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.

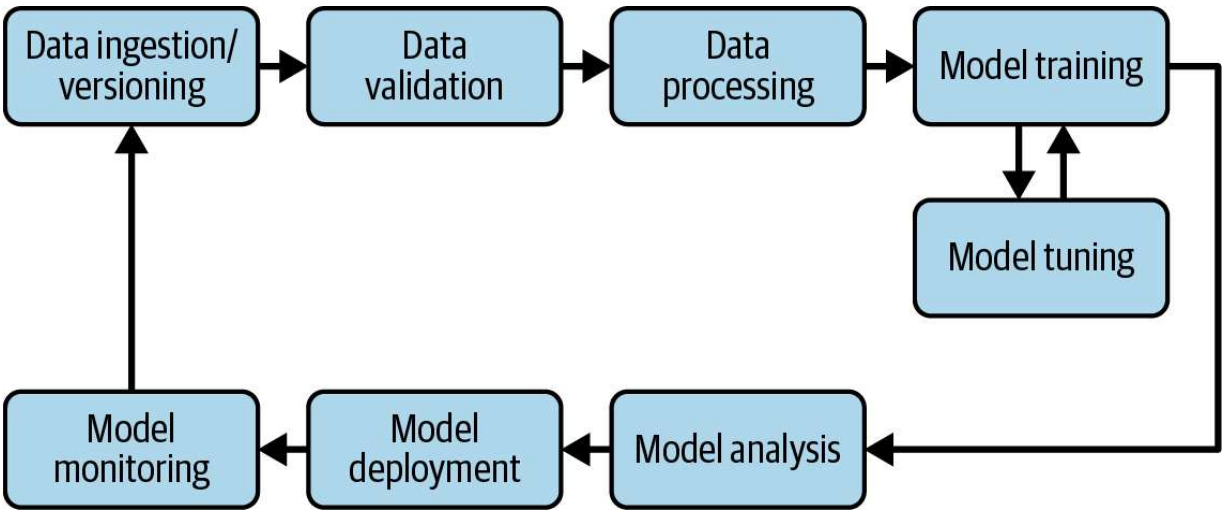

Figure 1-1. The steps in an ML pipeline

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.