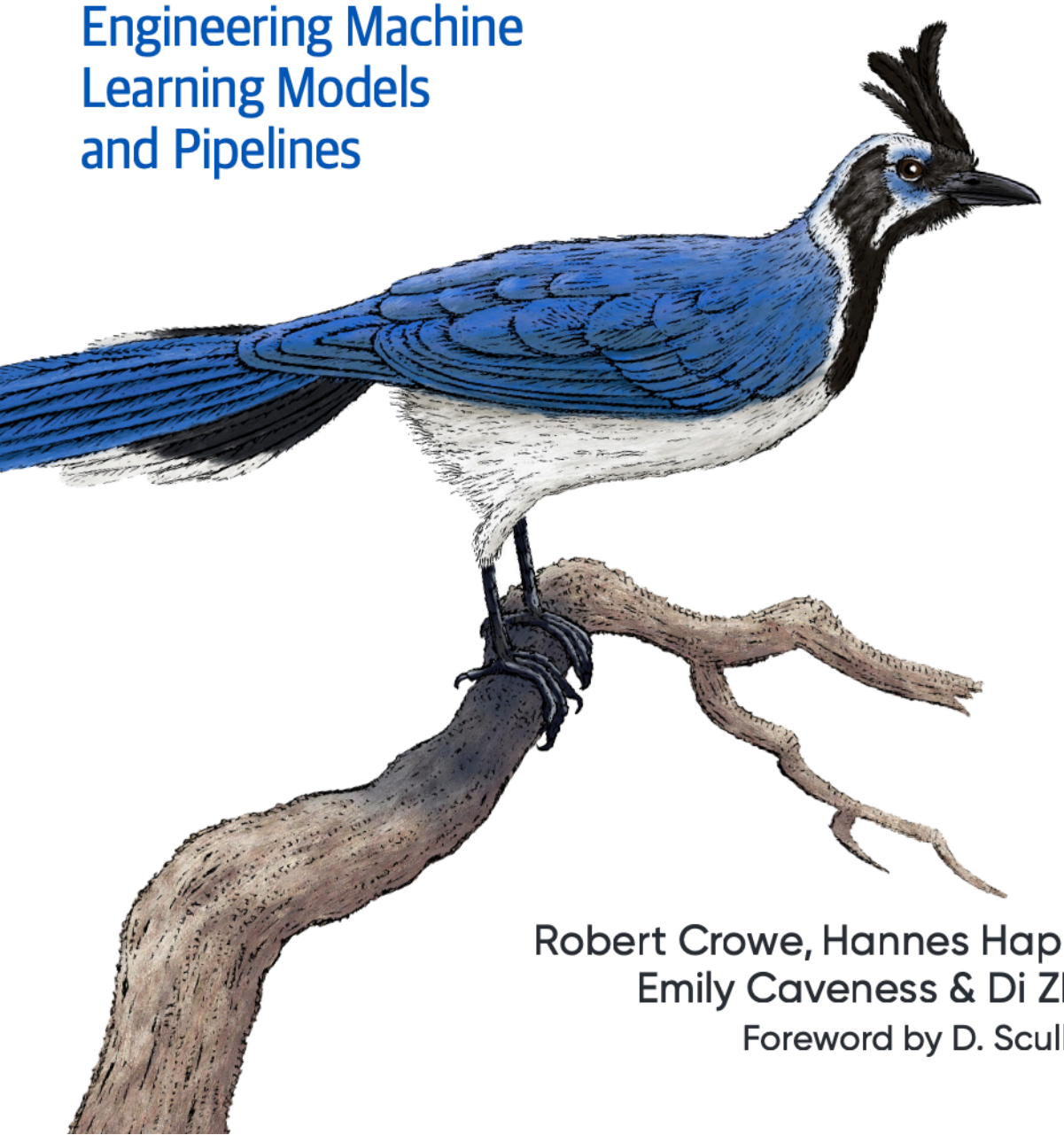
O'REILLY®

Machine Learning Production Systems

Engineering Machine Learning Models and Pipelines



Robert Crowe, Hannes Hapke,
Emily Caveness & Di Zhu
Foreword by D. Sculley

"A comprehensive book that gives you a holistic view of the entire process of building, deploying, and managing ML systems in production. It takes you through everything you need to know—from getting the most out of your data all the way through training models, deploying those models to scalable infrastructure, and managing the details to keep them running smoothly."

Laurence Moroney
AI consultant, teacher, and author

Machine Learning Production Systems

The world of machine learning (ML) and artificial intelligence (AI) is exploding, with new research, models, and technologies arriving nearly every day. Given this wealth of options, it's easy for data scientists, ML engineers, and software developers to get lost among the many steps necessary to take an AI/ML model from experiment stage into production.

This practical book focuses on *production machine learning*, a process that enables you to bring ML models into viable products and applications. Production machine learning covers all areas of ML, taking you beyond simple model training. This book places special emphasis on ML pipelines that will help you build the foundation for your ML production systems.

You'll explore a broad range of technologies you need to put ML applications into production, as well as the issues and approaches you need to consider. Critical ML engineering topics include:

- **Data collection, validation, storage, and feature engineering**
- **Model analysis, serving, monitoring, and logging**
- **Orchestrating machine learning pipelines using TensorFlow Extended (TFX) and other tools**

This publication provides in-depth examples, including end-to-end ML pipelines for NLP and computer vision models.

Robert Crowe, product manager for JAX and GenAI at Google, helps developers quickly learn what they need to be productive.

Hannes Hapke, principal machine learning engineer at Digits, has coauthored multiple machine learning publications.

Emily Caveness, software engineer at Google, currently works on ML data analysis and validation.

Di Zhu, software engineer at Google, has worked on a variety of projects, including MLOps infrastructure and applied machine learning solutions.

DATA

US $79.99 CAN $99.99
ISBN: 978-1-098-15601-5

O'REILLY®

Machine Learning Production Systems

Engineering Machine Learning Models and Pipelines

Robert Crowe, Hannes Hapke, Emily Caveness, and Di Zhu

Foreword by D. Sculley

Beijing • Boston • Farnham • Sebastopol • Tokyo

O'REILLY®

Machine Learning Production Systems
by Robert Crowe, Hannes Hapke, Emily Caveness, and Di Zhu

Printed in the United States of America.

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

Acquisitions Editor: Nicole Butterfield
Development Editor: Jeff Bleiel
Production Editor: Katherine Tozer
Copyeditor: Audrey Doyle
Proofreader: Piper Editorial Consulting, LLC

Indexer: WordCo Indexing Services, Inc.
Interior Designer: David Futato
Cover Designer: Karen Montgomery
Illustrator: Kate Dullea

October 2024: First Edition

Revision History for the First Edition
2024-10-01: First Release

See *http://oreilly.com/catalog/errata.csp?isbn=9781098156015* for release details.

978-1-098-15601-5

[LSI]

Table of Contents

Foreword

My first big break in AI and machine learning (ML) came about 20 years ago. It was during a time when the internet still felt like a brand new technology. The world was noticing that the power of free communication had drawbacks as well as benefits—with those drawbacks being most notable in the form of email spam. These unwanted messages were clogging up inboxes everywhere with shady offers for pills or scams seeking bank account information.

Email spam was a raging problem because the available spam filters (being based largely on hand-crafted rules and patterns) were ineffective. Spammers would fool these filters with all kinds of tricks, like int3nt!onal mi$$pellings or o t h e r h a c k y m e t h o d s that were hard for a fixed rule to adapt to. As a grad student at the time, I became part of the community of researchers that believed a funny technology called machine learning might be the right solution for this set of problems. I was even lucky enough to create a model that won one of the early benchmark competitions for email spam filtering.

I remember that early model for two reasons. First, it was kind of cool that it worked well by using a simple but very flexible representation—something that we would now call an early precursor to a one-dimensional convolution on strings. Second, I can look back and say with certainty that it would have been an absolute mess to put into a production environment. It had been designed under the pressures of academic research, in which velocity trumps reliability, and quick fixes and patches that work once are more than good enough. I didn't know any better. I had never actually met anyone who had run an ML pipeline in production. Back then I don't think I had ever even heard the words production and machine learning used together in the same sentence.

The first real production system I got to design and build was an early system at Google for detecting and removing ads that violated policies—basically ads that were scammy or spammy. This was important work, and I felt it was extremely rewarding to protect our users this way. It was also a time when creating an ML production

system meant building everything from scratch. There weren't reliable scalable libraries—this was well before PyTorch or TensorFlow—and infrastructure for data storage, model training, and serving all had to be built from scratch. As you might guess, this meant that I got hit with every single pitfall imaginable in production ML: validation, monitoring, safety checks, rollout plans, update mechanisms, dealing with churn, dealing with noise, handling unreliable labels, encountering unstable data dependencies—the list goes on. It was a hard way to learn these lessons, but the experience definitely made an impression.

A few years later, I was leading Google's systems for search ads click-through prediction. At the time, this was perhaps one of the largest and—from a business standpoint—most impactful ML systems in the world. Because of that, reliability was of the utmost importance, and much of the work that my colleagues and I did revolved around strengthening the production robustness of our system. This included both system-level robustness from an infrastructure perspective, and statistical robustness to ensure that changes in data over time would be handled well. Because running ML systems at this scale and importance was still quite new, we had to invent much of this for ourselves. We ended up writing a few papers on this experience, one of which was cheerfully titled "Machine Learning: The High Interest Credit Card of Technical Debt," hoping to share what we had learned with others in the field. And I got to help put some of these thoughts into general practice through some of the early designs of TensorFlow Extended (TFX).

Now here we are in the present day. AI and ML are more important than ever, and the emergent capabilities of large language models (LLMs) and generative AI (GenAI) are incredibly promising. There is also more awareness of the importance of production-grade safety, reliability, responsibility, and robustness—along with a keen understanding of just how difficult these problems can be. It might feel daunting to be taking on the challenge of building a new AI or ML pipeline.

Fortunately, today, you are not alone. The field has come a long way from those early days; we have some incredible benefits now. One incredible benefit is that the level of production-grade infrastructure has advanced considerably, and best practices have been codified into off-the-shelf offerings through TFX and similar offerings that significantly simplify building a robust pipeline.

But even more important than the infrastructure is the people in the field. There are folks like the authors of this book—Robert Crowe, Hannes Hapke, Emily Caveness, and Di Zhu—who are willing to serve as your guide through these pipeline jungles, providing painstakingly detailed knowledge. They will ensure you don't have to learn the way I did—by hitting pitfall after unexpected pitfall—and can put you on a well-lit path to success.

I have known Hannes and Robert for many years. Hannes and I first met at a Google Developer Advisory Board meeting, where he provided a ton of useful feedback on ways that Google could support ML developers even better, and I could tell from the first conversation that he was someone who had lived these problems and their solutions in the trenches for many years. Robert and I have been colleagues at Google for quite some time, and I have always been struck by both his technical expertise and by his ability to articulate clear and simple explanations for complex systems.

So you are in good hands, and you are in for an exciting journey. I very much hope that you don't just read this book— that you also build along with it and create something amazing, something that pushes forward the cutting edge of what AI and ML can do, and most of all, something that will not wake you up at 3 a.m. with a production outage.

Very best wishes for your journey!

— D. Sculley
CEO, Kaggle
August 2024

Preface

As we write this book in 2024, the world of machine learning and artificial intelligence (ML/AI) is exploding, with new research, models, and technologies arriving nearly every day. While large language models (LLMs) and diffusion models are the exciting new things, the technologies for building those new models rest on a foundation of years of advancement in deep learning and even earlier classic approaches. All the previous work in this field seems to have reached a turning point where we are beginning to see the exponential growth of new applications, built on new capabilities, that will fundamentally accelerate progress in a wide range of fields and directly impact people's lives. It's an incredibly exciting time to be working in this field!

This gets us to the focus of this book, which is to take those technologies and use them to create new products and services.

Who Should Read This Book

If you're working in ML/AI or if you want to work in ML/AI in any way other than pure research, this book is for you. It's primarily focused on people who will have a job title of "ML engineer" or something similar, but in many cases, they'll also be considered data scientists (the difference between the two job descriptions is often murky). On a more fundamental level, this book is for people who need to know about taking ML/AI technologies and using them to create new products and services. Putting models and applications into production might be the main focus of your job, or it might be something that you do occasionally, or it might even be something done by a team you collaborate with. In all cases, the topics we discuss in this book will help you understand the issues and approaches that need to be considered and applied when putting ML/AI applications into production.

Why We Wrote This Book

While this book is fairly comprehensive, it is intended in many cases to only introduce you to the topics involved and give you enough background to know when you need to dig deeper. Nearly all of these topics have entire books written about them, and the field is constantly evolving. Knowing the landscape, and knowing when you need to know more or check for new developments, are skills you will apply throughout your career.

We've seen books that covered many subsets of these topics, but when we went looking for a more comprehensive view of production ML, we found big gaps in what was available. That's what inspired us to write this book and attempt to present you with a more complete picture of the state of the art along the entire range of technologies that are used to put ML/AI applications into production. As you'll see, it's a very broad range of topics. It would have been great to have a book like this when we were starting our careers, but many of these technologies were just beginning to be developed. So, while we learned about them through working in this field, this book will give you a big head start.

Navigating This Book

It's not a bad idea to read the chapters in this book in order, but you can also just pick a chapter in an area you're interested in and start reading. Each chapter is fairly self-contained, with references to other chapters identified. The chapters are organized as follows:

- Chapter 1 provides a quick overview of ML for production applications.
- Chapters 2 through 5 focus on data.
- Chapters 6 through 10 focus on specialized topics:
  - Chapter 6 focuses on model resource management.
  - Chapter 7 focuses on performance and accelerators.
  - Chapters 8 and 9 focus on analyzing and interpreting models.
  - Chapter 10 focuses on AutoML.
- Chapters 11 through 16 focus on model serving and inference.
- Chapter 17 focuses on privacy and legal issues related to ML/AI.
- Chapters 18 through 21 focus on ML pipelines.
- Chapter 22 focuses on LLMs, diffusion models, and generative AI.
- Chapter 23 focuses on the future.

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.

This element signifies a tip or suggestion.

This element signifies a general note.

This element indicates a warning or caution.

Using Code Examples

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

If you have a technical question or a problem using the code examples, please email *support@oreilly.com*. The authors can also be reached at *mlproductionsystems@googlegroups.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: "*Machine Learning Production Systems* by Robert Crowe, Hannes Hapke, Emily Caveness, and Di Zhu (O'Reilly). Copyright 2025 Robert Crowe, Hannes Hapke, Emily Caveness, and Di Zhu, 978-1-098-15601-5."

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

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 *https://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
800-889-8969 (in the United States or Canada)
707-827-7019 (international or local)
707-829-0104 (fax)

support@oreilly.com
https://oreilly.com/about/contact.html

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at *https://oreil.ly/ML-Production-Systems*.

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

Find us on LinkedIn: *https://linkedin.com/company/oreilly-media*.

Watch us on YouTube: *https://youtube.com/oreillymedia*.

Acknowledgments

We've had so much support from so many people throughout the process of writing this book! Thank you so much to everyone who helped make it a reality. We would like to give an especially big thank-you to the following people:

Writing this book has been a labor of love for the authors. The book was inspired by Robert's development of a series of Coursera courses with Andrew Ng and Laurence Moroney that focused on production ML. It was also inspired by the O'Reilly book *Building Machine Learning Pipelines*, which Hannes wrote with Catherine Nelson.

So to start, we'd like to thank Andrew Ng, Laurence Moroney, and Catherine Nelson for their efforts on those earlier works. Their efforts helped guide our team toward the focus of this book.

We'd also like to thank Jarek Kazmierczak for his work on shaping the outline for the series of Coursera courses that Robert worked on. His experience and perspective were invaluable for identifying the range of topics that needed to be covered.

We'd also like to thank the technical reviewers of this book: Margaret Maynard-Reid, Ashwin Raghav, Stef Ruinard, Vikram Tiwari, and Glen Yu. Their time and efforts to read through the initial draft of the book and provide comments, suggestions, and corrections made a major contribution toward improving the book's quality overall.

Everyone at O'Reilly has been fantastic to work with throughout the book's lifecycle, starting with Mike Loukides, who originally proposed the idea of a book on production ML, and continuing with Nicole Butterfield and Jeff Bleiel, who worked with us to shape and edit the book. When we were ready to move to the production process, Katherine Tozer, Audrey Doyle, and Kristen Brown were just amazing. A big thank-you to the entire O'Reilly team!

We'd also like to thank the team of Googlers who developed many of the technologies that we discuss in the book. There are really too many to list, but they include everyone in the TFX, TFDV, TFT, TFMA, MLMD, TF Serving, and Vertex teams. In

many cases, you were doing groundbreaking work that set the standard for this field going forward! Thank you for your hard work, dedication, and vision.

Robert

I couldn't have even considered dedicating the necessary time to work on this project without the love and support of my family: my loving wife, Jayne; my daughter, Zoe; and my son, Michael. Your love and support throughout this process have given me the space and inspiration that have made this effort possible, and without you, this book would not exist.

I'd also like to thank my managers at Google, who supported my effort to write this book, starting with Laurence Moroney and continuing with Joe Spisak, Eve Phillips, and Grace Chen. Your support in this effort, and in all my efforts, was hugely appreciated and impactful and was really critical to the success of this project.

Finally, I'd like to thank my coauthors, Hannes, Emily, and Di! You have been fantastic and inspiring to work with, and we're finally at the finish line!

Hannes

In this ever-changing world of machine learning, writing a technical book is a daunting task. Thank you, Robert, Emily, and Di, for letting me join this exciting and insightful journey.

I am deeply grateful to the entire team at Digits, especially Jeff Seibert, Cole Howard, and Jo Pu, for their endless support and for the opportunity to let me implement production machine learning systems from scratch. We have learned so much in our endeavors to bring machine learning into production.

To the staff at O'Reilly Media, especially Jeff Bleiel, Nicole Butterfield, and Katie Tozer, thank you for lending your time and expertise to review early drafts and guide the project. Your feedback was essential in shaping the content.

Finally, this book wouldn't exist without the unwavering support, endless patience, love, and ability to always make me smile by my partner, Whitney. Thank you for being a rock. Thank you to my family, especially my parents, who let me follow my dreams throughout the world.

Emily

I'd like to thank my coauthors, Robert, Hannes, and Di, for giving me the opportunity to participate in this project. I've learned so much from you, am impressed with your breadth of knowledge, and value all the hard work you put into making this book a reality.

I'd also like to thank my manager and colleagues at Google, who not only have connected me with this opportunity but have supported my continued growth and learning in the dynamic space covered in this book.

Finally, and above all, I would like to thank my family for supporting all I do professionally.

Di

I want to thank my parents and friends for encouraging me throughout my career and life and for supporting me during the challenging times while writing this book.

I am grateful to my coauthors, Robert, Hannes, and Emily, for joining me on this journey. I learned a lot from our collaboration. I also appreciate the staff at O'Reilly, especially Jeff, for working with us throughout the process.

I would like to thank my manager and colleagues at Google for providing valuable technical insights. They have greatly expanded my understanding of the various domains.

CHAPTER 1

Introduction to Machine Learning Production Systems

The field of machine learning engineering is so vast that it can be easy to get lost in the different steps that are necessary to get a model from an experiment into a production deployment. Over the last few years, machine learning, novel machine learning concepts such as attention, and more recently, large language models (LLMs) have been in the news almost every day. However, very little discussion has focused on production machine learning, which brings machine learning into products and applications.

Production machine learning covers all areas of machine learning beyond simply training a machine learning model. Production machine learning can be viewed as a combination of machine learning development practices and modern software development practices. Machine learning pipelines build the foundation for production machine learning. Implementing and executing machine learning pipelines are key aspects of production machine learning.

In this chapter, we will introduce the concept of production machine learning. We'll also introduce what machine learning pipelines are, look at their benefits, and walk through the steps of a machine learning pipeline.

What Is Production Machine Learning?

In an academic or research setting, modeling is relatively straightforward. Typically, you have a dataset (often a standard dataset that is supplied to you, already cleaned and labeled), and you're going to use that dataset to train your model and evaluate the results.

The result you're trying to achieve is simply a model that makes good predictions. You'll probably go through a few iterations to fully optimize the model, but once you're satisfied with the results, you're typically done.

Production machine learning (ML) requires a lot more than just a model. We've found that a model usually contains only about 5% of the code that is required to put an ML application into production. Over their lifetimes, production ML applications will be deployed, maintained, and improved so that you can consistently deliver a high-quality experience to your users.

Let's look at some of the differences between ML in a nonproduction environment (generally research or academia) and ML in a production environment:

- In an academic or research environment, you're typically using a static dataset. Production ML uses real-world data, which is dynamic and usually shifting.
- For academic or research ML, there is one design priority, and usually it is to achieve the highest accuracy over the entire training set. But for production ML, there are several design priorities, including fast inference, fairness, good interpretability, acceptable accuracy, and cost minimization.
- Model training for research ML is based on a single optimal result, and the tuning and training necessary to achieve it. Production ML requires continuous monitoring, assessment, and retraining.
- Interpretability and fairness are very important for any type of ML modeling, but they are absolutely crucial for production ML.
- And finally, while the main challenge with academic and research ML is to find and tune a high-accuracy model, the main challenge with production ML is a high-accuracy model plus the rest of the system that is required to operate the model in production.

In a production ML environment, you're not just producing a single result; you're developing a product or service that is often a mission-critical part of your offering. For example, if you're doing supervised learning, you need to make sure your labels are accurate. You also need to make sure your training dataset has examples that cover the same feature space as the requests your model will receive. In addition, you want to reduce the dimensionality of your feature vector to optimize system performance while retaining or enhancing the predictive information in your data.

Throughout all of this, you need to consider and measure the fairness of your data and model, especially for rare conditions. In fields such as health care, for example, rare but important conditions may be absolutely critical to success.

On top of all that, you're putting a piece of software into production. This requires a system design that includes all the things necessary for any production software deployment, including the following:

- Data preprocessing methods
- Parallelized model training setups
- Repeatable model analysis
- Scalable model deployment

Your production ML system needs to run automatically so that you're continuously monitoring model performance, ingesting new data, retraining as needed, and redeploying to maintain or improve performance.

And of course, you need to try to build your production ML system so that it achieves maximal performance at a minimal cost. That might seem like a daunting task, but the good news is that there are well-established tools and methodologies for doing this.

Benefits of Machine Learning Pipelines

When new training data becomes available, a workflow that includes data validation, preprocessing, model training, analysis, and deployment should be triggered. The key benefit of ML pipelines lies in automation of the steps in the model lifecycle. We have observed too many data science teams manually going through these steps, which is both costly and a source of errors. Throughout this book, we will introduce tools and solutions to automate your ML pipelines.

Let's take a more detailed look at the benefits of building ML pipelines.

Focus on Developing New Models, Not on Maintaining Existing Models

Automated ML pipelines free up data scientists from maintaining existing models for large parts of their lifecycle. It's not uncommon for data scientists to spend their days keeping previously developed models up-to-date. They run scripts manually to preprocess their training data, they write one-off deployment scripts, or they manually tune their models. Automated pipelines allow data scientists to develop new models—the fun part of their job. Ultimately, this will lead to higher job satisfaction and retention in a competitive job market.

Prevention of Bugs

Automated pipelines can prevent bugs. As we will explain in later chapters, newly created models will be tied to a set of versioned data, and preprocessing steps will be tied to the developed model. This means that if new data is collected, a new version of the model will be generated. If the preprocessing steps are updated, the training data will become invalid and a new model will be generated.

In manual ML workflows, a common source of bugs is a change in the preprocessing step after a model was trained. In such a case, we would deploy a model with different processing instructions than what we trained the model with. These bugs might be really difficult to debug, since an inference of the model is still possible but is simply incorrect. With automated workflows, these errors can be prevented.

Creation of Records for Debugging and Reproducing Results

In a well-structured pipeline, experiment tracking generates a record of the changes made to a model. This form of model release management enables data scientists to keep track of which model was ultimately selected and deployed. This record is especially valuable if the data science team needs to re-create the model, create a new variant of the model, or track the model's performance.

Standardization

Standardized ML pipelines improve the work experience of a data science team. Not only can data scientists be onboarded quickly, but they also can move across teams and find the same development environments. This improves efficiency and reduces the time spent getting set up on a new project.

The Business Case for ML Pipelines

In short, the implementation of automated ML pipelines leads to four key benefits for a data science team:

- More development time to spend on novel models
- Simpler processes to update existing models
- Less time spent on reproducing models
- Good information about previously developed models

All of these aspects will reduce the costs of data science projects. Automated ML pipelines will also do the following:

- Help detect potential biases in the datasets or trained models, which can prevent harm to people who interact with the model (e.g., Amazon's ML-powered resume screener was found to be biased against females).
- Create a record (via experiment tracking and model release management) that will assist if questions arise around data protection laws, such as AI regulations in Europe or an AI Bill of Rights in the United States.
- Free up development time for data scientists and increase their job satisfaction.

When to Use Machine Learning Pipelines

Production ML and ML pipelines provide a variety of advantages, but not every data science project needs a pipeline. Sometimes data scientists simply want to experiment with a new model, investigate a new model architecture, or reproduce a recent publication. Pipelines wouldn't be useful in these cases. However, as soon as a model has users (e.g., it is being used in an app), it will require continuous updates and fine-tuning. In these situations, you need an ML pipeline. If you're developing a model that is intended to go into production and you feel fairly confident about the design, starting in a pipeline will save time later when you're ready to graduate your model to production.

Pipelines also become more important as an ML project grows. If the dataset or resource requirements are large, the ML pipeline approach allows for easy infrastructure scaling. If repeatability is important, even when you're only experimenting, it is provided through the automation and the audit trail of ML pipelines.

Steps in a Machine Learning Pipeline

An ML pipeline starts with the ingestion of new training data and ends with the receipt of some kind of feedback on how your newly trained model is performing. This feedback can be a production performance metric, or it can be feedback from users of your product. The pipeline comprises a number of steps, including data preprocessing, model training, model analysis, and model deployment.

As you can see in Figure 1-1, the pipeline is actually a recurring cycle. Data can be continuously collected, and therefore, ML models can be updated. More data generally means improved models. And because of this constant influx of data, automation is key.

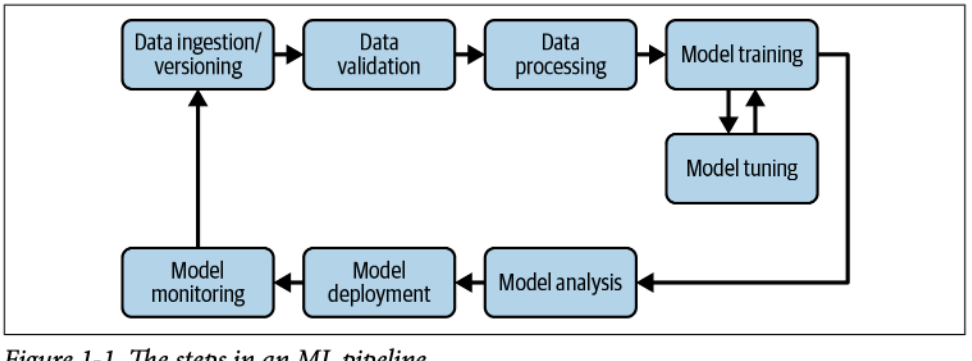


Figure 1-1. The steps in an ML pipeline

In real-world applications, you want to retrain your models frequently. If you don't, in many cases accuracy will decrease because the training data is different from the new data on which the model is making predictions. If retraining is a manual process, where it is necessary to manually validate the new training data or analyze the updated models, a data scientist or ML engineer would have no time to develop new models for entirely different business problems.

Let's discuss the steps that are most commonly included in an ML pipeline.

Data Ingestion and Data Versioning

Data ingestion occurs at the beginning of every ML pipeline. During this step, we process the data into a format that the components that follow can digest. The data ingestion step does not perform any feature engineering; this happens after the data validation step. This is also a good time to version the incoming data to connect a data snapshot with the trained model at the end of the pipeline.

Data Validation

Before training a new model version, we need to validate the new data. Data validation (discussed in detail in Chapter 2) focuses on checking that the statistics of the new data—for example, the range, number of categories, and distribution of categories—are as expected. It also alerts the data scientist if any anomalies are detected.

For example, say you are training a binary classification model, and 50% of your training data consists of Class X samples and 50% consists of Class Y samples. Data validation tools would alert you if the 50/50 split between these classes changes to, say, 70/30. If a model is being trained with such an imbalanced training set and you haven't adjusted the model's loss function or over-/under-sampled one of the sample categories, the model predictions could be biased toward the dominant category.

Data validation tools will also allow a data scientist to compare datasets and highlight anomalies. If the validation highlights anything out of the ordinary, the pipeline can be stopped and the data scientist can be alerted. If a shift in the data is detected, the data scientist or the ML engineer can either change the sampling of the individual classes (e.g., only pick the same number of examples from each class), or change the model's loss function, kick off a new model build pipeline, and restart the lifecycle.

Feature Engineering

It is highly likely that you cannot use your freshly collected data and train your ML model directly. In almost all cases, you will need to preprocess the data to use it for your training runs. That preprocessing is referred to as *feature engineering*. Labels often need to be converted to one-hot or multi-hot vectors. The same applies to the

model inputs. If you train a model from text data, you want to convert the characters of the text to indices, or convert the text tokens to word vectors. Since preprocessing is only required prior to model training and not with every training epoch, it makes the most sense to run the preprocessing in its own lifecycle step before training the model.

Data preprocessing tools can range from a simple Python script to elaborate graph models. It's important that, when changes to preprocessing steps happen, the previous training data should become invalid and force an update of the entire pipeline.

Model Training and Model Tuning

Model training is the primary goal of most ML pipelines. In this step, we train a model to take inputs and predict an output with the lowest error possible. With larger models, and especially with large training sets, this step can quickly become difficult to manage. Since memory is generally a finite resource for our computations, efficient distribution of model training is crucial.

Model tuning has seen a great deal of attention lately because it can yield significant performance improvements and provide a competitive edge. Depending on your ML project, you may choose to tune your model before you start to think about ML pipelines, or you may want to tune it as part of your pipeline. Because our pipelines are scalable thanks to their underlying architecture, we can spin up a large number of models in parallel or in sequence. This lets us pick out the optimal model hyperparameters for our final production model.

Model Analysis

Generally, we would use accuracy or loss to determine the optimal set of model parameters. But once we have settled on the final version of the model, it's extremely useful to carry out a more in-depth analysis of the model's performance. This may include calculating other metrics such as precision, recall, and area under the curve (AUC), or calculating performance on a larger dataset than the validation set used in training.

An in-depth model analysis should also check that the model's predictions are fair. It's impossible to tell how the model will perform for different groups of users unless the dataset is sliced and the performance is calculated for each slice. We can also investigate the model's dependence on features used in training and explore how the model's predictions would change if we altered the features of a single training example.

Similar to the model-tuning step and the final selection of the best-performing model, this workflow step requires a review by a data scientist. The automation will keep the analysis of the models consistent and comparable against other analyses.

Model Deployment

Once you have trained, tuned, and analyzed your model, it is ready for prime time. Unfortunately, too many models are deployed with one-off implementations, which makes updating models a brittle process.

Model servers allow you to update model versions without redeploying your application. This will reduce your application's downtime and reduce the amount of communication necessary between the application development team and the ML team.

Looking Ahead

In Chapters 20 and 21, we will introduce two examples of a production ML process in which we implement an ML pipeline from end to end. In those examples, we'll use TensorFlow Extended (TFX), an open source, end-to-end ML platform that lets you implement ML pipelines exactly as you would for production systems.

But first, we will discuss the ML pipeline steps in more detail. We'll start with data collection, labeling, and validation, covered next.

CHAPTER 2

Collecting, Labeling, and Validating Data

In production environments, you discover some interesting things about the importance of data. We asked ML practitioners at Uber and Gojek, two businesses where data and ML are mission critical, about it. Here's what they had to say:

> *Data is the hardest part of ML and the most important piece to get right...Broken data is the most common cause of problems in production ML systems.*
>
> —ML practitioner at Uber

> *No other activity in the machine learning lifecycle has a higher return on investment than improving the data a model has access to.*
>
> —ML practitioner at Gojek

The truth is that if you ask any production ML team member about the importance of data, you'll get a similar answer. This is why we're talking about data: it's incredibly important to success, and the issues for data in production environments are very different from those in the academic or research environment that you might be familiar with.

OK, now that we've gotten that out of the way, let's dive in!

Important Considerations in Data Collection

In programming language design, a *first-class citizen* in a given programming language is an entity that supports all the operations generally available to other entities. In ML, data is a first-class citizen. Finding data with predictive content might sound easy, but in reality it can be incredibly difficult.

When collecting data, it's important to ensure that the data represents the application you are trying to build and the problem you are trying to solve. By that we mean you need to ensure that the data has feature space coverage that is close to that of the prediction requests you will receive.

Another key part of data collection is sourcing, storing, and monitoring your data responsibly. This means that when you're collecting data, it is important to identify potential issues with your dataset. For example, the data may have come from different measurements of different types (e.g., the dataset may mix some measurements that come from two different types of thermometers that produce different measurements). In addition, simple things like the difference between an integer and a float, or how a missing value is encoded, can cause problems. As another example, if you have a dataset that measures elevation, does an entry of 0 feet mean no elevation (sea level), or that no elevation data was received for that record? If the output of other ML models is the input dataset for your model, you also need to be aware of the potential for errors to propagate over time. And you want to make sure you're looking for potential problems early in the process by monitoring data sources for system issues and outages.

When collecting data, you will also need to understand data effectiveness by dissecting which features have predictive value. Feature engineering helps maximize the predictive signal of your data, and feature selection helps measure the predictive signal.

Responsible Data Collection

In this section, we will discuss how to responsibly source data. This involves ensuring data security and user privacy, checking for and ensuring fairness, and designing labeling systems that mitigate bias.

ML system data may come from different sources, including synthetic datasets you build, open source datasets, web scraping, and live data collection. When collecting data, data security and data privacy are important. *Data security* refers to the policies, methods, and means to secure personal data. *Data privacy* is about proper usage, collection, retention, deletion, and storage of data.

Data management is not only about the ML product. Users should also have control over what data is being collected. In addition, it is important to establish mechanisms to prevent systems from revealing user data inadvertently. When thinking about user privacy, the key is to protect personal identifiable information (PII). Aggregating, anonymizing, redacting, and giving users control over what data they share can help prevent issues with PII. How you handle data privacy and data security depends on the nature of the data, the operating conditions, and regulations currently in place

(an example is the General Data Protection Regulation or GDPR, a European Union regulation on information privacy).

In addition to security and privacy, you must consider fairness. ML systems need to strike a delicate balance in being fair, accurate, transparent, and explainable. However, such systems can fail users in the following ways:

Representational harm
: When a system amplifies or reflects a negative stereotype about particular groups

Opportunity denial
: When a system makes predictions that have negative real-life consequences, which could result in lasting impacts

Disproportionate product failure
: When you have skewed outputs that happen more frequently for a particular group of users

Harm by disadvantage
: When a system infers disadvantageous associations between different demographic characteristics and the user behaviors around them

When considering fairness, you need to check that your model does not consistently predict different experiences for some groups in a problematic way, by ensuring group fairness (demographic parity and equalized odds) and equal accuracy.

One aspect of this is looking at potential bias in human-labeled data. For supervised learning, you need accurate labels to train your model on and to serve predictions. These labels usually come from two sources: automated systems and human raters. *Human raters* are people who look at the data and assign a label to it. There are various types of human raters, including generalists, trained subject matter experts, and users. Humans are able to label data in different ways than automated systems can. In addition, the more complicated the data is, the more you may require a human expert to look at that data.

When considering fairness with respect to human-labeled data, there are many things to think about. For instance, you will want to ensure rater pool diversity, and you will want to account for rater context and incentives. In addition, you'll want to evaluate rater tools and consider cost, as you need a sufficiently large dataset. You will also want to consider data freshness requirements.

Labeling Data: Data Changes and Drift in Production ML

When thinking about data, you must also consider the fact that data changes often. There are numerous potential causes of data changes or problems, which can be categorized as those that cause gradual changes or those that cause sudden changes.

Gradual changes might reflect changes in the data and/or changes in the world that affect the data. Gradual data changes include those due to trends or seasonality, changes in the distribution of features, or changes in the relative importance of features. Changes in the world that affect the data include changes in styles, scope and process changes, changes in competitors, and expansion of a business into different markets or areas.

Sudden changes can involve both data collection problems and system problems. Examples of data collection problems that cause sudden changes in data include moved, disabled, or malfunctioning sensors or cameras, or problems in logging. Examples of system problems that can cause sudden changes in data include bad software updates, loss of network connectivity, or a system delay or failure.

Thinking about data changes raises the issues of data drift and concept drift. With *data drift*, the distribution of the data input to your model changes. Thus, the data distribution on which the model was trained is different from the current input data to the model, which can cause model performance to decay in time. As an example of data drift, if you have a model that predicts customer clothing preferences that was trained with data collected mainly from teenagers, the accuracy of that model would be expected to degrade if data from older adults is later fed to the model.

With *concept drift*, the relationship between model inputs and outputs changes over time, which can also lead to poorer model performance. For example, a model that predicts consumer clothing preferences might degrade over time as new trends, seasonality, and other previously unseen factors change the customer preferences themselves.

To handle potential data change, you must monitor your data and model performance continuously, and respond to model performance decays over time. When ground truth changes slowly (i.e., over months or years), handling data change tends to be relatively easy. Model retraining can be driven by model improvements, better data, or changes in software or systems. And in this case, you can use curated datasets built using crowd-based labeling.

When ground truth changes more quickly (i.e., over weeks), handling data change tends to become more difficult. In these cases, model retraining can be driven by the factors noted previously, but also by declining model performance. Here, datasets tend to be labeled using direct feedback or crowd-based labeling.

When ground truth changes even more quickly (i.e., over days, hours, or minutes), things become even more difficult. Here, model retraining can be driven by declining model performance, the desire to improve models, better training data availability, or software system changes. Labeling in this scenario could be through direct feedback (discussed next), or through weak supervision for applying labels quickly.

Labeling Data: Direct Labeling and Human Labeling

Training datasets need to be created using the data available to the organization, and models often need to be retrained with new data at some frequency. To create a current training dataset, examples must be labeled. As a result, labeling becomes an ongoing and mission-critical process for organizations doing production ML.

We will start our discussion of labeling data by taking a look at direct labeling and human labeling. *Direct labeling* involves gleaning information from your system—for example, by tracking click-through rates. *Human labeling* involves having a person label examples with ground truth values—for example, by having a cardiologist label MRI scans as a subject matter expert rater. There are also other methods, including semi-supervised labeling, active learning, and weak supervision, which we will discuss in later chapters that address advanced labeling methods.

Direct labeling has several advantages: it allows for a training dataset to be continuously created, as labels can be added from logs or other system-collected information as data arrives; it allows labels to evolve and adapt quickly as the world changes; and it can provide strong label signals. However, there are situations in which direct labeling is not available or has disadvantages. For example, for some types of ML problems, labels cannot be gleaned from your system. In addition, direct labeling can require custom designs to fit your labeling processes with your systems.

In cases where direct labeling is useful, there are open source tools that you can use for log analysis. Two such tools are Logstash and Fluentd. Logstash is a data processing pipeline for collecting, transforming, and storing logs from different sources. Collected logs can then be sent to one of several types of outputs. Fluentd is a data collector that can collect, parse, transform, and analyze data. Processed data can then be stored or connected with various platforms. In addition, Google Cloud provides log analytics services for storing, searching, analyzing, monitoring, and alerting on logging data and events from Google Cloud and Amazon Web Services (AWS). Other systems, such as AWS Elasticsearch and Azure Monitor, are also available for log processing and can be used in direct labeling.

With human labeling, raters examine data and manually assign labels. Typically, raters are recruited and given instructions to guide their assignment of ground truth values. Unlabeled data is collected and divided among the raters, often with the same data being assigned to more than one rater to improve quality. The labels are collected, and conflicting labels are resolved.

Human labeling allows more labels to be annotated than might be possible through other means. However, there are disadvantages to this approach. Depending on the dataset, it might be difficult for raters to assign the correct label, resulting in a low-quality dataset. Quality might also suffer due to rater inexperience and other factors. Human labeling can also be an expensive and slow process, and can result

in a smaller training dataset than could be created through other methods. This is particularly the case for domains that require significant specialization or expertise to be able to label the data, such as medical imaging. In addition, human labeling is subject to the fairness considerations discussed earlier in this chapter.

Validating Data: Detecting Data Issues

As discussed, there are many ways in which your data can change or in which the systems that impact your data can cause unanticipated issues. Especially in light of the importance of data to ML systems, detecting such issues is essential. In this section, we will discuss common issues to look for in your data, and the concepts involved in detecting those issues. In the next section, we'll explore a specific tool for detecting such data issues.

As we noted earlier, issues can arise due to differences in datasets. One such issue or group of issues is drift, which, as we mentioned previously, involves changes in data over time. With *data drift*, the statistical properties of the input features change due to seasonality, events, or other changes in the world. With *concept drift*, the statistical properties of the labels change over time, which can invalidate the mapping found during training.

Skew involves changes between datasets, often between training datasets and serving datasets. *Schema skew* occurs when the training and serving datasets do not conform to the same schema. *Distribution skew* occurs when the distribution of values in the training and serving datasets differs.

Validating Data: TensorFlow Data Validation

Now that you understand the basics of data issues and detection workflows, let's take a look at TensorFlow Data Validation (TFDV), a library that allows you to analyze and validate data using Python and Apache Beam. Google uses TFDV to analyze and validate petabytes of data every day across hundreds or thousands of different applications that are in production. The library helps users maintain the health of their ML pipelines by helping them understand their data and detect data issues like those discussed in this chapter.

TFDV allows users to do the following:

- Generate summary statistics over their data
- Visualize those statistics, including visually comparing two datasets
- Infer a schema to express the expectations for their data

- Check the data for anomalies using the schema
- Detect drift and training–serving skew

Data validation in TFDV starts with generating summary statistics for a dataset. These statistics can include feature presence, values, and valency, among other things. TFDV leverages Apache Beam's data processing capabilities to compute these statistics over large datasets.

Once TFDV has computed these summary statistics, it can automatically create a schema that describes the data by defining various constraints including feature presence, value count, type, and domain. Although it is useful to have an automatically inferred schema as a starting point, the expectation is that users will tweak or curate the generated schema to better reflect their expectations about their data.

With a refined schema, a user can then run anomaly detection using TFDV. TFDV can do several types of anomaly detection, including comparison of a single set of summary statistics to a schema to ensure that the data from which the statistics were generated conforms to the user's expectations. TFDV can also compare the data distributions between two datasets—again using TFDV-generated summary statistics—to help identify potential drift or training–serving skew (discussed further in the next section).

The results of TFDV's anomaly detection process can help users further refine the schema or identify potentially problematic inconsistencies in their data. The schema can then be maintained over time and used to validate new data as it arrives.

Skew Detection with TFDV

Let's take a closer look at TFDV's ability to detect anomalies such as data drift and training–serving skew between datasets. For our discussion, *drift* refers to differences across iterations of training data and *skew* refers to differences between training and serving data.

You can use TFDV to detect three types of skew: schema skew, feature skew, and distribution skew, as shown in Figure 2-1.

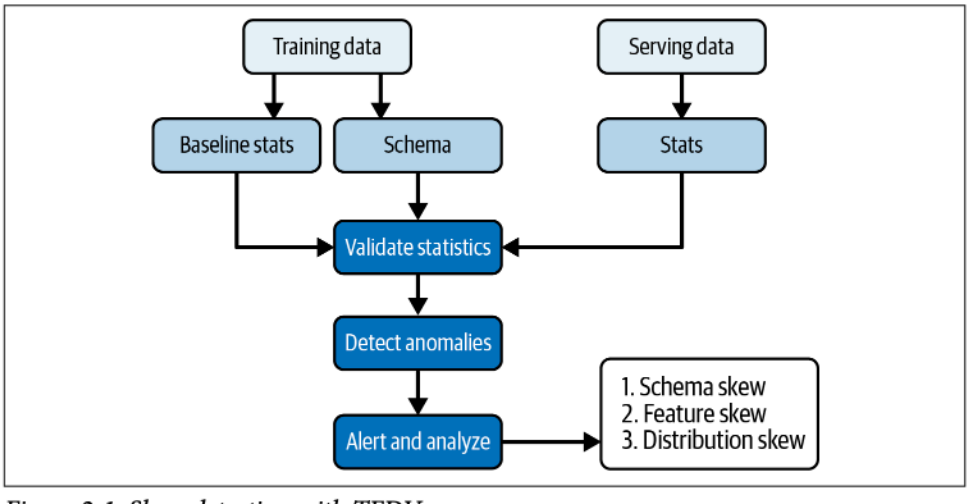


Figure 2-1. Skew detection with TFDV

Types of Skew

Schema skew occurs when the training data and serving data do not conform to the same schema; for example, if Feature A is a float in the training data but an integer in the serving data. Schema skew is detected similarly to single-dataset anomaly detection, which compares the dataset to a specified schema.

Feature skew occurs where feature values that are supposed to be the same in both training data and serving data differ. To identify feature skew, TFDV joins the training and serving examples on one or more specified identifier features, and then compares the feature values to identify the resulting pairs. If they differ, TFDV reports the difference as feature skew. Because feature skew is computed using examples and not summary statistics, it is computed separately from the other validation steps.

Distribution skew occurs when there is a shift in the distribution of feature values across two datasets. TFDV uses L-infinity distance (for categorical features only) and Jensen–Shannon divergence (for numeric and categorical features) to identify and measure such shifts. If the measure exceeds a user-specified threshold, TFDV will raise a distribution skew anomaly noting the difference.

Various factors can cause the distribution of serving and training datasets to differ significantly, including faulty sampling during training, use of different data sources for training and serving, and trend, seasonality, or other changes over time. Once TFDV helps identify potential skew, you can investigate the shift to determine whether it's a problem that needs to be remedied.

Example: Spotting Imbalanced Datasets with TensorFlow Data Validation

Let's say you want to visually and programmatically detect whether your dataset is imbalanced. We consider datasets to be *imbalanced* if the sample quantities per label are vastly different (e.g., you have 100 samples for one category and 1,000 samples for another category). Real-world datasets will almost always be imbalanced for various reasons—for example, because the costs of acquiring samples for a certain category might be too high—but datasets that are too imbalanced hinder the model training process to generalize the overall problem.

TFDV offers simple ways to generate statistics of your datasets and check for imbalance. In this section, we'll take you through the steps of using TFDV to spot imbalanced datasets.

Let's start by installing the TFDV library:

```
$ pip install tensorflow-data-validation
```

If you have TFX installed, TFDV will automatically be installed as one of the dependencies.

With a few lines of code, we can analyze the data. First, let's generate the data statistics:

```
import tensorflow_data_validation as tfdv
stats = tfdv.generate_statistics_from_csv(
    data_location='your_data.csv',
    delimiter=',')
```

TFDV provides functions to load the data from a variety of formats, such as Pandas data frames (`generate_statistics_from_dataframe`) or TensorFlow's TFRecords (`generate_statistics_from_tfrecord`):

```
stats = tfdv.generate_statistics_from_tfrecord(
    data_location='your_data.tfrecord')
```

It even allows you to define your own data connectors. For more information, refer to the TFDV documentation.

If you want to programmatically check the label distribution, you can read the generated statistics. In our example, we loaded a spam detection dataset with data samples marked as `spam` or `ham`. As in every real-world example, the dataset contains more nonspam examples than spam examples. But how many? Let's check:

```
print(stats.datasets[0].features[0].string_stats.rank_histogram)
buckets {
  label: "ham"
  sample_count: 4827.0
}
```