Top 30 ML in Production Resources of 2019

Luigi Patruno

Founder of *MLinProduction.com*

Contents

1 About the Author

Luigi Patruno is a data scientist and consultant. He is currently the Director of Data Science at 2U, where he leads a team of data scientists responsible for building machine learning models and infrastructure. As a consultant Luigi helps companies generate value by applying modern data science methods to strategic business and product initiatives. He founded MLinProduction.com to collect and share best practices for operationalizing machine learning and he has taught graduate courses in statistics, data analysis, and big data engineering. He holds an M.S. Computer Science and a B.S. Mathematics from Fordham University.

2 Introduction

"What's a *data scientist*?"

"Uhh well, do you know how Amazon recommends you things to buy and Netflix recommends you movies to watch? Data scientists are the people who develop the algorithms to make that happen."

That's usually what I tell people when I explain what I do for work. Since the term "Data Science" is relatively new, many people outside of tech don't know what it means or what the role entails. But most people I speak with recognize that companies like Amazon and Netflix provide personalized experiences and recommend goods and services to buy. Since I can relate my job to common experiences like shopping or watching movies, it's much easier to explain to others what my job entails.

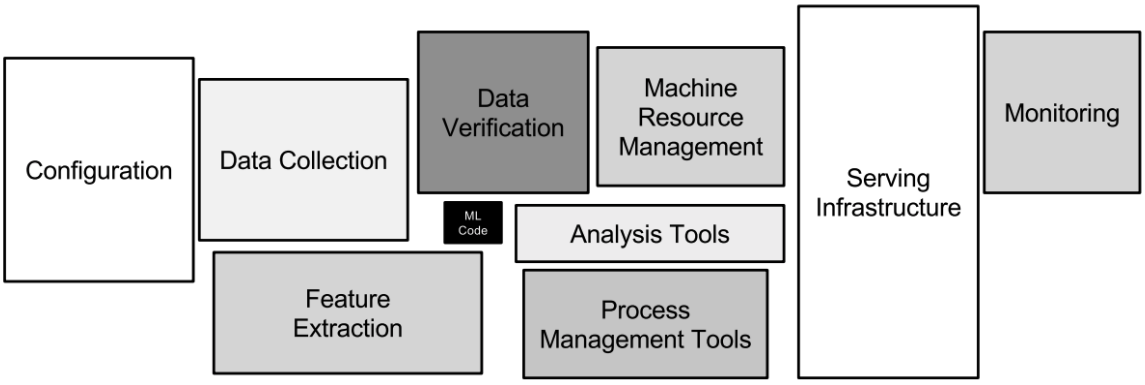
But it's not exactly *true*; I don't actually build recommender systems. This makes my analogy a bit disingenous, but it's much easier for me. So if you run into a data science lay person who claims all data scientists do is build recommenders, you have me to blame.

About a year ago I realized that much of the data science community is guilty of similar imprecision.

I was online searching for resources describing how to monitor machine learning models once they've been deployed to production. This should have been easy; monitoring a technology that's impacting millions of users is not exactly a ground-breaking idea. But it turned out to be much harder than I imagined. Instead, I found tons of content dedicated to a small cluster of topics in ML. How to train a deep convolutional neural

network on ImageNet. How to use reinforcement learning algorithms to play Atari. How to generate fake text with a generative adversarial network.

This was odd to me for a few reasons. First, I know a fair number of data scientists across different industries and almost none of them have worked on reinforcement learning outside of academia. Second, a recurring theme of the applied ML conferences I've been to is how few companies are actually using deep learning. Finally, my practical experience working in data science at startups and public companies demonstrates that the day-to-day life of a data scientist looks like this:



Source: **Hidden Technical Debt in Machine Learning Systems**

Most of my time as a data scientist wasn't spent coding up learning algorithms from scratch or doing image classification. It was spent learning about the problem domain, combing through disparate datasets, engineering relevant features, and building reproducible ML pipelines that made it easy to deploy and operationalize models. I spent far more time configuring distributed data processing systems like Spark than I did building neural nets.

What I realized was that a mismatch exists between the internet and reality when it comes to what data scientists work on. Reality is much closer to the figure above. The

"ML code" is a small piece of the puzzle. The vast majority of code in an ML system is NOT dedicated to machine learning; it's dedicated to all of the systems that support the machine learning. And while this difference might seem insignificant, it's led to a glut of junior data scientists entering the job market with unrealistic sets of expectations of what the job entails.

This insight led me to start MLinProduction.com. The site is dedicated to helping data scientists, machine learning engineers, and other ML practitioners build, deploy, and run real world machine learning systems in production. MLinProduction.com started as a weekly newsletter where I hand select and share the best content I discover on topics like ML product management, model deployment, MLOps. But it has since evolved into a blog where I create and share content focused on the tools, patterns, platforms, and systems that make the ML community productive, efficient, and effective. I've written about tools like Sacred for storing model metadata, platforms like AWS SageMaker and Kubernetes, and how to know when to retrain your model. The blog and newsletter also provide a platform for me to write about and share the knowledge and experience I've collected in my career - something that I think's missing from a lot of the tutorial-style online content out there.

The newsletter has been a great way to learn about the topics and content that's especially relevant to the ML engineers and data scientists in my community. To prepare this book, I reviewed each issue from 2019 and analyzed the viewership data to learn which blog posts, journal articles, and case studies readers found most valuable. These have been organized into 5 categories: data science product management & process, metrics & evaluation, model deployment & operations, ML platforms, and company use-cases. Each chapter can be read completely independently of other

chapters. It's my hope that the type of content in this book continues to proliferate and enrich the ML community in the future.

Without further ado, enjoy the best of MLinProduction.com of 2019.

3 Data Science Product Management & Process

As a team leader in an organization I spend a lot of time thinking about the *process of data science*. That is, how do you effectively utilize data science and machine learning to solve real business problems. Data science process isn't about which model to use or what features to engineer. Instead, the focus is on ensuring that you and your team are aligned with stakeholders by solving the most valuable problems as efficiently as possible.

A key part of solving the most valuable problems is identifying the right problems to solve. It may sound obvious, but this should happen *before* you start building models. Data scientists and machine learning engineers should be working with product teams to clearly frame the problem that needs to be solved. Once the problem has been articulated and agreed upon, you should define success. Maybe success means higher mobile app engagement or more accurate financial forecasts. Regardless of how you define success, it's important to have the business problem and a successful outcome explicitly spelled out to help guide you in the right direction.

Here are 5 steps you can take to ensure that your ML efforts are properly aligned with the business outcomes you wish to achieve.

1. **Determine the desired business outcomes** - Before beginning any technical work, it's imperative that you have a clear understanding of what the business wishes to achieve. One way to think about this is to imagine how the business would be different after the successful implementation of a project. Does the business wish to reduce customer churn? Is a department seeking to eliminate a manual task? Each of these are examples of outcomes that can be achieved

through the application of machine learning. Neglecting this step by rushing into the model building process can lead to a great deal of time, effort, and resources put into building solutions that address the wrong questions. Clearly defining the desired business goals at the outset of an ML project provides a north star to follow.

2. **Define the success criteria** - Once the desired business outcomes are understood, you need to define the measures of success that will track your progress towards those goals. These metrics will guarantee that your team's work is moving the business towards achieving its goals. The best metrics are specific and measurable. Specific metrics reduce ambiguity and increase focus. And the easier it is to measure your success, the more certain you can be that you're achieving the right results.
3. **Translate the business success criteria into a machine learning metric** - After defining the business objectives and success criteria, it's time to define the successful outcome in technical terms. For machine learning projects, this means selecting and optimizing an ML metric that directly relates to the business metrics. Note that selecting an out-of-the-box metric like accuracy or AUROC may not be good enough. Optimizing the wrong metric can lead to a model that looks great from the perspective of accuracy but still misses the most costly transactions.
4. **Establish a baseline** - Regardless of the metric you decide to optimize, it's necessary to establish a baseline measure of performance. This baseline lets you judge the rate of return of increasing the complexity of your modeling solution.
5. **Deploy an MVP quickly and begin iterating immediately** - In his Rules of Ma-

chine Learning, Google Research Scientist Martin Zinkevich stresses that developers should *"Keep the first model simple and get the infrastructure right"* (Rule #4). Deprioritizing modeling gains allows machine learning engineers to sort out the numerous infrastructure issues that need to be solved before the benefits of machine learning models can be fully realized. The simple model provides baseline metrics and behavior to compare against, and a reliable pipeline gives you the ability to test more complex models.

In order for ML projects to have a massive business impact practitioners need to relentlessly focus on asking the right questions. Data scientists must understand the business problem as specifically as possible before diving into technical details and implementation. This involves speaking with stakeholders and translating their needs into technical requirements. Once that's done, data scientists should integrate their models into products as soon as possible and get feedback from customers. This feedback loop will help guarantee that ML efforts are driving business success.

- Managing Machine Learning Projects - An extensive paper describing Amazon's best practices for managing machine learning projects. It describes how ML projects can fit within an economic framework and offers several techniques like risk scorecards and incremental investment approaches for mitigating the risk associated with machine learning projects. My favorite sections discuss the dichotomy between research and production ML, how to document the data catalogue and pipeline of a project, and assessing the economic value of a project. A must read for anyone managing data science projects in production.
- Building machine learning products: a problem well-defined is a problem half-solved. - How do you develop the requirements for a machine learning project

when you're given a vague problem to solve? This post blends ideas from product development, design, and user experience testing to describe a process for turning a vague idea into a concrete ML product. The author imagines designing a model to organize users' photos to illustrate his ideas. Excellent read about how to create machine learning products filled with resources for blending AI and design.

- Data Science Best Practices - A startup shares its template for running an efficient machine learning infrastructure and team. Their guideline is broken down into 4 goals: all new data scientists will build, train, and deploy production models within their first week, automate the automatable and use humans for the rest, deploy models incrementally and often, end users will never notice a model change unless its an improvement. I especially enjoyed the section on interviewing new data scientists and deploying models in "dark mode" for testing.
- So You're Going to Manage a Data Science Team - Fantastic piece from a Data Science leader on the *people and processes* required to building a data team that works. The author drives home the point that managing a team has very little to do with hands on tasks such as feature engineering and model evaluation. Instead, leaders should focus on taking what your team produces and turning it into repeatable, measurable processes. The majority of your time should be spent figuring out how make your data and models available to the rest of your company.
- The Product/Data Fit Strategy - Data product strategy is about building products that rely on data and analytics to generate business value. Understanding the value generated by machine learning involves weighing the costs of training

accurate models against the value generated from correct predictions. This post does a great job of describing how to increase the returns from your investments in machine learning.

- The power of the full-stack data science generalist and the perils of division of labor through function - This article argues that *"data science roles need to be made more general, with broad responsibilities agnostic to technical function."* The author argues that specialization in data science increases coordination costs between teammates, extends wait-times between work, and results in less rewarding work for employees. *"By contrast, generalist roles provide all the things that drive job satisfaction: autonomy, mastery, and purpose."*

4 Metrics & Evaluation

When you begin a new machine learning project, one of the most important choices you face is which evaluation metric to optimize. Plenty has already been written comparing different classification and regression metrics, but most of this content focuses on technical aspects of different metrics. While the technical details are important, it's critical to consider the choice of metric from a product perspective.

What do I mean by that? As Xavier Amatrain puts it, *"The value of a machine learning model is the value it brings to the product."* Before deciding on which ML metric to optimize, you should define what it means for the product to succeed. Maybe this means increased user engagement or increased customer retention. The machine learning metric you choose to optimize should be correlated to the product metric that tracks your progress towards product success.

After choosing the metric you need to determine an acceptable level of predictive performance to achieve before deploying the model to production. To do this you have to understand the ***cost of making an error*** with that model.

When errors are inexpensive, the threshold for predictive performance doesn't need to be very high. If Amazon recommends you a product you don't need you might get a bit annoyed, but the recommendation doesn't drastically reduce your experience. Plus negative user experiences from less-than-perfect models can be mitigated through enhancements to the user interface. For instance, rather than automatically fill your cart with its recommendations, Amazon just *suggests* products you might like. And these suggestions are often presented in unobtrusive ways, such as in sidebars or towards the bottom of your screen. When the cost of a model error is low, other metrics

might warrant deploying the model much sooner. If a model achieves 60-70% accuracy but simplifies a burdensome, but non-critical, operational task, it may make sense to deploy that model sooner.

The story is very different when the cost of a model error is high. Self-driving cars provide an illustrative example. The success of autonomous vehicles is going to depend in large part on legal policy and public perception. One death from an autonomous vehicle generates more negative press than a thousand deaths from human drunk driving. Performing statistically better than humans is just not enough. The medical field has similar requirements. A false negative from a cancer detection model could result in death, leading to heartache for a patient's family and malpractice suits for doctors. When errors are expensive, models need to be extremely performant.

But how can you be sure that your model will maximize business impact? Building a model that maximizes AUC doesn't guarantee that model will maximize business results. Suppose your company seeks to increase user engagement on its mobile application. The mobile app currently recommends content to users based on global metrics such as most viewed/clicked within the last 7 days, but the Product team hypothesizes that recommending more personalized content will increase engagement. So the data science team prototypes a recommender system, validates the model on historical data, and deploys it as a microservice that the mobile app can call to generate recommendations. Time to pop the champagne, right?

Well, no. Before deploying this model to all users, we need to determine whether the new recommendations produce the outcome we desire, namely, increased user engagement. The right way to do this is to run an A/B test on a subset of the user population. We serve one group of users the original heuristic-based recommendations

while another group of users see the recommendations from the new model. After some observation period, we determine whether the new recommendation system leads to higher user engagement according to a metric such as mean time spent on the mobile app. If higher engagement is observed, then Product may decide to roll out the model to a larger group of users or even to deploy it across all users. If engagement doesn't increase, then the data scientists can iterate on the model and run another online experiment in the future.

Implementing such a system involves several components including an A/B testing framework and a mechanism to run and track multiple models in production. While there's a wealth of information available on offline model evaluation, much less has been published on online model evaluation. I suspect this will change as more companies deploy models tasked with serving predictions to users in real time.

Below are some excellent resources for learning about both online and offline model evaluation.

- Product Driven Machine Learning (and Parking Tickets in NYC) - To derive business value from machine learning models, business goals have to be framed in statistical terms. The product manager defines business metrics and business goals. Data scientists then translate these to modeling metrics and modeling goals. The post includes a very illustrative use-case using NYC parking tickets.
- 150 Successful Machine Learning Models: 6 Lessons Learned at Booking.com - This fantastic paper contains a large scale study of the impact of machine learning across many different products at Booking.com and distills the learnings into 6 specific lessons covering all phases of a machine learning project (Inception, Modeling, Deployment, Monitoring and Evaluation). The authors list the chal-

lenges faced while building models and describe the techniques they used to address each challenge. Two of my favorite lessons from the paper: 1) They found that offline metrics are poorly correlated to business gains and 2) the same ML problem can be formulated in several ways, but often certain subsets of these are easier to solve.

- Predictive Model Performance: Offline and Online Evaluations - Offline evaluation metrics like AUC are used to estimate a model's predictive performance in offline settings. But sometimes there are substantial differences between the offline and online performance of a model. This fantastic paper discusses the shortcomings of offline evaluation and proposes a method to simulate the online experience using historical data in the context of online advertising.
- Training models with unequal economic error costs using Amazon SageMaker - It's extremely important to understand the impact of different types of errors when building classification models. In many applications, one kind of error can be much more consequential than another. This excellent blog post demonstrates how to use a custom cost function to incorporate the cost of different types of errors in the model training process. The advantage of this approach is that it explicitly links a model's outcomes to the business framework for decision-making.
- Modeling conversion rates and saving millions of dollars using Kaplan-Meier and gamma distributions - Computing conversion rates is often straightforward, but is complicated when there's a delay until the conversion event. For example, suppose it takes weeks or months from when a new lead enters your system to when that lead makes a purchase. This post discusses how to model conversion

rates in such situations and introduces an open source library giving you the ability to forecast future conversion rates for new leads.

- Driving Business Decisions Using Data Science and Machine Learning - LinkedIn discusses how they connect machine learning to their business challenges by answering two questions: how do you determine the right metric (KPI) for a business goal and how do you test out a new feature on the site to make business decisions? Answering the first question often involves translating fuzzy business questions such as “what is a highly engaged customer” into simple and interpretable metrics that can be analyzed rigorously with data science. The second question involves a highly robust data pipeline solution that enables A/B testing, model monitoring, and the ability to inspect and explain model predictions.

5 Model Deployment & Operations

One question data scientists often ask me is how to deploy trained models so that other users can generate predictions. This topic is rarely discussed when machine learning is taught. Boot camps, data science graduate programs, and online courses tend to focus on training procedures and algorithms because these are "core" machine learning ideas. I don't disagree that those topics are important, but I believe that a data scientist who can't deploy a model can't add value to a business.

There are essentially 3 different ways to handle model deployment.

The first way is to package the model directly with an application. Consider the Uber Eats mobile app. When a customer orders food through Uber Eats, the app displays the estimated time-to-delivery of the order. The model generating that prediction could be packaged directly with the Uber mobile app or web application. While this is possible, it's highly unlikely. One reason is that if a model is tightly coupled with an application updating or retrain the model is quite difficult. Another is that a tightly coupled architecture makes reusing using the same model across different applications very hard.

A better approach is to expose the model as a microservice. This means deploying a REST API that accepts incoming requests and returns predictions. The implementation of the API is responsible for preprocessing incoming data, joining these inputs to other feature sets, and generating a prediction. Updating a model with this approach is simple: train the model, deploy another microservice, and point traffic to the new version. Model reuse is also simple because the model is completely decoupled from any application.

The third approach involves generating predictions in batch on a recurring schedule. Batch prediction typically requires a job scheduler to trigger inference jobs. An inference job begins by retrieving input data from a database and preparing the data for prediction. A model is retrieved from a model store, deserialized, and then used to predict the dataset. These predictions are stored in a database where they become available to any users or services with priveleged access to that database. Rather than query the model, users retrieve precomputed predictions from the database.

To determine which strategy is right for your situation, you need to answer two questions. First, how often will the model be queried in production? Is the model expected to deliver predictions to users in real time at random intervals? Or will the model be queried on a recurring schedule? Second, how many samples will the model handle in a single query? Will the model generate predictions for a single sample at a time? A subset of samples? Or both? Answering both of these questions determines the *infrastructure* necessary for your deployment as well as the *interface* the logic supporting your model should implement.

After deployment ML engineers should focus on model operations. Models can fail in many different and unpredictable ways, which can lead to dramatic problems for end users and businesses. For instance, ML practitioners must implement specialized software to detect *silent failures* - errors that occur when model training and inference processes operate on inaccurate or stale data without throwing exceptions. Although these errors don't show up in normal applications logs, silent failures cause seriously flawed model outputs which result in unhappy customers and lost revenue. How do you prevent silent failures? The answer is thorough logging, robust monitoring, and reliable data validation. The goal of these operational topics is to detect issues before

they reach your end users.

Check out the following links below to learn more about machine learning deployment and operations.

- Continuous Delivery for Machine Learning - An incredible post that walks readers through the technical components of a continuous delivery for machine learning (CD4ML) pipeline. The authors describe CD4ML as *"a software engineering approach in which a cross-functional team produces machine learning applications based on code, data, and models in small and safe increments that can be reproduced and reliably released at any time, in short adaptation cycles."* Using a sample ML application, the authors explain the concepts and demonstrate how different tools can be used together to implement the full end-to-end process including model and dataset versioning, testing, and deployment.
- How to Deploy Machine Learning Models: A Guide - This post provides a fantastic summary of the challenges involved in deploying and maintaining machine learning systems. The author dives into what makes machine learning systems hard, different systems architectures, and key principles to keep in mind when designing an ML system. He also provides an overview of useful tooling and hints at useful testing strategies for ML systems.
- Machine learning requires a fundamentally different deployment approach - Most of what we know about "production" software comes from web apps running ecommerce and social media apps at scale. But ML applications differ from traditional software; ML systems must be evaluated against metrics rather than strict specifications. Doing this well requires rethinking ideas like version control and testing. To quote the author: *"The biggest issue facing machine learning isn't*

whether we will discover better algorithms . . . [it's] how we'll put ML systems into production."

- Models and microservices should be running on the same continuous delivery stack - According to this engineer, the process for deploying models to production looks strikingly similar to the continuous deployment process for microservices. For instance, model deployment involves a build process (model training triggered by new data or a code change), unit testing (evaluating a trained model against holdout data), and a measurement of online production metrics. Since these processes resemble one another, he argues, engineers should seek to leverage existing devops infrastructure for ML problems.
- On Being Model-driven: Metrics and Monitoring - A trained model's predictive performance is expected to decline over time after being deployed to production. Monitoring model metrics helps identify this drift - but what metrics should you capture? This post introduces what metrics to monitor and why. P.S. I did some research to find the slide deck referenced in the article.
- Machine Learning Models Monitoring Architectures - Continuing with the monitoring theme, this post presents 3 system architectures for monitoring models and applying anomaly and drift detection to the streams. The post compares and contrasts the different architectures and includes high-level diagrams. Check out the follow up post for implementation details.

6 Machine Learning Platforms

Over the last several years a number of machine learning platforms have emerged. These platforms seek to centralize and commoditize commonly performed tasks such as experiment tracking, model deployment, and the provisioning of compute resources. While large companies such as Uber, Google, Facebook, and Airbnb have built internal platforms for these tasks, vendors offering ML-Platform-as-a-Service (MLPaaS) solutions have emerged. Data science leaders who have understood the importance of platforms are now deciding whether to build their own ML platforms or buy a vendor offering. For many of these leaders this isn't an easy decision to make.

Although the MLPaaS space is fairly new, there's already a large number of platform solutions to choose from. How do you choose among these different platforms and companies? Building a platform may seem like a viable option, especially if you have sufficient head count and you're willing to adapt various open source components for your needs. But building a custom solution is time consuming and potentially very expensive.

Since I've been on teams that have built custom ML platforms and on teams that have bought vendor solutions, I thought it might be useful to outline a useful decision making progress. The most important question to ask yourself is: *"Does building an ML platform provide [your company] differentiated value?"*

If the answer to this question is *no*, then you should definitely buy. Building a platform is an extensive and expensive process. Recent estimates I've heard from teams who've built their own platform say it takes at least 5-10 engineers and 1-2 years worth of effort. In my experience, you can build something basic in much less time, but the cost of

supporting and maintaining the platform will remain high for the remainder of that platform's lifetime. If the headcount and timing estimates aren't daunting enough, consider the *opportunity cost* of the project. Engineering resources spent building commoditized software won't be utilized building more valuable projects.

If you decide to buy, the very next question is what platform should you buy? How do you choose between the different vendors out there? My recommendation: before you look at vendors, determine exactly what functionality is important for your data science team. Formalizing these requirements into a set of acceptance criteria you can use to objectively evaluate different vendors. For example, is experiment tracking something your team needs? Or do you care more about automating deployment? What about A/B testing deployed models? The more extensive and detailed the criteria, the more efficient and effective your search through vendor-space will be.

If building a platform does provide differentiated value, ask yourself if you have the time and resources. If the resources are available, then building a platform might be the right solution for you. I still recommend a vendor review, though you might spend much less time on it by only reviewing the top N vendors. If you're determined to build a platform, here are two suggestions. First, learn about what other companies have built and what their experiences have been. Second, spend time looking through open source components. You may not need to build everything from scratch.

To help get you started, here is some of the best content I've found on building ML platforms and components. And if you're interested in hearing more about my experience building and buying ML platforms, shoot me an email at luigi@mlinproduction.com.

- Meet Michelangelo: Uber's Machine Learning Platform - If you haven't yet come

across Uber's ML-as-a-service platform, Michelangelo, you're overdo. In its introductory blog post on the tool, Uber's team describes the motivation and architecture of the end-to-end system and how it powers their ML models. A really comprehensive post that's worth a read. And while you're at it check out their follow-up piece, Scaling Machine Learning at Uber with Michelangelo as well.

- Bighead: Airbnb's End-to-End ML Platform - Bighead is airbnb's end-to-end platform for building and deploying machine learning models to production. It's a collection of tools built around open source technologies for handling all parts of the ML in production stack including model management, feature engineering, online inference, and experimentation. The slides do a great job of describing how Airbnb largely relies on Docker for reproducibility and scalability.
- Productionizing ML with workflows at Twitter - Cortex, Twitter's machine learning engineering team, describes how and why they developed ML Workflows, a tool designed to automate, schedule, and share machine learning pipelines. Before ML Workflows, data scientists at Twitter struggled to manage machine learning pipelines for processes like model retraining and hyperparameter optimization. After its adoption, teams have seen the time for retraining and deployment drop down from four weeks to one week.
- Overton: A Data System for Monitoring and Improving Machine-Learned Products - Overton is an ML platform out of Apple that reimagines the way engineers interact with and build production machine learning systems. Overton automates model construction, deployment, and monitoring by having engineers focus on supervision and data without writing any code in frameworks like Tensorflow

or PyTorch. Rather than manipulate code, engineers using Overton manipulate data files that specify tasks. This feels like a groundbreaking approach to me!

- Accelerating the Machine Learning Lifecycle with MLflow - MLflow is an open source ML platform that encourages data scientists to use the latest models, libraries, and frameworks while still promoting reproducibility and deployability. It does this through three main components that can be used together or separately: Tracking, Projects, and Models. These provide an API for experiment tracking, a format for packaging code into reusable projects, and a generic format for packaging and deploying models.
- Kubeflow - Kubeflow is an ML toolkit dedicated to making the deployment of machine learning on Kubernetes simple, portable, and scalable. The goal of the project is to simplify the machine learning deployment process by leveraging kubernetes to deploy across diverse infrastructure and scale with demand. The KFServing component enables serverless inference on Kubernetes and provides a custom resource definition for serving models for frameworks like Tensorflow, XGBoost, sklearn, and PyTorch. Check out this tutorial on using kubeflow to deploy an XGBoost model and view other examples here.

7 Company Use Cases

Experience has taught me that the best way to learn a new topic is through action. Whenever I want to learn a new skill, like surfing or martial arts, I start doing that new thing as soon as possible. This is much more effective than reading books or listening to podcasts about the topic. Often this is pretty simple - learning a new martial art like Muay Thai just means finding a good Muay Thai gym.

But what if you want to learn how to build and serve machine learning models to 100 million users? Your best bet in this case is to find someone who's done the work and ask them as many questions as possible. While learning from someone who's "done the job" is a great way to learn, it's not the only way. So how do you learn how to deploy and scale ML models if you don't know anyone who's done it before, and if there's a scarcity of resources and best practices on the topic?

One way is to take an existing ML product and try to reverse engineer it. You don't need to code up the entire system or even train a model. I like to start by describing the system architecture at a high level. Think about the constraints that might be involved. What target should the model predict? How quickly must predictions be returned to a user? What input data is available at runtime? What kind of historical data is available? What sorts of features can we build?

As an example, consider Uber Eats' estimated-time-to-delivery feature. Each time a user orders food through the app, a machine learning model estimates the delivery time of the order based on data including the time of day, the customer's location, and the average time needed to prepare a meal at the restaurant. This feature requires significant investment from various teams:

- **Machine Learning** - Data scientists build models to predict the duration of a complex multi-step process. Features must encode information related to the complexity of the meal being prepared, the load the restaurant is currently facing, and the time required for a driver to be dispatched, stop to pick up the food, and drive to the final delivery location.
- **Data Infrastructure** - The model's input features are generated from historical data as well as real time information. The historical data is probably stored in multiple disparate data systems, so it must be possible to query and retrieve the necessary information within an acceptable latency. The real time data also needs to be available to the models, so a set of microservices to surface these data is likely necessary.
- **Product** - The predictions must be returned to millions of users around the world. These users access the app across different devices. Ideally, the predictions should be returned to users within 1-2 seconds of completing an order.

Although we haven't written any code, this thought experiment forces us to think about the end-to-end pipeline required to deploy this machine learning feature. Luckily many companies have written extensively about their ML models and pipelines. As you read the following posts, actively think about the efforts involved in bringing these machine learning products to production.

- Maintainable ETLs: Tips for Making Your Pipelines Easier to Support and Extend - While writing ETLs is not the most enjoyable part of a data scientist's day, this Stitch Fix post explains, a poorly constructed ETL will create obstacles to building new models and deriving insights. The author offers 4 suggestions for writing better ETLs: build a chain of simple tasks, use a workflow management tool,

leverage SQL where possible, and implement data quality checks. The less you currently care about ETLs and data pipelines, the more I suggest you read this article.

- Applying Deep Learning to AirBnB Search - I know what you're thinking: "A journal paper on deep learning?! I thought we were here to talk about machine learning in production…" I've included this paper because it details the lessons Airbnb learned by building deep learning solutions. Rather than focus on architectures, the authors describe how they iterated towards their current solutions, the common deep learning tricks they tried that failed, and the features they engineered to make their models work. A wonderful **applied** deep learning paper.
- How Uber Organizes Around Machine Learning - This is a summary of several posts from Uber's engineering blog related to machine learning and AI. I enjoyed this piece because it summarizes the major applications of machine learning at Uber and then describes how Uber leverages ML through technology, process, and organizational structure.
- Scaling Machine Learning Productivity at LinkedIn - LinkedIn started the "Productive Machine Learning" (ProML) initiative to increase the effectiveness of their machine learning engineers and democratize their AI tools across the company. They broke their efforts into 6 layers: exploration, training, deployment, running (operational), health assurance, and a feature marketplace. This provides an interesting discussion of what it means to scale a machine learning practice as the impact of ML increases within a company. The model starts with a specialized team of high density of ML knowledge to then building services and tools that empower other engineers by "baking-in" best practices.

- System Architectures for Personalization and Recommendation - Building machine learning products like Netflix's recommender system requires a software architecture that handles large volumes of existing data, is responsive to user interactions, and makes it easy to experiment with new recommendation approaches. The Netflix architecture combines online computation that responds quickly to events and uses the most recent data, with offline computation, which allows for more choices in algorithmic approach and has less limitations on the amount of data that can be used. Combining these approaches allows for flexibility - new approaches can be developed and easily substituted.
- Building Lyft's Marketing Automation Platform - Lyft describes Symphony, an orchestration system that takes a business objective, predicts future user value, allocates marketing budgets, and publishes that budget to drive new users to Lyft. Since acquiring news users through different channels like search and paid social media involves making thousands of decisions each day, their growth team decided to use machine learning to automate many of these decisions by forecasting customer lifetime value, allocating budget based on customer LTV, and then adjusting bidding strategies accordingly. As a machine learning practitioner who works on marketing problems I really enjoyed this piece and hope Lyft decides to publish more on the topic!

8 Conclusion

One of my goals in writing this book is to educate others on the challenges and opportunities facing teams that are building and operating real world production systems. The challenges are much broader than what type of training algorithm or neural network architecture to use. In fact, machine learning in production requires a broad range of skill sets from statistics, software engineering, devops, and product management. The field is incredibly dynamic and there are all kinds of interesting technical challenges to be solved. I've sought to provide a taste of these challenges and introduce some of the solutions built so far. But we're very much in the early days of production machine learning systems. I'd like to leave you with several challenges and open problems facing machine learning teams:

1. **Feature Stores** - Numerous companies have emerged in the ML platform space but I haven't yet seen vendors selling feature stores. One of the main challenges is that data infrastructure varies widely from company to company. This makes it hard to build a solution that works across teams. But every ML team has to reliably compute predictive features, so feature-store-as-a-service solutions would be valuable.
2. **Model Deployment** - Model deployment is more complicated than wrapping a model in a Flask API. For example, how should you set up logging and what should you log? What telemetry do you collect during batch inference? Setting this up today is manual work and requires web development knowledge.
3. **Concept Drift Detection** - As soon as a model is deployed its predictive performance begins to decline. This is known as concept drift, the idea that the input

data a model sees in production will diverge from the data originally used for training. Automatically detecting concept drift helps data scientists ensure that predictive performance remains adequate.

4. **Model Monitoring** - Detecting concept drift is just one example of model monitoring. Like any software application, ML systems need to be monitored so that runtime issues can be discovered and fixed. However these systems require an entirely new set of metrics to be monitored. While some of these metrics, like the the distributions of the input data and predictions, are consistent across ML projects, others are more specialized and need to be determined on a case-by-case basis.
5. **Online Evaluation of ML Models** - A model that optimizes an offline evaluation metric like AUC isn't guaranteed to maximize the metric a business cares about. For example, a recommender system might achieve high marks on an offline dataset but could fail to drive important key performance indicators (KPIs). Such models need to be evaluated in an online fashion through mechanisms like A/B testing.
6. **Evaluation Schema Design** - Most data scientists are familiar with evaluation metrics like accuracy and RMSE. While these metrics are easy to understand and compute, they don't factor in the costs of specific business errors. For instance, accuracy treats false positives and false negatives equally, but these types of errors usually have different costs. Crafting evaluation metrics that explicitly model these costs will become more common.
7. **Model Governance** - Governance includes understanding who is authorized to access a model, storing metadata that describes when models were trained,

tested, and deployed, model versioning, who is responsible for a model, and monitoring. Special governnce considerations need to be taken in highly regulated industries such as finance and healthcare. As models become more central to a company's operations, they will need to be treated as important assets similar to their datasets and codebases.

8. **Explainability** - Sometimes it's not enough for a model to be accurate - users also want to know **why** a model generates a prediction. This is easy with simple models like linear regression but is more difficult when using black-box algorithms. Researchers have developed a suite of tools to help explain how complex models generate predictions, but it's not clear how useful these diagnostics are.

I look forward to learning about your solutions to these challenges and sharing your work in a future issue of the newsletter!

9 Did you enjoy the guide?

Then please do me a favor and share it on LinkedIn (@ML in Production), Twitter (@MLinProduction), or your social media of choice. Your friends will appreciate it too.

Share on Twitter.