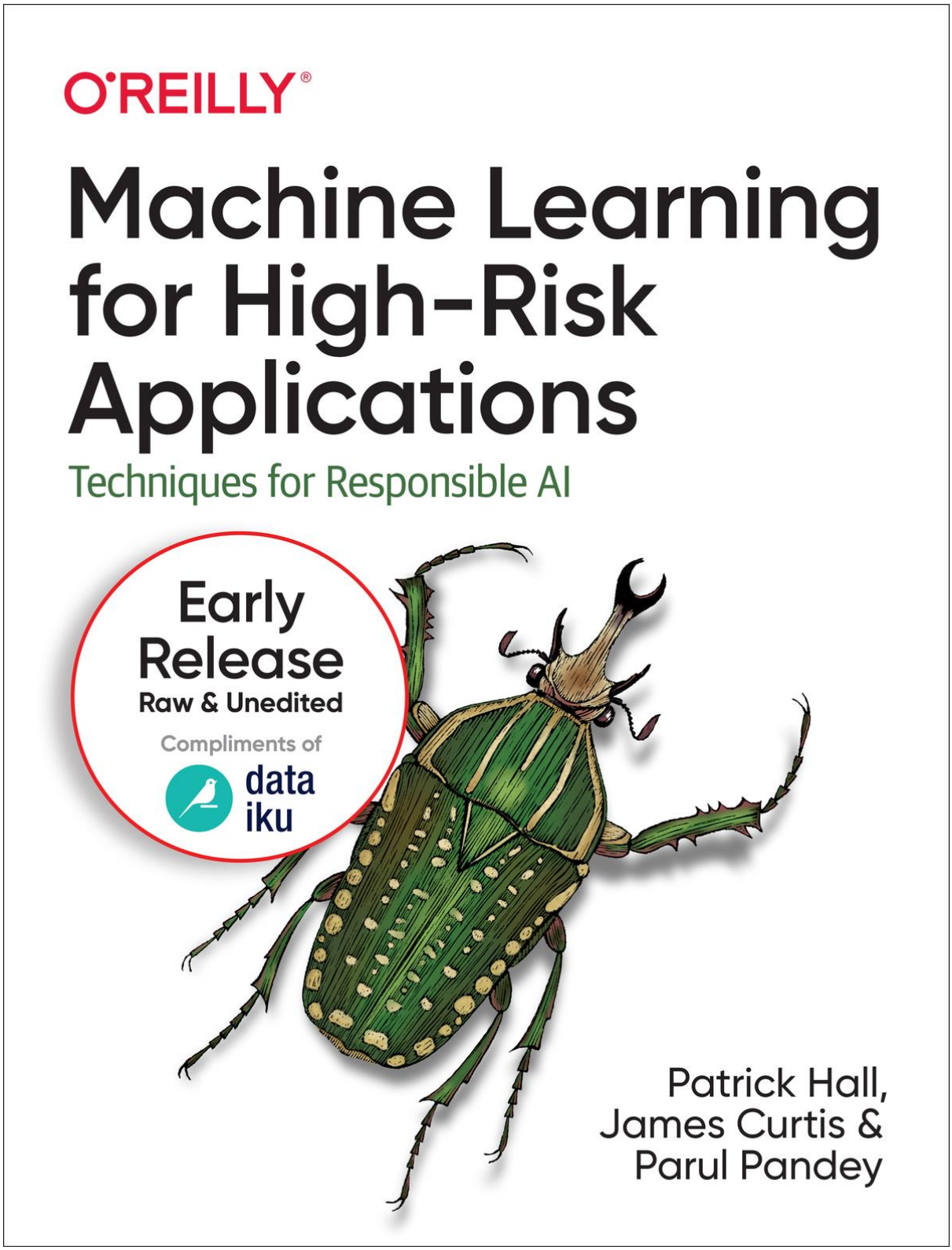
NOTE: This is only partial eBook, not a completed. If you want to Read full e-Book, You should have be buy it from HERE.



Dataiku

AI Governance With Dataiku
The Platform for Everyday AI

Leveraging one central solution for AI means more transparent, repeatable, and scalable AI. Dataiku gives people (whether technical and working in code or on the business side and low- or no-code) the ability to make better day-to-day decisions with data.

Deploy, Monitor, & Manage Machine Learning Projects in Production

Keep critical AI initiatives up and running with Dataiku, which offers model monitoring, drift detection, automatic model retraining, easy production project model updates, automated CI/CD, and more.

Manage Risk & Ensure Compliance at Scale

Advanced AI Governance including customizable governance plans, production sign-off, risk and value analysis together with permissions management, user directory integration, audit trails, and secure API access make Dataiku the perfect choice for streamlined AI Governance across the organization.

Understand Outputs, Increase Trust, & Identify Potential Bias

Dataiku provides critical capabilities for explainable and responsible AI, including reports on feature importance, partial dependence plots, subpopulation analysis, model fairness, and individual prediction explanations.

©2022 dataiku | dataiku.com

Machine Learning for High-Risk Applications

Techniques for Responsible AI

With Early Release ebooks, you get books in their earliest form—the author's raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

Patrick Hall, James Curtis, and Parul Pandey

Machine Learning for High-Risk Applications

by Patrick Hall

Copyright © 2022 Patrick Hall, James Curtin, and Parul Pandey. All rights reserved.

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (http://oreilly.com). For more information, contact our corporate/institutional sales department: 800-998-9938 or corporate@oreilly.com.

- Editors: Michele Cronin and Rebecca Novack
- Production Editor: Beth Kelly
- Copyeditor:
- Proofreader:
- Indexer:
- Interior Designer: David Futato
- Cover Designer: Karen Montgomery
- Illustrator: Kate Dullea

- August 2022: First Edition

Revision History for the Early Release

- 2021-05-26: First Release
- 2021-08-19: Second Release
- 2021-12-08: Third Release
- 2022-02-08: Fourth Release

See http://oreilly.com/catalog/errata.csp?isbn=9781098102432 for release details.

The O'Reilly logo is a registered trademark of O'Reilly Media, Inc. *Machine Learning for High-Risk Applications*, the cover image, and related trade dress are trademarks of O'Reilly Media, Inc.

The views expressed in this work are those of the author, and do not represent the publisher's views. While the publisher and the author have used good faith efforts to ensure that the information and instructions contained in this work are accurate, the publisher and the author disclaim all responsibility for errors or omissions, including without limitation responsibility for damages resulting from the use of or reliance on this work. Use of the information and instructions contained in this work is at your own risk. If any code samples or other technology this work contains or describes is subject to open source licenses or the intellectual property rights of others, it is your responsibility to ensure that your use thereof complies with such licenses and/or rights.

This work is part of a collaboration between O'Reilly and Dataiku. See our statement of editorial independence.

978-1-098-10243-2

[FILL IN]

Preface

Today, machine learning (ML) is the most commercially viable subdiscipline of artificial intelligence (AI). ML systems are used to make high-stakes decisions in employment, bail, parole, lending and in many other high-risk applications throughout the world's economies. In a corporate setting, ML systems are used in all parts of an organization — from consumer-facing products, to employee assessments, back-office automation, and more. Indeed, the past decade has brought with it wider adoption of ML technologies. But it has also proven that ML presents risks to it's operators and consumers. Unfortunately, and like nearly all other technologies, ML can fail — whether by unintentional misuse or intentional abuse. As of today, there have been over 1,000 public reports of algorithmic discrimination, data privacy violations, training data security breaches and other harmful incidents. Such risks must be mitigated before organizations, and the general public, can realize the true benefits of this exciting technology. This requires action from practitioners. While regulations, to which this book aims to adhere, are beginning to take shape, the practice of ML is still largely unregulated and lacks broadly accepted professional standards. That means it's largely up to individual practitioners to hold themselves accountable for the good and bad outcomes their technology may evoke when deployed into the real world. *Machine Learning for High-Risk Applications* will arm practitioners with a solid understanding of model governance processes and a new way to use common Python tools for training interpretable models and debugging them for performance, safety, fairness, security and privacy issues.

Who Should Read This Book

This is a technical book for engineers and data scientists, and the examples use Python. (We have tremendous respect for the R data community, and we have attempted to give many recommendations for R users wherever we can!) That said, this book probably isn't for every data scientist and engineer out there coding in Python. This book is for you if you want to learn some model governance basics and update your workflow to accommodate basic risk controls. This book is for you if your work needs to comply with certain nondiscrimination, transparency, privacy or security regulations. (Although we can't guarantee compliance or provide legal advice!). This book is for you if you want to train interpretable models, and learn to edit and debug them. This book is for you if you're coming from a field like econometrics or psychometrics, and want to learn how to blend newer machine learning techniques with established domain expertise and notions of causality. Finally, this book is for you if you're concerned that your work in ML maybe leading to unintended consequences relating to sociological bias, data privacy violations, security vulnerabilities, or other known problems caused by automated decision-making writ large, and you want to do something about it.

What Readers Will Learn

Readers of this book will be exposed to both traditional model governance and how to blend it with computer security best practices, like incident response, bug bounties, and red-teaming, to apply battle-tested risk controls to ML workflows and systems. This book will introduce a number of older and newer interpretable models, and explanation techniques that make ML systems even more transparent. Once we've setup a solid foundation of highly transparent models, we'll dig into testing models for performance and safety. That's a lot easier when you can see how your model works! We'll go way beyond quality measurements in holdout data to explore how to apply well-known diagnostic techniques like residual analysis, sensitivity analysis, and benchmarking to new types of ML models. We'll then progress to structuring models for fairer outcomes, testing for bias, and fixing bias from a technical perspective. Finally, we'll discuss data privacy and security basics and security specifics for ML pipelines and APIs.

Preliminary Book Outline

The book is broken into two parts. The first part discusses issues from a practical application perspective, with dashes of theory where necessary. Part Two contains long-form Python coding examples, addressing the topics in Part One from both structured and unstructured data perspectives.

Part 1

Part One begins with a deep dive into pending regulations, discussions of product liability and the Hand Rule, and a thorough treatment of traditional model risk management. Because many of these practices assume a somewhat staid and professional approach to modeling — a far cry from today's common "go fast and break things" ethos — we'll also discuss how to incorporate computer security best practices that assume failure into model governance. Chapter Two presents the burgeoning ecosystem of interpretable models. We cover the generalized additive model (GAM) family in the most depth, but also discuss many other types of high-quality and high-transparency estimators. Chapter Two also outlines many different post-hoc explanation techniques, but with an eye toward rigor and known problems with this somewhat over-hyped sub-field of responsible ML techniques. Chapter Three tackles model validation, but in a way that actually tests model's assumptions and real-world performance. We'll go over software testing basics as well as draw out highlights in the new field of model debugging. Chapter Four overviews the technical aspects of fairness and bias in ML, starting with appropriate experimental design, data collection and proxies. Chapter Four then treats bias testing in some detail, including tests for disparate impact, differential validity, and uncovering drivers of bias with Shapley values and other explanation techniques. Chapter Four also addresses both established and conservative methods for bias remediation, and more cutting edge dual-objective, adversarial, and pre-, in-, and post-processing remediation techniques. Chapter Five closes Part One by laying out how to red-team ML systems, starting with the basics of computer security and moving into discussions of common ML attacks, adversarial ML, and robust ML.

Part 2

Part Two expands on the ideas in Part One with a series of code example Chapters. Chapter Six puts explainable boosting machines (EBM), XGBoost, and explainable AI techniques

through their paces in a consumer finance example. Chapter Seven applies post-hoc explanation techniques to a PyTorch image classifier. In Chapter Eight, we'll debug our consumer finance models for performance problems, and do the same for our image classifier in Chapter Nine. Chapter Ten contains detailed examples relating to bias testing and bias remediation, and Chapter Eleven provides examples of ML attacks and countermeasures.

Stop Going Fast and Breaking Things

We end the book in Chapter Twelve with a plea to think through the materiality of your use case. For some low risk use cases, it might be fine to apply a quick and dirty approach. But as ML is used more in high-risk applications, the consequences of breaking things are becoming more serious. Our hope with the first edition of this text is to provide a legitimate alternative to the standard black-box, compressed-time-frame workflows that are common in ML today. This book should provide a set of vocabulary, ideas, tools, and techniques that enable practitioners to be more deliberate in their very important work.

Conventions Used in This Book

The following typographical conventions are used in this book:

Italic

: Indicates new terms, URLs, email addresses, filenames, and file extensions.

`Constant width`

: Used for program listings, as well as within paragraphs to refer to program elements such as variable or function names, databases, data types, environment variables, statements, and keywords.

`Constant width bold`

: Shows commands or other text that should be typed literally by the user.

`Constant width italic`

: Shows text that should be replaced with user-supplied values or by values determined by context.

TIP

This element signifies a tip or suggestion.

NOTE

This element signifies a general note.

WARNING

This element indicates a warning or caution.

Using Code Examples

Supplemental material (code examples, exercises, etc.) is available for download at *https://github.com/oreillymedia/title_title*.

If you have a technical question or a problem using the code examples, please send email to *bookquestions@oreilly.com*.

This book is here to help you get your job done. In general, if example code is offered with this book, you may use it in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing examples from O'Reilly books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.

We appreciate, but generally do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*Book Title* by Some Author (O'Reilly). Copyright 2012 Some Copyright Holder, 978-0-596-xxxx-x."

If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at *permissions@oreilly.com*.

O'Reilly Online Learning

NOTE

For more than 40 years, *O'Reilly Media* has provided technology and business training, knowledge, and insight to help companies succeed.

Our unique network of experts and innovators share their knowledge and expertise through books, articles, and our online learning platform. O'Reilly's online learning platform gives you on-demand access to live training courses, in-depth learning paths, interactive coding environments, and a vast collection of text and video from O'Reilly and 200+ other publishers. For more information, visit *http://oreilly.com*.

How to Contact Us

Please address comments and questions concerning this book to the publisher:

- O'Reilly Media, Inc.
- 1005 Gravenstein Highway North

- Sebastopol, CA 95472
- 800-998-9938 (in the United States or Canada)
- 707-829-0515 (international or local)
- 707-829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at *https://learning.oreilly.com/library/view/machine-learning-for/9781098102425/*.

Email *bookquestions@oreilly.com* to comment or ask technical questions about this book.

For news and information about our books and courses, visit *http://oreilly.com*.

Find us on Facebook: *http://facebook.com/oreilly*

Follow us on Twitter: *http://twitter.com/oreillymedia*

Watch us on YouTube: *http://www.youtube.com/oreillymedia*

Acknowledgments

Chapter 1. Contemporary Model Governance

A NOTE FOR EARLY RELEASE READERS

With Early Release ebooks, you get books in their earliest form—the author's raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 1st chapter of the final book. Please note that the GitHub repo will be made active later on.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at *mcronin@oreilly.com*.

Complex systems tend to drift toward unsafe conditions unless constant vigilance is maintained.

— *Closing the AI Accountability Gap*, Google Research

Building the best AI system starts with cultural competencies and business processes. Along with a case that illustrates what happens when an AI system is built without proper rigor, Chapter 1 presents numerous cultural and procedural approaches you can use to improve AI performance and safeguard your organization's AI against real-world safety and performance problems. The primary goal of the methodologies discussed in this chapter is to create better AI systems. This might mean improved *in silica* test data performance. But it really means training models that perform as expected once deployed *in vivo*, so you don't lose money, hurt people or cause other harms.

Chapter 1 begins with a discussion of basic legal standards, to inform system developers of their fundamental obligations when it comes to safety and performance. Because those who do not study history are bound to repeat it, Chapter 1 then highlights AI incidents, and discusses why understanding AI incidents is important for proper safety and performance in AI systems. Since many AI safety concerns require thinking beyond technical specifications, Chapter 1 then blends model risk management (MRM) and information technology (IT) security best practices to put forward numerous ideas for improving AI safety culture and processes within organizations. Chapter 1 will close with a case study focusing on safety culture, legal ramifications, and AI incidents.

Basic Legal Obligations

As makers of consumer products, data scientists and ML engineers have a fundamental obligation to create safe systems. To quote a recent Brookings Institute report, *Products liability law as a way to address AI harms*, "Manufacturers have an obligation to make products that will be safe when used in reasonably foreseeable ways. If an AI system is used in a foreseeable way and yet becomes a source of harm, a plaintiff could assert that the manufacturer was negligent in not recognizing the possibility of that outcome." Just like car or power tool manufacturers, makers of AI systems are subject broad legal standards for negligence and safety. Product safety has been the subject of large amounts of legal and economic analysis, but this subsection will focus on one of the first and simplest standards for negligence: the Hand Rule. Named after Judge Learned Hand, and coined in 1947, provides a viable framework for AI product makers to think about negligence and due diligence. The Hand Rule says that a product maker takes on a burden of care, and that such care should always be greater than the probability that an incident involving the product occurs multiplied by the expected loss related to an incident. Stated algebraically:

$B \geq PL$

In more plain terms, organizations are expected to apply care, i.e. time, resources, or money, to a level commensurate to the cost associated with a risk. Otherwise legal liability can ensue.

In Figure 1, Burden is the parabolicly increasing line, and risk, or Probability multiplied by Loss, is the parabolicly decreasing line. While these lines are not related to a specific measurement, their parabolic shape is meant to reflect the last mile problem in removing all AI system risk, and that the application of additional care beyond a reasonable threshold leads to diminishing returns for decreasing risk as well.

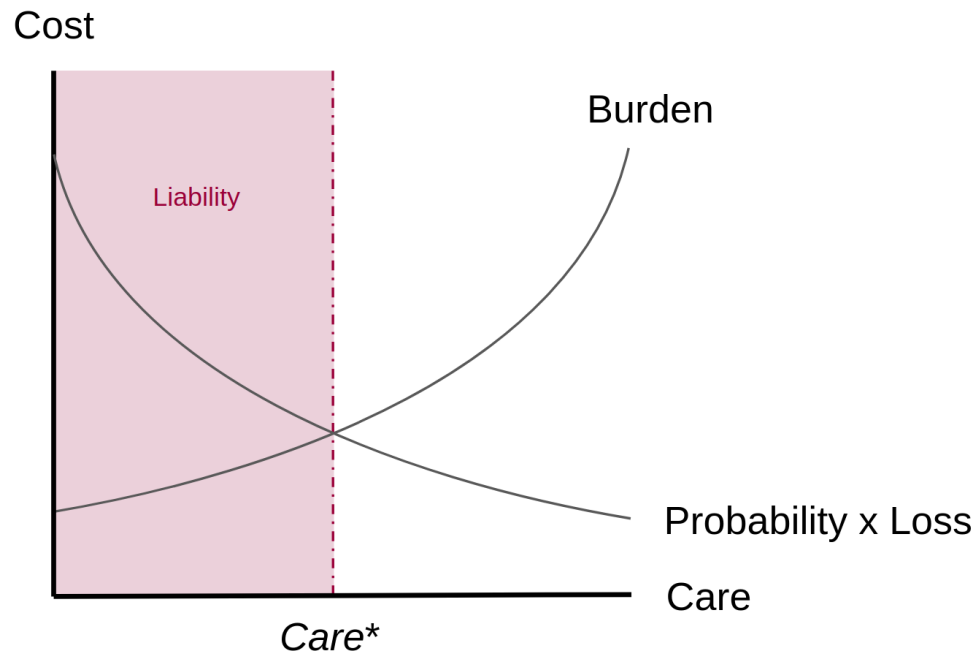


Figure 1-1. An illustration of the Hand Rule. Adapted from Economic Analysis of Alternative Standards of Liability in Accident Law.

While it's probably too resource intensive to calculate the quantities in the Hand Rule exactly, it is important to think about these concepts of negligence and liability when designing an AI system. For a given AI system, if the probability of an incident is high, if the monetary, or other, loss associated with a system incident is large, or both quantities are large, your organization needs to spend extra resources on ensuring safety for that system. Moreover, your organization should document to the best of your ability that due diligence exceeds the estimated failure probabilities multiplied by the estimated losses.

Of course there are legal considerations beyond product liability. The United States (US) Federal Trade Commission's (FTC) recent rounds of AI guidance are also important for safety and performance. The FTC is urging organizations deploying AI to prioritize fairness, transparency, accountability, and mathematical soundness. While many of those subjects are better suited for other chapters, accountability is crucial for safety and performance. In this context, accountability often means holding yourself to independent standards and allowing for independent oversight. The FTC has also been crystal clear about deceptive practices. AI cannot be used to deceive consumers, or you may face serious enforcement activities. When the FTC found that the photosharing app EverAlbum was being used to collect training data for a facial recognition system operating under a parallel line of business named Paravision, they forced the deletion of the facial recognition system and levied orders to prevent revenue generation based off the deceptive practices.

Much like the EU General Data Protection Regulation (GDPR) has changed the way companies handle data in US, any EU AI regulations will likely have an out-sized impact on US AI deployments. Well, the EU did recently propose sweeping and wide-ranging AI regulations. These regulations cover nearly every aspect of the commercial use of AI, and for safety and performance they mandate risk-tiering, and differing levels of system documentation, quality management, and monitoring based on risk determination. The remainder of Chapter 1 and much of Chapter 3 will be provide helpful information on addressing these requirements and more.

AI Incidents

In many ways, the fundamental goal of the AI safety processes and related model debugging discussed in Chapter 3, is to prevent and mitigate AI incidents. Here, we'll loosely define AI incidents as any outcome of the system that could cause harm. And using the Hand rule as a guide, the severity of an AI incident is increased by the loss the incident causes, and decreased by the care taken by the operators to mitigate those losses.

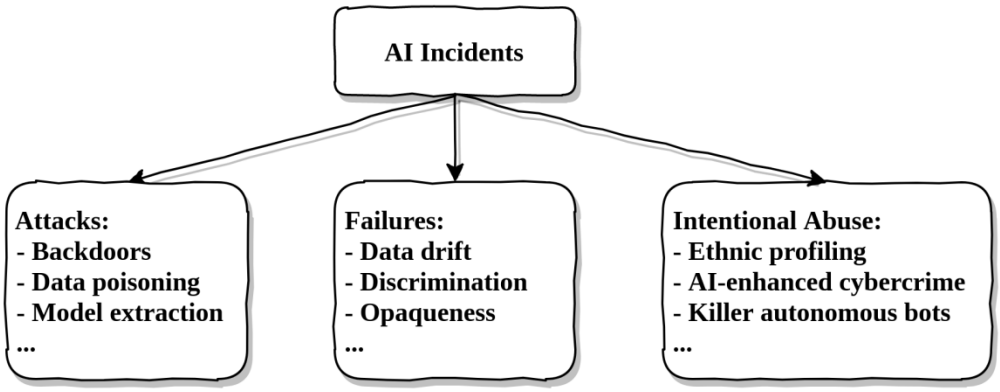


Figure 1-2. A basic taxonomy of AI incidents. Adapted from What to Do When AI Fails.

Because complex systems drift toward failure, there is no shortage of AI incidents to discuss as examples. AI incidents can range from annoying to deadly — from mall security robots falling down stairs, to self-driving cars killing pedestrians, to mass-scale diversion of healthcare resources away from those who need them most. As pictured in Figure 2, AI incidents can be roughly divided into three buckets.

- **Abuses**: AI can be used for nefarious purposes, apart from specific hacks and attacks on AI systems. The day may already be have come where hackers use AI to increase the efficiency and potency of their more general attacks. What the future could hold is even more frightening. Specters like autonomous drone attacks and ethnicity profiling by authoritarian regimes are already on the horizon.
- **Attacks**: Examples of all major types of attacks - confidentiality, integrity, and availability attacks (see Chapter 5 for more information) - have been published by researchers. Confidentiality attacks involve the exfiltration of training data or model logic from AI system end-points. Integrity attacks include adversarial manipulation of training data or model outcomes, either through adversarial examples, evasion, impersonation, or poisoning. Availability attacks can be conducted through more standard denial-of-service approaches, or via algorithmic discrimination induced by some adversary to deny system services to certain groups of users.
- **Failures**: AI system failures tend to involve algorithmic discrimination, safety and performance lapses, data privacy violations, inadequate transparency, or problems in third party system components.

AI incidents are a reality. And like the systems from which they arise, AI incidents can be complex. AI incidents have multiple causes: failures, attacks, and abuses. They also tend to blend traditional notions of computer security, with concerns like data privacy and algorithmic discrimination.

The 2016 Tay chatbot incident is an informative example. Tay was a state-of-the-art chatbot trained by some of the world's leading experts at Microsoft Research for the purpose of interacting with people on Twitter to increase awareness about AI. Sixteen hours after its release - and 96,000 tweets later - later Tay had spiraled into a neo-nazi pornagrapher and had to be shut down. What happnened? Twitter users quickly learned that Tay's adpative learning system could easily be poisoned. Racist and sexual content tweeted at the bot was quickly incorporated into its training data, and just as quickly resulted in offensive output. Data poisoning is an integrity attack, but due to the context in which it was carried out, this attack resulted in algorthmic discrimination. It's also important to note that Tay's designers, being

world-class experts at an extremely well-funded research center, seemed to have put some guide rails in place. Tay would respond to certain hot-button issues with pre-canned responses. But that was not enough, and Tay devolved into a public security and algorithmic discrimination incident for Microsoft Research.

Likely because of nothing more than silly hype, Tay was released without counter-measures for attacks, and due to the complexity of its operating environment this security breach morphed into a large-scale algorithmic discrimination incident. Think this was a one-off incident? Wrong. Just recently, again due to hype and failure to think through performance, safety, and security risks systematically, many of Tay's most obvious failures were repeated in ScatterLab's release of it's Lee Luda chatbot. When designing AI systems, plans should be compared to past known incidents in hope of preventing future similar incidents. This is precisely the point of recent AI incident database efforts and associated publications.

AI incidents can also be an apolitical motivator for responsible technology development. For better or worse, cultural and political viewpoints on topics like algorithmic discrimination and data privacy can vary widely. Getting a team to agree on ethical considerations can be very difficult. It might be easier to get them working to prevent embarrassing and potentially costly or dangerous incidents, which should be a baseline goal of any serious data science team. The notion of AI incidents is central to understanding AI safety and a central theme of this chapter's content is cultural competencies and business processes that can be used to prevent and mitigate AI incidents. We'll dig into those mitigants in the next sections and take a deep dive into a real incident to close the chapter.

Organizational and Cultural Competencies for Responsible AI

An organization's culture is an essential aspect of responsible AI. This section will discuss the cultural competencies like accountability, drinking your-own champagne, domain expertise, and the stale adage, "go fast and break things."

Accountability

A key to the successful mitigation of AI risks is real accountability within organizations for AI incidents. If no one's job is at stake when an AI system fails, gets attacked, or is abused for nefarious purposes, then it's entirely possible that no one in that organization really cares about AI safety and performance. In addition to developers who think through risks, apply software quality assurance (QA) techniques, and model debugging methods, organizations need individuals or teams who validate AI system technology and audit associated processes. Organizations also need someone to be responsible for AI incident response plans. All of this is why leading financial institutions, whose use of predictive modeling has been regulated for decades, employ a practice known as model risk management (MRM). MRM is patterned off the Federal Reserve's S.R. 11-7 model risk management guidance, that arose out the of the financial crisis during the Great Recession. Notably, implementation of MRM often involves accountable executives and several teams that are responsible for safety and performance of models and AI systems.

Leadership and Teams: Chief Model Risk Officer and the Three Lines of Defense

Implementation of MRM standards usually requires several different teams and executive leadership. Two key tenants form the cultural backbone for MRM:

- **Effective Challenge**: Effective challenge dictates that personnel who did not build an AI system perform validation and auditing of such systems. MRM practices typically distribute effective challenge across three "lines of defense," where system developers make up the first line of defense and independent technical validators and process auditors make up the second and third lines, respectively.
- **Accountable Leadership**: A specific executive within an organization should be accountable for ensuring AI incidents do not happen. This position is referred to as *chief model risk officer* (CMRO). It's also not uncommon for CMRO terms of employment and compensation structure to be linked to AI system performance. The role of CMRO offers a very straightforward cultural check on AI safety and performance. When your boss really cares about AI system safety and performance, then you start to care too.
- **Incentives**: Data science staff and management must be incentivized to implement AI responsibly. Often, compressed product timelines can incentivize the creation of a minimum viable product first, with rigorous testing and remediation relegated to the end of the model life cycle immediately before deployment to production. Moreover, AI testing and validation teams are often evaluated by the same criterion as AI development teams, leading to a fundamental misalignment where testers and validators are encouraged to move quickly rather than assure quality. Aligning timeline, performance evaluation, and pay incentives to team function helps solidify a culture of responsible AI and risk mitigation.

Of course, small or young organizations may not be able to spare an entire full-time employee to monitor ML AI system risk. But it's important to have an individual or group responsible and held accountable if AI systems cause incidents. If an organization assumes everyone is accountable for ML risk and AI incidents, the reality is that no one is accountable.

Cultural effective challenge

Whether your organization is ready to adopt full-blown MRM practices, or not, you can still benefit from certain aspects of MRM. In particular, the cultural competency of effective challenge can be applied outside of the MRM context. At it's core, effective challenge means actively challenging and questioning steps in the development of AI systems. An organizational culture that encourages serious questioning of AI system designs will be more likely to develop effective AI systems or products, and to catch problems before they explode into harmful incidents. Note that effective challenge cannot be abusive, and it must apply equally to all personnel developing an AI system, especially so-called "rockstar" engineers and data scientists. Effective challenge should also be structured, such as weekly meetings where current design thinking is questioned and alternative design choices are considered.

Drinking Your Own Champagne

Also known as "eating your own dog food," the practice of drinking your own champagne refers to using your own software or products inside of your own organization. Often a form

of pre-alpha or pre-beta testing, drinking your own champagne can identify problems that emerge from the complexity of real-world deployment environments before bugs and failures affect customers, users or the general public. Because serious issues like concept drift, algorithmic discrimination, shortcut learning or underspecification are notoriously difficult to identify using standard ML development processes, drinking your own champagne provides a limited and controlled, but also realistic, test bed for AI systems. Of course, when organizations employ demographically and professional diverse teams, including domain experts in the field where the AI system will be deployed, drinking your own champagne is more likely to catch a wider variety of problems. Drinking your own champagne also brings the classical Golden Rule into AI. If you're not comfortable using a system on yourself or your own organization, then you probably shouldn't deploy that system.

Diverse and Experienced Teams

Diverse teams can bring wider and uncorrelated perspectives to bear on design, development, and testing AI systems. Non-diverse teams often do not. Many have documented the unfortunate outcomes that can arise as a result of data scientists not considering demographic diversity in the training or results of AI systems. A potential solution to these kinds of oversights is increasing demographic diversity on AI teams from its current woeful levels. Business or other domain experience is also important when building teams. Domain experts are instrumental in feature selection and engineering, and in the testing of system outputs. In the mad rush to develop AI systems, domain expert participation can also serve as a safety check. Generalist data scientists often lack the experience necessary to deal with domain-specific data and results. Misunderstanding the meaning of input data or output results is a recipe for disaster that can lead to AI incidents when a system is deployed. Unfortunately, the social sciences deserve a special emphasis when it comes to data scientists forgetting or ignoring the importance of domain expertise. In a trend referred to as "tech's quiet colonization of the social sciences," several organizations have pursued regrettable AI projects that seek to replace decisions that should be made by trained social scientists or that simply ignore the collective wisdom of social science domain expertise altogether.

"Going Fast and Breaking Things"

The mantra, "go fast and break things," is almost a religious belief for many "rock-star" engineers and data scientists. Sadly, these top practitioners also seem to forget that when they go fast and break things, things get broken. As AI systems make more high-impact decisions that implicate autonomous vehicles, credit, employment, grades and university attendance, medical diagnoses and resource allocation, mortgages, pre-trial bail, parole and more, breaking things means more than buggy apps. It can mean that a small group of data scientists and engineers causes real harm at scale to many people. Participating in the design and implementation of high-impact AI systems requires a mindset change to prevent egregious performance and safety problems. Practitioners must change from prioritizing the number of software features they can push or the test data accuracy of an ML model, to recognizing the implications and downstream risks of their work.

Organizational Processes for Responsible AI

Organizational processes play a key role in assuring AI systems are safe and performant. Like the cultural competencies discussed in the previous section, organizational processes are a key non-technical determinant of reliability in AI systems. This section on processes starts out by urging practitioners to consider, document, and attempt to mitigate any known or foreseeable failure modes for their AI systems. This section then discusses a mature and tested process framework for governing predictive models known as model risk management (MRM). While the culture section focused on the people and mindsets necessary to make MRM a success, this section will outline the different processes MRM uses to mitigate risks in advanced predictive modeling and ML systems. While MRM is an incredible process standard to which we can all aspire, there are additional important process controls that are not typically part of MRM. We'll look beyond traditional MRM in this section and highlight crucial processes for change management, pair or double programming, and security permission requirements for code deployment. This section will close with a discussion of AI incident response. Nearly all powerful commercial technologies suffer incidents. AI is no different. No matter how hard we work to minimize harms while designing and implementing an AI system, we still have to prepare for failures and attacks.

Forecasting Failure Modes

AI safety and ethics experts roughly agree on the importance of thinking through, documenting, and attempting to mitigate foreseeable failure modes for AI systems. Unfortunately they also mostly agree that this is a nontrivial task. Happily, new resources and scholarship on this topic have emerged in recent years that can help AI system designers forecast incidents in more systematic ways. If holistic categories of potential failures can be identified, it makes hardening AI systems for better real-world performance and safety a more pro-active and efficient task. In this subsection, we'll discuss one such strategy, along with a few additional processes for brainstorming future incidents in AI systems.

Known Past Failures

As discussed in *Preventing Repeated Real World AI Failures by Cataloging Incidents: The AI Incident Database*, one the most efficient ways to mitigate potential AI incidents in your AI systems is to compare your system design to past failed designs. Much like transportation professionals investigating incidents, cataloging incidents, using the findings to prevent related incidents, and to test new technologies, several AI researchers, commentators, and trade organizations have begun to collect and analyze AI incidents in hopes of preventing repeated and related failures. Likely the most high-profile and mature AI incident repository is the Partnership on AI's AI Incident Database. This searchable and interactive resource allows registered users to search a visual database with keywords and locate different types of information about publicly recorded incidents. Others have begun collecting AI incidents in simpler GitHub repositories:

- AI Incident Tracker
- Awful AI
- Learning from the past to create Responsible AI

Consult these resources while developing your AI systems. If you see something that looks familiar, stop and think about what you're doing. If a system similar to the one you're designing, implementing, or deploying has caused an incident in the past, this is one of strongest indicators that your new system could cause an incident.

Failures of Imagination

Imagining the future with context and detail is never easy. And it's often the context in which AI systems operate, accompanied by unforeseen or unknowable details, that lead to AI incidents. In a recent workshop paper, the authors of *Overcoming Failures of Imagination in AI Infused System Development and Deployment* put forward some structured approaches to hypothesize about those hard-to-imagine future risks. In addition to deliberating on the *who* (e.g., investors, customers, vulnerable non-users), *what* (e.g., well-being, opportunities, dignity), *when* (e.g., immediately, frequently, over long periods of time), and *how* (e.g., taking an action, altering beliefs) of AI incidents, they also urge system designers to consider:

- Assumptions that the impact of the system will be only beneficial, and to admit when uncertainty in system impacts exists.
- The problem domain and applied use cases of the system, as opposed to just the math and technology.
- Any unexpected or surprising results, user interactions, and responses to the system.

Causing AI incidents is embarrassing, if not costly or illegal, for organizations. AI incidents can also hurt consumers and the general public. Yet, with some foresight, many of the currently known AI incidents could have been mitigated, if not wholly avoided. It's also possible that in performing the due diligence of researching and conceptualizing AI failures, you find that your design or system must be completely reworked. If this is the case, take comfort that a delay in system implementation or deployment is likely less costly than the harms your organization or the public could experience if the flawed system was released.

Model Risk Management

The process aspects of MRM mandate thorough documentation of modeling systems, human review of systems, and ongoing monitoring of systems. These processes represent the bulk of the governance burden for the Federal Reserve's SR 11-7 MRM guidance, which is overseen by the Federal Reserve and the Office of the Comptroller of the Currency (OCC) for predicitve models deployed in material consumer finance applications. MRM represents the culmination of decades of predictive modeling governance in consumer finance, and lessons learned from incidents during those same years. While only large organizations will be able to fully embrace all that MRM has to offer, any serious AI practitioner can learn something from the discipline. The subsection below breaks MRM processes down into smaller components so that you can start thinking through using aspects of MRM in your organization.

Risk-tiering

As outlined in the opening of Chapter 1, the product of the probability of a harm occurring and likely loss resulting from that harm is an accepted way to rate the risk of given AI system deployment. The product of risk and loss has a more formal name in the context of MRM, *materiality*. Materiality is a powerful concept that enables organizations to assign realistic risk levels to AI systems. More importantly, this risk-tiering allows for the efficient allocation of limited development, validation, and audit resources. Of course, the highest materiality applications should recieve the greatest human attention and review, while the lowest materiality applications could potentially be handled by automatic machine learning

(AutoML) systems and undergo minimal validation. Because risk mitigation for AI systems is an ongoing task, proper resource allocation between high, medium, and low risk systems is a must for effective governance.

Model Documentation

MRM standards also require that systems be thoroughly documented. Moreover, documentation is where the rubber hits the road for compliance. Documentation templates, illustrated by the section list below, are documents that data scientists and engineers fill in as they move through a standardized workflow or in the later stages of model development. Documentation templates should include all the steps that a responsible practitioner should move through to build a sound model. If parts of the document aren't filled out, that points to sloppiness in the training process. Since most documentation templates and frameworks also call for adding one's name and contact information to the finished model document, there should be no mystery about who is not pulling their weight. For reference, the section list below is a rough combination of typical sections in MRM documentation and the sections recommended by the Annexes to the EU Artificial Intelligence Act.

- Basic Information
    - Names of Developers and Stakeholders
    - Current Date and Revision Table
    - Summary of Model System
    - Business or Value Justification
    - Intended Uses and Users
    - Potential Harms and Ethical Considerations
- Development Data Information
    - Source for Development Data
    - Data Dictionary
    - Privacy Impact Assessment
    - Assumptions and Limitations
    - Software Implementation for Data Preprocessing
- Model Information
    - Description of Training Algorithm with Peer-reviewed References
    - Specification of Model
    - Performance Quality
    - Assumptions and Limitations
    - Software Implementation for Training Algorithm
- Testing Information
    - Quality Testing and Remediation
    - Discrimination Testing and Remediation
    - Security Testing and Remediation
    - Assumptions and Limitations
    - Software Implementation for Testing
- Deployment Information
    - Monitoring Plans and Mechanisms
    - Up- and Down-stream Dependencies
    - Appeal and Override Plans and Mechanisms
    - Audit Plans and Mechanisms
    - Change Management Plans
    - Incident Response Plans

- References (If you're doing science, then you're building on the shoulders of giants and you will have several peer-reviewed references in a formatted bibliography!)

Of course, these documents can be hundreds of pages long, especially for high-materiality systems. If you're feeling like that sounds impossible for your organization today, then maybe these two simpler frameworks might work instead. First, documentation should enable accountability for system stakeholders, ongoing system maintenance, and a degree of incident response. Second, documentation must be standardized across systems, for the most efficient audit and review processes. The proposed datasheet and model card standards may also be helpful for smaller or younger organizations to meet these goals.

Model Monitoring

A primary tenant of AI safety is that AI system performance in the real-world is hard to predict and performance must be monitored. Hence, deployed system performance should be monitored frequently and until a system is decommissioned. Systems can be monitored for any number of problematic conditions, the most common being input drift. While AI system training data encodes information about a system's operating environment in a static snapshot, the world is anything but static. Competitors can enter markets, new regulations can be passed, consumer tastes can change, and pandemics or other disasters can happen. Any of these can change the live data that's entering your AI system away from the characteristics of its training data, resulting in decreased, or even dangerous, system performance. To avoid such unpleasant surprises, the best AI systems are monitored both for drifting input and output distributions and for decaying quality, often known as *model decay*. While performance quality is the most common quantity to monitor, AI systems can also be monitored for anomalous inputs or predictions, specific attacks and hacks, and for drifting fairness characteristics.

Model Inventories

Any organization that is deploying AI should be able to answer straightforward questions like:

- How many AI systems are currently deployed?
- How many customers or users do these systems affect?
- Who are the accountable stakeholders for each system?

MRM achieves this goal through the use of model inventories. A model inventory is a curated and up-to-date database of all an organization's AI systems. Model inventories can serve as a repository for crucial information in documentation, but should also link to monitoring plans and results, auditing plans and results, important past and upcoming system maintenance and changes, and plans for incident response.

System Validation and Auditing

Under traditional MRM practices, an AI system undergoes two primary reviews before its release. The first review is a technical validation of the system, where skilled validators, not uncommonly Ph.D. data scientists, attempt to poke holes in system design and implementation, and work with system developers to fix any discovered problems. The second review investigates processes. Audit and compliance personnel carefully analyze the

system design, development, and deployment, along with documentation and future plans, to ensure all regulatory and internal process requirements are meant. Only after these two reviews, would an executive sign-off for deployment begin. Moreover, because AI systems change and drift over time, review must take place whenever a system undergoes a major update or at an agreed upon future cadence.

You may be thinking (again) that your organization doesn't have the resources for such heavy-handed reviews. Of course that is a reality for many small or younger organizations. The keys for validation and auditing, that should work at nearly an organization, are having technicians who did not develop the system test it, having a function to review non-technical internal and external obligations, and having sign-off oversight for important AI system deployments.

Beyond Model Risk Management

MRM is not the only place to draw inspiration for improved AI safety and performance processes. There are also lots of lessons to be learned from software development best practices and from IT security. This subsection will shine a light on pair programming, least privilege, change management and incident response from an AI safety and performance perspective.

Pair and Double Programming

Because they tend to be complex and stochastic, it's hard to know if any given ML algorithm implementation is correct! This is why some leading AI organizations implement ML algorithms twice as a quality assurance (QA) mechanism. Such double implementation is usually achieved by one of two methods: pair programming or double programming. In the pair programming approach, two technical experts code an algorithm without collaborating. Then they join forces and work out any discrepancies between their implementations. In double programming, the same practitioner implements the same algorithm twice, but in very different programming languages, such as Python (object-oriented) and SAS (procedural). They must then reconcile any differences between their two implementations. Either approach tends to catch numerous bugs that would otherwise go unnoticed until the system was deployed. Pair and double programming can also align with the more standard workflow of data scientists prototyping algorithms, while dedicated engineers harden them for deployment. However, for this to work, engineers must be free to challenge and test data science prototypes and not relegated to simply re-coding prototypes.

Security Permissions for Code Deployment

The concept of *least privilege* from IT security states that no system user should ever have more permissions than they need. Least privilege is a fundamental process control that, likely because AI systems touch so many other IT systems, tends to be thrown out the window for AI build-outs and for so-called "rock star" data scientists. Unfortunately, this is an AI safety and performance anti-pattern. Outside the world of over-hyped AI and rock star data science, it's long been understood that engineers cannot adequately test their own code and that others in a product organization, product managers, attorneys, or executives, should make the final call as to when software is released.

For these reasons, the IT permissions necessary to deploy an AI system should be distributed across several teams within an IT organizations. During development sprints, data scientists and engineers certainly must retain full control over their development environments. But, as important releases or reviews approach, the IT permissions to push fixes, enhancements, or new features to user-facing products are transferred away from data scientists and engineers to product managers, legal, executives or others. Such process controls provide a gate that prevents unapproved code from being deployed.

Change Management

Like all complex software applications, AI systems tend to have a large number of different components. From backend ML code, to application programming interfaces (APIs), to user interfaces, changes in any component of the system can cause side-effects in other components. Add in issues like data drift, emergent data privacy and anti-discrimination regulations, and complex dependencies on third-party software, and change management in AI systems becomes a serious concern. There are many frameworks and project management approaches for change management. If you're in the planning or design phase of a mission-critical AI system, you'll likely need to make change management a first-class process control. Without explicit planning and resources for change management, process or technical mistakes that arise through the evolution of the system, like using data without consent or API mismatches, are very difficult to prevent. Furthermore, without change management, such problems might not even be detected until they cause an incident.

AI Incidents Response

According to the vaunted SR 11-7 guidance, "even with skilled modeling and robust validation, model risk cannot be eliminated". If risks from AI systems and ML models cannot be eliminated, then such risks will eventually lead to incidents. Incident response is already a mature practice in the field of computer security. Venerable institutions like NIST and SANS have published computer security incident response guidelines for years. Given that AI is a less mature and higher-risk technology than general purpose enterprise computing, formal AI incident response plans and practices are a must for high-impact or mission critical AI systems.

Formal AI incident response plans enable organizations to respond more quickly and effectively to inevitable incidents. Incident response also plays into the Hand Rule discussed at the beginning of Chapter 1. With rehearsed incident response plans in place, organizations may be able to identify, contain, and eradicate AI incidents before they spiral into costly or dangerous public spectacles. Although only mandated by regulation in a few specific verticals as of today, AI incident response plans are one of the most basic and universal ways to mitigate AI-related risks. Before a system is deployed, incident response plans should be drafted and tested. For young or small organizations that cannot fully implement model risk management, AI incident response is a primary and potent AI risk control to consider. Borrowing from computer incident response, AI incident response can be thought of in six phases:

Phase 1: Preparation

In addition to clearly defining an AI incident for your organization, preparation for AI incidents includes personnel, logistical, and technology plans for when an incident occurs.

Budget must be set aside for response, communication strategies must be put in place, and technical safeguards for standardizing and preserving model documentation, out-of-band communications, and shutting down AI systems must be implemented. One of the best ways to prepare and rehearse for AI incidents are table top discussion exercises, where key organizational personnel work through a realistic incident. Good starter questions for an AI incident table top include:

- Who has the organizational budget and authority to respond to an AI incident?
- Can the AI system in question be taken offline? By whom? At what cost? What upstream processes will be affected?
- Which regulators or law enforcement agencies need to be contacted? Who will contact them?
- Which external law firms, insurance agencies, or public relation firms need to be contacted? Who will contact them?
- Who will manage communications? Internally, between responders? Externally, with customers or users?

Phase 2: Identification

Identification is when organizations spot AI failures, attacks, or abuses. In practice, this tends to involve more general attack identification approaches, like network intrusion monitoring, and more specialized monitoring for AI system failures, like monitoring for concept drift or algorithmic discrimination. Identification also means staying vigilant for AI-related abuses. Often the last step of the identification phase is to notify management, incident responders, and others specified in incident response plans.

Phase 3: Containment

Containment refers to mitigating the incident's immediate harms. Keep in mind that harms are rarely limited to the system where the incident began. Like more general computer incidents, AI incidents can have network effects that spread throughout an organizations' and its customers' technologies. Actual containment strategies will vary depending on whether the incident stemmed from an external adversary, and internal failure, or an off-label use or abuse of an AI system. If necessary, containment is also a good place to start communicating with the public.

Phase 4: Eradication

Eradication involves remediating any affected systems. For example, sealing off any attacked systems from vectors of in- or ex-filtration, or shutting down a discriminatory AI system and temporarily replacing it with a trusted rule-based system. After eradication, there should be no new harms caused by the incident.

Phase 5: Recovery

Recovery means ensuring all affected systems are back to normal and that controls are in place to prevent similar incidents in the future. Recovery often means re-training or re-implementing AI systems, and testing that they are performing at documented pre-incident levels. Recovery can also require careful analysis of technical or security protocols for personnel, especially in the case of an accidental failure or insider attack.

Phase 6: Lessons Learned

Lessons learned refers to corrections or improvements of AI incident response plans based on the the successes and challenges encountered while responding to the current incident. Response plan improvements can be process- or technology-oriented.

For a sneak peek at a free and open AI incident response plan, see the Sample Incident Response Plan provided by the specialty law firm bnh.ai.

Case Study: Death by Autonomous Vehicle

On the night of March 18th 2018, Elaine Herzberg was walking a bicycle across a wide intersection in Tempe, Arizona. In what has become one of the most high-profile AI incidents, she was struck by an autonomous Uber test vehicle traveling at roughly 40 mph. According to the National Safety Transportation Board (NTSB), the test vehicle driver, who was obligated to take control of the vehicle in emergency situations, was distracted by a smart phone. The self driving AI system also failed to save Ms. Herzberg. The system did not identify Ms. Herzberg until 1.2 seconds before impact, too late to prevent a brutal crash.

Fallout

Autonomous vehicles are thought to offer safety benefits over today's status quo of human-operated vehicles. Indeed, self-driving cars have driven millions of miles with no fatalities. Yet, the NTSB's report states that Uber's, "system design did not include a consideration for jaywalking pedestrians," and criticized lax risk assessements and immature safety culture at the company. Furthermore, an Uber employee raised serious concerns about 37 crashes in the previous 18 months and common problems with test vehicle drivers just days before the Tempe incident. As a result of the Tempe crash, Uber's autonomous vehicle testing was stopped in four other cities and governments around the US and Canada began re-examining safety protocols for self-driving vehicle tests. The driver has been charged with negligent homicide. Uber has been excused from criminal liability, but came to a monetary settlement with the deceased's family. The city of Tempe and State of Arizona were also sued by Ms. Herzberg's family for $10 million each.

An Unprepared Legal System

It must be noted that the legal system in the US is not yet prepared for the reality of AI incidents, leaving employees, consumers and the general public largely unprotected from the unique dangers of AI systems operating in our midst. The EU Parliament has put forward a liability regime for AI systems that would mostly prevent large technology companies from escaping their share of the consequences in future incidents. In the US, any plans for Federal AI product safety regulations are still in a highly preliminary phase. In the interim, individual cases of AI safety incidents will likely be decided by lower courts with little education and experience in handling AI incidents, enabling Big Tech and other AI system operators to bring vastly asymmetric legal resources to bear against individuals caught up in incidents related to complex AI systems. Even for the companies and AI system operators, this legal limbo is not ideal. While the lack of regulation seems to benefit those with the most resources and expertise, it makes risk management and predicting the outcomes of AI incidents more

difficult. Regardless, future generations may judge us harshly for allowing the criminal liability of one of the first AI incidents, involving many data scientists and other highly paid professionals, to be pinned solely on a test driver of a supposedly automated vehicle.

Lessons Learned

What lessons learned from Chapter 1 could be applied to this case?

- **Lesson 1: Culture is important.** A mature safety culture is a broad risk control, bringing safety to the forefront of design and implementation work, and picking up the slack in corner cases that processes and technology miss. Learned from the last generation of life-changing commercial technologies, like aerospace travel and nuclear power, a more mature safety culture at Uber could have prevented this incident, especially since an employee raised serious concerns in the days before the crash.
- **Lesson 2: Mitigate foreseeable failure modes.** The NTSB concluded that Uber's software did not specifically consider jaywalking pedestrians as a failure mode. For anyone who's driven a car with pedestrians around, this should have been an easily foreseeable problem for which any self-driving car should be prepared. AI systems generally are not prepared for incidents unless their human engineers make them prepared. This incident shows us what happens when those preparations are not made in advance.
- **Lesson 3: Test AI systems in their operating domain.** After the crash, Uber stopped and reset its self-driving car program. After improvements, they were able to show via simulation that their new software would have started breaking 4 seconds before impact. Why wasn't the easily foreseeable reality of jaywalking pedestrians tested with these same in-domain simulations before the March 2018 crash? The public may never know. But enumerating failure modes and testing them in realistic scenarios could prevent you or your organization from having to answer these kinds of unpleasant questions.

A potential bonus lesson here is to consider not only accidental failures, like the Uber crash, but also malicious hacks against AI systems and the abuse of AI systems to commit violence. Terrorists have turned motor vehicles into deadly weapons before, so this is a known failure mode. Precautions must be taken in autonomous vehicles, and in driving assistance features, to prevent hacking and violent outcomes in these systems. Regardless of whether its an accident or a malicious attack, AI incidents will certainly kill more people. Our hope is that governments and other organizations will take AI safety seriously, and minimize the number of these somber incidents in the future.

Chapter 2. Debugging Machine Learning Systems for Safety and Performance

A NOTE FOR EARLY RELEASE READERS

With Early Release ebooks, you get books in their earliest form—the author's raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 3rd chapter of the final book. Please note that the GitHub repo will be made active later on.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at *mcronin@oreilly.com*.

Make sure that your AI models are validated and revalidated to ensure that they work as intended, and do not illegally discriminate.

— Using AI and Algorithms, United States Federal Trade Commission

For decades, error or accuracy on hold-out test data has been the standard by which machine learning (ML) models are judged. Unfortunately, as machine learning (ML) models are embedded into artificial intelligence (AI) systems that are deployed more broadly and for more sensitive applications, the standard approaches for ML model assessment have proven inadequate. For instance, test data area under the curve (AUC) tells us almost nothing about algorithmic discrimination, lack of transparency, privacy harms, or security vulnerabilities. Yet, these problems are often why AI systems fail once deployed. For acceptable *in vivo* performance, we simply must push beyond traditional *in silica* assessments designed primarily for research prototypes. Moreover, the best results for safety and performance occur when organizations are able to mix and match appropriate cultural competencies and process controls from Chapter 1 with AI technology that promotes trust. Chapter 3 presents sections on training, debugging, and deploying AI systems that delve into the numerous technical approaches for testing and improving safety, performance, and trust in AI.

Training

The discussion of training ML algorithms begins with reproducibility, because without that, it's impossible to know if any one version of an AI system is really any better than another. Data and feature engineering will be addressed briefly and the training section closes by outlining key points for model specification.

Reproducibility

Without reproducibility, you're building on sand. Reproducibility is fundamental to all scientific efforts, including AI. Without reproducible results, it's very hard to know if day-to-day efforts are improving, or even changing, an AI system. Reproducibility helps ensure proper implementation and testing, and some customers may simply demand it. The techniques discussed below are some of the most common that data scientists and ML engineers use to establish a solid, reproducible foundation for their AI systems.

- **Benchmark Models**: Benchmark models are important safety and performance tools for training, debugging, and deploying AI systems. They'll be addressed several times in Chapter 3. In the context of model training and reproducibility, you should always build from a reproducible benchmark model. This allows a checkpoint for rollbacks if reproducbility is lost, but it also enables real progress. If yesterday's benchmark is reproducible, and today's gains above and beyond that benchmark are also reproducible, that's real and measurable progress!
- **Hardware**: As AI systems often leverage hardware acceleration via graphical processing units (GPUs) and other specialized system components, hardware is still of special interest for preserving reproducibility. If possible, try to keep hardware as similar as possible across development, testing, and deployment systems.
- **Environments**: AI systems always operate in some computational environment, specified by the system hardware, system software, and your data and AI software stack. Changes in any of these can affect the reproducibility of ML outcomes. Thankfully, tools like Python virtual environments and Docker containers that preserve software environments have become commonplace in the practice of data science. Additional specialized environment management software from Domino Data Labs, gigantum, TensorFlow TFX, and Kubeflow can provide even more expansive control of computational environments.
- **Metadata**: Data about data is essential for reproducibility. Track all artifacts associated with the model, e.g., datasets, preprocessing steps, model, data and model validation results, human sign offs, and deployment details. Not only does this allow rolling-back to a specific version of a dataset or model, but it also allows for detailed debugging and forensic investigations of AI incidents. For an open-source example of a nice tool for tracking metadata, checkout TensorFlow ML Metadata.
- **Random Seeds**: Set by data scientists and engineers in specific code blocks, random seeds are the plow-horse of ML reproducibility. Unfortunately, they often come with language- or package-specific instructions. Seeds can take some time learn in different software, but when combined with careful testing, random seeds enable the building blocks of intricate and complex AI systems to retain reproducibility. This is a prerequisite for overall reproducibility.
- **Version Control**: Minor code changes can lead to drastic changes in ML results. Changes to your own code plus it's dependencies must be tracked in a professional version control tool for any hope of reproducibility. Git and GitHub are free and ubiquitous resources for software version control, but there are plenty of other options to explore. Crucially, data can also be version-controlled with tools like Pachyderm and DVC.

Though it may take some experimentation, some combination of these approaches and technologies should work to assure reproducibility in your AI system. Once this fundamental

safety and performance control is in place, it's time to consider other baseline factors like data quality and feature engineering.

Data Quality and Feature Engineering

Entire books have been written about data quality and feature engineering for ML and AI systems. This short subsection highlights some of the most critical aspects of this vast practice from a safety and performance perspective.

Starting with the basics, both the size and shape of a dataset are important safety and performance considerations. ML algorithms, that form the guts of today's AI systems, are hungry for data. Both small data and wide, sparse data can lead to catastrophic performance failures in the real-world, because both give rise to scenarios in which system performance appears normal on test data, but is simply untethered to real-world phenomena. Small data can make it hard to detect underfitting, underspecification, overfitting, or other fundamental performance problems. Sparse data can lead to over-confident predictions for certain input values. If an ML algorithm did not see certain data ranges during training due to sparsity issues, most ML algorithms will issue a predictions in those ranges with no warning that the prediction is based on almost nothing.

A number of other data quality problems can cause safety worries, mostly due to misrepresentation of important information or propensity to cause overfitting or pipeline process issues. In Chapter 3, misrepresentation means data is distorted in a way that is consistent across training data partitions, but results in incorrect decisions once a model trained on the incorrect data is applied to new, live data. Overfitting refers to memorizing noise in training data and resulting optimistic error estimates, and pipeline issues are problems that arise from combining different stages of data preparation and modeling components into one prediction-generating executable. The short checklist below can be applied to most standard ML problems to help identify common data quality problems with safety and performance implications.

Misrepresentation

- No duplicate data over-emphasizing unimportant information
- No incorrect encoding or decoding of character or binary data
- No incorrect use of time or date formats
- Minimal one-hot encoding — reducing unnecessary sparsity
- Minimal outliers — diminishing influence of outliers on model parameters or rules
- Minimal use of simplistic imputation — preventing distortion of training distributions
- Minimal correlation — increasing stability in model parameters or rules
- Thorough normalization of character values — preventing mistreatment of modeled entities

Overfitting

- No improper treatment of high cardinality categorical features
- No naive target encoding
- Minimal duplicate entities across training, validation, or test data partitions
- Training data with time or date feature not split randomly for validation and testing

Pipeline Issues

- All data cleaning and transformation steps applied during inference
- Re-adjusting for over- or under-sampling during inference

Of course, many other problems can arise in data preparation and feature engineering, especially as the types of data that ML algorithms can accept for training becomes more varied. Applying the lens of misrepresentation, overfitting, or pipeline issues to think through how discrepancies in data or pipelines can lead to errors once the system is deployed is usually a helpful exercise for spotting pitfalls. Tools that detect and address such problems are also an important part of the the data science toolkit. For Python Pandas users, the pandas-profiling tool is a visual aide that helps to detect many basic data quality problems. R users also have options, as discussed by Mateusz Staniak and Przemysław Biecek in *The Landscape of R Packages for Automated Exploratory Data Analysis*.

Model Specification

Once your data preparation and feature engineering pipeline is hardened, it's time to think about ML model specification. Considerations for real-world performance and safety are quite different from getting published or maximizing performance on ML contest leaderboards. While measurement of validation and test error remain important, bigger questions of accurately representing data and commonsense real-world phenomena have the highest priority. This subsection address model specification for safety and performance by highlighting the importance of benchmarks and alternative models, discussing the many hidden assumptions of ML models, and previewing the emergent disciplines of robust ML and ML safety and reliability.

Benchmarks and Alternatives

When starting a ML modeling task, it's best to begin with a peer-reviewed training algorithm, and ideally to replicate any benchmarks associated with that algorithm. While academic algorithms rarely meet all the needs of complex business problems, starting from a well-known algorithm and benchmarks provides a baseline assurance that the training algorithm is implemented correctly. Once this sanity check is addressed, then think about tweaking a complex algorithm to address specific quirks of a given problem.

Along with comparison to benchmarks, evaluation of numerous alternative algorithmic approaches is another best practice that can improve safety and performance outcomes. The exercise of training many different algorithms and judiciously selecting the best of many options for final deployment typically results in higher-quality models because it increases the number of models evaluated and forces users to understand differences between them. Moreover, evaluation of alternative approaches is important in complying with a broad set of U.S. non-discrimination and negligence standards. In general, these standards require evidence that different technical options were evaluated and an appropriate trade-off between consumer protection and business need was made before deployment.

Hidden Assumptions

Like undetected misrepresentation problems in training data causing major problems once an AI system is deployed, ML algorithms that don't align with fundamental structures in training

data or in the real-world domain can cause serious incidents. Often times data scientists think they matched their models to the problem domain, but a wide array of hidden assumptions can still cause problems.

Selecting to use an ML algorithm for a modeling problem also comes with a lot of basic assumptions — essentially that high-degree interactions and nonlinearity in input features are important drivers of the predicted phenomenon. Conversely, choosing to use a linear model implicitly downplays interactions and nonlinearities. If those qualities are important for high-quality predictions, they'll have to be specified explicitly for the linear model. In either case, it's important to take stock of how main effects, correlations and local dependencies, interactions, nonlinearities, clusters, outliers, and hierarchies in training data, or in reality, will be handled by a modeling algorithm and to test those mechanisms. For optimal safety and performance once deployed, dependencies on time, geographical locations, or connections between entities must also be represented within ML models. Testing for independence of errors between rows in training data or plotting model residuals and looking for strong patterns are general and time-tested methods for ensuring such dependencies have been addressed. It is also possible to directly constrain ML algorithms to reflect reality. Monotonicity constraints ensure known monotonic relationships are reflected in ML models. Interaction constraints can prevent arbitrary or undesirable combinations of input features from affecting model predictions.

Another often unstated assumption that comes with many learning algorithms involves squared loss functions. Many ML algorithms use a squared loss function by default. This means your model thinks that your modeling target variable follows a normal distribution. Some modeling targets are normally distributed, but many are not. Matching your target distribution to your loss function is an important step in aligning your ML algorithm to the problem domain. Hyperparameters for ML algorithms are yet another place where hidden assumptions can cause safety and performance problems. Hyperparameters can be selected based on domain knowledge or via technical approaches like grid search and Bayesian optimization. The key is doing this process systematically and not assuming the large number of default settings in most ML algorithms will work for your data and your problem.

The Future of Safe and Robust Machine Learning

The new field of robust ML is churning out new algorithms with improved stability and security characteristics. Various researchers are creating new learning algorithms with guarantees for optimality, like optimal sparse decision trees. And researchers have put together excellent tutorial materials on ML safety and reliability. Today, these approaches require custom implementations and extra work, but hopefully soon these safety and performance advances will be more widely available.

Model Debugging

Once a model has been properly specified and trained, the next step in the technical safety and performance assurance process is testing and debugging. In years past, such assessments focused on accuracy and error rates in holdout data. As ML models are incorporated in public-facing AI systems, and the number of publicly reported AI incidents is increasing dramatically, it's clear that more rigorous validation is required. The new field of model debugging is rising to meet this need. Model debugging treats ML models more like code and

less like abstract mathematics. It applies a number of different testing methods to find software flaws, logical errors, inaccuracies, and security vulnerabilities in ML models and AI system pipelines. Of course, these bugs must also be fixed when they are found. This section of Chapter 3 explores model debugging in some detail, starting with basic and traditional approaches, moving onto specialized testing techniques, delineating the common bugs we're trying to find, and closing with a discussion of bug remediation methods.

Software Testing

Basic software testing becomes much more important when we stop thinking of pretty figures and impressive tables of results as the end goal of an ML model training task. When AI systems are deployed, they need to work correctly under various circumstances. Almost more than anything else related to AI and ML systems, making software work is an exact science. Best practices for software testing are well-known and can even be made automatic in many cases. At a minimum, mission-critical AI systems should undergo:

- **Unit testing**: All functions, method, subroutines or other code blocks should have tests associated with them to ensure they behave as expected, accurately, and are reproducible. This ensures the building blocks of an AI system are solid.
- **Integration testing**: All APIs and interfaces between modules, tiers, or other subsystems should be tested to ensure proper communication. API mismatches after backend code changes are a classic failure mode for AI systems. Use integration testing to catch this and other integration fails.
- **Functional Testing**: Functional testing should be applied to AI system user interfaces and endpoints to ensure that they behave as expected once deployed.
- **Chaos Testing**: Testing under chaotic and adversarial conditions can lead to better outcomes when your AI systems faces complex and surprising *in vivo* scenarios.

Two additional ML-specific tests should be added into the mix to increase quality further:

- **Random Attack**: Random attacks expose AI systems or ML models to vast amounts of random data to catch both software and math problems. The real world is a chaotic place. Your AI system will encounter data for which it's not prepared. Random attacks can decrease those occurrences and any associated glitches or incidents.
- **Benchmarking**: Use benchmarks to track system improvements over time. AI systems can be incredibly complex. How can you know if the 3 lines of code an engineer changes today made a difference in the performance of the system as a whole? If system performance is reproducible, and benchmarked before and after changes, it's much easier to answer such questions.

ML is software. So, all the testing that's done on traditional enterprise software assets should be done on important AI systems as well. **If you don't know where to start with modeling debugging, start with random attack.** You may very well be shocked at the math or software bugs random data can expose in your systems. When you add benchmarks to your organization's continuous integration/continuous development (CI/CD) pipelines, that's the another big step toward assuring the safety and performance of AI systems.

Traditional Model Assessment

Once you feel confident that the code in your AI systems is functioning as expected, it's easier to concentrate on testing the math of your ML algorithms. Looking at standard performance metrics is important. But it's not the end of the validation and debugging process — it's the beginning. While exact values and decimal points matter, from a safety and performance standpoint, they matter much less than they do on the leaderboard of an ML contest. When considering in-domain performance, it's less about exact numeric values of assessment statistics, and more about mapping *in silica* performance to *in vivo* performance.

If possible, try to select assessment statistics that have a logical interpretation and practical or statistical thresholds. For instance, root mean squared error (RMSE) can be calculated for many types of prediction problems, and crucially, it can be interpreted in units of the target variable. Area under the curve (AUC), for classification tasks, is bounded between 0.5 at the low end and 1.0 at the high end. Such assessment measures allow for commonsense interpretation of ML model performance and for comparison to widely accepted thresholds for determining quality. It's also important to analyze performance metrics across important segments in your data and across training, validation, and testing data partitions. When comparing performance across segments within training data, it's important that all those segments exhibit roughly equivalent and high quality performance. Amazing performance on one large customer segment, and poor performance on everyone else, will look fine in average assessment statistic values like RMSE. But, it won't look fine if it leads to public brand damage due to many unhappy customers. Varying performance across segments can also be a sign of underspecification, a serious ML bug that will be addressed in more detail later in Chapter 3. Performance across training, validation and test datasets are usually analyzed for under and overfitting. Like underspecification, these are serious bugs that will receive more detailed treatment below.

Another practical consideration related to traditional model assessment is selecting a probability cutoff threshold. Most ML models for classification generate numeric probabilities, not discrete decisions. Selecting the numeric probability cutoff to associate with actual decisions can be done in various ways. While it's always tempting to maximize some sophisticated assessment measure, it's also a good idea to consider real-world impact. Let's consider a classic lending example. Say a probability of default model threshold is originally set at 0.15, meaning that everyone who scores less than a 0.15 probability of default is approved for a loan, and those that score at the threshold or over are denied. Think through questions like:

- What is the expected monetary return for this threshold?
- What is the risk?
- How many people will get the loan at this threshold?
- How many women? How many minority group members?

Outside of the probability cutoff thresholds, it's always a good idea to estimate in-domain performance, because that's what we really care about. Assessment measures are nice, but what matters is making money versus losing money, or even saving lives versus taking lives. You can take a first crack at understanding real-world value by assigning monetary, or other, values to each cell of a confusion matrix for classification problems or to each residual unit for regression problems. Do a back-of-the-napkin calculation. Does it look like your model will make money or lose money? Once you get the gist of this kind of valuation, you can

even incorporate value levels for different model outcomes directly into ML loss functions, and optimize towards the best-suited model for real-world deployment.

Error and accuracy metrics will always be important for ML. But once ML algorithms are used in deployed AI systems, numeric values and comparisons matter less than they do for publishing papers and data science competitions. So, keep using traditional assessment measures, but try to map them to in-domain safety and performance.

Residual Analysis for Machine Learning

Residual analysis is another type of traditional model assessment that can be highly effective for ML models and AI systems. At it's most basic level, residual analysis means learning from mistakes. That's an important thing to do in life, as well as in organizational AI systems. Moreover, residual analysis is a tried and true model diagnostic technique. This subsection will use an example and three generally applicable residual analysis techniques to apply this established discipline to ML.

Example Setup

Since residual analysis is a fairly technical topic, we'll use an example problem and some related figures for clarity's sake. The figures below are based on the well-known Taiwanese credit card dataset from the University of California Irvine (UCI) ML repository. In this dataset, the object is to predict if someone will pay, `DEFAULT_NEXT_MONTH = 0`, or default, `DEFAULT_NEXT_MONTH = 1`, on their next credit card payment. Features about payments are used to generate probabilities of default, or `p_DEFAULT_NEXT_MONTH`. The example ML model is trained on payment features including `PAY_0 – PAY_6`, or a customer's most recent through six months prior repayment statuses (higher values are late payments), `PAY_AMT1 – PAY_AMT6`, or a customer's most recent through six months prior payment amounts, and `BILL_AMT1 – BILL_AMT6`, or a customer's most recent through six months prior bill amounts. All monetary values are reported in Taiwanese Dollars (NT$).

Some of the figures also contain the features `LIMIT_BAL` and `r_DEFAULT_NEXT_MONTH`. `LIMIT_BAL` is a customer's credit limit. `r_DEFAULT_NEXT_MONTH` is a logloss residual value, or a numeric measure of how far off the prediction, `p_DEFAULT_NEXT_MONTH`, is from the known correct answer, `DEFAULT_NEXT_MONTH`. You may also see demographic features in the dataset, like `SEX`, that are used for bias testing. For the most part, Chapter 3 treats the example credit lending problem as a general predictive modeling exercise, and does not consider applicable regulations.

Analysis and Visualizations of Residuals

Plotting residuals and examining them for tell-tale patterns of different kinds of problems is a long-running model diagnostic technique. And it can be applied to ML algorithms to great benefit with a bit of creativity and elbow grease. Simply plotting residuals for an entire dataset can be helpful, especially to spot outlying rows causing very large numeric errors or to analyze overall trends in errors. However, breaking residual values and plots down by feature and level, as illustrated in Figure 1, is likely to be more informative. In the top row of Figure 1, favorable values for `PAY_0`, `-2,-1,0` representing paying on time or not using credit, are associated with large residuals for customers who default. In the bottom rows the exact

opposite, but still obvious, behavior is displayed. Customer's with unfavorable values for `PAY_0` cause large residuals when they suddenly pay on time. What's the lesson here? Figure 1 shows that the ML model in question would make the same mistakes a human, or a simple business rule, would make. Now that we know about these simplistic decision processes, which happen to arise from thousands of machine-learned rules, the model can be improved, or just replaced with a more transparent and secure business rule: `IF PAY_0 < 2 THEN APPROVE, ELSE DENY.`

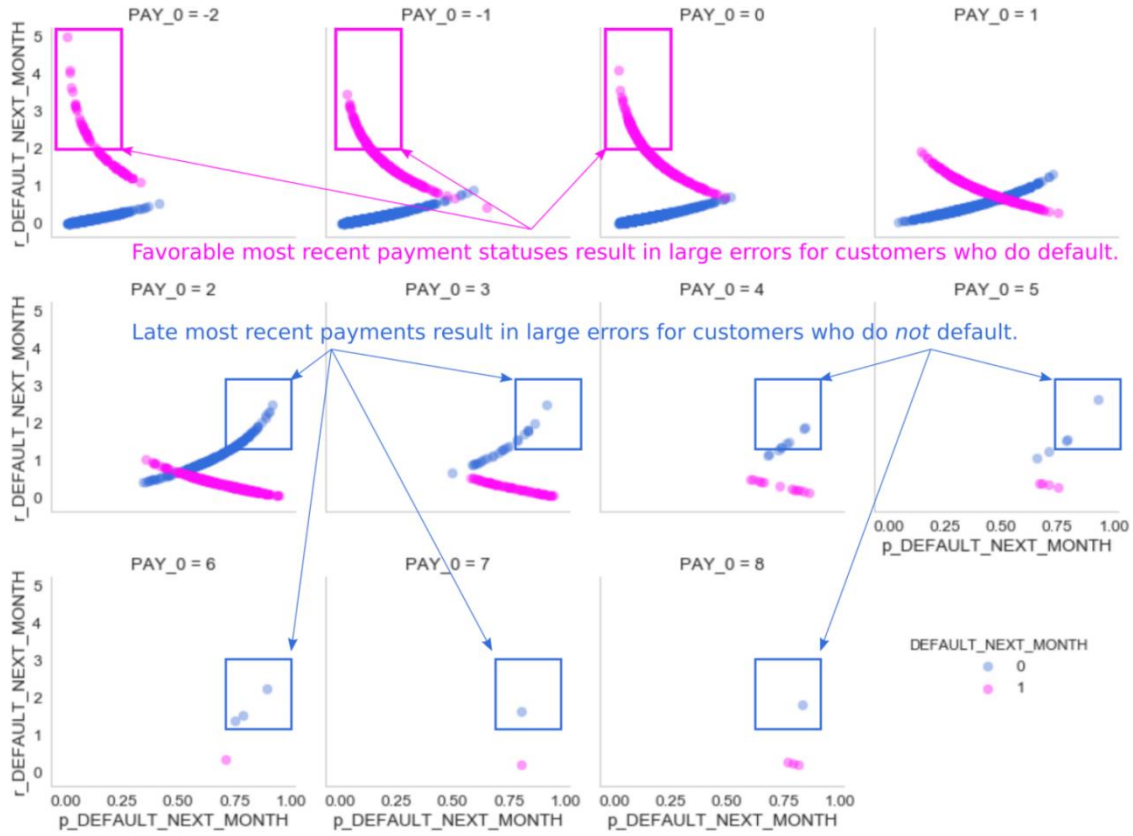


Figure 2-1. Residuals show that a model makes the same obvious mistakes a human might. Customers with good payment track records who default suddenly cause large residuals, as do customers with poor payment track records who suddenly start paying on time. Adapted from Responsible Machine Learning with Python.

Do you have a lot features or features with many categorical levels? You're not off the hook! Start with the most important features and their most common levels. Residual analysis is considered standard practice for important linear regression models. ML models are arguably higher-risk and more failure prone, so they need even more residual analysis and model debugging.

Modeling Residuals

Modeling residuals with interpretable models is another great way to learn more about the mistakes your AI system could make. Figure 2 displays a decision tree model of the example ML model's residuals for `DEFAULT_NEXT_MONTH = 1`, or customer's who default. While it reflects what was discovered in Figure 1, it does so in a very direct way that exposes the logic of the failures. In fact, it's even possible to build programmatic rules about when the model is

likely to fail the worst. The worst residuals occur when: `PAY_0 < 0.5 AND PAY_AMT2 >= 2802.5 AND PAY_4 < 1 AND LIMIT_BAL >= 256602.0` Or more plainly, this model fails when a customer exhibits excellent payment behavior, but then suddenly defaults.

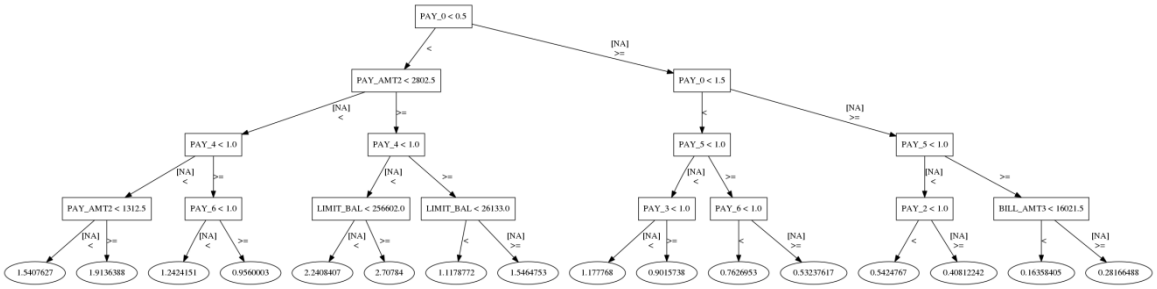


Figure 2-2. An interpretable decision tree model of the example ML model's residuals displays patterns that can be used to spot failure modes and design mitigation approaches. Adapted from Responsible Machine Learning with Python.

In general this technique helps uncover failure modes. Once failure modes are known, they can be mitigated to increase performance and safety. For the example, if the patterns in Figure 4 can be isolated from patterns that result in non-default, this can lead to precise remediation strategies in the form of model assertions. Model assertions, also known as business rules, are a mitigation technique that can be applied directly to ML model predictions to address such discovered failure modes. Model assertions will be discussed in greater detail below, but don't be shy about applying other common sense actions to increase safety and performance.

Local Contribution to Residuals

Plotting and modeling residuals are older techniques that are well-known to skilled practitioners. A more recent breakthrough has made it possible to calculate accurate Shapley value contributions to model errors! This means for any feature or row of any dataset, we can now know which features are driving model predictions, and which features are driving model errors. What this advance really means for ML is yet to be determined, but the possibilities are certainly intriguing. One obvious application for this new Shapley value technique is to compare feature importance for predictions to feature importance for residuals, as illustrated in Figure 3.

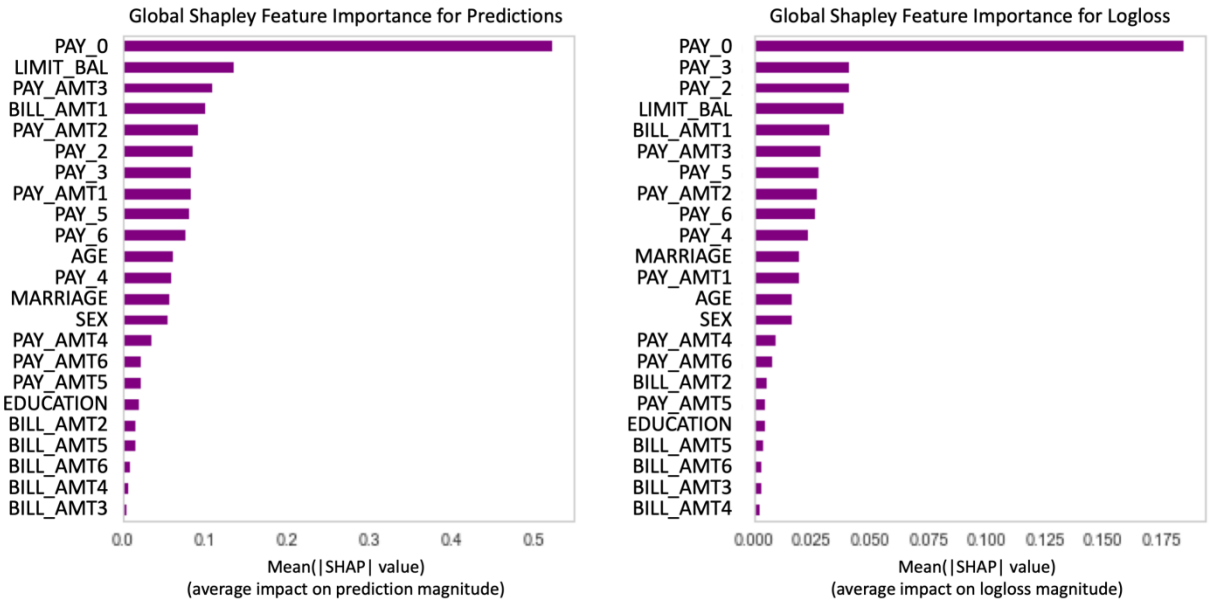


Figure 2-3. It is now possible to calculate Shapley value contributions to model errors! This means we can know which features are driving predictions, which features are driving errors, and consider mitigation approaches for non-robust features that contribute more to errors than to predictions. Adapted from Responsible Machine Learning with Python.

Figure 3 shows accurate Shapley value feature importance on the left, and accurate Shapley values contributions to residuals on the right, both aggregated over an entire dataset. This reveals that non-robust features like `PAY_2` and `PAY_3` are actually more important to model residuals than model predictions! Meaning these features should be examined carefully, have strong regularization or noise injection applied to them, or even be dropped from the analysis.

This ends our brief tour of residual analysis for ML. Of course there are other ways to study the errors of ML models. If you prefer another way, then go for it! The important thing is do some kind of residual analysis for all high-stakes AI systems. Along with sensitivity analysis, to be discussed in the next subsection, residual analysis is a major component of the toolkit for ML model debugging.

Sensitivity Analysis

Unlike linear models, it's very hard to understand how ML models extrapolate or perform on new data, without testing them explicitly. That's the simple and powerful idea behind sensitivity analysis. Find or simulate data for interesting scenarios, then see how your model performs on that data. You really won't know how your AI system will perform in these scenarios unless you do basic sensitivity analysis. Of course, there are structured and more efficient variants of sensitivity analysis, such as in the interpret library from Microsoft Research. Another great option for sensitivity analysis, and a good place to start with more advanced model debugging techniques is random attacks, discussed in the Software Testing subsection. Many other approaches, like stress-testing, visualization, and adversarial example searches also provide standardized ways to conduct sensitivity analysis.

- **Stress-Testing**: Stress-testing involves simulating data that represents realistic adverse scenarios, like recessions or pandemics, and making sure your ML models

and any downstream business processes will hold up to the stress of the adverse situation.

- **Visualizations**: Visualizations like plots of accumulated local affect (ALE), individual conditional local effect (ICE), and partial dependence curves are a well-known and highly structured way to observe the performance of ML algorithms across various real or simulated values of input features.
- **Adversarial Example Searches**: Adversarial examples are rows of data that evoke surprising responses from ML models. Deep learning approaches can be used to generate adversarial examples for unstructured data, and ICE and genetic algorithms can be used to generate adversarial examples for structured data. Adversarial examples, and searching for them, are a great way to find local areas in your ML response functions or decision boundaries that can cause incidents once deployed.

Figure 4 shows the results of combining visualizations and adversarial example searches in model debugging. The response surfaces in Figure 4 were formed by randomly perturbing a certain interesting row of data many thousands of times and plotting the associated predictions. How was this interesting row of data selected for the basis of an adversarial example search? ICE. The surfaces are based on an ICE curve that showed a particularly drastic swing for different values of input variables.

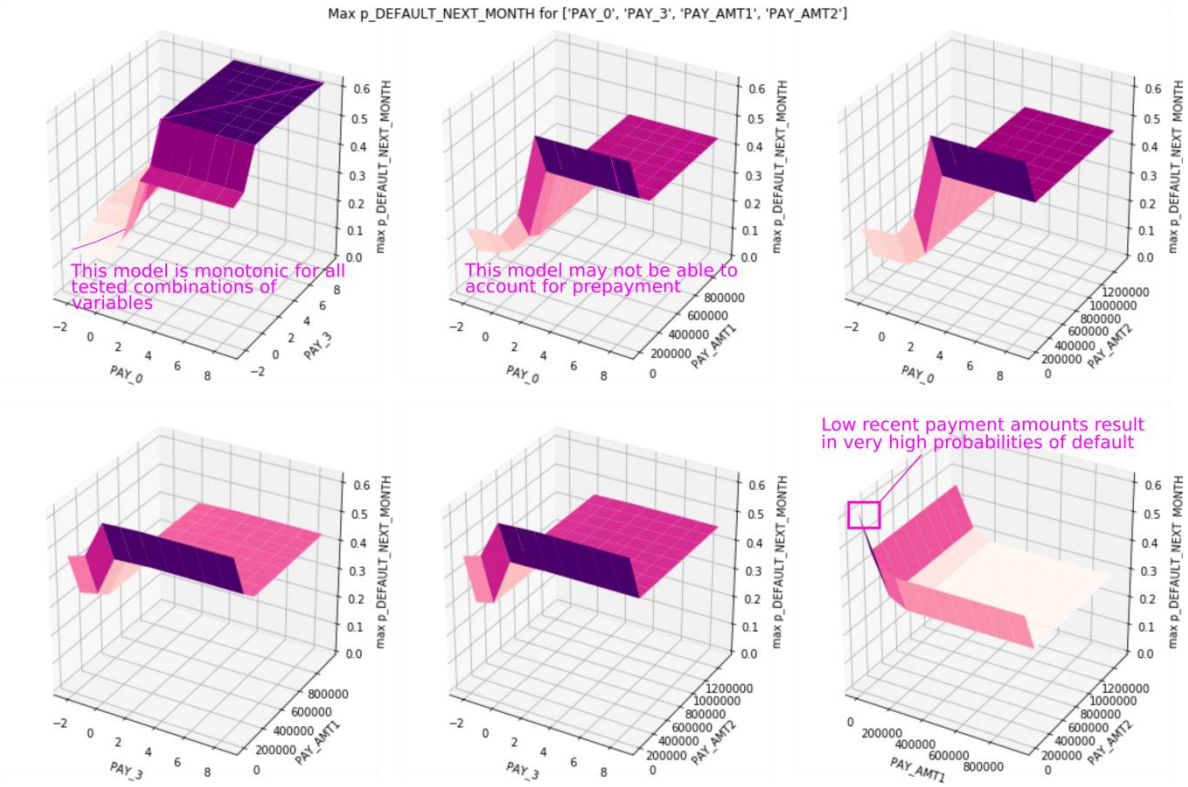


Figure 2-4. Adversarial example search shows several interesting positive and negative results.

Getting back to the example data and ML model discussed in previous sections,
Figure 4 shows the results of six adversarial example searches, seeded by an ICE curve, and presents some positive and negative findings. On the positive side, each response surface shows monotonicity. These simulations confirm that monotonic constraints, supplied at training time and based on domain knowledge, held up during training. On the negative side, a potential logical flaw and a vulnerability to adversarial attack were also discovered.

According to one of the response surfaces, the example model will issue high probability of default predictions once customers become two months late on their most recent payment (`PAY_0`). This harsh treatment is applied even in the circumstance where a customer repays (`PAY_AMT1`) over their credit limit. This potential logical flaw could prevent pre-payment or over-penalize good customers who failed pay their bill while on vacation. Another surface also shows a surprisingly steep spike in predictions for low first (`PAY_AMT1`) and second payment (`PAY_AMT2`) values. This could present an avenue for adversarial examples to evoke surprisingly high probability of default predictions once the system is deployed.3, 15>> is anticipating how an AI system will perform in the real-world and making sure it's performance in the lab is relevant to it's *in vivo* performance, so that appropriate controls can be applied before deployment. Sensitivity analysis is one of most direct tools for simulating real-world performance and making sure your AI system is ready for what it may face once deployed.

Benchmark Models

Benchmark models have been discussed at numerous points within 3. They are a very important safety and performance tool, with uses throughout the AI lifecycle. This subsection will discuss benchmark models in the context of model debugging and also summarize other critical uses.

The first way to use a benchmark model for debugging is to compare performance between a benchmark and the AI system in question. If the AI system does not outperform a simple benchmark, and many may not, it's back to the drawing board! Assuming a system passes this initial baseline test, benchmark models are a comparison tool used to interrogate mechanisms and find bugs within an AI system. For instance, data scientists can ask the question: "which predictions does my benchmark get right and my AI system get wrong?" Given that the benchmark should be well understood, it should be clear why it is correct, and this understanding should also provide some clues as to what the AI system is getting wrong. Benchmarks can also be used for reproducibility and model monitoring purposes as follows:

Reproducibility Benchmarks: Before making changes to a complex AI system it is imperative to have a reproducible benchmark from which to measure performance gains, or losses. A reproducible benchmark model is an ideal tool for such measurement tasks. If this model can be built into CI/CD processes that enable automated testing for reproducibility and comparison of new system changes to established benchmarks, even better!

Debugging Benchmarks: As discussed above, comparing complex ML model mechanisms and predictions to a trusted, well-understood benchmark model's mechanisms and predictions is an effective way to spot ML bugs.

Monitoring Benchmarks: Comparing real-time predictions between a trusted benchmark model and a complex AI system is a way to catch serious ML bugs in real-time. If a trusted benchmark model and a complex AI system give noticeably different predictions for the same instance of new data, this can be a sign of an ML hack, data drift, or even algorithmic discrimination. In such cases, benchmark predictions can be issued in place of AI system predictions, or predictions can be withheld until human analysts determine if the AI system prediction is valid.

If you set benchmarks up efficiently, it may even be possible to use the same model for all three tasks. A benchmark can be run before starting work to establish a baseline from which to improve performance, and that same model can be used in comparisons for debugging and model monitoring. When a new version of the system outperforms an older version in a reproducible manner, the ML model at the core of the system can become the new benchmark. If your organization can establish this kind of workflow, you'll be benchmarking and iterating your way to increased AI safety and performance.

Machine Learning Bugs

Now that we know how to find ML bugs with software QA, traditional model assessment, residual analysis, sensitivity analysis and benchmark models, what exactly are we looking for? As discussed elsewhere in 3, there's a lot that can go wrong, and thinking about an AI system's operating environment is key to understanding failure modes. But when it comes to the math of ML, there are a few emergent gotchas and many well-known pratfalls. This subsection will discuss some usual suspects and some dark horse bugs including distributional shifts, instability, looped inputs, overfitting, underfitting, and underspecification.

Distributional Shifts

Shifts in the underlying data between different training data partitions and after model deployment are common failure modes for AI systems. Whether it's a new competitor entering a market or a devastating world-wide pandemic, the world is a dynamic place. Unfortunately most of today's AI systems learn patterns from static snapshots of training data and try to apply those patterns in new data. Sometimes that data is holdout validation or testing partitions. Sometimes it's live data in a production scoring queue. Regardless, drifting distributions of input features is a serious bug that must be caught and squashed. (Before you jump to the conclusion that adaptive, on-line, or reinforcement learning AI systems will solve the problem of distributional drift, I'd like to point out that such systems currently implicate untenable risks for *in vivo* instability and insecurity.) When training ML models, watch out for distributional shifts between training, cross-validation, validation, or test sets using KS-, t-, or other appropriate statistical tests. If a feature has a different distribution from training partition to another, drop it or regularize it heavily. Another smart test for distributional shifts to conduct during debugging is to simulate distributional shifts for potential deployment conditions. Worried about how your model will perform during a recession? Simulated distributional shifts to simulate more late payments, lower cash flow, and higher credit balances and see how your model performs. It's also crucial to record information about distributions in training data so that drift after deployment can be detected easily.

Instability

ML models can exhibit instability in the training process or when making predictions on live data. Instability in training is often related to small training data, sparse regions of training data, highly correlated features within training data, or high-variance model forms, such as deep single decision trees. Cross-validation is a typical tool for detecting instability during training. If a model displays noticeably different error or accuracy properties across cross-validation folds then you have an instability problem. Training instability can often be remediated with better data and lower-variance model forms such as decision tree ensembles. Plots of ALE or ICE also tend to reveal prediction instability in sparse regions of training

data, and instability in predictions can be analyzed using sensitivity analysis: simulations, stress-testing, and adversarial example searches. If probing your response surface or decision boundary with these techniques uncovers wild swings in predictions, or your ALE or ICE curves are bouncing around, especially in high or low ranges of feature values, you also have an instability problem. This type of instability can often be fixed with constraints and regularization.

Looped Inputs

As AI systems are incorporated into broader digitalization efforts, or implemented as part of larger decision support efforts, multiple data-driven systems often interact. In these cases, error propagation and feedback loop bugs can occur. Error propagation occurs when small errors in one system cause or amplify errors in another system. Feedback loops are a way an AI system can fail by being right. Feedback loops occur when an AI system affects its environment and then those effects are re-incorporated into system training data. Examples of feedback loops include when predictive policing leads to over-policing of certain neighborhoods or when employment algorithms intensify diversity problems in hiring by continually recommending correct, but non-diverse, candidates. Dependencies between systems must be documented and deployed models must be monitored so that debugging efforts can detect error propagation or feedback loop bugs.

Leakage

Information leakage between training, validation, and test data partitions happens when information from validation and testing partitions leaks into a training partition, resulting in overly optimistic error and accuracy measurements. Leakage can happen for a variety of reasons, including:

- **Feature Engineering**: If used incorrectly, certain feature engineering techniques such as imputation or principal components analysis (PCA) may contaminate training data with information from validation and test data. To avoid this kind of leakage, perform feature engineering uniformly, but separately, across training data partitions. Or ensure that information, like means and modes used for imputation, are calculated in training data and applied to validation and testing data, and not vice-versa.
- **Mistreatment of Temporal Data**: Don’t use the future to predict the past. Most data has some association with time, whether explicit as in time-series data, or some other implicit relationship. Mistreating or breaking this relationship with random sampling is a common cause of leakage. If you’re dealing with data where time plays a role, time needs to be used in constructing model validation schemes. The most basic rule is that the earliest data should be in training partitions while later data should be divided into validation and test partitions, also according to time.
- **Multiple Identical Entities**: Sometimes the same person, financial or computing transaction, or other modeled entity will be in multiple training data partitions. When this occurs, care should be taken to ensure that ML models do not memorize characteristics of these individuals then apply those individual-specific patterns to different entities in new data.

Keeping an untouched, time-aware holdout set for an honest estimate of real-world performance can help with many of these different leakage bugs. If error or accuracy on such a holdout set looks a lot less rosy than on partitions used in model development, you might

have a leakage problem. More complex modeling schemes involving stacking, gates, or bandits can make leakage much harder to prevent and detect. However, a basic rule of thumb still applies: do not use data involved in learning or model selection to make realistic performance assessments. Using stacking, gates, or bandits means you need more holdout data for the different stages of these complex models to make an accurate guess at *in vivo* quality. More general controls such as careful documentation of data validation schemes and model monitoring in deployment are also necessary for any AI system.

Overfitting

Overfitting happens when a complex ML algorithm memorizes too much specific information from training data, but does not learn enough generalizable concepts to be useful once deployed. Overfitting is often caused by high-variance models, or models that are too complex for the data at hand. Overfitting usually manifests in much better performance on training data than on validation, cross-validation, and test data partitions. Since overfitting is a ubiquitous problem, there are many possible solutions, but most involve decreasing the variance in your chosen model. Examples of these solutions include:

- **Ensemble Models**: Ensemble techniques, particularly bootstrap aggregation (i.e., bagging) and gradient boosting are known to reduce error from single high-variance models. So, try one of these ensembling approaches if you encounter overfitting. Just keep in mind that when switching from one model to many, you can decrease overfitting and instability but you'll also likely loose interpretability.
- **Reducing Architectural Complexity**: Neural networks can have too many hidden layers or hidden units. Ensemble models can have to many base learners. Trees can be too deep. If you think you're observing overfitting, make your model architecture less complex.
- **Regularization**: Regularization refers to many sophisticated mathematical approaches for reducing the strength, complexity, or number of learned rules or parameters in an ML model. In fact, many types of ML models now incorporate multiple options for regularization, so make sure you employ these options to decrease the likelihood of overfitting.
- **Simpler Hypothesis Model Families**: Some ML models will be more complex than others out-of-the-box. If your neural network or gradient boosting machine (GBM) look to be overfit, you can try a less complex decision tree or linear model.

Overfitting is traditionally seen as the Achilles' heal of ML. While it is one the most likely bugs to encounter, it's also just one of many possible technical risks to consider from a safety and performance perspective. As with leakage, as ML systems become more complex, overfitting becomes harder to detect. Always keep an untouched holdout set with which to estimate real-world performance before deployment. More general controls like documentation of validation schemes, model monitoring, and A/B testing of models on live data also need to be applied to prevent overfitting.

Shortcut Learning

Shortcut learning occurs when a complex AI system is thought to be learning and making decisions about one subject, say anomalies in lung scans or job interview performance, but it's actually learned about some simpler related concept, such as machine identification numbers or Zoom video call backgrounds. Use interpretable models and explainable AI

(XAI) techniques to understand what learned mechanisms are driving model decisions, and make sure you understand how your AI system makes decisions.

Underfitting

If someone tells you a statistic about a set of data, you might wonder how much data that statistic is based on, and whether that data was of high enough quality to be trustworthy. What if someone told you they had millions, billions, or even trillions of statistics for you, they would need lots of data to make a case that all these statistics were meaningful. Just like averages and other statistics, each parameter or rule within an ML model is learned from data. Big ML models need lots of data to learn enough to make their millions, billions, or trillions of learned induction mechanisms meaningful. Underfitting happens when a complex ML algorithm doesn't have enough training data, constraints, or other input information, and it learns just a few generalizable concepts from training data, but not enough specifics to be useful when deployed. Underfitting can be diagnosed by markedly better performance in validation, cross-validation, and test data versus training data. It can be mitigated by using simpler models or with additional training data, by constraining ML algorithms, or by providing additional input information, such as a Bayesian prior.

Underspecification

Forty Google researchers recently published *Underspecification Presents Challenges for Credibility in Modern Machine Learning*. This paper gives a name to a problem that has existed for decades, *underspecification*. Underspecification arises from the core ML concept of the multiplicity of good models, sometimes also called the Rashomon effect. For any given data set, there are many accurate ML models. How many? Vastly more than human operators have any chance of understanding in most cases. While we use validation data to select a good model from many models attempted during training, validation-based model selection is not a strong enough control to ensure we picked the best model - or even a servicable model - for deployment. Say that for some dataset there are a million total good ML models based on training data and the large number of potential hypothesis models. Selecting by validation data may cut that number of models down to a pool of one hundred total models. Even in this simple scenario, we'd still only have a 1 in 100 chance of picking the right model for deployment. How can we increase those odds? By injecting domain knowledge into ML models. By combining validation-based model selection with domain-informed constraints, we have a much better chance at selecting a viable model for the job at hand.

Happily, testing for underspecification can be fairly straightforward. One major symptom of underspecification is model performance that's dependent on computational hyperparameters that are not related to the structure of the domain, data, or model. If your model's performance varies due to random seeds, number of threads or GPUs, or other computational settings, your model is probably underspecified. Another test for underspecification is illustrated in Figure 5.

Error Metrics for PAY_0

PAY_0	Prevalence	Accuracy	True Positive Rate	Precision	Specificity	Negative Predicted Value	False Positive Rate	False Discovery Rate	False Negative Rate	False Omissions Rate
-2	0.124	0.864	0.099	0.333	0.972	0.884	0.028	0.667	0.901	0.116
-1	0.168	0.816	0.206	0.406	0.939	0.854	0.061	0.594	0.794	0.146
0	0.121	0.867	0.107	0.341	0.972	0.888	0.028	0.659	0.893	0.112
1	0.325	0.491	0.903	0.381	0.292	0.862	0.708	0.619	0.097	0.138
2	0.709	0.709	1	0.709	0	0.5	1	0.291	0	0.5
3	0.748	0.748	1	0.748	0	0.5	1	0.252	0	0.5
4	0.571	0.571	1	0.571	0	0.5	1	0.429	0	0.5
5	0.444	0.444	1	0.444	0	0.5	1	0.556	0	0.5
6	0.25	0.25	1	0.25	0	0.5	1	0.75	0	0.5
7	0.5	0.5	1	0.5	0	0.5	1	0.5	0	0.5
8	0.75	0.75	1	0.75	0	0.5	1	0.25	0	0.5

Error Metrics for SEX

SEX	Prevalence	Accuracy	True Positive Rate	Precision	Specificity	Negative Predicted Value	False Positive Rate	False Discovery Rate	False Negative Rate	False Omissions Rate
Male	0.235	0.782	0.626	0.531	0.83	0.879	0.17	0.469	0.374	0.121
Female	0.209	0.797	0.552	0.514	0.862	0.879	0.138	0.486	0.448	0.121

Figure 2-5. Anayzing accuracy and errors across key segments is an important debugging method for detecting bias, underspecification, and other serious ML bugs.

Figure 5 displays several error and accuracy measures across important segments in the example training data and model. Here a noticeable shift in performance for segments defined by higher values of the important feature `PAY_0` points to a potential underspecification problem, likely due to data sparsity in that region of the training data. (Performance across segments defined by `SEX` is more equally balanced, which a good sign from a bias testing perspective, but certainly not the only test to be considered for bias problems.) Fixing underspecification tends to involve applying real-world knowledge to ML algorithms. Such domain-informed mechanisms include graph connections, monotonicity constraints, interaction constraints, beta constraints, or other architectural constraints.

Each of the ML bugs discussed in this subsection has real-world safety and performance ramifications. A unifying theme across these bugs is they cause systems to perform differently than expected when deployed and over time. Unexpected performance leads to unexpected failures and AI incidents. Using knowledge of potential bugs and bug detection methods discussed here to ensure estimates of validation and test performance are relevant to deployed performance will go a long way toward preventing real-world incidents.

Remediation: Fixing Bugs

The last step in debugging is fixing bugs. The previous subsections have outlined testing strategies, bugs to be on the lookout for, and a few specific fixes. This subsection outlines general ML bug-fixing approaches and discusses how they might be applied in the example debugging scenario. General strategies to consider during ML model debugging include:

- **Anomaly Detection**: Strange inputs and outputs are usually bad news for ML systems. These can be evidence of real-time security, discrimination, and safety and

performance problem. Monitor ML system data queues and predictions for anomalies, record the occurrence of anomalies, and alert stakeholders to their presence when necessary.

- **Experimental Design and Data Augmentation**: Collecting better data is a often a fix-all for ML bugs. What's more, data collection doesn't have to be done in a trial-and-error fashion, nor do data scientists have to rely on data exhaust by-products of other organizational processes for selecting training data. The mature science of design of experiment (DOE) has been used by data practitioners for decades to ensure they collect the right kind and amount of data for model training. Arrogance related to the perceived omnipotence of "big" data and overly compressed deployment time lines are the most common reasons data scientists don't practice DOE. Unfortunately, these are not scientific reasons to ignore DOE.
- **Model Assertions**: Model assertions are business rules applied to ML model predictions that correct for shortcomings in learned ML model logic. Using business rules to improve predictive models is a time-honored remediation technique that will likely be with us for decades to come. If there is a simple, logical rule that can be applied to correct a foreseeable ML model failure, don't be shy about implementing it. The best practitioners and organizations in the predictive analytics space have used this trick for decades.
- **Model Editing**: Given that ML models are software, that software artifact can be edited to correct for any discovered bugs. Certain models, like GA2Ms or explainable boosting machines (EBMs) are designed to be edited for the purposes of model debugging. Other types of models may require more creativity to edit. Either way, editing must be justified by domain considerations, as it's likely to make performance on training data appear worse. For better or worse, ML models optimize toward lower error. If you edit this highly-optimized structure to make in-domain performance better, you'll likely worsen traditional assessment statistics.
- **Model Management and Monitoring**: ML models and the AI systems that house them are dynamic entities that must be monitored to the extent that resources allow. All mission-critical ML systems should be well-documented, inventoried, and monitored for security, discrimination, and safety and performance problems in real-time. When something starts to go wrong, stake-holders need to be alerted quickly. The next major section of Chapter 3 gives more detailed treatment of model monitoring.
- **Monotonicity and Interaction Constraints**: Many ML bugs occur because ML models have too much flexibility. Constraining that flexibility with real-world knowledge is a general solution to several types of ML bugs. Monotonicity and interaction constraints, in popular tools like XGBoost, can help ML practitioners enforce logical domain assumptions in complex ML models.
- **Noise Injection and Strong Regularization**: Many ML algorithms come with options for regularization. However, if an ML model is over-emphasizing a certain feature, stronger or external regularization might need to be applied. L_0 regularization can be used to limit the number of rules or parameters in a model directly, and manual noise injection can be used to corrupt signal from certain features to de-emphasize any undue importance in ML models.

There's more detailed information regarding model debugging and the example data and model in the Resources Subsection at the end of Chapter 3. For now, we've learned quite a bit about model debugging and it's time to turn our attention to safety and performance for deployed AI systems.

Deployment

Once bugs are found and fixed, it's time to deploy your AI system to make real-world decisions. AI systems are much more dynamic than most traditional software systems. Even if system operators don't change any code or setting of the system, the results can still change. Once deployed, AI systems must be checked for in-domain safety and performance, they must be monitored, and their operators must be able to shut them off quickly. This last major subsection of Chapter 3 will cover how to enhance safety and performance once an AI system is deployed: domain safety, model monitoring, and kill switches.

Domain Safety

Domain safety means safety in the real world. This is very different than standard model assessment, or even enhanced model debugging. How can practitioners work toward real-world safety goals? A/B testing and champion challenger methodologies allow for some amount of testing in real-time operating environments. Process controls, like enumerating foreseeable incidents, implementing controls to address those potential incidents, and testing those controls under realistic or stressful conditions are also important for solid *in vivo* performance. To make up for incidents that can't be predicted, apply chaos testing, random attacks, and manual prediction limits to your AI system outputs.

- **Foreseeable real-world incidents**: A/B testing and champion-challenger approaches, in which models are tested against one another on live data streams or under other realistic conditions are a first step toward robust in-domain testing. Beyond these somewhat standard practices, resources should be spent on thinking through possible incidents. For example common failure modes in credit lending include algorithmic discrimination, lack of transparency, and poor performance during recessions. For other applications, say autonomous vehicles, there are numerous ways they could accidentally or intentionally cause harm. Once potential incidents are recorded, then safety controls can be adopted for the most likely or most serious potential incidents. In credit lending, models are tested for discrimination, explanations are provided to consumers via adverse action notices, and models are monitored to catch performance degradation quickly. In autonomous vehicles, we still have a lot to learn (as the Chapter 1 case showed). Regardless of the application, safety controls must be tested, and these tests should be realistic and performed in collaboration with domain experts. When it comes to human safety, simulations run by data scientists are not enough. Safety controls need to be tested and hardened *in vivo* and in coordination with people who have a deep understanding of safety in the application domain.
- **Unforeseeable real-world incidents**: Interactions between AI systems and their environments can be complex and surprising. For high-stakes AI systems, it's best to admit that unforeseeable incidents can occur. We can try to catch some of these potential surprises before they occur with chaos testing and random attacks. Important AI systems should be tested in strange and chaotic use cases and exposed to large amounts of random input data. While these are time- and resource-consuming tests, they are one of the few tools available to test for so-called "unknown unknowns." Given that no testing regime can catch every problem, it's also ideal to apply common sense prediction limits to systems. For instance, large loans or interest rates should not be issued without some kind of human oversight. Nor should autonomous vehicles be allowed to travel at very high speeds without human intervention. Some actions

simply should not be performed purely automatically as of today and prediction limits are one way to implement that kind of control.

Another key aspect of domain safety is knowing if problems are occurring. Sometimes glitches can be caught before they grow into harmful incidents. To catch problems quickly, AI systems must be monitored. If incidents are detected, incident response plans or kill-switches may need to be activated.

Model Monitoring

It's been mentioned numerous times in Chapter 3, but important AI systems must be monitored once deployed. This subsection focuses on the technical aspects of model monitoring. It outlines the basics of model decay and concept drift bugs, how to detect and address drift, the importance of measuring multiple key performance indicators (KPIs) in monitoring, and it briefly highlights a few other notable model monitoring concepts.

Model Decay and Concept Drift

No matter what you call it, the data coming into an AI system is likely to drift away from the data on which the system was trained. The change in the distribution of input values over time is sometimes labeled "data drift." The statistical properties of what you're trying to predict can also drift, and sometimes this is known specifically as "concept drift." The COVID-19 crisis is likely one of history's best examples of these phenomena. At the height of the pandemic, there was likely a very strong drift toward more cautious consumer behavior accompanied by an overall change in late payment and credit default distributions. These kinds of shifts are painful to live through, and they can wreak havoc an AI system's accuracy.

Detecting and Addressing Drift

The best approach to detect drift is to monitor the statistical properties of live data — both input variables and predictions. Once a mechanism has been put in place to monitor statistical properties, you can set alerts or alarms to notify stakeholders when there is a significant drift. Testing inputs is usually the easiest way to start detecting drift. This is because sometimes true data labels, i.e., true outcome values associated with AI system predictions, cannot be known for long periods of time. In contrast, input data values are available immediately whenever an AI system must generate a prediction or output. So, if current input data properties have changed from the training data properties, you likely have a problem on your hands. Watching AI system outputs for drift can be difficult due to the information needed to compare current and training quality might not be available immediately. (Think about mortgage default versus online advertising — default doesn't happen at the same pace as clicking on an online advertisement.) The basic idea for monitoring predictions is to watch predictions in real-time and look for drift and anomalies, potentially using methodologies such as statistical tests, control limits, and rules or ML algorithms to catch outliers. And when known outcomes become available, test for degradation in model performance and fairness quickly and frequently.

There are known strategies to address inevitable drift and model decay. These include:

- Refreshing an AI system with extended training data containing some amount of new data.

- Refreshing or retraining AI systems frequently.
- Refreshing or retraining an AI system when drift is detected.

It should be noted that any type of retraining of ML models in production should be subject to the risk mitigation techniques discussed in Chapter 3 and elsewhere in this book — just like they should be applied to the initial training of an AI system.

Monitoring Multiple Key Performance Indicators

Most discussions of model monitoring focus on model accuracy as the primary key performance indicator (KPI). Yet, discrimination, security vulnerabilities, and privacy harms can, and likely should, be monitored as well. The same discrimination testing that was done at training time can be applied when new known outcomes become available. Numerous other strategies, discussed elsewhere in the book, can be used to detect malicious activities that could compromise system security or privacy. Perhaps the most crucial KPI to measure, if at all possible, is the actual impact of the ML system. Whether it's saving or generating money, or saving lives, measuring the intended outcome and actual value of the ML system can lead to critical organizational insights. Assign monetary or other values to confusion matrix cells in classifications problems, and to residual units in regression problems, as a first step toward estimating actual business value.

Out-of-range Values

Training data can never cover all of the data an AI system might encounter once deployed. Most ML algorithms and prediction functions do not handle out-of-range data well, and may simply issue an average prediction or crash, and do so without notifying application software or system operators. AI system operators should make specific arrangements to handle data, such as large magnitude numeric values, rare categorical values, or missing values that were not encountered during training so that AI systems will operate normally when they encounter out-of-range data.

Anomaly Detection and Benchmark Models

Anomaly detection and benchmark models round out the technical discussion of model monitoring in this subsection. These topics have been treated elsewhere in Chapter 3, and are touched on briefly here.

- **Anomaly Detection**: Strange input or output values in an AI system can be indicative of stability problems or security and privacy vulnerabilities. It's possible to use statistics, ML, and business rules to monitor anomalous behavior in both inputs and outputs, and across an entire AI systems. Record any such detected anomaly, report them to stakeholders, and be ready to take more drastic action when necessary.
- **Benchmark Models**: Comparing simpler benchmark models and AI system predictions as part of model monitoring can help to catch stability, fairness, or security anomalies in near real-time. A benchmark model should be more stable, easier to confirm as minimally discriminatory, and should be harder to hack. Use a highly transparent benchmark model and your more complex AI system together when scoring new data, then compare your AI system predictions against the trusted benchmark prediction in real-time. If the difference between the AI system and the

benchmark is above some reasonable threshold, then fall back to issuing the benchmark model's prediction or send the row of data for more review.

Whether it's out-of-range values in new data, disappointing KPIs, drift, or anomalies — these real-time problems are where rubber meets road for AI incidents. If you're monitoring detects these issues, a natural inclination will be to turn the system off, and the next subsection addresses kill switches for AI systems.

Kill Switches

Kill switches are rarely single switches or scripts, but a set of business and technical processes bundled together that serve to turn an AI system off — to the degree that's possible. There's a lot to consider before flipping a proverbial kill switch. AI system outputs often feed into downstream business processes, sometimes including other AI systems. These systems and business processes can be mission critical, for example, an AI system used for credit underwriting or e-retail payment verification. To turn off an AI system, you'll not only need the right technical know-how and personnel available, but you also need an understanding of the system's place inside of broader organizational processes. During an ongoing AI incident is a bad time to start thinking about turning off a fatally flawed AI system. So, kill processes and kill switches are a great addition to your ML system documentation and AI incident response plans (see Chapter 1). This way, when the time comes to kill an AI system, your organization can be ready to make a quick and informed decision. Hopefully you'll never be in a position where flipping an AI system kill switch is necessary, but unfortunately AI incidents have grown more common in recent years. When technical remediation methods are applied along side cultural competencies and business processes for risk mitigation, safety and performance of AI systems is enhanced. When these controls are not applied, bad things can happen.

Case Study: Remediating the Strawman

The Chapter 3 case study takes a deeper look at how to fix the strawman GBM model discussed in the technical examples. Recall that we found this model to pathologically over-emphasize a customer's most recent repayment status (`PAY_0`), we found that some seemingly important input variables were more important to the loss than to the predictions, we found vulnerable spikes and potential logical errors in the response surface, and we saw poor performance, if not underspecification, for `PAY_0` values greater than 1. All of these issues conspired to make a seemingly passable ML model less appealing than a simple business rule. Because many ML models are not adequately debugged before deployment, it's likely that you could find yourself with similar bugs to handle if you applied the debugging techniques in Chapter 3 to one of your organization's models. For the example data and model, several techniques could be applied to remediate the highlighted bugs. Let's try to address them one by one as an example of how you could take on these bugs at your job.

- **Logical errors**: For the logical errors that cause high probability of default to be issued, even after very large payments are made, model assertions or business rules are a likely solution. For customer's who just recently became two month delinquent, use a model assertion or business rule to check if a large payment was also made recently before posting the adverse default prediction. A residual model like the one

in Figure 2, focused on that small group of customers could help suggest or refine more targeted assertions or rules.

- **Over-emphasis of `PAY_0`**: Perhaps the biggest problem with the strawman model, and many other ML models, is bad training data. In this case, training data should be augmented with new, relevant features to spread the primary decision-making mechanisms within the model across more than one feature. Noise injection to corrupt `PAY_0` could also be used to mitigate over-emphasis, but only if there are other accurate signals available in better training data. Look into design of experiment as well. Collecting appropriate training data is seen as a mature science in many other data-driven disciplines that weren't as accepting of by big data hype.
- **Non-robust input variables**: For `PAY_2` and `PAY_3`, the seemingly important input features that ended up being more important to logloss residuals than to model predictions, several different experiments could be performed. Assuming better training data doesn't mitigate these problems, an A/B or champion-challenger platform would need to be established, so that test results can be measured *in vivo*, instead of just *in silica*. Once you can test model changes on realistic data, the first thing to test would be simply retraining the model without `PAY_2` and `PAY_3`. Other interesting experiments could include the application of strong regularization or noise injection to test whether a model would emphasize these features in the same non-robust ways.
- **Security vulnerabilities**: In general, best practices like API throttling and authentication, coordinated with real-time model monitoring, help a lot with ML security. In the case of the example model, the problems is particularly thorny. In many circumstances, the model should give higher probabilities of default for customers who make small payments. The issue here is the dramatic spike in output probabilities for low `PAY_1` and `PAY_2` values. Regularization or robust ML techniques could be applied during training to smooth out swings in the response function, and this region of the response function could be monitored more carefully with an eye toward adversarial manipulation. For instance, if a row of data appears in the live scoring queue with low values for `PAY_1` and `PAY_2` and random values for other features, this would certainly be cause for human review and delaying predictions.
- **Poor performance for `PAY_0 > 1`**: Like many of the other problems with the strawman, this model needs better data to learn more about customers who end up defaulting. In the absence this information, observation weights or over-sampling could be used increase the influence of the small number of customers who did default. However, in the `PAY_0 > 1` region, the model predictions may simply be unusable. The model's monotonicity constraints, discussed briefly in previous sections, are one of the best mitigants to try when faced with sparse training data. The monotonicity constraints enforce well-understood real-world controls on the model. Yet, model performance for `PAY_0 > 1` is extremely poor even with these constraints. Predictions in this range may have to handled by a more specialized model, a rule-based system, or even human case workers.

Like AI in general, model debugging is likely still in its infancy. Where it goes from here is largely up to us. Whenever AI is adopted for business- or life-critical decisions, I personally hope to see a lot more responsible AI and model debugging. One final note to address before closing Chapter 3 is that in-depth analysis of discrimination testing and remediation is left for other chapters of the book. This is an incredibly important topic, a very common failure mode for AI systems, and one deserving fulsome coverage and analysis on its own.

Resources

Further reading and code examples related to model debugging examples:

- Further reading: *Real-World Strategies for Model Debugging*
- Code examples:
  - *https://nbviewer.jupyter.org/github/jphall663/interpretable_machine_learning_with_python/blob/master/resid_sens_analysis.ipynb*
  - *https://nbviewer.jupyter.org/github/jphall663/interpretable_machine_learning_with_python/blob/master/debugging_sens_analysis_redux.ipynb*
  - *https://nbviewer.jupyter.org/github/jphall663/interpretable_machine_learning_with_python/blob/master/debugging_resid_analysis_redux.ipynb*
- Talks, workshops, and educational materials related to model debugging and ML safety and performance:
  - Debugging Machine Learning Models
  - Safe and Reliable Machine Learning
  - Testing and Debugging in Machine Learning.
- Toolkits for model debugging:
  - checklist
  - cleverhans
  - manifold
  - What-If Tool