from the N models, where the weights

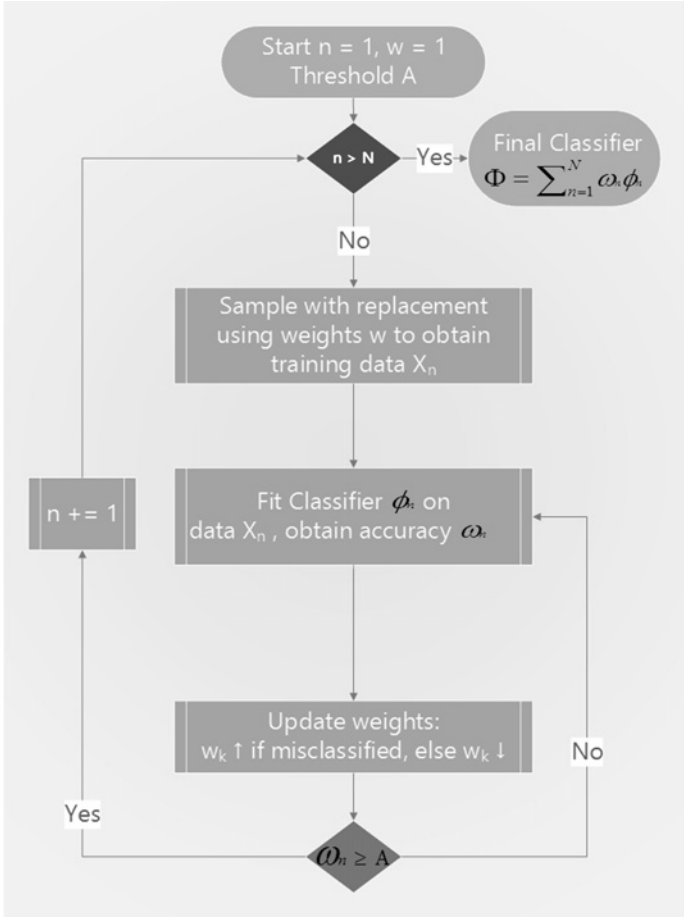


FIGURE 6.3 AdaBoost decision flow

are determined by the accuracy of the individual estimators. There are many boosting algorithms, of which AdaBoost is one of the most popular (Geron [2017]). Figure 6.3 summarizes the decision flow of a standard AdaBoost implementation.

6.6 BAGGING VS. BOOSTING IN FINANCE

From the above description, a few aspects make boosting quite different from bagging:[4]

- Individual classifiers are fit sequentially.
- Poor-performing classifiers are dismissed.

[4] For a visual explanation of the difference between bagging and boosting, visit: https://quantdare.com/what-is-the-difference-between-bagging-and-boosting/.

- Observations are weighted differently in each iteration.
- The ensemble forecast is a weighted average of the individual learners.

Boosting's main advantage is that it reduces both variance and bias in forecasts. However, correcting bias comes at the cost of greater risk of overfitting. It could be argued that in financial applications bagging is generally preferable to boosting. Bagging addresses overfitting, while boosting addresses underfitting. Overfitting is often a greater concern than underfitting, as it is not difficult to overfit an ML algorithm to financial data, because of the low signal-to-noise ratio. Furthermore, bagging can be parallelized, while generally boosting requires sequential running.

6.7 BAGGING FOR SCALABILITY

As you know, several popular ML algorithms do not scale well with the sample size. Support vector machines (SVMs) are a prime example. If you attempt to fit an SVM on a million observations, it may take a while until the algorithm converges. And even once it has converged, there is no guarantee that the solution is a global optimum, or that it is not overfit.

One practical approach is to build a bagging algorithm, where the base estimator belongs to a class that does not scale well with the sample size, like SVM. When defining that base estimator, we will impose a tight early stopping condition. For example, in sklearn's SVM implementation, you could set a low value for the `max_iter` parameter, say 1E5 iterations. The default value is `max_iter=-1`, which tells the estimator to continue performing iterations until errors fall below a tolerance level. Alternatively, you could raise the tolerance level through the parameter `tol`, which has a default value `tol=1E-3`. Either of these two parameters will force an early stop. You can stop other algorithms early with equivalent parameters, like the number of levels in an RF (`max_depth`), or the minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node (`min_weight_fraction_leaf`).

Given that bagging algorithms can be parallelized, we are transforming a large sequential task into many smaller ones that are run simultaneously. Of course, the early stopping will increase the variance of the outputs from the individual base estimators; however, that increase can be more than offset by the variance reduction associated with the bagging algorithm. You can control that reduction by adding more independent base estimators. Used in this way, bagging will allow you to achieve fast and robust estimates on extremely large datasets.

EXERCISES

6.1 Why is bagging based on random sampling with replacement? Would bagging still reduce a forecast's variance if sampling were without replacement?

6.2 Suppose that your training set is based on highly overlap labels (i.e., with low uniqueness, as defined in Chapter 4).

(a) Does this make bagging prone to overfitting, or just ineffective? Why?

(b) Is out-of-bag accuracy generally reliable in financial applications? Why?

6.3 Build an ensemble of estimators, where the base estimator is a decision tree.

(a) How is this ensemble different from an RF?

(b) Using sklearn, produce a bagging classifier that behaves like an RF. What parameters did you have to set up, and how?

6.4 Consider the relation between an RF, the number of trees it is composed of, and the number of features utilized:

(a) Could you envision a relation between the minimum number of trees needed in an RF and the number of features utilized?

(b) Could the number of trees be too small for the number of features used?

(c) Could the number of trees be too high for the number of observations available?

6.5 How is out-of-bag accuracy different from stratified k-fold (with shuffling) cross-validation accuracy?

REFERENCES

Geron, A. (2017): *Hands-on Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*, 1st edition. O'Reilly Media.

Kearns, M. and L. Valiant (1989): "Cryptographic limitations on learning Boolean formulae and finite automata." In Proceedings of the 21st Annual ACM Symposium on Theory of Computing, pp. 433–444, New York. Association for Computing Machinery.

Schapire, R. (1990): "The strength of weak learnability." *Machine Learning*. Kluwer Academic Publishers. Vol. 5 No. 2, pp. 197–227.

BIBLIOGRAPHY

Gareth, J., D. Witten, T. Hastie, and R. Tibshirani (2013): *An Introduction to Statistical Learning: With Applications in R*, 1st ed. Springer-Verlag.

Hackeling, G. (2014): *Mastering Machine Learning with Scikit-Learn*, 1st ed. Packt Publishing.

Hastie, T., R. Tibshirani and J. Friedman (2016): *The Elements of Statistical Learning*, 2nd ed. Springer-Verlag.

Hauck, T. (2014): *Scikit-Learn Cookbook*, 1st ed. Packt Publishing.

Raschka, S. (2015): *Python Machine Learning*, 1st ed. Packt Publishing.

CHAPTER 7

Cross-Validation in Finance

7.1 MOTIVATION

The purpose of cross-validation (CV) is to determine the generalization error of an ML algorithm, so as to prevent overfitting. CV is yet another instance where standard ML techniques fail when applied to financial problems. Overfitting will take place, and CV will not be able to detect it. In fact, CV will contribute to overfitting through hyper-parameter tuning. In this chapter we will learn why standard CV fails in finance, and what can be done about it.

7.2 THE GOAL OF CROSS-VALIDATION

One of the purposes of ML is to learn the general structure of the data, so that we can produce predictions on future, unseen features. When we test an ML algorithm on the same dataset as was used for training, not surprisingly, we achieve spectacular results. When ML algorithms are misused that way, they are no different from file lossy-compression algorithms: They can summarize the data with extreme fidelity, yet with zero forecasting power.

CV splits observations drawn from an IID process into two sets: the *training* set and the *testing* set. Each observation in the complete dataset belongs to one, and only one, set. This is done as to prevent leakage from one set into the other, since that would defeat the purpose of testing on unseen data. Further details can be found in the books and articles listed in the references section.

There are many alternative CV schemes, of which one of the most popular is k-fold CV. Figure 7.1 illustrates the k train/test splits carried out by a k-fold CV, where $k = 5$. In this scheme:

1. The dataset is partitioned into k subsets.
2. For $i = 1,\ldots,k$

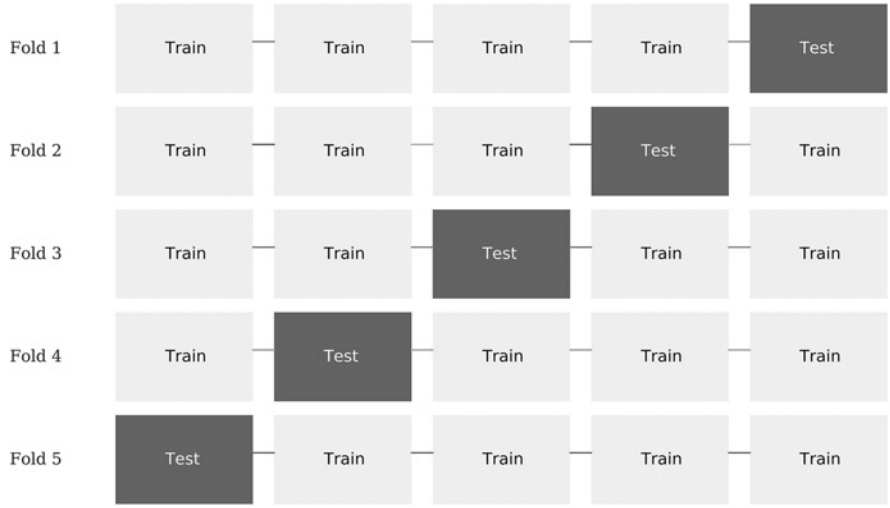


FIGURE 7.1 Train/test splits in a 5-fold CV scheme

(a) The ML algorithm is trained on all subsets excluding *i*.

(b) The fitted ML algorithm is tested on *i*.

The outcome from k-fold CV is a *kx1* array of cross-validated performance metrics. For example, in a binary classifier, the model is deemed to have learned something if the cross-validated accuracy is over 1/2, since that is the accuracy we would achieve by tossing a fair coin.

In finance, CV is typically used in two settings: model development (like hyperparameter tuning) and backtesting. Backtesting is a complex subject that we will discuss thoroughly in Chapters 10–16. In this chapter, we will focus on CV for model development.

7.3 WHY K-FOLD CV FAILS IN FINANCE

By now you may have read quite a few papers in finance that present k-fold CV evidence that an ML algorithm performs well. Unfortunately, it is almost certain that those results are wrong. One reason k-fold CV fails in finance is because observations cannot be assumed to be drawn from an IID process. A second reason for CV's failure is that the testing set is used multiple times in the process of developing a model, leading to multiple testing and selection bias. We will revisit this second cause of failure in Chapters 11–13. For the time being, let us concern ourselves exclusively with the first cause of failure.

Leakage takes place when the training set contains information that also appears in the testing set. Consider a serially correlated feature X that is associated with labels Y that are formed on overlapping data:

- Because of the serial correlation, $X_t \approx X_{t+1}$.
- Because labels are derived from overlapping datapoints, $Y_t \approx Y_{t+1}$.

By placing t and $t + 1$ in different sets, information is leaked. When a classifier is first trained on (X_t, Y_t), and then it is asked to predict $\mathrm{E}[Y_{t+1}|X_{t+1}]$ based on an observed X_{t+1}, this classifier is more likely to achieve $Y_{t+1} = \mathrm{E}[Y_{t+1}|X_{t+1}]$ even if X is an irrelevant feature.

If X is a predictive feature, leakage will enhance the performance of an already valuable strategy. The problem is leakage in the presence of irrelevant features, as this leads to false discoveries. There are at least two ways to reduce the likelihood of leakage:

1. Drop from the training set any observation i where Y_i is a function of information used to determine Y_j, and j belongs to the testing set.
   (a) For example, Y_i and Y_j should not span overlapping periods (see Chapter 4 for a discussion of sample uniqueness).
2. Avoid overfitting the classifier. In this way, even if some leakage occurs, the classifier will not be able to profit from it. Use:
   (a) Early stopping of the base estimators (see Chapter 6).
   (b) Bagging of classifiers, while controlling for oversampling on redundant examples, so that the individual classifiers are as diverse as possible.
      i. Set `max_samples` to the average uniqueness.
      ii. Apply sequential bootstrap (Chapter 4).

Consider the case where X_i and X_j are formed on overlapping information, where i belongs to the training set and j belongs to the testing set. Is this a case of informational leakage? Not necessarily, as long as Y_i and Y_j are independent. For leakage to take place, it must occur that $(X_i, Y_i) \approx (X_j, Y_j)$, and it does not suffice that $X_i \approx X_j$ or even $Y_i \approx Y_j$.

7.4 A SOLUTION: PURGED K-FOLD CV

One way to reduce leakage is to purge from the training set all observations whose labels overlapped in time with those labels included in the testing set. I call this process "purging." In addition, since financial features often incorporate series that exhibit serial correlation (like ARMA processes), we should eliminate from the training set observations that immediately follow an observation in the testing set. I call this process "embargo."

7.4.1 Purging the Training Set

Suppose a testing observation whose label Y_j is decided based on the information set Φ_j. In order to prevent the type of leakage described in the previous section, we would

like to purge from the training set any observation whose label Y_i is decided based on the information set Φ_i, such that $\Phi_i \cap \Phi_j = \emptyset$.

In particular, we will determine that there is informational overlap between two observations i and j whenever Y_i and Y_j are concurrent (see Chapter 4, Section 4.3), in the sense that both labels are contingent on at least one common random draw. For example, consider a label Y_j that is a function of observations in the closed range $t \in [t_{j,0}, t_{j,1}]$, $Y_j = f[[t_{j,0}, t_{j,1}]]$ (with some abuse of notation). For example, in the context of the triple-barrier labeling method (Chapter 3), it means that the label is the sign of the return spanning between price bars with indices $t_{j,0}$ and $t_{j,1}$, that is $\text{sgn}[r_{t_{j,0},t_{j,1}}]$. A label $Y_i = f[[t_{i,0}, t_{i,1}]]$ overlaps with Y_j if any of the three sufficient conditions is met:

1. $t_{j,0} \leq t_{i,0} \leq t_{j,1}$
2. $t_{j,0} \leq t_{i,1} \leq t_{j,1}$
3. $t_{i,0} \leq t_{j,0} \leq t_{j,1} \leq t_{i,1}$

Snippet 7.1 implements this purging of observations from the training set. If the testing set is contiguous, in the sense that no training observations occur between the first and last testing observation, then purging can be accelerated: The object `testTimes` can be a pandas series with a single item, spanning the entire testing set.

SNIPPET 7.1 PURGING OBSERVATION IN THE TRAINING SET

```
def getTrainTimes(t1,testTimes):
    '''
    Given testTimes, find the times of the training observations.
    —t1.index: Time when the observation started.
    —t1.value: Time when the observation ended.
    —testTimes: Times of testing observations.
    '''
    trn=t1.copy(deep=True)
    for i,j in testTimes.iteritems():
        df0=trn[(i<=trn.index)&(trn.index<=j)].index # train starts within test
        df1=trn[(i<=trn)&(trn<=j)].index # train ends within test
        df2=trn[(trn.index<=i)&(j<=trn)].index # train envelops test
        trn=trn.drop(df0.union(df1).union(df2))
    return trn
```

When leakage takes place, performance improves merely by increasing $k \to T$, where T is the number of bars. The reason is that the larger the number of testing splits, the greater the number of overlapping observations in the training set. In many cases, purging suffices to prevent leakage: Performance will improve as we increase k, because we allow the model to recalibrate more often. But beyond a certain value

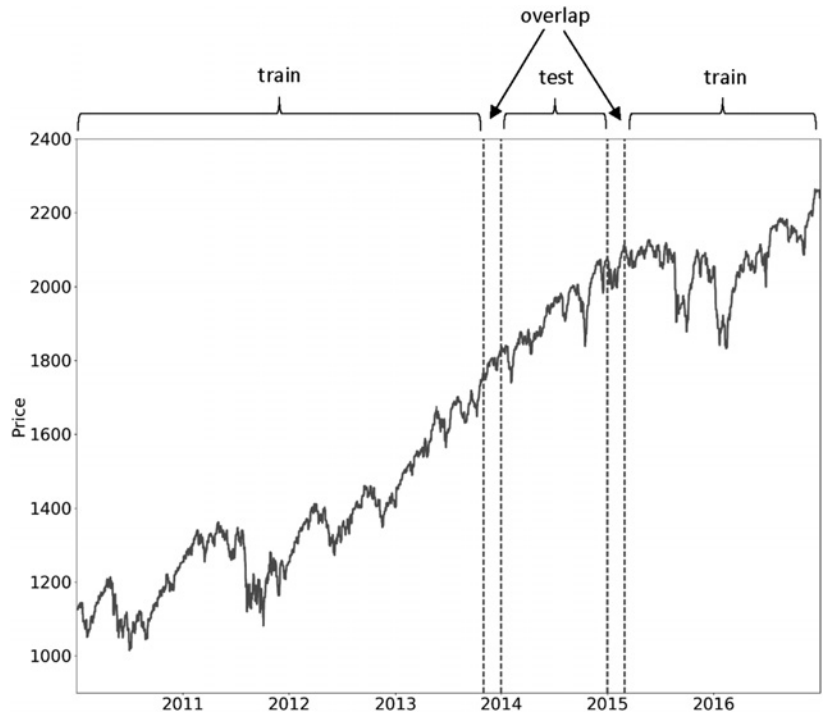


FIGURE 7.2 Purging overlap in the training set

k^*, performance will not improve, indicating that the backtest is not profiting from leaks. Figure 7.2 plots one partition of the k-fold CV. The test set is surrounded by two train sets, generating two overlaps that must be purged to prevent leakage.

7.4.2 Embargo

For those cases where purging is not able to prevent all leakage, we can impose an embargo on training observations *after* every test set. The embargo does not need to affect training observations prior to a test set, because training labels $Y_i = f[[t_{i,0}, t_{i,1}]]$, where $t_{i,1} < t_{j,0}$ (training ends before testing begins), contain information that was available at the testing time $t_{j,0}$. In other words, we are only concerned with training labels $Y_i = f[[t_{i,0}, t_{i,1}]]$ that take place immediately after the test, $t_{j,1} \leq t_{i,0} \leq t_{j,1} + h$. We can implement this embargo period h by setting $Y_j = f[[t_{j,0}, t_{j,1} + h]]$ before purging. A small value $h \approx .01T$ often suffices to prevent all leakage, as can be confirmed by testing that performance does not improve indefinitely by increasing $k \to T$. Figure 7.3 illustrates the embargoing of train observations immediately after the testing set. Snippet 7.2 implements the embargo logic.

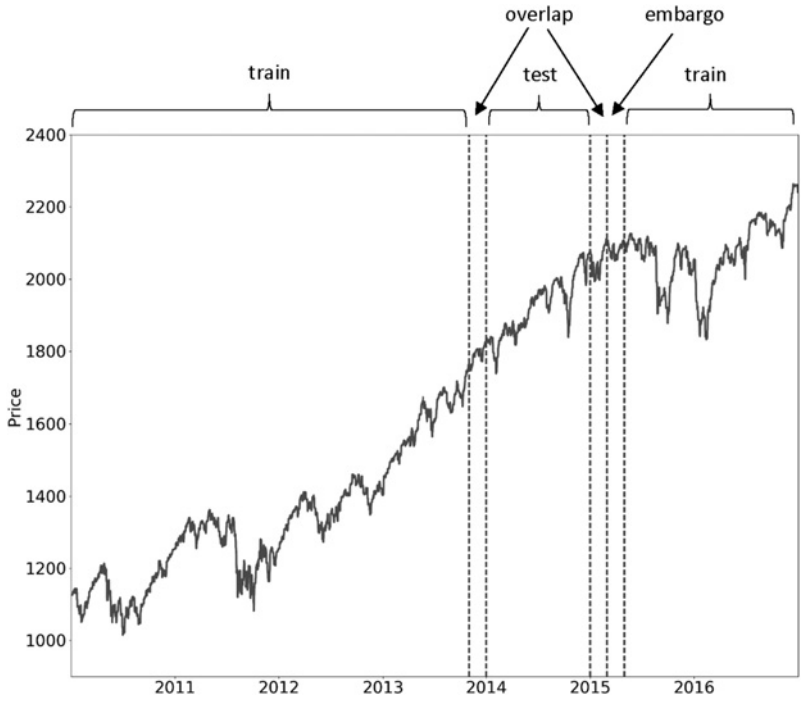


FIGURE 7.3 Embargo of post-test train observations

SNIPPET 7.2 EMBARGO ON TRAINING OBSERVATIONS

```
def getEmbargoTimes(times,pctEmbargo):
    # Get embargo time for each bar
    step=int(times.shape[0]*pctEmbargo)
    if step==0:
        mbrg=pd.Series(times,index=times)
    else:
        mbrg=pd.Series(times[step:],index=times[:-step])
        mbrg=mbrg.append(pd.Series(times[-1],index=times[-step:]))
    return mbrg
#——————————————————————————————————
testTimes=pd.Series(mbrg[dt1],index=[dt0]) # include embargo before purge
trainTimes=getTrainTimes(t1,testTimes)
testTimes=t1.loc[dt0:dt1].index
```

7.4.3 The Purged K-Fold Class

In the previous sections we have discussed how to produce training/testing splits when labels overlap. That introduced the notion of purging and embargoing, in the

particular context of model development. In general, we need to purge and embargo overlapping training observations whenever we produce a train/test split, whether it is for hyper-parameter fitting, backtesting, or performance evaluation. Snippet 7.3 extends scikit-learn's `KFold` class to account for the possibility of leakages of testing information into the training set.

SNIPPET 7.3 CROSS-VALIDATION CLASS WHEN OBSERVATIONS OVERLAP

```
class PurgedKFold(_BaseKFold):
    '''
    Extend KFold class to work with labels that span intervals
    The train is purged of observations overlapping test-label intervals
    Test set is assumed contiguous (shuffle=False), w/o training samples in between
    '''
    def __init__(self,n_splits=3,t1=None,pctEmbargo=0.):
        if not isinstance(t1,pd.Series):
            raise ValueError('Label Through Dates must be a pd.Series')
        super(PurgedKFold,self).__init__(n_splits,shuffle=False,random_state=None)
        self.t1=t1
        self.pctEmbargo=pctEmbargo
    def split(self,X,y=None,groups=None):
        if (X.index==self.t1.index).sum()!=len(self.t1):
            raise ValueError('X and ThruDateValues must have the same index')
        indices=np.arange(X.shape[0])
        mbrg=int(X.shape[0]*self.pctEmbargo)
        test_starts=[(i[0],i[-1]+1) for i in \
            np.array_split(np.arange(X.shape[0]),self.n_splits)]
        for i,j in test_starts:
            t0=self.t1.index[i] # start of test set
            test_indices=indices[i:j]
            maxT1Idx=self.t1.index.searchsorted(self.t1[test_indices].max())
            train_indices=self.t1.index.searchsorted(self.t1[self.t1<=t0].index)
            if maxT1Idx<X.shape[0]: # right train (with embargo)
                train_indices=np.concatenate((train_indices,indices[maxT1Idx+mbrg:]))
            yield train_indices,test_indices
```

7.5 BUGS IN SKLEARN'S CROSS-VALIDATION

You would think that something as critical as cross-validation would be perfectly implemented in one of the most popular ML libraries. Unfortunately that is not the case, and this is one of the reasons you must always read all the code you run, and a strong point in favor of open source. One of the many upsides of open-source code is

that you can verify everything and adjust it to your needs. Snippet 7.4 addresses two known sklearn bugs:

1. Scoring functions do not know `classes_`, as a consequence of sklearn's reliance on numpy arrays rather than pandas series: https://github.com/scikit-learn/scikit-learn/issues/6231
2. `cross_val_score` will give different results because it passes weights to the fit method, but not to the `log_loss` method: https://github.com/scikit-learn/scikit-learn/issues/9144

SNIPPET 7.4 USING THE `PurgedKFold` CLASS

```
def cvScore(clf,X,y,sample_weight,scoring='neg_log_loss',t1=None,cv=None,cvGen=None,
            pctEmbargo=None):
    if scoring not in ['neg_log_loss','accuracy']:
        raise Exception('wrong scoring method.')
    from sklearn.metrics import log_loss,accuracy_score
    from clfSequential import PurgedKFold
    if cvGen is None:
        cvGen=PurgedKFold(n_splits=cv,t1=t1,pctEmbargo=pctEmbargo) # purged
    score=[]
    for train,test in cvGen.split(X=X):
        fit=clf.fit(X=X.iloc[train,:],y=y.iloc[train],
                    sample_weight=sample_weight.iloc[train].values)
        if scoring=='neg_log_loss':
            prob=fit.predict_proba(X.iloc[test,:])
            score_=-log_loss(y.iloc[test],prob,
                sample_weight=sample_weight.iloc[test].values,labels=clf.classes_)
        else:
            pred=fit.predict(X.iloc[test,:])
            score_=accuracy_score(y.iloc[test],pred,sample_weight= \
                sample_weight.iloc[test].values)
        score.append(score_)
    return np.array(score)
```

Please understand that it may take a long time until a fix for these bugs is agreed upon, implemented, tested, and released. Until then, you should use `cvScore` in Snippet 7.4, and avoid running the function `cross_val_score`.

EXERCISES

7.1 Why is shuffling a dataset before conducting k-fold CV generally a bad idea in finance? What is the purpose of shuffling? Why does shuffling defeat the purpose of k-fold CV in financial datasets?

7.2 Take a pair of matrices (X, y), representing observed features and labels. These could be one of the datasets derived from the exercises in Chapter 3.

(a) Derive the performance from a 10-fold CV of an RF classifier on (X, y), without shuffling.

(b) Derive the performance from a 10-fold CV of an RF on (X, y), with shuffling.

(c) Why are both results so different?

(d) How does shuffling leak information?

7.3 Take the same pair of matrices (X, y) you used in exercise 2.

(a) Derive the performance from a 10-fold purged CV of an RF on (X, y), with 1% embargo.

(b) Why is the performance lower?

(c) Why is this result more realistic?

7.4 In this chapter we have focused on one reason why k-fold CV fails in financial applications, namely the fact that some information from the testing set leaks into the training set. Can you think of a second reason for CV's failure?

7.5 Suppose you try one thousand configurations of the same investment strategy, and perform a CV on each of them. Some results are guaranteed to look good, just by sheer luck. If you only publish those positive results, and hide the rest, your audience will not be able to deduce that these results are false positives, a statistical fluke. This phenomenon is called "selection bias."

(a) Can you imagine one procedure to prevent this?

(b) What if we split the dataset in three sets: training, validation, and testing? The validation set is used to evaluate the trained parameters, and the testing is run only on the one configuration chosen in the validation phase. In what case does this procedure still fail?

(c) What is the key to avoiding selection bias?

BIBLIOGRAPHY

Bharat Rao, R., G. Fung, and R. Rosales (2008): "On the dangers of cross-validation: An experimental evaluation." White paper, IKM CKS Siemens Medical Solutions USA. Available at http://people.csail.mit.edu/romer/papers/CrossVal_SDM08.pdf.

Bishop, C. (1995): *Neural Networks for Pattern Recognition*, 1st ed. Oxford University Press.

Breiman, L. and P. Spector (1992): "Submodel selection and evaluation in regression: The X-random case." White paper, Department of Statistics, University of California, Berkeley. Available at http://digitalassets.lib.berkeley.edu/sdtr/ucb/text/197.pdf.

Hastie, T., R. Tibshirani, and J. Friedman (2009): *The Elements of Statistical Learning*, 1st ed. Springer.

James, G., D. Witten, T. Hastie and R. Tibshirani (2013): *An Introduction to Statistical Learning*, 1st ed. Springer.

Kohavi, R. (1995): "A study of cross-validation and bootstrap for accuracy estimation and model selection." International Joint Conference on Artificial Intelligence. Available at http://web.cs.iastate.edu/~jtian/cs573/Papers/Kohavi-IJCAI-95.pdf.

Ripley, B. (1996): *Pattern Recognition and Neural Networks*, 1st ed. Cambridge University Press.

CHAPTER 8

Feature Importance

8.1 MOTIVATION

One of the most pervasive mistakes in financial research is to take some data, run it through an ML algorithm, backtest the predictions, and repeat the sequence until a nice-looking backtest shows up. Academic journals are filled with such pseudo-discoveries, and even large hedge funds constantly fall into this trap. It does not matter if the backtest is a walk-forward out-of-sample. The fact that we are repeating a test over and over on the same data will likely lead to a false discovery. This methodological error is so notorious among statisticians that they consider it scientific fraud, and the American Statistical Association warns against it in its ethical guidelines (American Statistical Association [2016], Discussion #4). It typically takes about 20 such iterations to discover a (false) investment strategy subject to the standard significance level (false positive rate) of 5%. In this chapter we will explore why such an approach is a waste of time and money, and how feature importance offers an alternative.

8.2 THE IMPORTANCE OF FEATURE IMPORTANCE

A striking facet of the financial industry is that so many very seasoned portfolio managers (including many with a quantitative background) do not realize how easy it is to overfit a backtest. How to backtest properly is not the subject of this chapter; we will address that extremely important topic in Chapters 11–15. The goal of this chapter is to explain one of the analyses that must be performed *before* any backtest is carried out.

Suppose that you are given a pair of matrices (X, y), that respectively contain features and labels for a particular financial instrument. We can fit a classifier on (X, y) and evaluate the generalization error through a purged k-fold cross-validation (CV), as we saw in Chapter 7. Suppose that we achieve good performance. The next

natural question is to try to understand what features contributed to that performance. Maybe we could add some features that strengthen the signal responsible for the classifier's predictive power. Maybe we could eliminate some of the features that are only adding noise to the system. Notably, understanding feature importance opens up the proverbial black box. We can gain insight into the patterns identified by the classifier if we understand what source of information is indispensable to it. This is one of the reasons why the black box mantra is somewhat overplayed by the ML skeptics. Yes, the algorithm has learned without us directing the process (that is the whole point of ML!) in a black box, but that does not mean that we cannot (or should not) take a look at what the algorithm has found. Hunters do not blindly eat everything their smart dogs retrieve for them, do they?

Once we have found what features are important, we can learn more by conducting a number of experiments. Are these features important all the time, or only in some specific environments? What triggers a change in importance over time? Can those regime switches be predicted? Are those important features also relevant to other related financial instruments? Are they relevant to other asset classes? What are the most relevant features across all financial instruments? What is the subset of features with the highest rank correlation across the entire investment universe? This is a much better way of researching strategies than the foolish backtest cycle. Let me state this maxim as one of the most critical lessons I hope you learn from this book:

SNIPPET 8.1 MARCOS' FIRST LAW OF BACKTESTING—IGNORE AT YOUR OWN PERIL

"Backtesting is not a research tool. Feature importance is."

—Marcos López de Prado
Advances in Financial Machine Learning (2018)

8.3 FEATURE IMPORTANCE WITH SUBSTITUTION EFFECTS

I find it useful to distinguish between feature importance methods based on whether they are impacted by substitution effects. In this context, a substitution effect takes place when the estimated importance of one feature is reduced by the presence of other related features. Substitution effects are the ML analogue of what the statistics and econometrics literature calls "multi-collinearity." One way to address linear substitution effects is to apply PCA on the raw features, and then perform the feature importance analysis on the orthogonal features. See Belsley et al. [1980], Goldberger [1991, pp. 245–253], and Hill et al. [2001] for further details.

8.3.1 Mean Decrease Impurity

Mean decrease impurity (MDI) is a fast, explanatory-importance (in-sample, IS) method specific to tree-based classifiers, like RF. At each node of each decision tree, the selected feature splits the subset it received in such a way that impurity is

decreased. Therefore, we can derive for each decision tree how much of the overall impurity decrease can be assigned to each feature. And given that we have a forest of trees, we can average those values across all estimators and rank the features accordingly. See Louppe et al. [2013] for a detailed description. There are some important considerations you must keep in mind when working with MDI:

1. Masking effects take place when some features are systematically ignored by tree-based classifiers in favor of others. In order to avoid them, set `max_features=int(1)` when using sklearn's RF class. In this way, only one random feature is considered per level.
   (a) Every feature is given a chance (at some random levels of some random trees) to reduce impurity.
   (b) Make sure that features with zero importance are not averaged, since the only reason for a 0 is that the feature was not randomly chosen. Replace those values with `np.nan`.
2. The procedure is obviously IS. Every feature will have some importance, even if they have no predictive power whatsoever.
3. MDI cannot be generalized to other non-tree based classifiers.
4. By construction, MDI has the nice property that feature importances add up to 1, and every feature importance is bounded between 0 and 1.
5. The method does not address substitution effects in the presence of correlated features. MDI dilutes the importance of substitute features, because of their interchangeability: The importance of two identical features will be halved, as they are randomly chosen with equal probability.
6. Strobl et al. [2007] show experimentally that MDI is biased towards some predictor variables. White and Liu [1994] argue that, in case of single decision trees, this bias is due to an unfair advantage given by popular impurity functions toward predictors with a large number of categories.

Sklearn's `RandomForest` class implements MDI as the default feature importance score. This choice is likely motivated by the ability to compute MDI on the fly, with minimum computational cost.[1] Snippet 8.2 illustrates an implementation of MDI, incorporating the considerations listed earlier.

SNIPPET 8.2 MDI FEATURE IMPORTANCE

```
def featImpMDI(fit,featNames):
    # feat importance based on IS mean impurity reduction
    df0={i:tree.feature_importances_ for i,tree in enumerate(fit.estimators_)}
    df0=pd.DataFrame.from_dict(df0,orient='index')
    df0.columns=featNames
    df0=df0.replace(0,np.nan) # because max_features=1
```

[1] http://blog.datadive.net/selecting-good-features-part-iii-random-forests/.

```
    imp=pd.concat({'mean':df0.mean(),'std':df0.std()*df0.shape[0]**-.5},axis=1)
    imp/=imp['mean'].sum()
    return imp
```

8.3.2 Mean Decrease Accuracy

Mean decrease accuracy (MDA) is a slow, predictive-importance (out-of-sample, OOS) method. First, it fits a classifier; second, it derives its performance OOS according to some performance score (accuracy, negative log-loss, etc.); third, it permutates each column of the features matrix (X), one column at a time, deriving the performance OOS after each column's permutation. The importance of a feature is a function of the loss in performance caused by its column's permutation. Some relevant considerations include:

1. This method can be applied to any classifier, not only tree-based classifiers.
2. MDA is not limited to accuracy as the sole performance score. For example, in the context of meta-labeling applications, we may prefer to score a classifier with F1 rather than accuracy (see Chapter 14, Section 14.8 for an explanation). That is one reason a better descriptive name would have been "permutation importance." When the scoring function does not correspond to a metric space, MDA results should be used as a ranking.
3. Like MDI, the procedure is also susceptible to substitution effects in the presence of correlated features. Given two identical features, MDA always considers one to be redundant to the other. Unfortunately, MDA will make both features appear to be outright irrelevant, even if they are critical.
4. Unlike MDI, it is possible that MDA concludes that all features are unimportant. That is because MDA is based on OOS performance.
5. The CV must be purged and embargoed, for the reasons explained in Chapter 7.

Snippet 8.3 implements MDA feature importance with sample weights, with purged k-fold CV, and with scoring by negative log-loss or accuracy. It measures MDA importance as a function of the improvement (from permutating to not permutating the feature), relative to the maximum possible score (negative log-loss of 0, or accuracy of 1). Note that, in some cases, the improvement may be negative, meaning that the feature is actually detrimental to the forecasting power of the ML algorithm.

SNIPPET 8.3 MDA FEATURE IMPORTANCE

```
def featImpMDA(clf,X,y,cv,sample_weight,t1,pctEmbargo,scoring='neg_log_loss'):
    # feat importance based on OOS score reduction
    if scoring not in ['neg_log_loss','accuracy']:
        raise Exception('wrong scoring method.')
    from sklearn.metrics import log_loss,accuracy_score
    cvGen=PurgedKFold(n_splits=cv,t1=t1,pctEmbargo=pctEmbargo) # purged cv
    scr0,scr1=pd.Series(),pd.DataFrame(columns=X.columns)
```

```
for i,(train,test) in enumerate(cvGen.split(X=X)):
    X0,y0,w0=X.iloc[train,:],y.iloc[train],sample_weight.iloc[train]
    X1,y1,w1=X.iloc[test,:],y.iloc[test],sample_weight.iloc[test]
    fit=clf.fit(X=X0,y=y0,sample_weight=w0.values)
    if scoring=='neg_log_loss':
        prob=fit.predict_proba(X1)
        scr0.loc[i]=-log_loss(y1,prob,sample_weight=w1.values,
                              labels=clf.classes_)
    else:
        pred=fit.predict(X1)
        scr0.loc[i]=accuracy_score(y1,pred,sample_weight=w1.values)
    for j in X.columns:
        X1_=X1.copy(deep=True)
        np.random.shuffle(X1_[j].values) # permutation of a single column
        if scoring=='neg_log_loss':
            prob=fit.predict_proba(X1_)
            scr1.loc[i,j]=-log_loss(y1,prob,sample_weight=w1.values,
                                    labels=clf.classes_)
        else:
            pred=fit.predict(X1_)
            scr1.loc[i,j]=accuracy_score(y1,pred,sample_weight=w1.values)
imp=(-scr1).add(scr0,axis=0)
if scoring=='neg_log_loss':imp=imp/-scr1
else:imp=imp/(1.-scr1)
imp=pd.concat({'mean':imp.mean(),'std':imp.std()*imp.shape[0]**-.5},axis=1)
return imp,scr0.mean()
```

8.4 FEATURE IMPORTANCE WITHOUT SUBSTITUTION EFFECTS

Substitution effects can lead us to discard important features that happen to be redundant. This is not generally a problem in the context of prediction, but it could lead us to wrong conclusions when we are trying to understand, improve, or simplify a model. For this reason, the following single feature importance method can be a good complement to MDI and MDA.

8.4.1 Single Feature Importance

Single feature importance (SFI) is a cross-section predictive-importance (out-of-sample) method. It computes the OOS performance score of each feature in isolation. A few considerations:

1. This method can be applied to any classifier, not only tree-based classifiers.
2. SFI is not limited to accuracy as the sole performance score.
3. Unlike MDI and MDA, no substitution effects take place, since only one feature is taken into consideration at a time.
4. Like MDA, it can conclude that all features are unimportant, because performance is evaluated via OOS CV.

The main limitation of SFI is that a classifier with two features can perform better than the bagging of two single-feature classifiers. For example, (1) feature B may be useful only in combination with feature A; or (2) feature B may be useful in explaining the splits from feature A, even if feature B alone is inaccurate. In other words, joint effects and hierarchical importance are lost in SFI. One alternative would be to compute the OOS performance score from subsets of features, but that calculation will become intractable as more features are considered. Snippet 8.4 demonstrates one possible implementation of the SFI method. A discussion of the function `cvScore` can be found in Chapter 7.

SNIPPET 8.4 IMPLEMENTATION OF SFI

```
def auxFeatImpSFI(featNames,clf,trnsX,cont,scoring,cvGen):
    imp=pd.DataFrame(columns=['mean','std'])
    for featName in featNames:
        df0=cvScore(clf,X=trnsX[[featName]],y=cont['bin'],sample_weight=cont['w'],
                    scoring=scoring,cvGen=cvGen)
        imp.loc[featName,'mean']=df0.mean()
        imp.loc[featName,'std']=df0.std()*df0.shape[0]**-.5
    return imp
```

8.4.2 Orthogonal Features

As argued in Section 8.3, substitution effects dilute the importance of features measured by MDI, and significantly underestimate the importance of features measured by MDA. A partial solution is to orthogonalize the features before applying MDI and MDA. An orthogonalization procedure such as principal components analysis (PCA) does not prevent all substitution effects, but at least it should alleviate the impact of linear substitution effects.

Consider a matrix $\{X_{t,n}\}$ of stationary features, with observations $t = 1, \ldots, T$ and variables $n = 1, \ldots, N$. First, we compute the standardized features matrix Z, such that $Z_{t,n} = \sigma_n^{-1}(X_{t,n} - \mu_n)$, where μ_n is the mean of $\{X_{t,n}\}_{t=1,\ldots,T}$ and σ_n is the standard deviation of $\{X_{t,n}\}_{t=1,\ldots,T}$. Second, we compute the eigenvalues Λ and eigenvectors W such that $Z'ZW = W\Lambda$, where Λ is an NxN diagonal matrix with main entries sorted in descending order, and W is an NxN orthonormal matrix. Third, we derive the orthogonal features as $P = ZW$. We can verify the orthogonality of the features by noting that $P'P = W'Z'ZW = W'W\Lambda W'W = \Lambda$.

The diagonalization is done on Z rather than X, for two reasons: (1) centering the data ensures that the first principal component is correctly oriented in the main direction of the observations. It is equivalent to adding an intercept in a linear regression; (2) re-scaling the data makes PCA focus on explaining correlations rather than variances. Without re-scaling, the first principal components would be dominated by the

columns of X with highest variance, and we would not learn much about the structure or relationship between the variables.

Snippet 8.5 computes the smallest number of orthogonal features that explain at least 95% of the variance of Z.

SNIPPET 8.5 COMPUTATION OF ORTHOGONAL FEATURES

```
def get_eVec(dot,varThres):
    # compute eVec from dot prod matrix, reduce dimension
    eVal,eVec=np.linalg.eigh(dot)
    idx=eVal.argsort()[::-1] # arguments for sorting eVal desc
    eVal,eVec=eVal[idx],eVec[:,idx]
    #2) only positive eVals
    eVal=pd.Series(eVal,index=['PC_'+str(i+1) for i in range(eVal.shape[0])])
    eVec=pd.DataFrame(eVec,index=dot.index,columns=eVal.index)
    eVec=eVec.loc[:,eVal.index]
    #3) reduce dimension, form PCs
    cumVar=eVal.cumsum()/eVal.sum()
    dim=cumVar.values.searchsorted(varThres)
    eVal,eVec=eVal.iloc[:dim+1],eVec.iloc[:,:dim+1]
    return eVal,eVec
#---------------------------------------------------------------------
def orthoFeats(dfX,varThres=.95):
    # Given a dataframe dfX of features, compute orthofeatures dfP
    dfZ=dfX.sub(dfX.mean(),axis=1).div(dfX.std(),axis=1) # standardize
    dot=pd.DataFrame(np.dot(dfZ.T,dfZ),index=dfX.columns,columns=dfX.columns)
    eVal,eVec=get_eVec(dot,varThres)
    dfP=np.dot(dfZ,eVec)
    return dfP
```

Besides addressing substitution effects, working with orthogonal features provides two additional benefits: (1) orthogonalization can also be used to reduce the dimensionality of the features matrix X, by dropping features associated with small eigenvalues. This usually speeds up the convergence of ML algorithms; (2) the analysis is conducted on features designed to explain the structure of the data.

Let me stress this latter point. An ubiquitous concern throughout the book is the risk of overfitting. ML algorithms will always find a pattern, even if that pattern is a statistical fluke. You should always be skeptical about the purportedly important features identified by any method, including MDI, MDA, and SFI. Now, suppose that you derive orthogonal features using PCA. Your PCA analysis has determined that some features are more "principal" than others, without any knowledge of the labels (unsupervised learning). That is, PCA has ranked features without any possible overfitting in a classification sense. When your MDI, MDA, or SFI analysis selects as most important (using label information) the same features that PCA chose as

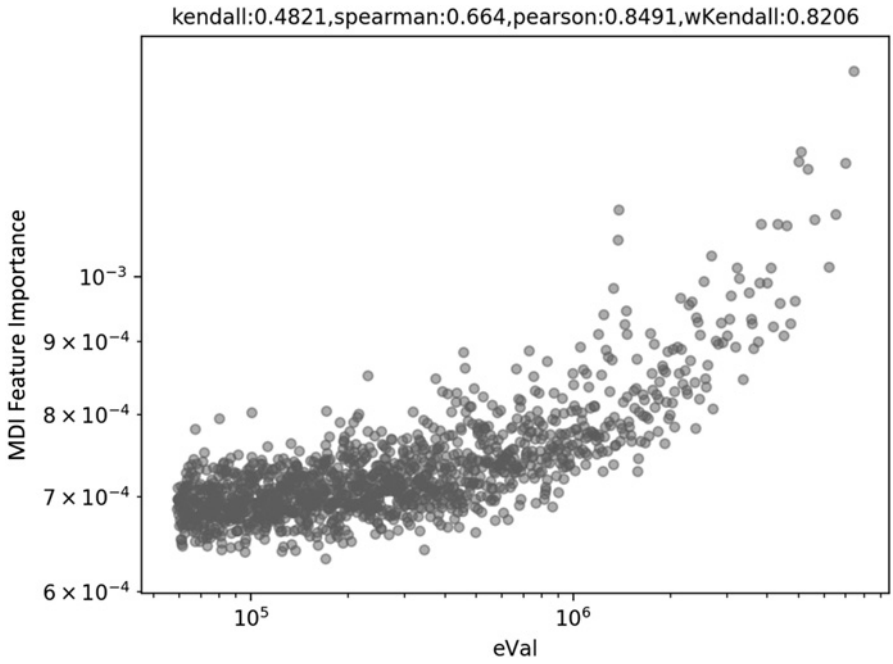


FIGURE 8.1 Scatter plot of eigenvalues (x-axis) and MDI levels (y-axis) in log-log scale

principal (ignoring label information), this constitutes confirmatory evidence that the pattern identified by the ML algorithm is not entirely overfit. If the features were entirely random, the PCA ranking would have no correspondance with the feature importance ranking. Figure 8.1 displays the scatter plot of eigenvalues associated with an eigenvector (x-axis) paired with MDI of the feature associated with an engenvector (y-axis). The Pearson correlation is 0.8491 (p-value below 1E-150), evidencing that PCA identified informative features and ranked them correctly without overfitting.

I find it useful to compute the weighted Kendall's tau between the feature importances and their associated eigenvalues (or equivalently, their inverse PCA rank). The closer this value is to 1, the stronger is the consistency between PCA ranking and feature importance ranking. One argument for preferring a weighted Kendall's tau over the standard Kendall is that we want to prioritize rank concordance among the most importance features. We do not care so much about rank concordance among irrelevant (likely noisy) features. The hyperbolic-weighted Kendall's tau for the sample in Figure 8.1 is 0.8206.

Snippet 8.6 shows how to compute this correlation using Scipy. In this example, sorting the features in descending importance gives us a PCA rank sequence very close to an ascending list. Because the `weightedtau` function gives higher weight to higher values, we compute the correlation on the inverse PCA ranking, `pcRank**-1`. The resulting weighted Kendall's tau is relatively high, at 0.8133.

SNIPPET 8.6 COMPUTATION OF WEIGHTED KENDALL'S TAU BETWEEN FEATURE IMPORTANCE AND INVERSE PCA RANKING

```
>>> import numpy as np
>>> from scipy.stats import weightedtau
>>> featImp=np.array([.55,.33,.07,.05]) # feature importance
>>> pcRank=np.array([1,2,4,3]) # PCA rank
>>> weightedtau(featImp,pcRank**-1.)[0]
```

8.5 PARALLELIZED VS. STACKED FEATURE IMPORTANCE

There are at least two research approaches to feature importance. First, for each security i in an investment universe $i = 1, \ldots, I$, we form a dataset (X_i, y_i), and derive the feature importance in parallel. For example, let us denote $\lambda_{i,j,k}$ the importance of feature j on instrument i according to criterion k. Then we can aggregate all results across the entire universe to derive a combined $\Lambda_{j,k}$ importance of feature j according to criterion k. Features that are important across a wide variety of instruments are more likely to be associated with an underlying phenomenon, particularly when these feature importances exhibit high rank correlation across the criteria. It may be worth studying in-depth the theoretical mechanism that makes these features predictive. The main advantage of this approach is that it is computationally fast, as it can be parallelized. A disadvantage is that, due to substitution effects, important features may swap their ranks across instruments, increasing the variance of the estimated $\lambda_{i,j,k}$. This disadvantage becomes relatively minor if we average $\lambda_{i,j,k}$ across instruments for a sufficiently large investment universe.

A second alternative is what I call "features stacking." It consists in stacking all datasets $\{(\tilde{X}_i, y_i)\}_{i=1,\ldots,I}$ into a single combined dataset (X, y), where $\tilde{X}_i$ is a transformed instance of X_i (e.g., standardized on a rolling trailing window). The purpose of this transformation is to ensure some distributional homogeneity, $\tilde{X}_i \sim X$. Under this approach, the classifier must learn what features are more important across all instruments simultaneously, as if the entire investment universe were in fact a single instrument. Features stacking presents some advantages: (1) The classifier will be fit on a much larger dataset than the one used with the parallelized (first) approach; (2) the importance is derived directly, and no weighting scheme is required for combining the results; (3) conclusions are more general and less biased by outliers or overfitting; and (4) because importance scores are not averaged across instruments, substitution effects do not cause the dampening of those scores.

I usually prefer features stacking, not only for features importance but whenever a classifier can be fit on a set of instruments, including for the purpose of model prediction. That reduces the likelihood of overfitting an estimator to a particular instrument or small dataset. The main disadvantage of stacking is that it may consume a lot of

memory and resources, however that is where a sound knowledge of HPC techniques will come in handy (Chapters 20–22).

8.6 EXPERIMENTS WITH SYNTHETIC DATA

In this section, we are going to test how these feature importance methods respond to synthetic data. We are going to generate a dataset (X, y) composed on three kinds of features:

1. Informative: These are features that are used to determine the label.
2. Redundant: These are random linear combinations of the informative features. They will cause substitution effects.
3. Noise: These are features that have no bearing on determining the observation's label.

Snippet 8.7 shows how we can generate a synthetic dataset of 40 features where 10 are informative, 10 are redundant, and 20 are noise, on 10,000 observations. For details on how sklearn generates synthetic datasets, visit: http://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_classification.html.

SNIPPET 8.7 CREATING A SYNTHETIC DATASET

```
def getTestData(n_features=40,n_informative=10,n_redundant=10,n_samples=10000):
    # generate a random dataset for a classification problem
    from sklearn.datasets import make_classification
    trnsX,cont=make_classification(n_samples=n_samples,n_features=n_features,
        n_informative=n_informative,n_redundant=n_redundant,random_state=0,
        shuffle=False)
    df0=pd.DatetimeIndex(periods=n_samples,freq=pd.tseries.offsets.BDay(),
                         end=pd.datetime.today())
    trnsX,cont=pd.DataFrame(trnsX,index=df0),
                            pd.Series(cont,index=df0).to_frame('bin')
    df0=['I_'+str(i) for i in xrange(n_informative)]+
        ['R_'+str(i) for i in xrange(n_redundant)]
    df0+=['N_'+str(i) for i in xrange(n_features-len(df0))]
    trnsX.columns=df0
    cont['w']=1./cont.shape[0]
    cont['t1']=pd.Series(cont.index,index=cont.index)
    return trnsX,cont
```

Given that we know for certain what feature belongs to each class, we can evaluate whether these three feature importance methods perform as designed. Now we need

a function that can carry out each analysis on the same dataset. Snippet 8.8 accomplishes that, using bagged decision trees as default classifier (Chapter 6).

SNIPPET 8.8 CALLING FEATURE IMPORTANCE FOR ANY METHOD

```
def featImportance(trnsX,cont,n_estimators=1000,cv=10,max_samples=1.,numThreads=24,
                   pctEmbargo=0,scoring='accuracy',method='SFI',minWLeaf=0.,**kargs):
    # feature importance from a random forest
    from sklearn.tree import DecisionTreeClassifier
    from sklearn.ensemble import BaggingClassifier
    from mpEngine import mpPandasObj
    n_jobs=(-1 if numThreads>1 else 1) # run 1 thread with ht_helper in dirac1
    #1) prepare classifier,cv. max_features=1, to prevent masking
    clf=DecisionTreeClassifier(criterion='entropy',max_features=1,
        class_weight='balanced',min_weight_fraction_leaf=minWLeaf)
    clf=BaggingClassifier(base_estimator=clf,n_estimators=n_estimators,
        max_features=1.,max_samples=max_samples,oob_score=True,n_jobs=n_jobs)
    fit=clf.fit(X=trnsX,y=cont['bin'],sample_weight=cont['w'].values)
    oob=fit.oob_score_
    if method=='MDI':
        imp=featImpMDI(fit,featNames=trnsX.columns)
        oos=cvScore(clf,X=trnsX,y=cont['bin'],cv=cv,sample_weight=cont['w'],
                    t1=cont['t1'],pctEmbargo=pctEmbargo,scoring=scoring).mean()
    elif method=='MDA':
        imp,oos=featImpMDA(clf,X=trnsX,y=cont['bin'],cv=cv,sample_weight=cont['w'],
                           t1=cont['t1'],pctEmbargo=pctEmbargo,scoring=scoring)
    elif method=='SFI':
        cvGen=PurgedKFold(n_splits=cv,t1=cont['t1'],pctEmbargo=pctEmbargo)
        oos=cvScore(clf,X=trnsX,y=cont['bin'],sample_weight=cont['w'],scoring=scoring,
            cvGen=cvGen).mean()
        clf.n_jobs=1 # paralellize auxFeatImpSFI rather than clf
        imp=mpPandasObj(auxFeatImpSFI,('featNames',trnsX.columns),numThreads,
            clf=clf,trnsX=trnsX,cont=cont,scoring=scoring,cvGen=cvGen)
    return imp,oob,oos
```

Finally, we need a main function to call all components, from data generation to feature importance analysis to collection and processing of output. These tasks are performed by Snippet 8.9.

SNIPPET 8.9 CALLING ALL COMPONENTS

```
def testFunc(n_features=40,n_informative=10,n_redundant=10,n_estimators=1000,
             n_samples=10000,cv=10):
    # test the performance of the feat importance functions on artificial data
    # Nr noise features = n_features—n_informative—n_redundant
    trnsX,cont=getTestData(n_features,n_informative,n_redundant,n_samples)
```

```
    dict0={'minWLeaf':[0.],'scoring':['accuracy'],'method':['MDI','MDA','SFI'],
        'max_samples':[1.]}
    jobs,out=(dict(izip(dict0,i)) for i in product(*dict0.values())),[]
    kargs={'pathOut':'./testFunc/','n_estimators':n_estimators,
        'tag':'testFunc','cv':cv}
    for job in jobs:
        job['simNum']=job['method']+'_'+job['scoring']+'_'+'%.2f'%job['minWLeaf']+ \
                    '_'+str(job['max_samples'])
        print job['simNum']
        kargs.update(job)
        imp,oob,oos=featImportance(trnsX=trnsX,cont=cont,**kargs)
        plotFeatImportance(imp=imp,oob=oob,oos=oos,**kargs)
        df0=imp[['mean']]/imp['mean'].abs().sum()
        df0['type']=[i[0] for i in df0.index]
        df0=df0.groupby('type')['mean'].sum().to_dict()
        df0.update({'oob':oob,'oos':oos});df0.update(job)
        out.append(df0)
    out=pd.DataFrame(out).sort_values(['method','scoring','minWLeaf','max_samples'])
    out=out['method','scoring','minWLeaf','max_samples','I','R','N','oob','oos']
    out.to_csv(kargs['pathOut']+'stats.csv')
    return
```

For the aesthetically inclined, Snippet 8.10 provides a nice layout for plotting feature importances.

SNIPPET 8.10 FEATURE IMPORTANCE PLOTTING FUNCTION

```
def plotFeatImportance(pathOut,imp,oob,oos,method,tag=0,simNum=0,**kargs):
    # plot mean imp bars with std
    mpl.figure(figsize=(10,imp.shape[0]/5.))
    imp=imp.sort_values('mean',ascending=True)
    ax=imp['mean'].plot(kind='barh',color='b',alpha=.25,xerr=imp['std'],
                        error_kw={'ecolor':'r'})
    if method=='MDI':
        mpl.xlim([0,imp.sum(axis=1).max()])
        mpl.axvline(1./imp.shape[0],linewidth=1,color='r',linestyle='dotted')
    ax.get_yaxis().set_visible(False)
    for i,j in zip(ax.patches,imp.index):ax.text(i.get_width()/2,
                    i.get_y()+i.get_height()/2,j,ha='center',va='center',
                    color='black')
    mpl.title('tag='+tag+' | simNum='+str(simNum)+' | oob='+str(round(oob,4))+
              ' | oos='+str(round(oos,4)))
    mpl.savefig(pathOut+'featImportance_'+str(simNum)+'.png',dpi=100)
    mpl.clf();mpl.close()
    return
```

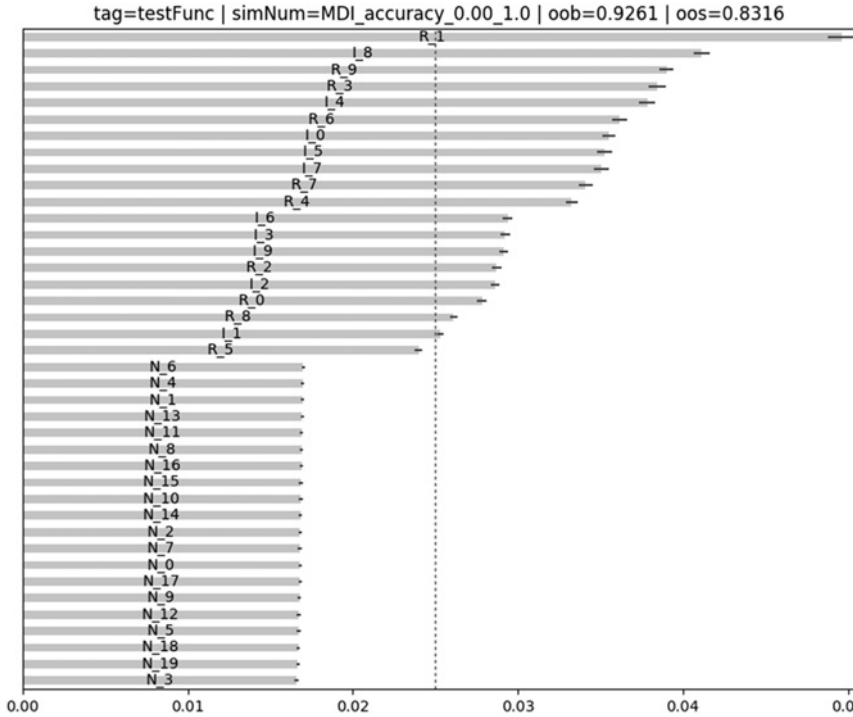


FIGURE 8.2 MDI feature importance computed on a synthetic dataset

Figure 8.2 shows results for MDI. For each feature, the horizontal bar indicates the mean MDI value across all the decision trees, and the horizontal line is the standard deviation of that mean. Since MDI importances add up to 1, if all features were equally important, each importance would have a value of 1/40. The vertical dotted line marks that 1/40 threshold, separating features whose importance exceeds what would be expected from undistinguishable features. As you can see, MDI does a very good job in terms of placing all informative and redundant features above the red dotted line, with the exception of R_5, which did not make the cut by a small margin. Substitution effects cause some informative or redundant features to rank better than others, which was expected.

Figure 8.3 shows that MDA also did a good job. Results are consistent with those from MDI's in the sense that all the informed and redundant features rank better than the noise feature, with the exception of R_6, likely due to a substitution effect. One not so positive aspect of MDA is that the standard deviation of the means are somewhat higher, although that could be addressed by increasing the number of partitions in the purged k-fold CV, from, say, 10 to 100 (at the cost of $10\times$ the computation time without parallelization).

Figure 8.4 shows that SFI also does a decent job; however, a few important features rank worse than noise (I_6, I_2, I_9, I_1, I_3, R_5), likely due to joint effects.

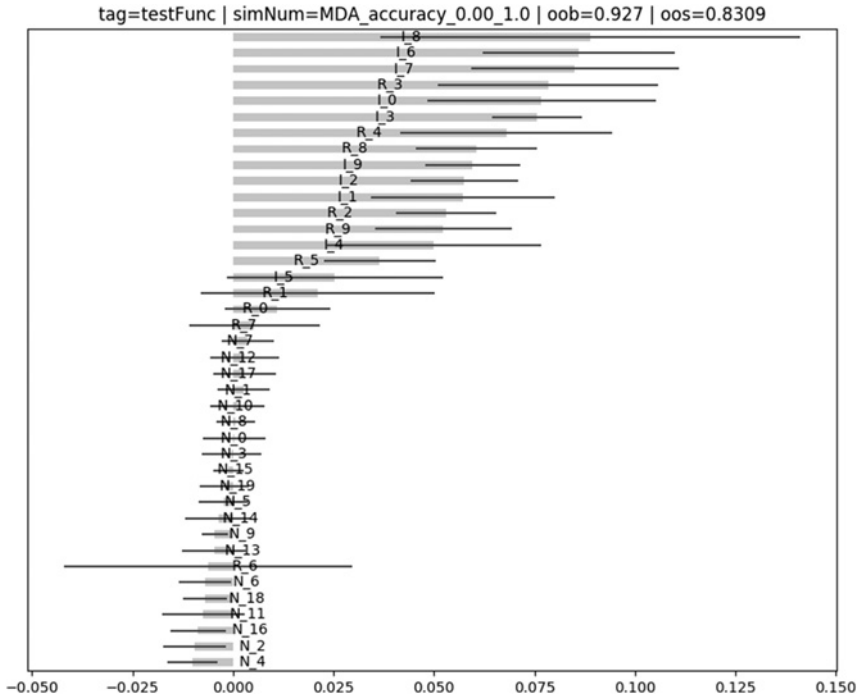


FIGURE 8.3 MDA feature importance computed on a synthetic dataset

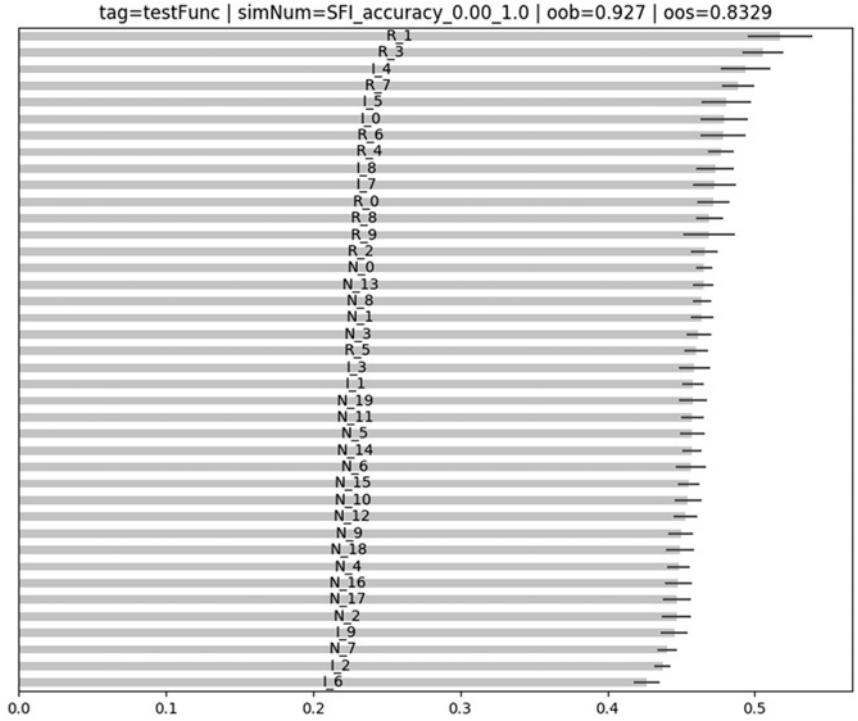


FIGURE 8.4 SFI feature importance computed on a synthetic dataset

The labels are a function of a combination of features, and trying to forecast them independently misses the joint effects. Still, SFI is useful as a complement to MDI and MDA, precisely because both types of analyses are affected by different kinds of problems.

EXERCISES

8.1 Using the code presented in Section 8.6:
- **(a)** Generate a dataset (X, y).
- **(b)** Apply a PCA transformation on X, which we denote $\dot{X}$.
- **(c)** Compute MDI, MDA, and SFI feature importance on $(\dot{X}, y)$, where the base estimator is RF.
- **(d)** Do the three methods agree on what features are important? Why?

8.2 From exercise 1, generate a new dataset $(\ddot{X}, y)$, where $\ddot{X}$ is a feature union of X and $\dot{X}$.
- **(a)** Compute MDI, MDA, and SFI feature importance on $(\ddot{X}, y)$, where the base estimator is RF.
- **(b)** Do the three methods agree on the important features? Why?

8.3 Take the results from exercise 2:
- **(a)** Drop the most important features according to each method, resulting in a features matrix $\ddot{X}$.
- **(b)** Compute MDI, MDA, and SFI feature importance on $(\ddot{X}, y)$, where the base estimator is RF.
- **(c)** Do you appreciate significant changes in the rankings of important features, relative to the results from exercise 2?

8.4 Using the code presented in Section 8.6:
- **(a)** Generate a dataset (X, y) of 1E6 observations, where 5 features are informative, 5 are redundant and 10 are noise.
- **(b)** Split (X, y) into 10 datasets $\{(X_i, y_i)\}_{i=1,\ldots,10}$, each of 1E5 observations.
- **(c)** Compute the parallelized feature importance (Section 8.5), on each of the 10 datasets, $\{(X_i, y_i)\}_{i=1,\ldots,10}$.
- **(d)** Compute the stacked feature importance on the combined dataset (X, y).
- **(e)** What causes the discrepancy between the two? Which one is more reliable?

8.5 Repeat all MDI calculations from exercises 1–4, but this time allow for masking effects. That means, do not set `max_features=int(1)` in Snippet 8.2. How do results differ as a consequence of this change? Why?

REFERENCES

American Statistical Association (2016): "Ethical guidelines for statistical practice." Committee on Professional Ethics of the American Statistical Association (April). Available at http://www.amstat.org/asa/files/pdfs/EthicalGuidelines.pdf.

Belsley, D., E. Kuh, and R. Welsch (1980): *Regression Diagnostics: Identifying Influential Data and Sources of Collinearity*, 1st ed. John Wiley & Sons.

Goldberger, A. (1991): *A Course in Econometrics*. Harvard University Press, 1st edition.

Hill, R. and L. Adkins (2001): "Collinearity." In Baltagi, Badi H. *A Companion to Theoretical Econometrics*, 1st ed. Blackwell, pp. 256–278.

Louppe, G., L. Wehenkel, A. Sutera, and P. Geurts (2013): "Understanding variable importances in forests of randomized trees." Proceedings of the 26th International Conference on Neural Information Processing Systems, pp. 431–439.

Strobl, C., A. Boulesteix, A. Zeileis, and T. Hothorn (2007): "Bias in random forest variable importance measures: Illustrations, sources and a solution." *BMC Bioinformatics*, Vol. 8, No. 25, pp. 1–11.

White, A. and W. Liu (1994): "Technical note: Bias in information-based measures in decision tree induction." *Machine Learning*, Vol. 15, No. 3, pp. 321–329.

CHAPTER 9

Hyper-Parameter Tuning with Cross-Validation

9.1 MOTIVATION

Hyper-parameter tuning is an essential step in fitting an ML algorithm. When this is not done properly, the algorithm is likely to overfit, and live performance will disappoint. The ML literature places special attention on cross-validating any tuned hyper-parameter. As we have seen in Chapter 7, cross-validation (CV) in finance is an especially difficult problem, where solutions from other fields are likely to fail. In this chapter we will discuss how to tune hyper-parameters using the purged k-fold CV method. The references section lists studies that propose alternative methods that may be useful in specific problems.

9.2 GRID SEARCH CROSS-VALIDATION

Grid search cross-validation conducts an exhaustive search for the combination of parameters that maximizes the CV performance, according to some user-defined score function. When we do not know much about the underlying structure of the data, this is a reasonable first approach. Scikit-learn has implemented this logic in the function `GridSearchCV`, which accepts a CV generator as an argument. For the reasons explained in Chapter 7, we need to pass our `PurgedKFold` class (Snippet 7.3) in order to prevent that `GridSearchCV` overfits the ML estimator to leaked information.

SNIPPET 9.1 GRID SEARCH WITH PURGED K-FOLD CROSS-VALIDATION

```
def clfHyperFit(feat,lbl,t1,pipe_clf,param_grid,cv=3,bagging=[0,None,1.],
                n_jobs=-1,pctEmbargo=0,**fit_params):
    if set(lbl.values)=={0,1}:scoring='f1' # f1 for meta-labeling
    else:scoring='neg_log_loss' # symmetric towards all cases
    #1) hyperparameter search, on train data
    inner_cv=PurgedKFold(n_splits=cv,t1=t1,pctEmbargo=pctEmbargo) # purged
    gs=GridSearchCV(estimator=pipe_clf,param_grid=param_grid,
        scoring=scoring,cv=inner_cv,n_jobs=n_jobs,iid=False)
    gs=gs.fit(feat,lbl,**fit_params).best_estimator_ # pipeline
    #2) fit validated model on the entirety of the data
    if bagging[1]>0:
        gs=BaggingClassifier(base_estimator=MyPipeline(gs.steps),
            n_estimators=int(bagging[0]),max_samples=float(bagging[1]),
            max_features=float(bagging[2]),n_jobs=n_jobs)
        gs=gs.fit(feat,lbl,sample_weight=fit_params \
            [gs.base_estimator.steps[-1][0]+'__sample_weight'])
        gs=Pipeline([('bag',gs)])
    return gs
```

Snippet 9.1 lists function `clfHyperFit`, which implements a purged `GridSearchCV`. The argument `fit_params` can be used to pass `sample_weight`, and `param_grid` contains the values that will be combined into a grid. In addition, this function allows for the bagging of the tuned estimator. Bagging an estimator is generally a good idea for the reasons explained in Chapter 6, and the above function incorporates logic to that purpose.

I advise you to use `scoring='f1'` in the context of meta-labeling applications, for the following reason. Suppose a sample with a very large number of negative (i.e., label '0') cases. A classifier that predicts all cases to be negative will achieve high `'accuracy'` or `'neg_log_loss'`, even though it has not learned from the features how to discriminate between cases. In fact, such a model achieves zero recall and undefined precision (see Chapter 3, Section 3.7). The `'f1'` score corrects for that performance inflation by scoring the classifier in terms of precision and recall (see Chapter 14, Section 14.8).

For other (non-meta-labeling) applications, it is fine to use `'accuracy'` or `'neg_log_loss'`, because we are equally interested in predicting all cases. Note that a relabeling of cases has no impact on `'accuracy'` or `'neg_log_loss'`, however it will have an impact on `'f1'`.

This example introduces nicely one limitation of sklearn's `Pipelines` : Their fit method does not expect a `sample_weight` argument. Instead, it expects a `fit_params` keyworded argument. That is a bug that has been reported in GitHub; however, it may take some time to fix it, as it involves rewriting and testing much functionality. Until then, feel free to use the workaround in Snippet 9.2. It creates a

new class, called `MyPipeline`, which inherits all methods from sklearn's `Pipeline`. It overwrites the inherited `fit` method with a new one that handles the argument `sample_weight`, after which it redirects to the parent class.

SNIPPET 9.2 AN ENHANCED PIPELINE CLASS

```
class MyPipeline(Pipeline):
    def fit(self,X,y,sample_weight=None,**fit_params):
        if sample_weight is not None:
            fit_params[self.steps[-1][0]+'__sample_weight']=sample_weight
        return super(MyPipeline,self).fit(X,y,**fit_params)
```

If you are not familiar with this technique for expanding classes, you may want to read this introductory Stackoverflow post: http://stackoverflow.com/questions/576169/understanding-python-super-with-init-methods.

9.3 RANDOMIZED SEARCH CROSS-VALIDATION

For ML algorithms with a large number of parameters, a grid search cross-validation (CV) becomes computationally intractable. In this case, an alternative with good statistical properties is to sample each parameter from a distribution (Begstra et al. [2011, 2012]). This has two benefits: First, we can control for the number of combinations we will search for, regardless of the dimensionality of the problem (the equivalent to a computational budget). Second, having parameters that are relatively irrelevant performance-wise will not substantially increase our search time, as would be the case with grid search CV.

Rather than writing a new function to work with `RandomizedSearchCV`, let us expand Snippet 9.1 to incorporate an option to this purpose. A possible implementation is Snippet 9.3.

SNIPPET 9.3 RANDOMIZED SEARCH WITH PURGED K-FOLD CV

```
def clfHyperFit(feat,lbl,t1,pipe_clf,param_grid,cv=3,bagging=[0,None,1.],
                rndSearchIter=0,n_jobs=-1,pctEmbargo=0,**fit_params):
    if set(lbl.values)=={0,1}:scoring='f1' # f1 for meta-labeling
    else:scoring='neg_log_loss' # symmetric towards all cases
    #1) hyperparameter search, on train data
    inner_cv=PurgedKFold(n_splits=cv,t1=t1,pctEmbargo=pctEmbargo) # purged
    if rndSearchIter==0:
        gs=GridSearchCV(estimator=pipe_clf,param_grid=param_grid,
            scoring=scoring,cv=inner_cv,n_jobs=n_jobs,iid=False)
    else:
```

```
    gs=RandomizedSearchCV(estimator=pipe_clf,param_distributions= \
        param_grid,scoring=scoring,cv=inner_cv,n_jobs=n_jobs,
        iid=False,n_iter=rndSearchIter)
gs=gs.fit(feat,lbl,**fit_params).best_estimator_ # pipeline
#2) fit validated model on the entirety of the data
if bagging[1]>0:
    gs=BaggingClassifier(base_estimator=MyPipeline(gs.steps),
        n_estimators=int(bagging[0]),max_samples=float(bagging[1]),
        max_features=float(bagging[2]),n_jobs=n_jobs)
    gs=gs.fit(feat,lbl,sample_weight=fit_params \
        [gs.base_estimator.steps[-1][0]+'__sample_weight'])
    gs=Pipeline([('bag',gs)])
return gs
```

9.3.1 Log-Uniform Distribution

It is common for some ML algorithms to accept non-negative hyper-parameters only. That is the case of some very popular parameters, such as `C` in the SVC classifier and `gamma` in the RBF kernel.[1] We could draw random numbers from a uniform distribution bounded between 0 and some large value, say 100. That would mean that 99% of the values would be expected to be greater than 1. That is not necessarily the most effective way of exploring the feasibility region of parameters whose functions do not respond linearly. For example, an SVC can be as responsive to an increase in `C` from 0.01 to 1 as to an increase in `C` from 1 to 100.[2] So sampling `C` from a $U[0, 100]$ (uniform) distribution will be inefficient. In those instances, it seems more effective to draw values from a distribution where the logarithm of those draws will be distributed uniformly. I call that a "log-uniform distribution," and since I could not find it in the literature, I must define it properly.

A random variable x follows a log-uniform distribution between $a > 0$ and $b > a$ if and only if $\log[x] \sim U\left[\log[a], \log[b]\right]$. This distribution has a CDF:

$$F[x] = \begin{cases} \dfrac{\log[x] - \log[a]}{\log[b] - \log[a]} & \text{for } a \leq x \leq b \\ 0 & \text{for } x < a \\ 1 & \text{for } x > b \end{cases}$$

From this, we derive a PDF:

$$f[x] = \begin{cases} \dfrac{1}{x \log\left[b/a\right]} & \text{for } a \leq x \leq b \\ 0 & \text{for } x < a \\ 0 & \text{for } x > b \end{cases}$$

Note that the CDF is invariant to the base of the logarithm, since $\dfrac{\log\left[\frac{x}{a}\right]}{\log\left[\frac{b}{a}\right]} = \dfrac{\log_c\left[\frac{x}{a}\right]}{\log_c\left[\frac{b}{a}\right]}$ for any base c, thus the random variable is not a function of c. Snippet 9.4 implements

[1] http://scikit-learn.org/stable/modules/metrics.html.

[2] http://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html.

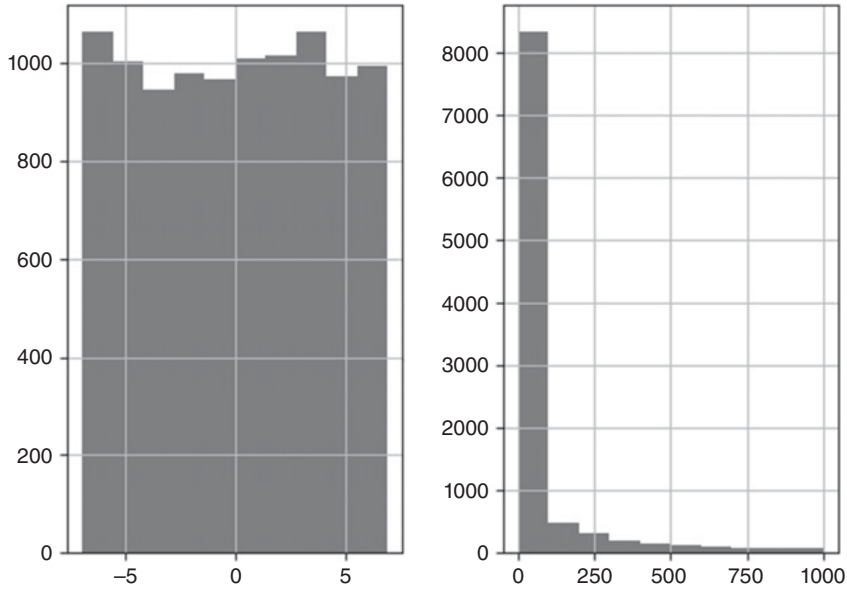


FIGURE 9.1 Result from testing the `logUniform_gen` class

(and tests) in `scipy.stats` a random variable where $[a, b] = [1E-3, 1E3]$, hence $\log[x] \sim U\left[\log[1E-3], \log[1E3]\right]$. Figure 9.1 illustrates the uniformity of the samples in log-scale.

SNIPPET 9.4 THE `logUniform_gen` CLASS

```
import numpy as np,pandas as pd,matplotlib.pyplot as mpl
from scipy.stats import rv_continuous,kstest
#———————————————————————————————
class logUniform_gen(rv_continuous):
    # random numbers log-uniformly distributed between 1 and e
    def _cdf(self,x):
        return np.log(x/self.a)/np.log(self.b/self.a)
def logUniform(a=1,b=np.exp(1)):return logUniform_gen(a=a,b=b,name='logUniform')
#———————————————————————————————
a,b,size=1E-3,1E3,10000
vals=logUniform(a=a,b=b).rvs(size=size)
print kstest(rvs=np.log(vals),cdf='uniform',args=(np.log(a),np.log(b/a)),N=size)
print pd.Series(vals).describe()
mpl.subplot(121)
pd.Series(np.log(vals)).hist()
mpl.subplot(122)
pd.Series(vals).hist()
mpl.show()
```

9.4 SCORING AND HYPER-PARAMETER TUNING

Snippets 9.1 and 9.3 set `scoring='f1'` for meta-labeling applications. For other applications, they set `scoring='neg_log_loss'` rather than the standard `scoring='accuracy'`. Although accuracy has a more intuitive interpretation, I suggest that you use `neg_log_loss` when you are tuning hyper-parameters for an investment strategy. Let me explain my reasoning.

Suppose that your ML investment strategy predicts that you should buy a security, with high probability. You will enter a large long position, as a function of the strategy's confidence. If the prediction was erroneous, and the market sells off instead, you will lose a lot of money. And yet, accuracy accounts equally for an erroneous buy prediction with high probability and for an erroneous buy prediction with low probability. Moreover, accuracy can offset a miss with high probability with a hit with low probability.

Investment strategies profit from predicting the right label with high confidence. Gains from good predictions with low confidence will not suffice to offset the losses from bad predictions with high confidence. For this reason, accuracy does not provide a realistic scoring of the classifier's performance. Conversely, log loss[3] (aka cross-entropy loss) computes the log-likelihood of the classifier given the true label, which takes predictions' probabilities into account. Log loss can be estimated as follows:

$$L[Y,P] = -\log[\text{Prob}[Y|P]] = -N^{-1}\sum_{n=0}^{N-1}\sum_{k=0}^{K-1} y_{n,k}\log\left[p_{n,k}\right]$$

where

- $p_{n,k}$ is the probability associated with prediction n of label k.
- Y is a 1-of-K binary indicator matrix, such that $y_{n,k} = 1$ when observation n was assigned label k out of K possible labels, and 0 otherwise.

Suppose that a classifier predicts two 1s, where the true labels are 1 and 0. The first prediction is a hit and the second prediction is a miss, thus accuracy is 50%. Figure 9.2 plots the cross-entropy loss when these predictions come from probabilities ranging [0.5, 0.9]. One can observe that on the right side of the figure, log loss is large due to misses with high probability, even though the accuracy is 50% in all cases.

There is a second reason to prefer cross-entropy loss over accuracy. CV scores a classifier by applying sample weights (see Chapter 7, Section 7.5). As you may recall from Chapter 4, observation weights were determined as a function of the observation's absolute return. The implication is that sample weighted cross-entropy loss estimates the classifier's performance in terms of variables involved in a PnL (mark-to-market profit and losses) calculation: It uses the correct label for the side, probability for the position size, and sample weight for the observation's return/outcome. That

[3] http://scikit-learn.org/stable/modules/model_evaluation.html#log-loss.

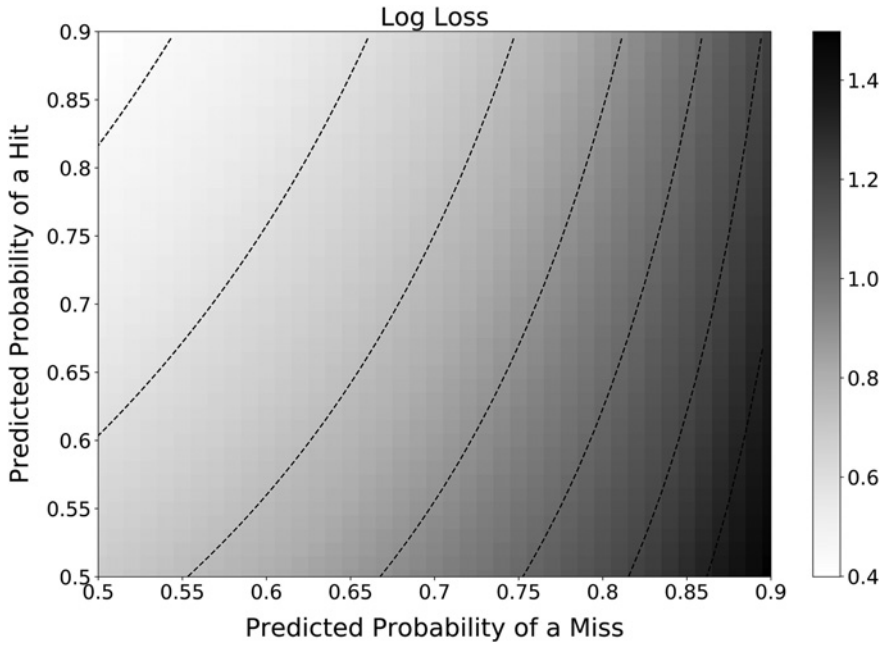


FIGURE 9.2 Log loss as a function of predicted probabilities of hit and miss

is the right ML performance metric for hyper-parameter tuning of financial applications, not accuracy.

When we use log loss as a scoring statistic, we often prefer to change its sign, hence referring to "neg log loss." The reason for this change is cosmetic, driven by intuition: A high neg log loss value is preferred to a low neg log loss value, just as with accuracy. Keep in mind this sklearn bug when you use `neg_log_loss`: https://github.com/scikit-learn/scikit-learn/issues/9144. To circumvent this bug, you should use the `cvScore` function presented in Chapter 7.

EXERCISES

9.1 Using the function `getTestData` from Chapter 8, form a synthetic dataset of 10,000 observations with 10 features, where 5 are informative and 5 are noise.

- **(a)** Use `GridSearchCV` on 10-fold CV to find the `C`, `gamma` optimal hyper-parameters on a SVC with RBF kernel, where `param_grid={'C':[1E-2,1E-1,1,10,100],'gamma':[1E-2,1E-1,1,10,100]}` and the scoring function is `neg_log_loss`.
- **(b)** How many nodes are there in the grid?
- **(c)** How many fits did it take to find the optimal solution?
- **(d)** How long did it take to find this solution?

(e) How can you access the optimal result?

(f) What is the CV score of the optimal parameter combination?

(g) How can you pass sample weights to the SVC?

9.2 Using the same dataset from exercise 1,

(a) Use `RandomizedSearchCV` on 10-fold CV to find the `C`, `gamma` optimal hyper-parameters on an SVC with RBF kernel, where `param_distributions={'C':logUniform(a=1E-2,b=1E2),'gamma':logUniform(a=1E-2,b=1E2)},n_iter=25` and `neg_log_loss` is the scoring function.

(b) How long did it take to find this solution?

(c) Is the optimal parameter combination similar to the one found in exercise 1?

(d) What is the CV score of the optimal parameter combination? How does it compare to the CV score from exercise 1?

9.3 From exercise 1,

(a) Compute the Sharpe ratio of the resulting in-sample forecasts, from point 1.a (see Chapter 14 for a definition of Sharpe ratio).

(b) Repeat point 1.a, this time with `accuracy` as the scoring function. Compute the in-sample forecasts derived from the hyper-tuned parameters.

(c) What scoring method leads to higher (in-sample) Sharpe ratio?

9.4 From exercise 2,

(a) Compute the Sharpe ratio of the resulting in-sample forecasts, from point 2.a.

(b) Repeat point 2.a, this time with `accuracy` as the scoring function. Compute the in-sample forecasts derived from the hyper-tuned parameters.

(c) What scoring method leads to higher (in-sample) Sharpe ratio?

9.5 Read the definition of log loss, $L[Y,P]$.

(a) Why is the scoring function `neg_log_loss` defined as the negative log loss, $-L[Y,P]$?

(b) What would be the outcome of maximizing the log loss, rather than the negative log loss?

9.6 Consider an investment strategy that sizes its bets equally, regardless of the forecast's confidence. In this case, what is a more appropriate scoring function for hyper-parameter tuning, accuracy or cross-entropy loss?

REFERENCES

Bergstra, J., R. Bardenet, Y. Bengio, and B. Kegl (2011): "Algorithms for hyper-parameter optimization." *Advances in Neural Information Processing Systems*, pp. 2546–2554.

Bergstra, J. and Y. Bengio (2012): "Random search for hyper-parameter optimization." *Journal of Machine Learning Research*, Vol. 13, pp. 281–305.

BIBLIOGRAPHY

Chapelle, O., V. Vapnik, O. Bousquet, and S. Mukherjee (2002): "Choosing multiple parameters for support vector machines." *Machine Learning*, Vol. 46, pp. 131–159.

Chuong, B., C. Foo, and A. Ng (2008): "Efficient multiple hyperparameter learning for log-linear models." *Advances in Neural Information Processing Systems*, Vol. 20. Available at http://ai.stanford.edu/~chuongdo/papers/learn_reg.pdf.

Gorissen, D., K. Crombecq, I. Couckuyt, P. Demeester, and T. Dhaene (2010): "A surrogate modeling and adaptive sampling toolbox for computer based design." *Journal of Machine Learning Research*, Vol. 11, pp. 2051–2055.

Hsu, C., C. Chang, and C. Lin (2010): "A practical guide to support vector classification." Technical report, National Taiwan University.

Hutter, F., H. Hoos, and K. Leyton-Brown (2011): "Sequential model-based optimization for general algorithm configuration." Proceedings of the 5th international conference on Learning and Intelligent Optimization, pp. 507–523.

Larsen, J., L. Hansen, C. Svarer, and M. Ohlsson (1996): "Design and regularization of neural networks: The optimal use of a validation set." Proceedings of the 1996 IEEE Signal Processing Society Workshop.

Maclaurin, D., D. Duvenaud, and R. Adams (2015): "Gradient-based hyperparameter optimization through reversible learning." Working paper. Available at https://arxiv.org/abs/1502.03492.

Martinez-Cantin, R. (2014): "BayesOpt: A Bayesian optimization library for nonlinear optimization, experimental design and bandits." *Journal of Machine Learning Research*, Vol. 15, pp. 3915–3919.

PART 3

Backtesting

CHAPTER 10

Bet Sizing

10.1 MOTIVATION

There are fascinating parallels between strategy games and investing. Some of the best portfolio managers I have worked with are excellent poker players, perhaps more so than chess players. One reason is bet sizing, for which Texas Hold'em provides a great analogue and training ground. Your ML algorithm can achieve high accuracy, but if you do not size your bets properly, your investment strategy will inevitably lose money. In this chapter we will review a few approaches to size bets from ML predictions.

10.2 STRATEGY-INDEPENDENT BET SIZING APPROACHES

Consider two strategies on the same instrument. Let $m_{i,t} \in [-1, 1]$ be the bet size of strategy i at time t, where $m_{i,t} = -1$ indicates a full short position and $m_{i,t} = 1$ indicates a full long position. Suppose that one strategy produced a sequence of bet sizes $[m_{1,1}, m_{1,2}, m_{1,3}] = [.5, 1, 0]$, as the market price followed a sequence $[p_1, p_2, p_3] = [1, .5, 1.25]$, where p_t is the price at time t. The other strategy produced a sequence $[m_{2,1}, m_{2,2}, m_{2,3}] = [1, .5, 0]$, as it was forced to reduce its bet size once the market moved against the initial full position. Both strategies produced forecasts that turned out to be correct (the price increased by 25% between p_1 and p_3), however the first strategy made money (0.5) while the second strategy lost money (−.125).

We would prefer to size positions in such way that we reserve some cash for the possibility that the trading signal strengthens before it weakens. One option is to compute the series $c_t = c_{t,l} - c_{t,s}$, where $c_{t,l}$ is the number of concurrent long bets at time t, and $c_{t,s}$ is the number of concurrent short bets at time t. This bet concurrency is derived, for each side, similarly to how we computed label concurrency in Chapter 4 (recall the `t1` object, with overlapping time spans). We fit a mixture of two Gaussians

on $\{c_t\}$, applying a method like the one described in López de Prado and Foreman [2014]. Then, the bet size is derived as

$$m_t = \begin{cases} \dfrac{F[c_t] - F[0]}{1 - F[0]} & \text{if } c_t \geq 0 \\ \dfrac{F[c_t] - F[0]}{F[0]} & \text{if } c_t < 0 \end{cases}$$

where $F[x]$ is the CDF of the fitted mixture of two Gaussians for a value x. For example, we could size the bet as 0.9 when the probability of observing a signal of greater value is only 0.1. The stronger the signal, the smaller the probability that the signal becomes even stronger, hence the greater the bet size.

A second solution is to follow a budgeting approach. We compute the maximum number (or some other quantile) of concurrent long bets, $\max_i\{c_{i,l}\}$, and the maximum number of concurrent short bets, $\max_i\{c_{i,s}\}$. Then we derive the bet size as $m_t = c_{t,l}\frac{1}{\max_i\{c_{i,l}\}} - c_{t,s}\frac{1}{\max_i\{c_{i,s}\}}$, where $c_{t,l}$ is the number of concurrent long bets at time t, and $c_{t,s}$ is the number of concurrent short bets at time t. The goal is that the maximum position is not reached before the last concurrent signal is triggered.

A third approach is to apply meta-labeling, as we explained in Chapter 3. We fit a classifier, such as an SVC or RF, to determine the probability of misclassification, and use that probability to derive the bet size. [1] This approach has a couple of advantages: First, the ML algorithm that decides the bet sizes is independent of the primary model, allowing for the incorporation of features predictive of false positives (see Chapter 3). Second, the predicted probability can be directly translated into bet size. Let us see how.

10.3 BET SIZING FROM PREDICTED PROBABILITIES

Let us denote $p[x]$ the probability that label x takes place. For two possible outcomes, $x \in \{-1, 1\}$, we would like to test the null hypothesis $H_0 : p[x = 1] = \frac{1}{2}$. We compute the test statistic $z = \frac{p[x=1]-\frac{1}{2}}{\sqrt{p[x=1](1-p[x=1])}} = \frac{2p[x=1]-1}{2\sqrt{p[x=1](1-p[x=1])}} \sim Z$, with $z \in (-\infty, +\infty)$ and where Z represents the standard Normal distribution. We derive the bet size as $m = 2Z[z] - 1$, where $m \in [-1, 1]$ and $Z[.]$ is the CDF of Z.

For more than two possible outcomes, we follow a one-versus-rest method. Let $X = \{-1, \ldots, 0, \ldots, 1\}$ be various labels associated with bet sizes, and $x \in X$ the predicted label. In other words, the label is identified by the bet size associated with it. For each label $i = 1, \ldots, \|X\|$, we estimate a probability p_i, with $\sum_{i=1}^{\|X\|} p_i = 1$. We define

[1] The references section lists a number of articles that explain how these probabilities are derived. Usually these probabilities incorporate information about the goodness of the fit, or confidence in the prediction. See Wu et al. [2004], and visit http://scikit-learn.org/stable/modules/svm.html#scores-and-probabilities.

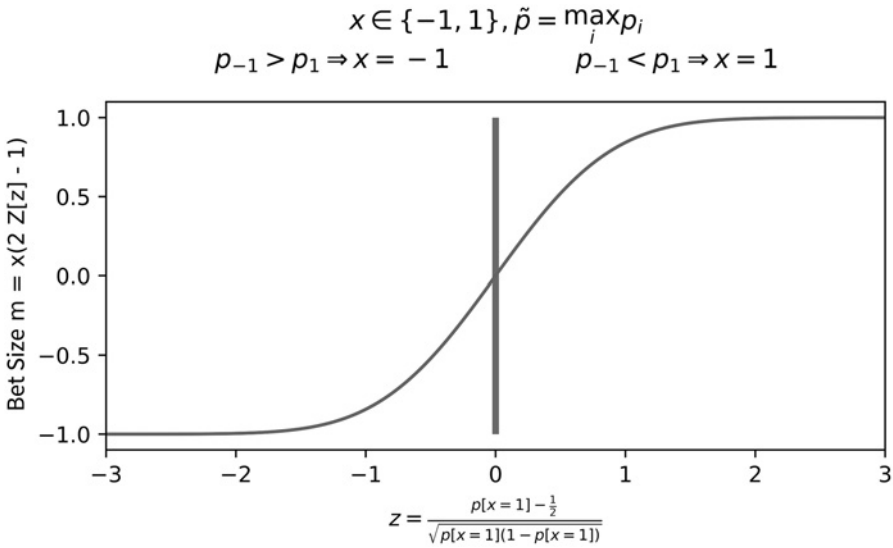


FIGURE 10.1 Bet size from predicted probabilities

$\tilde{p} = \max_i\{p_i\}$ as the probability of x, and we would like to test for $H_0 : \tilde{p} = \frac{1}{\|X\|}$.[2] We compute the test statistic $z = \frac{\tilde{p} - \frac{1}{\|X\|}}{\sqrt{\tilde{p}(1-\tilde{p})}} \sim Z$, with $z \in [0\ , +\infty)$. We derive the bet size as $m = x\underbrace{(2Z[z] - 1)}_{\in[0,1]}$, where $m \in [-1, 1]$ and $Z[z]$ regulates the size for a prediction x (where the side is implied by x).

Figure 10.1 plots the bet size as a function of test statistic. Snippet 10.1 implements the translation from probabilities to bet size. It handles the possibility that the prediction comes from a meta-labeling estimator, as well from a standard labeling estimator. In step #2, it also averages active bets, and discretizes the final value, which we will explain in the following sections.

SNIPPET 10.1 FROM PROBABILITIES TO BET SIZE

```
def getSignal(events,stepSize,prob,pred,numClasses,numThreads,**kargs):
    # get signals from predictions
    if prob.shape[0]==0:return pd.Series()
    #1) generate signals from multinomial classification (one-vs-rest, OvR)
    signal0=(prob-1./numClasses)/(prob*(1.-prob))**.5 # t-value of OvR
    signal0=pred*(2*norm.cdf(signal0)-1) # signal=side*size
```

[2] Uncertainty is absolute when all outcomes are equally likely.

```
    if 'side' in events:signal0*=events.loc[signal0.index,'side'] # meta-labeling
    #2) compute average signal among those concurrently open
    df0=signal0.to_frame('signal').join(events[['t1']],how='left')
    df0=avgActiveSignals(df0,numThreads)
    signal1=discreteSignal(signal0=df0,stepSize=stepSize)
    return signal1
```

10.4 AVERAGING ACTIVE BETS

Every bet is associated with a holding period, spanning from the time it originated to the time the first barrier is touched, `t1` (see Chapter 3). One possible approach is to override an old bet as a new bet arrives; however, that is likely to lead to excessive turnover. A more sensible approach is to average all sizes across all bets still active at a given point in time. Snippet 10.2 illustrates one possible implementation of this idea.

SNIPPET 10.2 BETS ARE AVERAGED AS LONG AS THEY ARE STILL ACTIVE

```
def avgActiveSignals(signals,numThreads):
    # compute the average signal among those active
    #1) time points where signals change (either one starts or one ends)
    tPnts=set(signals['t1'].dropna().values)
    tPnts=tPnts.union(signals.index.values)
    tPnts=list(tPnts);tPnts.sort()
    out=mpPandasObj(mpAvgActiveSignals,('molecule',tPnts),numThreads,signals=signals)
    return out
#-----------------------------------------
def mpAvgActiveSignals(signals,molecule):
  '''
  At time loc, average signal among those still active.
  Signal is active if:
      a) issued before or at loc AND
      b) loc before signal's endtime, or endtime is still unknown (NaT).
  '''
  out=pd.Series()
  for loc in molecule:
      df0=(signals.index.values<=loc)&((loc<signals['t1'])|pd.isnull(signals['t1']))
      act=signals[df0].index
      if len(act)>0:out[loc]=signals.loc[act,'signal'].mean()
      else:out[loc]=0 # no signals active at this time
  return out
```

10.5 SIZE DISCRETIZATION

Averaging reduces some of the excess turnover, but still it is likely that small trades will be triggered with every prediction. As this jitter would cause unnecessary

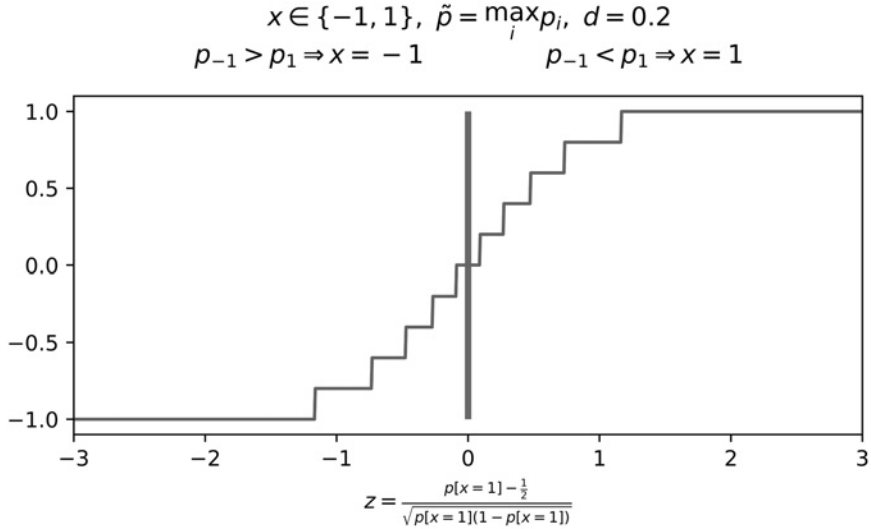


FIGURE 10.2 Discretization of the bet size, d = 0.2

overtrading, I suggest you discretize the bet size as $m^* = \text{round}\left[\frac{m}{d}\right] d$, where $d \in (0,\ 1]$ determines the degree of discretization. Figure 10.2 illustrates the discretization of the bet size. Snippet 10.3 implements this notion.

SNIPPET 10.3 SIZE DISCRETIZATION TO PREVENT OVERTRADING

```
def discreteSignal(signal0,stepSize):
    # discretize signal
    signal1=(signal0/stepSize).round()*stepSize # discretize
    signal1[signal1>1]=1 # cap
    signal1[signal1<-1]=-1 # floor
    return signal1
```

10.6 DYNAMIC BET SIZES AND LIMIT PRICES

Recall the triple-barrier labeling method presented in Chapter 3. Bar i is formed at time $t_{i,0}$, at which point we forecast the first barrier that will be touched. That prediction implies a forecasted price, $\mathrm{E}_{t_{i,0}}[p_{t_{i,1}}]$, consistent with the barriers' settings. In the period elapsed until the outcome takes place, $t \in [t_{i,0}, t_{i,1}]$, the price p_t fluctuates and additional forecasts may be formed, $\mathrm{E}_{t_{j,0}}[p_{t_{i,1}}]$, where $j \in [i+1, I]$ and $t_{j,0} \leq t_{i,1}$. In Sections 10.4 and 10.5 we discussed methods for averaging the active bets and

discretizing the bet size as new forecasts are formed. In this section we will introduce an approach to adjust bet sizes as market price p_t and forecast price f_i fluctuate. In the process, we will derive the order's limit price.

Let q_t be the current position, Q the maximum absolute position size, and $\hat{q}_{i,t}$ the target position size associated with forecast f_i, such that

$$\hat{q}_{i,t} = \text{int}[m[\omega, f_i - p_t]Q]$$

$$m[\omega, x] = \frac{x}{\sqrt{\omega + x^2}}$$

where $m[\omega, x]$ is the bet size, $x = f_i - p_t$ is the divergence between the current market price and the forecast, ω is a coefficient that regulates the width of the sigmoid function, and Int $[x]$ is the integer value of x. Note that for a real-valued price divergence x, $-1 < m[\omega, x] < 1$, the integer value $\hat{q}_{i,t}$ is bounded $-Q < \hat{q}_{i,t} < Q$.

The target position size $\hat{q}_{i,t}$ can be dynamically adjusted as p_t changes. In particular, as $p_t \to f_i$ we get $\hat{q}_{i,t} \to 0$, because the algorithm wants to realize the gains. This implies a breakeven limit price $\bar{p}$ for the order size $\hat{q}_{i,t} - q_t$, to avoid realizing losses. In particular,

$$\bar{p} = \frac{1}{|\hat{q}_{i,t} - q_t|} \sum_{j=|q_t + \text{sgn}[\hat{q}_{i,t} - q_t]|}^{|\hat{q}_{i,t}|} L\left[f_i, \omega, \frac{j}{Q}\right]$$

where $L[f_i, \omega, m]$ is the inverse function of $m[\omega, f_i - p_t]$ with respect to p_t,

$$L[f_i, \omega, m] = f_i - m\sqrt{\frac{\omega}{1 - m^2}}$$

We do not need to worry about the case $m^2 = 1$, because $|\hat{q}_{i,t}| < 1$. Since this function is monotonic, the algorithm cannot realize losses as $p_t \to f_i$.

Let us calibrate ω. Given a user-defined pair (x, m^*), such that $x = f_i - p_t$ and $m^* = m[\omega, x]$, the inverse function of $m[\omega, x]$ with respect to ω is

$$\omega = x^2(m^{*-2} - 1)$$

Snippet 10.4 implements the algorithm that computes the dynamic position size and limit prices as a function of p_t and f_i. First, we calibrate the sigmoid function, so that it returns a bet size of $m^* = .95$ for a price divergence of $x = 10$. Second, we compute the target position $\hat{q}_{i,t}$ for a maximum position $Q = 100$, $f_i = 115$ and $p_t = 100$. If you try $f_i = 110$, you will get $\hat{q}_{i,t} = 95$, consistent with the calibration of ω. Third, the limit price for this order of size $\hat{q}_{i,t} - q_t = 97$ is $p_t < 112.3657 < f_i$, which is between the current price and the forecasted price.

SNIPPET 10.4 DYNAMIC POSITION SIZE AND LIMIT PRICE

```
def betSize(w,x):
    return x*(w+x**2)**-.5
#---------------------------------------
def getTPos(w,f,mP,maxPos):
    return int(betSize(w,f-mP)*maxPos)
#---------------------------------------
def invPrice(f,w,m):
    return f-m*(w/(1-m**2))**.5
#---------------------------------------
def limitPrice(tPos,pos,f,w,maxPos):
    sgn=(1 if tPos>=pos else -1)
    lP=0
    for j in xrange(abs(pos+sgn),abs(tPos+1)):
        lP+=invPrice(f,w,j/float(maxPos))
    lP/=tPos-pos
    return lP
#---------------------------------------
def getW(x,m):
  # 0<alpha<1
    return x**2*(m**-2-1)
#---------------------------------------
def main():
    pos,maxPos,mP,f,wParams=0,100,100,115,{'divergence':10,'m':.95}
    w=getW(wParams['divergence'],wParams['m']) # calibrate w
    tPos=getTPos(w,f,mP,maxPos) # get tPos
    lP=limitPrice(tPos,pos,f,w,maxPos) # limit price for order
    return
#---------------------------------------
if __name__=='__main__':main()
```

As an alternative to the sigmoid function, we could have used a power function $\tilde{m}[\omega, x] = \text{sgn}[x]|x|^{\omega}$, where $\omega \geq 0$, $x \in [-1, 1]$, which results in $\tilde{m}[\omega, x] \in [-1, 1]$. This alternative presents the advantages that:

- $\tilde{m}[\omega, -1] = -1, \tilde{m}[\omega, 1] = 1$.
- Curvature can be directly manipulated through ω.
- For $\omega > 1$, the function goes from concave to convex, rather than the other way around, hence the function is almost flat around the inflexion point.

We leave the derivation of the equations for a power function as an exercise. Figure 10.3 plots the bet sizes (y-axis) as a function of price divergence $f - p_t$ (x-axis) for both the sigmoid and power functions.

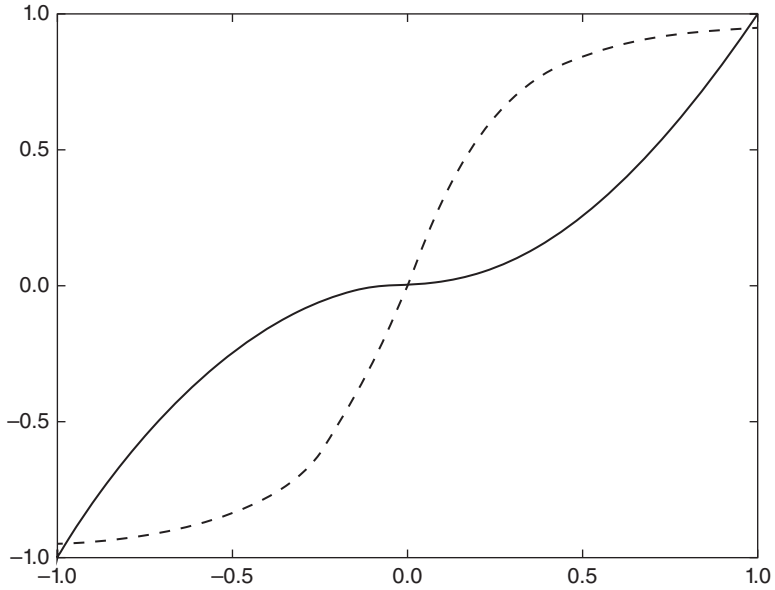


FIGURE 10.3 $f[x] = sgn[x]|x|^2$ (concave to convex) and $f[x] = x(.1 + x^2)^{-.5}$ (convex to concave)

EXERCISES

10.1 Using the formulation in Section 10.3, plot the bet size (m) as a function of the maximum predicted probability ($\tilde{p}$) when $\|X\| = 2, 3, \ldots, 10$.

10.2 Draw 10,000 random numbers from a uniform distribution with bounds $U[.5, 1.]$.

- **(a)** Compute the bet sizes m for $\|X\| = 2$.
- **(b)** Assign 10,000 consecutive calendar days to the bet sizes.
- **(c)** Draw 10,000 random numbers from a uniform distribution with bounds $U[1, 25]$.
- **(d)** Form a pandas series indexed by the dates in 2.b, and with values equal to the index shifted forward the number of days in 2.c. This is a `t1` object similar to the ones we used in Chapter 3.
- **(e)** Compute the resulting average active bets, following Section 10.4.

10.3 Using the `t1` object from exercise 2.d:

- **(a)** Determine the maximum number of concurrent long bets, $\bar{c}_l$.
- **(b)** Determine the maximum number of concurrent short bets, $\bar{c}_s$.
- **(c)** Derive the bet size as $m_t = c_{t,l}\frac{1}{\bar{c}_l} - c_{t,s}\frac{1}{\bar{c}_s}$, where $c_{t,l}$ is the number of concurrent long bets at time t, and $c_{t,s}$ is the number of concurrent short bets at time t.

10.4 Using the `t1` object from exercise 2.d:

(a) Compute the series $c_t = c_{t,l} - c_{t,s}$, where $c_{t,l}$ is the number of concurrent long bets at time t, and $c_{t,s}$ is the number of concurrent short bets at time t.

(b) Fit a mixture of two Gaussians on $\{c_t\}$. You may want to use the method described in López de Prado and Foreman [2014].

(c) Derive the bet size as $m_t = \begin{cases} \frac{F[c_t]-F[0]}{1-F[0]} & \text{if } c_t \geq 0 \\ \frac{F[c_t]-F[0]}{F[0]} & \text{if } c_t < 0 \end{cases}$, where $F[x]$ is the CDF of the fitted mixture of two Gaussians for a value x.

(d) Explain how this series $\{m_t\}$ differ from the bet size series computed in exercise 3.

10.5 Repeat exercise 1, where you discretize m with a `stepSize=.01`, `stepSize=.05`, and `stepSize=.1`.

10.6 Rewrite the equations in Section 10.6, so that the bet size is determined by a power function rather than a sigmoid function.

10.7 Modify Snippet 10.4 so that it implements the equations you derived in exercise 6.

REFERENCES

López de Prado, M. and M. Foreman (2014): "A mixture of Gaussians approach to mathematical portfolio oversight: The EF3M algorithm." *Quantitative Finance*, Vol. 14, No. 5, pp. 913–930.

Wu, T., C. Lin and R. Weng (2004): "Probability estimates for multi-class classification by pairwise coupling." *Journal of Machine Learning Research*, Vol. 5, pp. 975–1005.

BIBLIOGRAPHY

Allwein, E., R. Schapire, and Y. Singer (2001): "Reducing multiclass to binary: A unifying approach for margin classifiers." *Journal of Machine Learning Research*, Vol. 1, pp. 113–141.

Hastie, T. and R. Tibshirani (1998): "Classification by pairwise coupling." *The Annals of Statistics*, Vol. 26, No. 1, pp. 451–471.

Refregier, P. and F. Vallet (1991): "Probabilistic approach for multiclass classification with neural networks." Proceedings of International Conference on Artificial Networks, pp. 1003–1007.

CHAPTER 11

The Dangers of Backtesting

11.1 MOTIVATION

Backtesting is one of the most essential, and yet least understood, techniques in the quant arsenal. A common misunderstanding is to think of backtesting as a research tool. Researching and backtesting is like drinking and driving. Do not research under the influence of a backtest. Most backtests published in journals are flawed, as the result of selection bias on multiple tests (Bailey, Borwein, López de Prado, and Zhu [2014]; Harvey et al. [2016]). A full book could be written listing all the different errors people make while backtesting. I may be the academic author with the largest number of journal articles on backtesting[1] and investment performance metrics, and still I do not feel I would have the stamina to compile all the different errors I have seen over the past 20 years. This chapter is not a crash course on backtesting, but a short list of some of the common errors that even seasoned professionals make.

11.2 MISSION IMPOSSIBLE: THE FLAWLESS BACKTEST

In its narrowest definition, a backtest is a historical simulation of how a strategy would have performed should it have been run over a past period of time. As such, it is a hypothetical, and by no means an experiment. At a physics laboratory, like Berkeley Lab, we can repeat an experiment while controlling for environmental variables, in order to deduce a precise cause-effect relationship. In contrast, a backtest is *not* an experiment, and it does not prove anything. A backtest guarantees nothing, not even achieving that Sharpe ratio if we could travel back in time in our retrofitted DeLorean DMC-12 (Bailey and López de Prado [2012]). Random draws would have been different. The past would not repeat itself.

[1] http://papers.ssrn.com/sol3/cf_dev/AbsByAuth.cfm?per_id=434076; http://www.QuantResearch.org/.

What is the point of a backtest then? It is a sanity check on a number of variables, including bet sizing, turnover, resilience to costs, and behavior under a given scenario. A good backtest can be extremely helpful, but backtesting well is extremely hard. In 2014 a team of quants at Deutsche Bank, led by Yin Luo, published a study under the title "Seven Sins of Quantitative Investing" (Luo et al. [2014]). It is a very graphic and accessible piece that I would advise everyone in this business to read carefully. In it, this team mentions the usual suspects:

1. **Survivorship bias:** Using as investment universe the current one, hence ignoring that some companies went bankrupt and securities were delisted along the way.
2. **Look-ahead bias:** Using information that was not public at the moment the simulated decision would have been made. Be certain about the timestamp for each data point. Take into account release dates, distribution delays, and backfill corrections.
3. **Storytelling:** Making up a story *ex-post* to justify some random pattern.
4. **Data mining and data snooping:** Training the model on the testing set.
5. **Transaction costs:** Simulating transaction costs is hard because the only way to be certain about that cost would have been to interact with the trading book (i.e., to do the actual trade).
6. **Outliers:** Basing a strategy on a few extreme outcomes that may never happen again as observed in the past.
7. **Shorting:** Taking a short position on cash products requires finding a lender. The cost of lending and the amount available is generally unknown, and depends on relations, inventory, relative demand, etc.

These are just a few basic errors that most papers published in journals make routinely. Other common errors include computing performance using a non-standard method (Chapter 14); ignoring hidden risks; focusing only on returns while ignoring other metrics; confusing correlation with causation; selecting an unrepresentative time period; failing to expect the unexpected; ignoring the existence of stop-out limits or margin calls; ignoring funding costs; and forgetting practical aspects (Sarfati [2015]). There are many more, but really, there is no point in listing them, because of the title of the next section.

11.3 EVEN IF YOUR BACKTEST IS FLAWLESS, IT IS PROBABLY WRONG

Congratulations! Your backtest is flawless in the sense that everyone can reproduce your results, and your assumptions are so conservative that not even your boss could object to them. You have paid for every trade more than double what anyone could possibly ask. You have executed hours after the information was known by half the globe, at a ridiculously low volume participation rate. Despite all these egregious

costs, your backtest still makes a lot of money. Yet, this flawless backtest is probably wrong. Why? Because only an expert can produce a flawless backtest. Becoming an expert means that you have run tens of thousands of backtests over the years. In conclusion, this is not the first backtest you produce, so we need to account for the possibility that this is a false discovery, a statistical fluke that inevitably comes up after you run multiple tests on the same dataset.

The maddening thing about backtesting is that, the better you become at it, the more likely false discoveries will pop up. Beginners fall for the seven sins of Luo et al. [2014] (there are more, but who's counting?). Professionals may produce flawless backtests, and will still fall for multiple testing, selection bias, or backtest overfitting (Bailey and López de Prado [2014b]).

11.4 BACKTESTING IS NOT A RESEARCH TOOL

Chapter 8 discussed substitution effects, joint effects, masking, MDI, MDA, SFI, parallelized features, stacked features, etc. Even if some features are very important, it does not mean that they can be monetized through an investment strategy. Conversely, there are plenty of strategies that will appear to be profitable even though they are based on irrelevant features. Feature importance is a true research tool, because it helps us understand the nature of the patterns uncovered by the ML algorithm, regardless of their monetization. Critically, feature importance is derived *ex-ante*, before the historical performance is simulated.

In contrast, a backtest is not a research tool. It provides us with very little insight into the reason why a particular strategy would have made money. Just as a lottery winner may feel he has done something to deserve his luck, there is always some *ex-post* story (Luo's sin number three). Authors claim to have found hundreds of "alphas" and "factors," and there is always some convoluted explanation for them. Instead, what they have found are the lottery tickets that won the last game. The winner has cashed out, and those numbers are useless for the next round. If you would not pay extra for those lottery tickets, why would you care about those hundreds of alphas? Those authors never tell us about all the tickets that were sold, that is, the millions of simulations it took to find these "lucky" alphas.

The purpose of a backtest is to discard bad models, not to improve them. Adjusting your model based on the backtest results is a waste of time . . . and it's dangerous. Invest your time and effort in getting all the components right, as we've discussed elsewhere in the book: structured data, labeling, weighting, ensembles, cross-validation, feature importance, bet sizing, etc. By the time you are backtesting, it is too late. Never backtest until your model has been fully specified. If the backtest fails, start all over. If you do that, the chances of finding a false discovery will drop substantially, but still they will not be zero.

11.5 A FEW GENERAL RECOMMENDATIONS

Backtest overfitting can be defined as selection bias on multiple backtests. Backtest overfitting takes place when a strategy is developed to perform well on a backtest, by

monetizing random historical patterns. Because those random patterns are unlikely to occur again in the future, the strategy so developed will fail. Every backtested strategy is overfit to some extent as a result of "selection bias": The only backtests that most people share are those that portray supposedly winning investment strategies.

How to address backtest overfitting is arguably the most fundamental question in quantitative finance. Why? Because if there was an easy answer to this question, investment firms would achieve high performance with certainty, as they would invest only in winning backtests. Journals would assess with confidence whether a strategy may be a false positive. Finance could become a true science in the Popperian and Lakatosian sense (López de Prado [2017]). What makes backtest overfitting so hard to assess is that the probability of false positives changes with every new test conducted on the same dataset, and that information is either unknown by the researcher or not shared with investors or referees. While there is no easy way to prevent overfitting, a number of steps can help reduce its presence.

1. Develop models for entire asset classes or investment universes, rather than for specific securities (Chapter 8). Investors diversify, hence they do not make mistake *X* only on security *Y*. If you find mistake *X* only on security *Y*, no matter how apparently profitable, it is likely a false discovery.
2. Apply bagging (Chapter 6) as a means to both prevent overfitting and reduce the variance of the forecasting error. If bagging deteriorates the performance of a strategy, it was likely overfit to a small number of observations or outliers.
3. Do not backtest until all your research is complete (Chapters 1–10).
4. Record every backtest conducted on a dataset so that the probability of backtest overfitting may be estimated on the final selected result (see Bailey, Borwein, López de Prado, and Zhu [2017a] and Chapter 14), and the Sharpe ratio may be properly deflated by the number of trials carried out (Bailey and López de Prado [2014b]).
5. Simulate scenarios rather than history (Chapter 12). A standard backtest is a historical simulation, which can be easily overfit. History is just the random path that was realized, and it could have been entirely different. Your strategy should be profitable under a wide range of scenarios, not just the anecdotal historical path. It is harder to overfit the outcome of thousands of "what if" scenarios.
6. If the backtest fails to identify a profitable strategy, start from scratch. Resist the temptation of reusing those results. Follow the Second Law of Backtesting.

SNIPPET 11.1 MARCOS' SECOND LAW OF BACKTESTING

"Backtesting while researching is like drinking and driving.
Do not research under the influence of a backtest."

—Marcos López de Prado
Advances in Financial Machine Learning (2018)

11.6 STRATEGY SELECTION

In Chapter 7 we discussed how the presence of serial conditionality in labels defeats standard k-fold cross-validation, because the random sampling will spatter redundant observations into both the training and testing sets. We must find a different (true out-of-sample) validation procedure: a procedure that evaluates our model on the observations least likely to be correlated/redundant to those used to train the model. See Arlot and Celisse [2010] for a survey.

Scikit-learn has implemented a walk-forward timefolds method.[2] Under this approach, testing moves forward (in the time direction) with the goal of preventing leakage. This is consistent with the way historical backtests (and trading) are done. However, in the presence of long-range serial dependence, testing one observation away from the end of the training set may not suffice to avoid informational leakage. We will retake this point in Chapter 12, Section 12.2.

One disadvantage of the walk-forward method is that it can be easily overfit. The reason is that without random sampling, there is a single path of testing that can be repeated over and over until a false positive appears. Like in standard CV, some randomization is needed to avoid this sort of performance targeting or backtest optimization, while avoiding the leakage of examples correlated to the training set into the testing set. Next, we will introduce a CV method for strategy selection, based on the estimation of the probability of backtest overfitting (PBO). We leave for Chapter 12 an explanation of CV methods for backtesting.

Bailey et al. [2017a] estimate the PBO through the combinatorially symmetric cross-validation (CSCV) method. Schematically, this procedure works as follows.

First, we form a matrix M by collecting the performance series from the N trials. In particular, each column $n = 1, \ldots, N$ represents a vector of PnL (mark-to-market profits and losses) over $t = 1, \ldots, T$ observations associated with a particular model configuration tried by the researcher. M is therefore a real-valued matrix of order (TxN). The only conditions we impose are that (1) M is a true matrix, that is, with the same number of rows for each column, where observations are synchronous for every row across the N trials, and (2) the performance evaluation metric used to choose the "optimal" strategy can be estimated on subsamples of each column. For example, if that metric is the Sharpe ratio, we assume that the IID Normal distribution assumption can be held on various slices of the reported performance. If different model configurations trade with different frequencies, observations are aggregated (downsampled) to match a common index $t = 1, \ldots, T$.

Second, we partition M across rows, into an even number S of disjoint submatrices of equal dimensions. Each of these submatrices M_s, with $s = 1, \ldots, S$, is of order $(\frac{T}{S}xN)$.

Third, we form all combinations C_S of M_s, taken in groups of size $\frac{S}{2}$. This gives a total number of combinations

$$\binom{S}{S/2} = \binom{S-1}{S/2-1}\frac{S}{S/2} = \ldots = \prod_{i=0}^{S/2-1}\frac{S-i}{S/2-i}$$

For instance, if $S = 16$, we will form 12,780 combinations. Each combination $c \in C_S$ is composed of $S/2$ submatrices M_s.

Fourth, for each combination $c \in C_S$, we:

1. Form the *training set* J, by joining the $S/2$ submatrices M_s that constitute c. J is a matrix of order $\left(\frac{T}{S}\frac{S}{2}xN\right) = \left(\frac{T}{2}xN\right)$.
2. Form the *testing set* $\bar{J}$, as the complement of J in M. In other words, $\bar{J}$ is the $\left(\frac{T}{2}xN\right)$ matrix formed by all rows of M that are not part of J.
3. Form a vector R of performance statistics of order N, where the n-th item of R reports the performance associated with the n-th column of J (the training set).
4. Determine the element n^* such that $R_n \leq R_{n^*}$, $\forall n = 1, \ldots, N$. In other words, $n^* = \arg max_n\{R_n\}$.
5. Form a vector $\bar{R}$ of performance statistics of order N, where the n-th item of $\bar{R}$ reports the performance associated with the n-th column of $\bar{J}$ (the testing set).
6. Determine the relative rank of $\overline{R_{n^*}}$ within $\bar{R}$. We denote this relative rank as $\bar{\omega}_c$, where $\bar{\omega}_c \in (0, 1)$. This is the relative rank of the out-of-sample (OOS) performance associated with the trial chosen in-sample (IS). If the strategy optimization procedure does not overfit, we should observe that $\overline{R_{n^*}}$ systematically outperforms $\bar{R}$ (OOS), just as R_{n^*} outperformed R (IS).
7. Define the logit $\lambda_c = \log\left[\frac{\bar{\omega}_c}{1-\bar{\omega}_c}\right]$. This presents the property that $\lambda_c = 0$ when $\overline{R_{n^*}}$ coincides with the median of $\bar{R}$. High logit values imply a consistency between IS and OOS performance, which indicates a low level of backtest overfitting.

Fifth, compute the distribution of ranks OOS by collecting all the λ_c, for $c \in C_S$. The probability distribution function $f(\lambda)$ is then estimated as the relative frequency at which λ occurred across all C_S, with $\int_{-\infty}^{\infty} f(\lambda)d\lambda = 1$. Finally, the PBO is estimated as $PBO = \int_{-\infty}^{0} f(\lambda)d\lambda$, as that is the probability associated with IS optimal strategies that underperform OOS.

The x-axis of Figure 11.1 shows the Sharpe ratio IS from the best strategy selected. The y-axis shows the Sharpe ratio OOS for that same best strategy selected. As it can be appreciated, there is a strong and persistent performance decay, caused by backtest overfitting. Applying the above algorithm, we can derive the PBO associated with this strategy selection process, as displayed in Figure 11.2.

The observations in each subset preserve the original time sequence. The random sampling is done on the relatively uncorrelated subsets, rather than on the observations. See Bailey et al. [2017a] for an experimental analysis of the accuracy of this methodology.

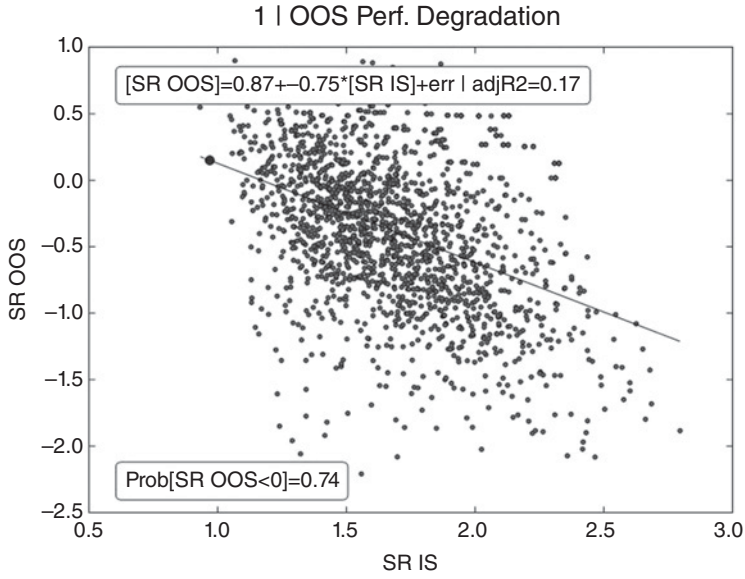


FIGURE 11.1 Best Sharpe ratio in-sample (SR IS) vs Sharpe ratio out-of-sample (SR OOS)

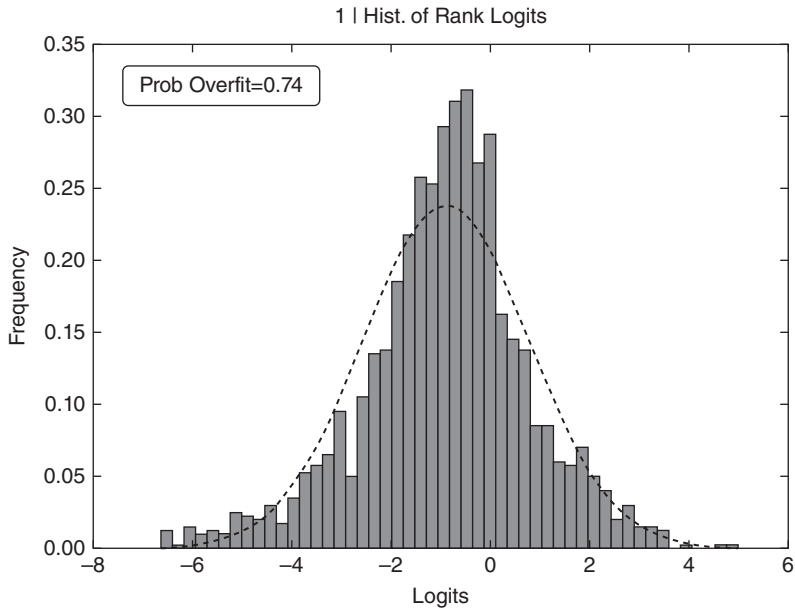


FIGURE 11.2 Probability of backtest overfitting derived from the distribution of logits

EXERCISES

11.1 An analyst fits an RF classifier where some of the features include seasonally adjusted employment data. He aligns with January data the seasonally adjusted value of January, etc. What "sin" has he committed?

11.2 An analyst develops an ML algorithm where he generates a signal using closing prices, and executed at close. What's the sin?

11.3 There is a 98.51% correlation between total revenue generated by arcades and computer science doctorates awarded in the United States. As the number of doctorates is expected to grow, should we invest in arcades companies? If not, what's the sin?

11.4 The *Wall Street Journal* has reported that September is the only month of the year that has negative average stock returns, looking back 20, 50, and 100 years. Should we sell stocks at the end of August? If not, what's the sin?

11.5 We download P/E ratios from Bloomberg, rank stocks every month, sell the top quartile, and buy the long quartile. Performance is amazing. What's the sin?

REFERENCES

Arlot, S. and A. Celisse (2010): "A survey of cross-validation procedures for model selection." *Statistics Surveys*, Vol. 4, pp. 40–79.

Bailey, D., J. Borwein, M. López de Prado, and J. Zhu (2014): "Pseudo-mathematics and financial charlatanism: The effects of backtest overfitting on out-of-sample performance." *Notices of the American Mathematical Society*, Vol. 61, No. 5 (May), pp. 458–471. Available at https://ssrn.com/abstract=2308659.

Bailey, D., J. Borwein, M. López de Prado, and J. Zhu (2017a): "The probability of backtest overfitting." *Journal of Computational Finance*, Vol. 20, No. 4, pp. 39–70. Available at http://ssrn.com/abstract=2326253.

Bailey, D. and M. López de Prado (2012): "The Sharpe ratio efficient frontier." *Journal of Risk*, Vol. 15, No. 2 (Winter). Available at https://ssrn.com/abstract=1821643.

Bailey, D. and M. López de Prado (2014b): "The deflated Sharpe ratio: Correcting for selection bias, backtest overfitting and non-normality." *Journal of Portfolio Management*, Vol. 40, No. 5, pp. 94–107. Available at https://ssrn.com/abstract=2460551.

Harvey, C., Y. Liu, and H. Zhu (2016): ". . . and the cross-section of expected returns." *Review of Financial Studies*, Vol. 29, No. 1, pp. 5–68.

López de Prado, M. (2017): "Finance as an industrial science." *Journal of Portfolio Management*, Vol. 43, No. 4, pp. 5–9. Available at http://www.iijournals.com/doi/pdfplus/10.3905/jpm.2017.43.4.005.

Luo, Y., M. Alvarez, S. Wang, J. Jussa, A. Wang, and G. Rohal (2014): "Seven sins of quantitative investing." White paper, Deutsche Bank Markets Research, September 8.

Sarfati, O. (2015): "Backtesting: A practitioner's guide to assessing strategies and avoiding pitfalls." Citi Equity Derivatives. CBOE 2015 Risk Management Conference. Available at https://www.cboe.com/rmc/2015/olivier-pdf-Backtesting-Full.pdf.

BIBLIOGRAPHY

Bailey, D., J. Borwein, and M. López de Prado (2016): "Stock portfolio design and backtest overfitting." *Journal of Investment Management*, Vol. 15, No. 1, pp. 1–13. Available at https://ssrn.com/abstract=2739335.

Bailey, D., J. Borwein, M. López de Prado, A. Salehipour, and J. Zhu (2016): "Backtest overfitting in financial markets." *Automated Trader*, Vol. 39. Available at https://ssrn.com/abstract=2731886.

Bailey, D., J. Borwein, M. López de Prado, and J. Zhu (2017b): "Mathematical appendices to: 'The probability of backtest overfitting.'" *Journal of Computational Finance (Risk Journals)*, Vol. 20, No. 4. Available at https://ssrn.com/abstract=2568435.

Bailey, D., J. Borwein, A. Salehipour, and M. López de Prado (2017): "Evaluation and ranking of market forecasters." *Journal of Investment Management*, forthcoming. Available at https://ssrn.com/abstract=2944853.

Bailey, D., J. Borwein, A. Salehipour, M. López de Prado, and J. Zhu (2015): "Online tools for demonstration of backtest overfitting." Working paper. Available at https://ssrn.com/abstract=2597421.

Bailey, D., S. Ger, M. López de Prado, A. Sim and, K. Wu (2016): "Statistical overfitting and backtest performance." In *Risk-Based and Factor Investing*, Quantitative Finance Elsevier. Available at ttps://ssrn.com/abstract=2507040.

Bailey, D. and M. López de Prado (2014a): "Stop-outs under serial correlation and 'the triple penance rule.'" *Journal of Risk*, Vol. 18, No. 2, pp. 61–93. Available at https://ssrn.com/abstract=2201302.

Bailey, D. and M. López de Prado (2015): "Mathematical appendices to: 'Stop-outs under serial correlation.'" *Journal of Risk*, Vol. 18, No. 2. Available at https://ssrn.com/abstract=2511599.

Bailey, D., M. López de Prado, and E. del Pozo (2013): "The strategy approval decision: A Sharpe ratio indifference curve approach." *Algorithmic Finance*, Vol. 2, No. 1, pp. 99–109. Available at https://ssrn.com/abstract=2003638.

Carr, P. and M. López de Prado (2014): "Determining optimal trading rules without backtesting." Working paper. Available at https://ssrn.com/abstract=2658641.

López de Prado, M. (2012a): "Portfolio oversight: An evolutionary approach." Lecture at Cornell University. Available at https://ssrn.com/abstract=2172468.

López de Prado, M. (2012b): "The sharp razor: Performance evaluation with non-normal returns." Lecture at Cornell University. Available at https://ssrn.com/abstract=2150879.

López de Prado, M. (2013): "What to look for in a backtest." Lecture at Cornell University. Available at https://ssrn.com/abstract=2308682.

López de Prado, M. (2014a): "Optimal trading rules without backtesting." Lecture at Cornell University. Available at https://ssrn.com/abstract=2502613.

López de Prado, M. (2014b): "Deflating the Sharpe ratio." Lecture at Cornell University. Available at https://ssrn.com/abstract=2465675.

López de Prado, M. (2015a): "Quantitative meta-strategies." *Practical Applications, Institutional Investor Journals*, Vol. 2, No. 3, pp. 1–3. Available at https://ssrn.com/abstract=2547325.

López de Prado, M. (2015b): "The Future of empirical finance." *Journal of Portfolio Management*, Vol. 41, No. 4, pp. 140–144. Available at https://ssrn.com/abstract=2609734.

López de Prado, M. (2015c): "Backtesting." Lecture at Cornell University. Available at https://ssrn.com/abstract=2606462.

López de Prado, M. (2015d): "Recent trends in empirical finance." *Journal of Portfolio Management*, Vol. 41, No. 4, pp. 29–33. Available at https://ssrn.com/abstract=2638760.

López de Prado, M. (2015e): "Why most empirical discoveries in finance are likely wrong, and what can be done about it." Lecture at University of Pennsylvania. Available at https://ssrn.com/abstract=2599105.

López de Prado, M. (2015f): "Advances in quantitative meta-strategies." Lecture at Cornell University. Available at https://ssrn.com/abstract=2604812.

López de Prado, M. (2016): "Building diversified portfolios that outperform out-of-sample." *Journal of Portfolio Management*, Vol. 42, No. 4, pp. 59–69. Available at https://ssrn.com/abstract=2708678.

López de Prado, M. and M. Foreman (2014): "A mixture of Gaussians approach to mathematical portfolio oversight: The EF3M algorithm." *Quantitative Finance*, Vol. 14, No. 5, pp. 913–930. Available at https://ssrn.com/abstract=1931734.

López de Prado, M. and A. Peijan (2004): "Measuring loss potential of hedge fund strategies." *Journal of Alternative Investments*, Vol. 7, No. 1, pp. 7–31, Summer 2004. Available at https://ssrn.com/abstract=641702.

López de Prado, M., R. Vince, and J. Zhu (2015): "Risk adjusted growth portfolio in a finite investment horizon." Lecture at Cornell University. Available at https://ssrn.com/abstract=2624329.

CHAPTER 12

Backtesting through Cross-Validation

12.1 MOTIVATION

A backtest evaluates out-of-sample the performance of an investment strategy using past observations. These past observations can be used in two ways: (1) in a narrow sense, to simulate the historical performance of an investment strategy, as if it had been run in the past; and (2) in a broader sense, to simulate scenarios that did not happen in the past. The first (narrow) approach, also known as walk-forward, is so prevalent that, in fact, the term "backtest" has become a *de facto* synonym for "historical simulation." The second (broader) approach is far less known, and in this chapter we will introduce some novel ways to carry it out. Each approach has its pros and cons, and each should be given careful consideration.

12.2 THE WALK-FORWARD METHOD

The most common backtest method in the literature is the walk-forward (WF) approach. WF is a historical simulation of how the strategy would have performed in past. Each strategy decision is based on observations that predate that decision. As we saw in Chapter 11, carrying out a flawless WF simulation is a daunting task that requires extreme knowledge of the data sources, market microstructure, risk management, performance measurement standards (e.g., GIPS), multiple testing methods, experimental mathematics, etc. Unfortunately, there is no generic recipe to conduct a backtest. To be accurate and representative, each backtest must be customized to evaluate the assumptions of a particular strategy.

WF enjoys two key advantages: (1) WF has a clear historical interpretation. Its performance can be reconciled with paper trading. (2) History is a filtration; hence, using trailing data guarantees that the testing set is out-of-sample (no leakage), as long as purging has been properly implemented (see Chapter 7, Section 7.4.1). It is a

common mistake to find leakage in WF backtests, where `t1.index` falls within the training set, but `t1.values` fall within the testing set (see Chapter 3). Embargoing is not needed in WF backtests, because the training set always predates the testing set.

12.2.1 Pitfalls of the Walk-Forward Method

WF suffers from three major disadvantages: First, a single scenario is tested (the historical path), which can be easily overfit (Bailey et al. [2014]). Second, WF is not necessarily representative of future performance, as results can be biased by the particular sequence of datapoints. Proponents of the WF method typically argue that predicting the past would lead to overly optimistic performance estimates. And yet, very often fitting an outperforming model on the reversed sequence of observations will lead to an underperforming WF backtest. The truth is, it is as easy to overfit a walk-forward backtest as to overfit a walk-backward backtest, and the fact that changing the sequence of observations yields inconsistent outcomes is evidence of that overfitting. If proponents of WF were right, we should observe that walk-backwards backtests systematically outperform their walk-forward counterparts. That is not the case, hence the main argument in favor of WF is rather weak.

To make this second disadvantage clearer, suppose an equity strategy that is backtested with a WF on S&P 500 data, starting January 1, 2007. Until March 15, 2009, the mix of rallies and sell-offs will train the strategy to be market neutral, with low confidence on every position. After that, the long rally will dominate the dataset, and by January 1, 2017, buy forecasts will prevail over sell forecasts. The performance would be very different had we played the information backwards, from January 1, 2017 to January 1, 2007 (a long rally followed by a sharp sell-off). By exploiting a particular sequence, a strategy selected by WF may set us up for a debacle.

The third disadvantage of WF is that the initial decisions are made on a smaller portion of the total sample. Even if a warm-up period is set, most of the information is used by only a small portion of the decisions. Consider a strategy with a warm-up period that uses t_0 observations out of T. This strategy makes half of its decisions $\left(\frac{T-t_0}{2}\right)$ on an average number of datapoints,

$$\left(\frac{T-t_0}{2}\right)^{-1}\left(t_0+\frac{T+t_0}{2}\right)\frac{T-t_0}{4}=\frac{1}{4}T+\frac{3}{4}t_0$$

which is only a $\frac{3}{4}\frac{t_0}{T}+\frac{1}{4}$ fraction of the observations. Although this problem is attenuated by increasing the warm-up period, doing so also reduces the length of the backtest.

12.3 THE CROSS-VALIDATION METHOD

Investors often ask how a strategy would perform if subjected to a stress scenario as unforeseeable as the 2008 crisis, or the dot-com bubble, or the taper tantrum, or

the China scare of 2015–2016, etc. One way to answer is to split the observations into two sets, one with the period we wish to test (testing set), and one with the rest (training set). For example, a classifier would be trained on the period January 1, 2009–January 1, 2017, then tested on the period January 1, 2008–December 31, 2008. The performance we will obtain for 2008 is not historically accurate, since the classifier was trained on data that was only available after 2008. But historical accuracy was not the goal of the test. The objective of the test was to subject a strategy *ignorant* of 2008 to a stress scenario such as 2008.

The goal of backtesting through cross-validation (CV) is not to derive historically accurate performance, but to infer future performance from a number of out-of-sample scenarios. For each period of the backtest, we simulate the performance of a classifier that knew everything except for that period.

Advantages

1. The test is not the result of a particular (historical) scenario. In fact, CV tests k alternative scenarios, of which only one corresponds with the historical sequence.
2. Every decision is made on sets of equal size. This makes outcomes comparable across periods, in terms of the amount of information used to make those decisions.
3. Every observation is part of one and only one testing set. There is no warm-up subset, thereby achieving the longest possible out-of-sample simulation.

Disadvantages

1. Like WF, a single backtest path is simulated (although not the historical one). There is one and only one forecast generated per observation.
2. CV has no clear historical interpretation. The output does not simulate how the strategy would have performed in the past, but how it may perform *in the future* under various stress scenarios (a useful result in its own right).
3. Because the training set does not trail the testing set, leakage is possible. Extreme care must be taken to avoid leaking testing information into the training set. See Chapter 7 for a discussion on how purging and embargoing can help prevent informational leakage in the context of CV.

12.4 THE COMBINATORIAL PURGED CROSS-VALIDATION METHOD

In this section I will present a new method, which addresses the main drawback of the WF and CV methods, namely that those schemes test a single path. I call it the "combinatorial purged cross-validation" (CPCV) method. Given a number φ of backtest paths targeted by the researcher, CPCV generates the precise number of combinations of training/testing sets needed to generate those paths, while purging training observations that contain leaked information.

	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14	S15	Paths
G1	x	x	x	x	x											5
G2	x					x	x	x	x							5
G3		x				x				x	x	x				5
G4			x				x			x			x	x		5
G5				x				x			x		x		x	5
G6					x				x			x		x	x	5

FIGURE 12.1 Paths generated for $\varphi[6,2] = 5$

12.4.1 Combinatorial Splits

Consider T observations partitioned into N groups without shuffling, where groups $n = 1, \ldots, N-1$ are of size $\lfloor T/N \rfloor$, the Nth group is of size $T - \lfloor T/N \rfloor (N-1)$, and $\lfloor . \rfloor$ is the floor or integer function. For a testing set of size k groups, the number of possible training/testing splits is

$$\binom{N}{N-k} = \frac{\prod_{i=0}^{k-1}(N-i)}{k!}$$

Since each combination involves k tested groups, the total number of tested groups is $k\binom{N}{N-k}$. And since we have computed all possible combinations, these tested groups are uniformly distributed across all N (each group belongs to the same number of training and testing sets). The implication is that from k-sized testing sets on N groups we can backtest a total number of paths $\varphi[N,k]$,

$$\varphi[N,k] = \frac{k}{N}\binom{N}{N-k} = \frac{\prod_{i=1}^{k-1}(N-i)}{(k-1)!}$$

Figure 12.1 illustrates the composition of train/test splits for $N = 6$ and $k = 2$. There are $\binom{6}{4} = 15$ splits, indexed as *S1,…,S15*. For each split, the figure marks with a cross (x) the groups included in the testing set, and leaves unmarked the groups that form the training set. Each group forms part of $\varphi[6,2] = 5$ testing sets, therefore this train/test split scheme allows us to compute 5 backtest paths.

Figure 12.2 shows the assignment of each tested group to one backtest path. For example, path 1 is the result of combining the forecasts from $(G1, S1)$, $(G2, S1)$,

	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14	S15	Paths
G1	1	2	3	4	5											5
G2	1					2	3	4	5							5
G3		1				2				3	4	5				5
G4			1				2			3			4	5		5
G5				1				2			3		4		5	5
G6					1				2			3		4	5	5

FIGURE 12.2 Assignment of testing groups to each of the 5 paths

$(G3, S2)$, $(G4, S3)$, $(G5, S4)$ and $(G6, S5)$. Path 2 is the result of combining forecasts from $(G1, S2)$, $(G2, S6)$, $(G3, S6)$, $(G4, S7)$, $(G5, S8)$ and $(G6, S9)$, and so on.

These paths are generated by training the classifier on a portion $\theta = 1 - k/N$ of the data for each combination. Although it is theoretically possible to train on a portion $\theta < 1/2$, in practice we will assume that $k \leq N/2$. The portion of data in the training set θ increases with $N \to T$ but it decreases with $k \to N/2$. The number of paths $\varphi[N, k]$ increases with $N \to T$ and with $k \to N/2$. In the limit, the largest number of paths is achieved by setting $N = T$ and $k = N/2 = T/2$, at the expense of training the classifier on only half of the data for each combination ($\theta = 1/2$).

12.4.2 The Combinatorial Purged Cross-Validation Backtesting Algorithm

In Chapter 7 we introduced the concepts of purging and embargoing in the context of CV. We will now use these concepts for backtesting through CV. The CPCV backtesting algorithm proceeds as follows:

1. Partition T observations into N groups without shuffling, where groups $n = 1, \ldots, N-1$ are of size $\lfloor T/N \rfloor$, and the Nth group is of size $T - \lfloor T/N \rfloor (N-1)$.
2. Compute all possible training/testing splits, where for each split $N - k$ groups constitute the training set and k groups constitute the testing set.
3. For any pair of labels (y_i, y_j), where y_i belongs to the training set and y_j belongs to the testing set, apply the `PurgedKFold` class to purge y_i if y_i spans over a period used to determine label y_j. This class will also apply an embargo, should some testing samples predate some training samples.
4. Fit classifiers on the $\binom{N}{N-k}$ training sets, and produce forecasts on the respective $\binom{N}{N-k}$ testing sets.
5. Compute the $\varphi[N, k]$ backtest paths. You can calculate one Sharpe ratio from each path, and from that derive the empirical distribution of the strategy's Sharpe ratio (rather than a single Sharpe ratio, like WF or CV).

12.4.3 A Few Examples

For $k = 1$, we will obtain $\varphi[N, 1] = 1$ path, in which case CPCV reduces to CV. Thus, CPCV can be understood as a generalization of CV for $k > 1$.

For $k = 2$, we will obtain $\varphi[N, 2] = N - 1$ paths. This is a particularly interesting case, because while training the classifier on a large portion of the data, $\theta = 1 - 2/N$, we can generate almost as many backtest paths as the number of groups, $N - 1$. An easy rule of thumb is to partition the data into $N = \varphi + 1$ groups, where φ is the number of paths we target, and then form $\binom{N}{N-2}$ combinations. In the limit, we can assign one group per observation, $N = T$, and generate $\varphi[T, 2] = T - 1$ paths, while training the classifier on a portion $\theta = 1 - 2/T$ of the data per combination.

If even more paths are needed, we can increase $k \to N/2$, but as explained earlier that will come at the cost of using a smaller portion of the dataset for training. In practice, $k = 2$ is often enough to generate the needed φ paths, by setting $N = \varphi + 1 \leq T$.

12.5 HOW COMBINATORIAL PURGED CROSS-VALIDATION ADDRESSES BACKTEST OVERFITTING

Given a sample of IID random variables, $x_i \sim Z$, $i = 1, \ldots, I$, where Z is the standard normal distribution, the expected maximum of that sample can be approximated as

$$\mathrm{E}[\max\{x_i\}_{i=1,\ldots,I}] \approx (1-\gamma) Z^{-1}\left[1 - \frac{1}{I}\right] + \gamma Z^{-1}\left[1 - \frac{1}{I}e^{-1}\right] \leq \sqrt{2\log[I]}$$

where $Z^{-1}[.]$ is the inverse of the CDF of Z, $\gamma \approx 0.5772156649\cdots$ is the Euler-Mascheroni constant, and $I \gg 1$ (see Bailey et al. [2014] for a proof). Now suppose that a researcher backtests I strategies on an instrument that behaves like a martingale, with Sharpe ratios $\{y_i\}_{i=1,\ldots,I}$, $\mathrm{E}[y_i] = 0$, $\sigma^2[y_i] > 0$, and $\frac{y_i}{\sigma[y_i]} \sim Z$. Even though the true Sharpe ratio is zero, we expect to find one strategy with a Sharpe ratio of

$$\mathrm{E}[\max\{y_i\}_{i=1,\ldots,I}] = \mathrm{E}[\max\{x_i\}_{i=1,\ldots,I}]\sigma[y_i]$$

WF backtests exhibit high variance, $\sigma[y_i] \gg 0$, for at least one reason: A large portion of the decisions are based on a small portion of the dataset. A few observations will have a large weight on the Sharpe ratio. Using a warm-up period will reduce the backtest length, which may contribute to making the variance even higher. WF's high variance leads to false discoveries, because researchers will select the backtest with the maximum *estimated* Sharpe ratio, even if the *true* Sharpe ratio is zero. That is the reason it is imperative to control for the number of trials *(I)* in the context of WF backtesting. Without this information, it is not possible to determine the Family-Wise Error Rate (FWER), False Discovery Rate (FDR), Probability of Backtest Overfitting (PBO, see Chapter 11) or similar model assessment statistic.

CV backtests (Section 12.3) address that source of variance by training each classifier on equal and large portions of the dataset. Although CV leads to fewer false discoveries than WF, both approaches still estimate the Sharpe ratio from a single path for a strategy i, y_i, and that estimation may be highly volatile. In contrast, CPCV derives the distribution of Sharpe ratios from a large number of paths, $j = 1, \ldots, \varphi$, with mean $\mathrm{E}[\{y_{i,j}\}_{j=1,\ldots,\varphi}] = \mu_i$ and variance $\sigma^2[\{y_{i,j}\}_{j=1,\ldots,\varphi}] = \sigma_i^2$. The variance of the sample mean of CPCV paths is

$$\sigma^2[\mu_i] = \varphi^{-2}\left(\varphi\sigma_i^2 + \varphi(\varphi-1)\sigma_i^2\bar{\rho}_i\right) = \varphi^{-1}\sigma_i^2\left(1 + (\varphi-1)\bar{\rho}_i\right)$$

where σ_i^2 is the variance of the Sharpe ratios across paths for strategy i, and $\bar{\rho}_i$ is the average off-diagonal correlation among $\{y_{i,j}\}_{j=1,\ldots,\varphi}$. CPCV leads to fewer false

discoveries than CV and WF, because $\bar{\rho}_i < 1$ implies that the variance of the sample mean is lower than the variance of the sample,

$$\varphi^{-1}\sigma_i^2 \leq \sigma^2\left[\mu_i\right] < \sigma_i^2$$

The more uncorrelated the paths are, $\bar{\rho}_i \ll 1$, the lower CPCV's variance will be, and in the limit CPCV will report the true Sharpe ratio $\mathrm{E}[y_i]$ with zero variance, $\lim_{\varphi\to\infty}\sigma^2[\mu_i] = 0$. There will not be selection bias, because the strategy selected out of $i = 1, \ldots, I$ will be the one with the highest *true* Sharpe ratio.

Of course, we know that zero variance is unachievable, since φ has an upper bound, $\varphi \leq \varphi\left[T, \frac{T}{2}\right]$. Still, for a large enough number of paths φ, CPCV could make the variance of the backtest so small as to make the probability of a false discovery negligible.

In Chapter 11, we argued that backtest overfitting may be the most important open problem in all of mathematical finance. Let us see how CPCV helps address this problem in practice. Suppose that a researcher submits a strategy to a journal, supported by an overfit WF backtest, selected from a large number of undisclosed trials. The journal could ask the researcher to repeat his experiments using a CPCV for a given N and k. Because the researcher did not know in advance the number and characteristics of the paths to be backtested, his overfitting efforts will be easily defeated. The paper will be rejected or withdrawn from consideration. Hopefully CPCV will be used to reduce the number of false discoveries published in journals and elsewhere.

EXERCISES

12.1 Suppose that you develop a momentum strategy on a futures contract, where the forecast is based on an AR(1) process. You backtest this strategy using the WF method, and the Sharpe ratio is 1.5. You then repeat the backtest on the reversed series and achieve a Sharpe ratio of –1.5. What would be the mathematical grounds for disregarding the second result, if any?

12.2 You develop a mean-reverting strategy on a futures contract. Your WF backtest achieves a Sharpe ratio of 1.5. You increase the length of the warm-up period, and the Sharpe ratio drops to 0.7. You go ahead and present only the result with the higher Sharpe ratio, arguing that a strategy with a shorter warm-up is more realistic. Is this selection bias?

12.3 Your strategy achieves a Sharpe ratio of 1.5 on a WF backtest, but a Sharpe ratio of 0.7 on a CV backtest. You go ahead and present only the result with the higher Sharpe ratio, arguing that the WF backtest is historically accurate, while the CV backtest is a scenario simulation, or an inferential exercise. Is this selection bias?

12.4 Your strategy produces 100,000 forecasts over time. You would like to derive the CPCV distribution of Sharpe ratios by generating 1,000 paths. What are the possible combinations of parameters (N, k) that will allow you to achieve that?

12.5 You discover a strategy that achieves a Sharpe ratio of 1.5 in a WF backtest. You write a paper explaining the theory that would justify such result, and submit

it to an academic journal. The editor replies that one referee has requested you repeat your backtest using a CPCV method with $N = 100$ and $k = 2$, including your code and full datasets. You follow these instructions, and the mean Sharpe ratio is –1 with a standard deviation of 0.5. Furious, you do not reply, but instead withdraw your submission, and resubmit in a different journal of higher impact factor. After 6 months, your paper is accepted. You appease your conscience thinking that, if the discovery is false, it is the journal's fault for not having requested a CPCV test. You think, "It cannot be unethical, since it is permitted, and everybody does it." What are the arguments, scientific or ethical, to justify your actions?

REFERENCES

Bailey, D. and M. López de Prado (2012): "The Sharpe ratio efficient frontier." *Journal of Risk*, Vol. 15, No. 2 (Winter). Available at https://ssrn.com/abstract=1821643.

Bailey, D. and M. López de Prado (2014): "The deflated Sharpe ratio: Correcting for selection bias, backtest overfitting and non-normality." *Journal of Portfolio Management*, Vol. 40, No. 5, pp. 94–107. Available at https://ssrn.com/abstract=2460551.

Bailey, D., J. Borwein, M. López de Prado, and J. Zhu (2014): "Pseudo-mathematics and financial charlatanism: The effects of backtest overfitting on out-of-sample performance." *Notices of the American Mathematical Society*, Vol. 61, No. 5, pp. 458–471. Available at http://ssrn.com/abstract=2308659.

Bailey, D., J. Borwein, M. López de Prado, and J. Zhu (2017): "The probability of backtest overfitting." *Journal of Computational Finance*, Vol. 20, No. 4, pp. 39–70. Available at https://ssrn.com/abstract=2326253.