The Next Evolutionary Step In

Model Performance Management

Machine learning troubleshooting is painful and time-consuming today, but it doesn't have to be. This paper charts the evolution that ML teams go through—from no monitoring to monitoring to full-stack ML observability—and offers a modernization blueprint for teams to implement ML performance tracing to solve problems faster.

arize

Table of contents

Part One:

From No Monitoring To Monitoring

Introduction: The Pain Is Real

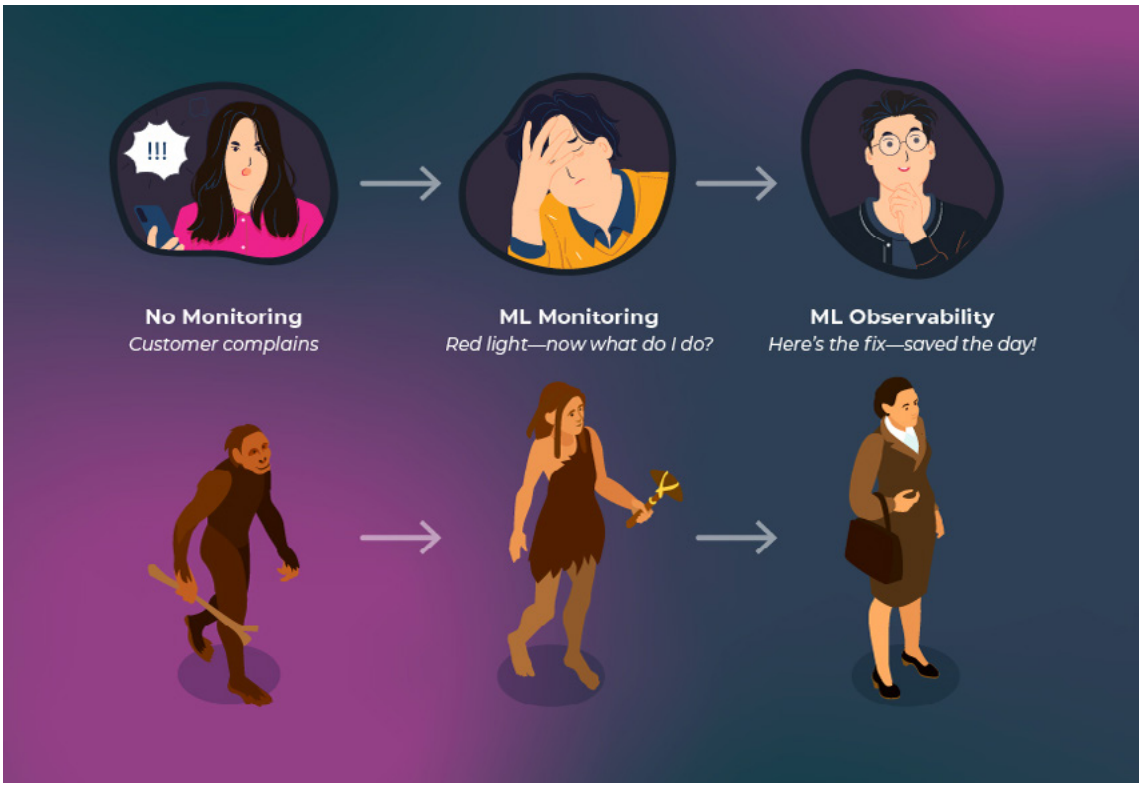
To paraphrase a common bit of wisdom, if a machine learning model runs in production and no one is complaining, does it mean the model is perfect? The unfortunate truth is that production models are usually left alone unless they lead to negative business impacts.

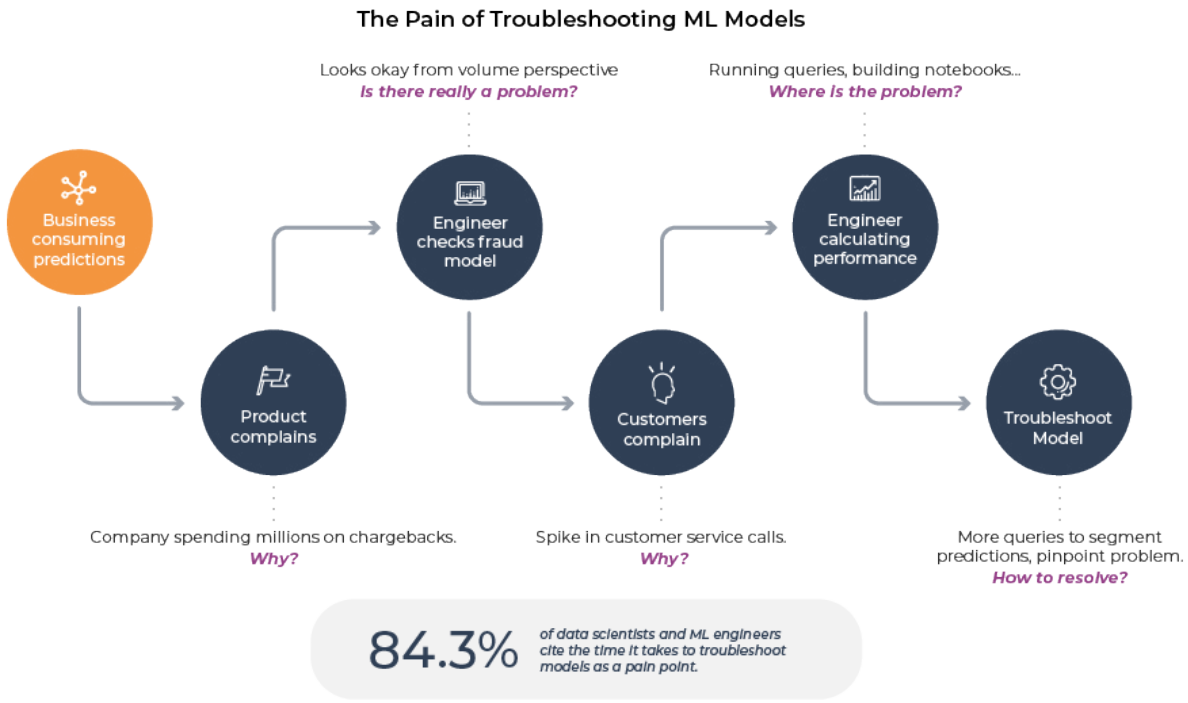
Let's look at an example of what may happen today:

As a machine learning engineer (MLE) for a fintech company, you maintain a fraud-detection model. It has been in production for a week, and you are enjoying your morning coffee when a product manager (PM) urgently complains that the customer support team has seen a significant increase in calls complaining about fraudulent transactions.

This costs the company a fortune in chargeback transactions. The company is spending tens of thousands of dollars every hour, and you have to fix it now.

Gulp. Is it your model? Software engineers tell you that the problem is not on their end.





You write a custom query to pull data from logs of the last million predictions that your model has made in the past three days. The query takes some time to run, you export the data, do some minimal preprocessing, import it into a Jupyter notebook, and eventually start calculating relevant metrics for the sample data you pulled.

There doesn't seem to be a problem in the overall data. Your PM and customers are still complaining, but all you see is maybe a slight increase in fraudulent activity.

More metrics, more analysis, more conversations with others. There's something going on, it's just not obvious. So you start digging through the data to find a common pattern on the fraud transactions that the model is missing. You're writing ad-hoc scripts to slice into the data.

This takes days or weeks of all-consuming effort. Everything else you were working on is now on pause until this issue is resolved because: 1) you know the model the best; and 2) every bad prediction is costing the company revenue.

Eventually you see something odd. If you slice by geographies, California seems to be performing somewhat worse than it did a few days ago. You filter to California and realize some of the merchant IDs belong to scam merchants that your model did not pick up. You retrain your model on these new merchants and save the day.

This example helps us see what it takes to troubleshoot a machine learning model today. It is many times more complex than troubleshooting traditional software. We are shipping AI blind.

There are many monitoring tools and techniques for traditional software engineering—things like Datadog and New Relic—that automatically surface performance problems. But what does monitoring look like for machine learning models?

Machine Learning Performance Monitoring

First, let's make sure we have a definition of what monitoring is: monitoring, at the most basic level, is data about how your systems are performing; it requires that data are made storable, accessible, and displayable in some reasonable way.

What Data Is Needed to Do Performance Monitoring?

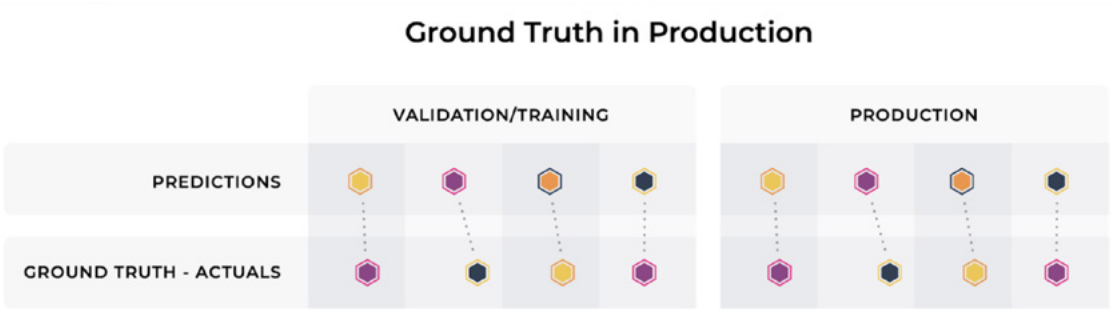
To monitor machine learning models' performance, you must begin with a **prediction** and **actual**.

A model has to make some predictions. This can be predicting the estimated time of arrival (ETA) of when the ride is going to arrive in a ride-sharing app. It can also be what loan amount to give a certain person. A model can predict if it will rain on Thursday. At a fundamental level, this is what machine learning systems do: they use data to make a prediction.

Since what you want is to predict the real world, and you want that prediction to be accurate, it is also useful to look at actuals (also known as ground truth). An actual is the right answer — it is what actually happened in the real world. Your ride arrived in five minutes, or it did rain on Thursday. Without comparison to the actuals, it is very difficult to quantify how the model is performing until your customers complain.

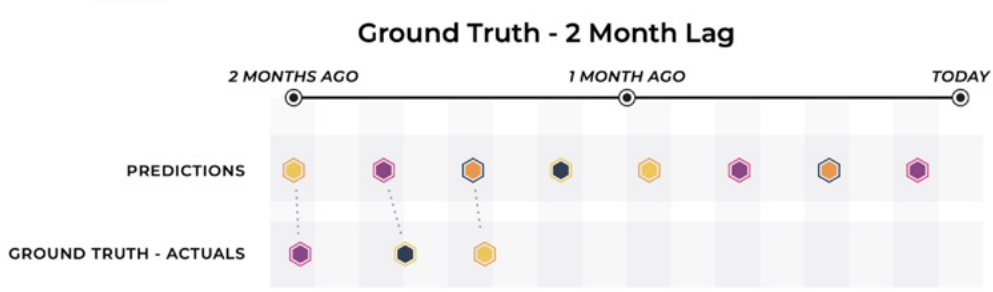
But getting the actuals is not a trivial endeavor. There are four cases here:

1. **Quick Actuals**: In the easiest case, actuals are surfaced to you for every prediction, and there is a direct link between predictions and actuals, allowing you to directly analyze the performance of your model in production. This can happen in the case of predicting the ETA of your ride, for example. At some point the ride will arrive, and you will know how long that took and whether the actual time matched your prediction.



2. **Delayed Actuals:** In the diagram below, while we see that actuals for the model are eventually determined, they come too late for the desired analysis.

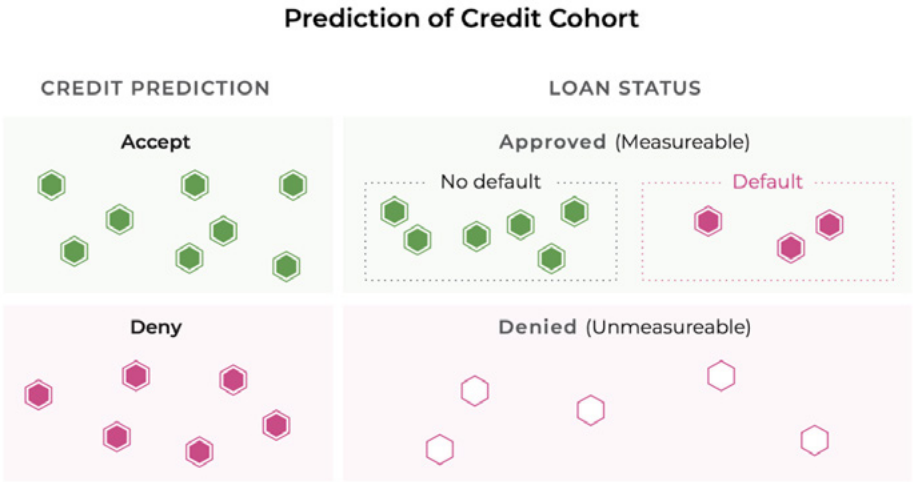
   When this actuals delay is small enough, this scenario doesn't differ too substantially from quick actuals. There is still a reasonable cadence for the model owner to measure performance metrics and update the model accordingly, as one would do in the real-time actuals scenario.



   However, in systems where there is a significant delay in receiving the actuals, teams may need to turn to proxy metrics. Proxy metrics are alternative signals that are correlated with the actuals that you're trying to approximate.

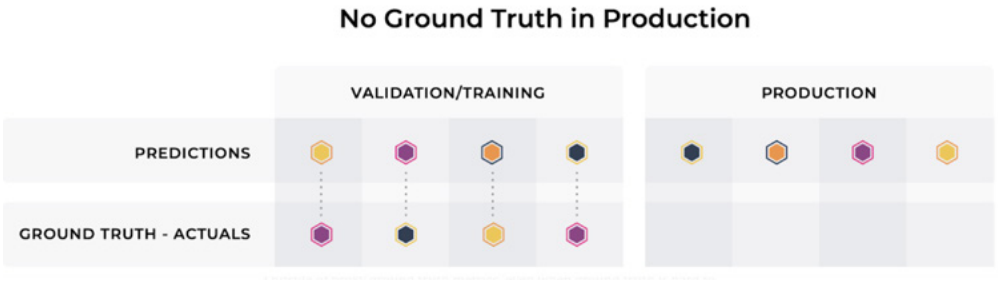
   For example, imagine you are using a model to determine which consumers are most likely to default on their credit card debt. A potential proxy metric in this scenario might be the percentage of customers to whom you have lent credit that make a late payment.

3. **Causal Influence on Actuals (Biased Actuals):** Not all actuals are created equal. In some cases, teams receive real-time actuals but the model's predictions have substantially affected the outcome. To take a lending example, when you decide to give loans to certain applicants, you will receive actuals on those applicants but not those you rejected. You will never know, therefore, whether your model accurately predicted that the rejected applicants would default.



4. **No Actuals:** Having no actuals to connect back to model performance is the worst-case scenario for a modeling team. One way to acquire ground truth data is to hire human annotators or labellers. Monitoring drift in the output predictions can also be used to signal aberrant model behavior even when no actuals are present.

For a deeper dive on these scenarios, see our "Playbook To Monitor Your Model Performance In Production."



Gathering your predictions and your actuals is the first step. But in order to do any meaningful monitoring, you need to have a formula for comparing your predictions and your actuals—you need the right metric.

What Are the Right Metrics for My Model?

The correct metric to monitor for any model depends on your model's use case. Let's look at some examples.

FRAUD

A fraud model is particularly hard to assess with simple measures of accuracy since the dataset is extremely unbalanced (a great majority of transactions are not fraudulent). Instead, we can measure:

- **Recall**, or what portion of fraud examples your model identified that are true positives.
- **False negative rate** measures fraud that a model failed to predict accurately. It is a key performance indicator since it's the most expensive to organizations in terms of direct financial losses, resulting in chargebacks and other stolen funds.
- **False positive rate** — or the rate at which a model predicts fraud for a transaction that is not actually fraudulent — is also important because inconveniencing customers has its own indirect costs, whether it's in healthcare where a patient's claim is denied or in consumer credit where a customer gets delayed buying groceries.

DEMAND FORECASTING

Demand forecasting predicts customer demand over a given time period. For example, an online retailer selling computer cases might need to forecast demand to make sure that they can meet customer needs and not buy too much inventory. Like other time-series forecasting models, it is best described by metrics like ME, MAE, MAPE, and MSE.

- **Mean error (ME)** is average historical error (bias). A positive value signifies an overprediction, while a negative value means underprediction. While mean error isn't typically the loss function that models optimize for in training, the fact that it measures bias is often valuable for monitoring business impact.
- **Mean absolute error (MAE)** is the absolute value difference between a model's predictions and actuals, averaged out across the dataset. It's a great first glance at model performance since it isn't skewed by extreme errors of a few predictions.
- **Mean absolute percentage error (MAPE)** measures the average magnitude of error produced by a model. It's one of the more common metrics of model prediction accuracy.
- **Mean squared error (MSE)** is the difference between the model's predictions and actuals, squared and averaged out across the dataset. MSE is used to check how close the predicted values are to the actual values. As with root mean square error (RMSE), this measure gives higher weight to large errors and therefore may be useful in cases where a business might want to heavily penalize large errors or outliers.

OTHER USE CASES

From click-through rate to lifetime value models, there are many machine learning use cases and associated model metrics. For a deeper dive on model metrics by use case, see this resource hub.

What Are the Right Thresholds?

So now you have your metric, and you're faced with a new problem: how good is good enough? What is a good accuracy rate? Is my false negative rate too high? What is considered a good RMSE?

Absolute measures are very difficult to define. Instead, machine learning practitioners must rely on relative metrics. In particular, you must determine a baseline performance. While you are training the model, your baseline could be an older model you have productized, a state-of-the-art model from literature, or human performance. But once the model is in production, it becomes its own benchmark. If you have a three percent false negative rate on day one and then a 10% false negative rate today, you should wake up your engineers!

Often, initial performance is not what is actually used; instead, you can use a rolling 30-day performance.

When the model shifts significantly (a standard deviation or more), an alert must be triggered. This should be an automated setup based on a baseline dataset so that you can be alerted proactively.

Part Two:

From Monitoring To Full-Stack ML Observability With ML Performance Tracing

With the essentials of model performance monitoring out of the way, many might understandably feel confident in their ability to continuously assess model accuracy and quality.

However, monitoring alone is not enough. To see why, let's revisit the fraud use case from the beginning of the paper. You are enjoying your morning coffee, but this time you have performance monitoring in place. Instead of getting a complaint from a product manager, you get a PagerDuty alert saying that "Fraud model performance declined."

Your product manager, customer support team, and customers are still blissfully unaware of the increase in fraudulent transactions and you are aware of the issue before it has a big impact on the company. The performance metric has crossed the threshold, and you see the red light – but now what? To pinpoint and fix the issue, ML observability is needed.

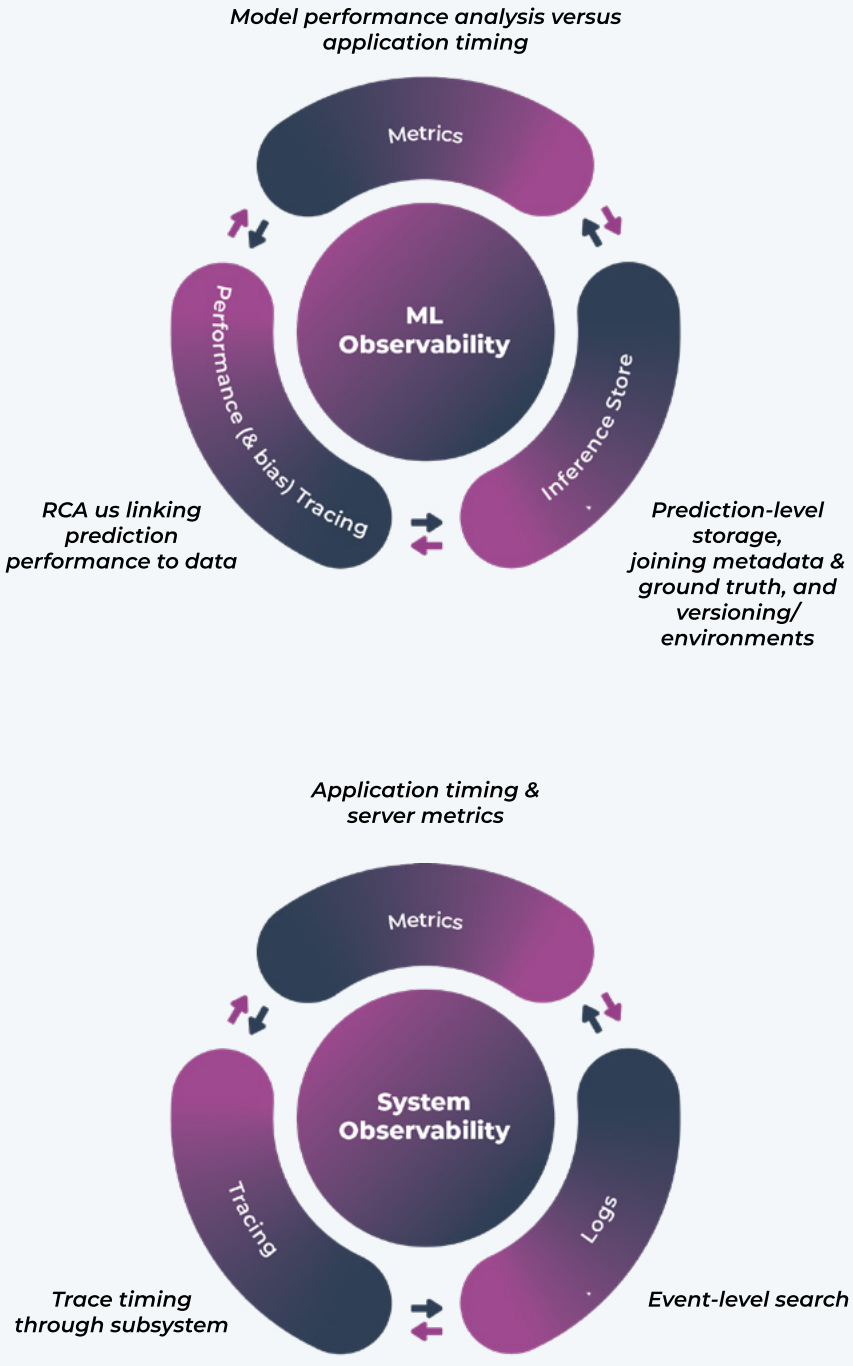
Key Components of Observability

In infrastructure and systems, logs, metrics, and tracing are all key to achieving observability. These components are also critical to achieving ML observability, which is the practice of obtaining a deep understanding into your model's data and performance across its lifecycle.

Key Components of ML Observability

- **Inference Store**: Records of ML prediction events that were logged from the model. These are the raw prediction events that hold granular information about the model's predictions. There are some key differences between what logs in system observability means and an inference store in ML Observability. Will cover this in an upcoming post!
- **Model Metrics**: Calculated metrics on the prediction events to determine overall model health over time—this includes drift, performance, and data quality metrics. These metrics can then be monitored.
- **ML Performance Tracing**: While logs and metrics might be adequate for understanding individual events or aggregate metrics, they rarely provide helpful information when debugging model performance. To troubleshoot model performance, you need another observability technique called ML performance tracing.

This paper covers how to use ML performance tracing for root cause analysis.



Key Components of Observability: Systems versus Machine Learning

System Observability	ML Observability
Logs • Records of an event that happened within an application. • Typically not mutable by an event ID. • Searchable by tags and unstructured indexes.	**Inference Store** • Records of ML prediction events that are logged from the model. • Raw prediction events that hold granular context about the models predictions. • Mutable by prediction ID and dataset.
Metrics • Measured values of system performance • Metrics comprise a set of attributes (ie. value, label, and timestamp) that convey information about SLAs, SLOs, and SLIs.	**Model Metrics** • Calculated metrics on the prediction events. • Provides ways to determine model health over time—this includes drift, performance, and data quality metrics. • Metrics can be monitored. • Metrics can be aggregate or slice-level.
Tracing • Provides context for other components of observability (logs, metrics). • Follows the entire lifecycle of a request or action across distributed systems.	**ML Performance Tracing** • ML performance tracing is the methodology for pinpointing the source of a model performance problem. • Involves mapping back to the data that caused the problem. • Necessarily a distinct discipline because logs and metrics are rarely helpful for debugging model performance.

Introduction to ML Performance Tracing

Definition: What Is ML Performance Tracing?

ML performance tracing is the methodology for pinpointing the source of a model performance problem and mapping back to the underlying data issue causing that problem.

In infrastructure observability, a trace represents the entire journey of a request or action as it moves through all the various nodes of a distributed system. In ML observability, a trace represents the model's performance across datasets and in various slices. It can also trace the model's performance through multiple dependency models to root cause which sub-model is causing the performance degradation. Most teams in industry today are single-model systems, but we see a growing set of model dependency chains.

In both infrastructure and ML observability, by analyzing trace data, you and your team can measure overall system health, pinpoint bottlenecks, identify and resolve issues faster, and prioritize high-value areas for optimization and improvements.

Let's dig into the ML performance tracing workflow. It follows three core steps:

Step 1: Comparing to another dataset;

Step 2: Performance breakdowns by slices;

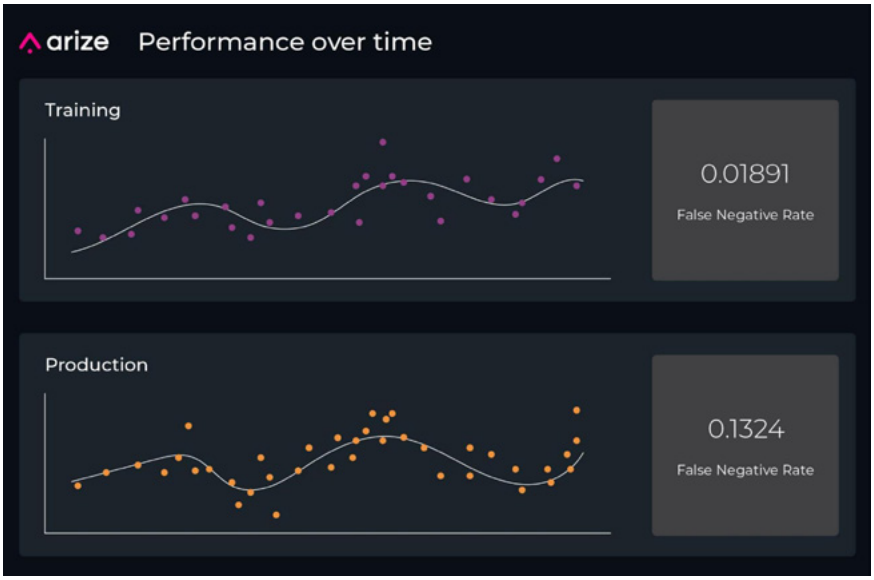
Step 3: Root cause and resolution.

Step 1: Compare to Something You Know

Model performance only really makes sense in relation to something—if the alert fired, something must have changed.

Machine learning models rely on data and code. One of those must be held constant for comparison while you change the other. So you either compare the same model on multiple datasets or multiple models on the same dataset.

Troubleshooting machine learning requires comparing across datasets.



What datasets do we have to compare?

1. **Training data.** Your model must have been trained on something, and you can look for differences between the training dataset and the data you are seeing in production. For example, perhaps a fraud detection model is having an issue in production. You can pull the original training dataset and see how the percent false negative changed since then.
2. **Validation data.** After training your model, you would have evaluated it on a validation dataset to understand how your model performs on data it did not see in training. How does the performance of your model now compare to when you validated it?
3. **Another window of time in production.** If your model was in production last week and the alert did not fire, what changed since then?

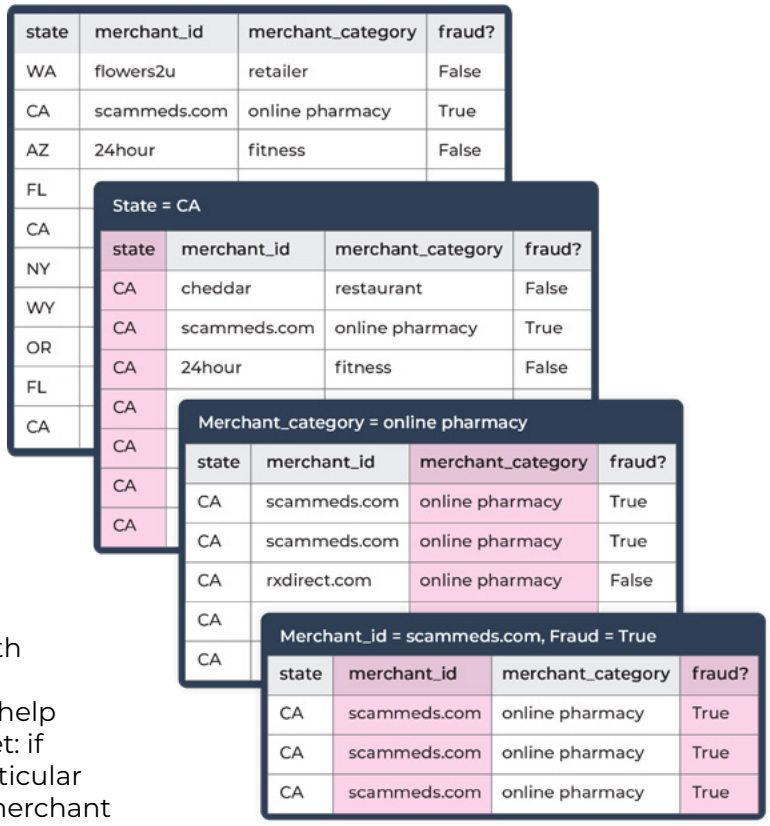
You can also compare your model's performance to a previous model that you had in production. Last month's model might give you more accurate ETAs for your food delivery, for example.

Step 2: Go Beyond Averages and Analyze Performance of Slices

Definition: What Is a Slice?

A dataset slice identifies a subset of your data that may behave qualitatively differently than the rest. For example, rideshare customers picked up from the airport may differ significantly from the "average" riders.

Comparing one metric across the whole dataset is fast, but averages often obscure interesting insights. Most frequently you are looking at a small slice, like a subset of a subset of data. If you can find the right slice, figuring out the problem becomes almost trivial. Ideally, this should not involve hundreds or thousands of SQL queries because you should be able to narrow your options quickly.

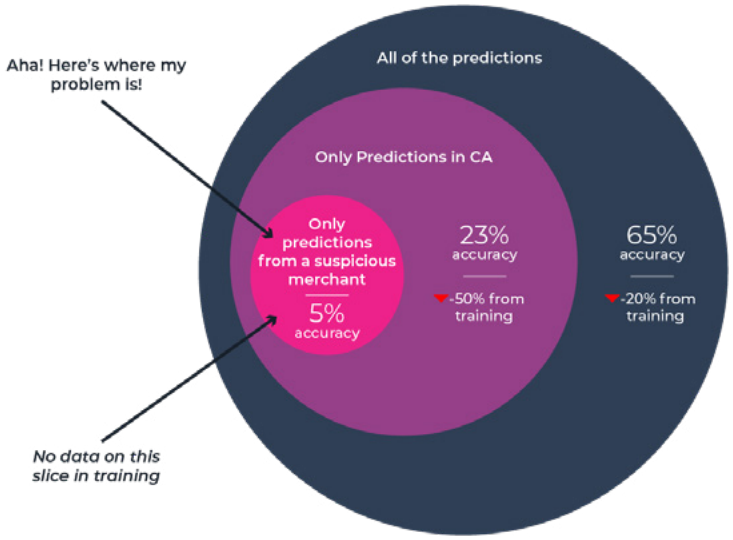


For example, if you saw the entire production dataset for your fraud detection model with slightly abnormal performance, it may not tell you much. If, on the other hand, you saw a smaller slice with significantly worse performance from your California transactions, that may help you identify what's going on. Better yet: if you narrow it down to California, a particular merchant category, and a particular merchant – and see that all or most transactions were fraudulent –that may help you identify the cause in minutes instead of days.

Real insights often lie several layers down.

You want to be able to quickly identify what is pulling your overall performance down. You want to know how your model is performing across different segments versus your comparison dataset.

This desire is complicated, however, by the exponential explosion in the number of possible combinations of segments. You may have thousands of features with dozens of categories each, and a slice can contain any number of features. So how do you find the ones that matter?



In order to automate this, you need some way to rank which segments are contributing the most to the issue you are seeing. If such a ranking existed, you could employ compute power to crunch through all the possible combinations and sort the amount of contribution from each segment.

Introducing: Performance Impact Score

Performance impact score is a measure of how much worse your metric of interest is on the slice compared to the average.

To calculate performance impact score for a slice, Arize takes the difference between performance in a slice and the average performance on the dataset and multiplies it by volume. We can then look at the maximum of all slices and surface the corresponding slice automatically.

The Calculation Behind Performance Impact Score

*max((Deltas between slice and global average metric) * volume)*

For example, if your metric of interest is MAE:

- *Compute (Slice MAE — Avg MAE) * Volume [for each slice]*
- *Max [slices]*

The volume should be normalized as the number of examples in the slice divided by the total number of examples in the dataset.

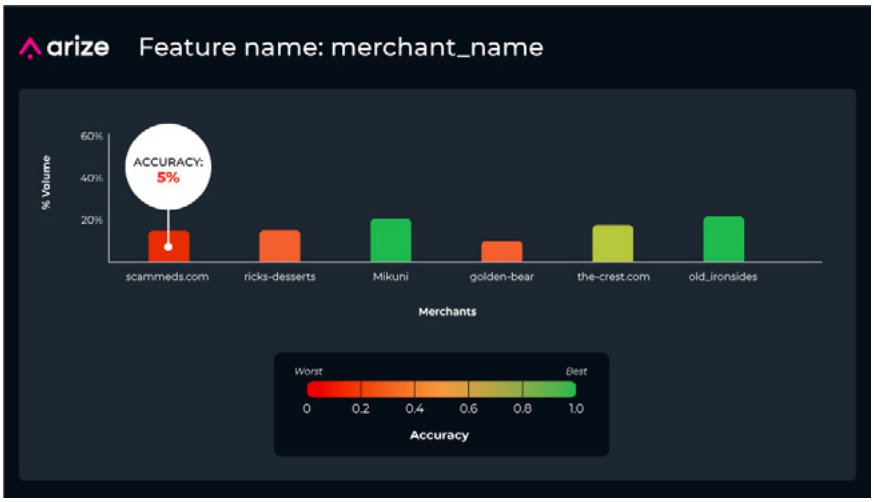
So the process for MAE (for example) is:

1. *Calculate MAE score on your dataset*
2. *For each slice find:*
   - *a) Number of examples in the slice divided by the number of examples in the dataset—let's call it normalized slice volume*
   - *b) Slice MAE minus total MAE (found in step 1)—let's call it delta*
   - *c) Delta (from 2b) divided by normalized slice volume (from 2a)*
3. *Take the maximum of all slices' results for 2c.*

Ideally, the ML engineer should see where the problem is at a glance. Good visualization and easy navigation can make this process very intuitive and help the engineer focus on providing insight—the job that humans are best at.

Continuing with the example of the fraud model, sorting by performance impact score enables you to narrow in on a slice – in this case, a specific merchant named "scammeds.com" in California – dragging down performance by 30% compared to the average. Since there was no data for this slice in training, it might indicate the need to retrain the model or revert to a different version.

Breaking out the feature “merchant_name” by accuracy and volume further reveals that the model’s accuracy for “scammeds.com” in production is only five percent – hence its drag on overall performance despite only representing a small share of overall transaction volume (~15%).



How Does Explainability Fit Into ML Observability?

Explainability in machine learning refers to the importance of the feature to a prediction. Some features may have much more impact on predicting fraud than others. It is tempting to look at explainability as the holy grail of segmentation, but you must be careful in doing so.

Explainability is the beginning of the journey to resolving the problem, not an end in itself. As Chip Huyen notes, explainability helps you understand the most important factors behind how your model works. Observability, on the other hand, helps you understand your entire system. Observability encompasses explainability and several other concepts.

Using feature importance can help you sort and prioritize where to troubleshoot.

Returning to the example of the fraud model, explainability can be illustrative as to where the problem lies. If you sort by which features have the most importance to the model, you will soon find that the features state (i.e. California) and merchant name (i.e. scammeds.com) are important to examine further to uncover the underlying performance issue.

While explainability is a terrific tool, it should not be used as a silver bullet to troubleshoot your models. Performance impact score offers more information by describing which segment has the biggest impact on why performance dropped.

Step 3: Root Cause & Resolve

You uncovered the needle in the haystack and found where the model is not doing well. Congratulations! Now, let's get to the harder question—why?

Here are the three most common reasons model performance can drop:

1. One or more of the features has a data quality issue;
2. One of more of the features has drifted, or is seeing unexpected values in production; or
3. There are labeling issues.

Let's look at those in a bit more detail.

1. One (or more) of the features has a data quality issue

Example: You are trying to figure out why the ETAs for a ride-sharing app are wrong, and you find that the feature "pickup location" is always 0.5 miles off from the actual pickup location.

Recommended solution: The data engineering team needs to go through the lifecycle of the "pickup location" feature and figure out where it gets corrupted. When they find the problem and can implement a feature fix, it should improve the ETAs.

2. One (or more) of the features has drifted, or is seeing unexpected values in production

Example: You see a spike of fraud transactions for your model, but your model is not picking them up. In other words, there is an increase in false negatives. This is coming from a specific merchant ID (which is a feature sent to your model). You are also receiving a huge spike from this merchant ID lately. You should see a drift in this merchant ID feature, showing that you are seeing more transactions from this merchant than before.

Recommended solution: In this case, you want to know what feature has changed either since you built the model or since before the performance decline. You want to find the root cause of the merchant ID distribution drift. After that, you may need to retrain your model, upsampling the new merchant ID that you didn't see as much of before. In some cases, you might even want to train another model just for this use case.

3. There are labeling issues

Example: A model predicting house prices is showing an extreme discrepancy across price distributions for a particular zip code. Zip code has very high importance. Upon further inspection, you find that the training data reveals this zip code is being labeled with two different city names, such as Valley Village and North Hollywood (the "Hollywood" city name yields higher house prices).

Recommended solution: Highlight the issue to the labeling provider, and provide clarification in labeling documentation.

Conclusion

The excitement around machine learning in the last decade has made it possible for ML models to get adopted quickly, solving very complex problems with large business impacts. Machine learning systems are usually built on top of data pipelines and other complex engineering systems that feed the models the data needed to make predictions.

However, real-life complexities often mean that an error in the smallest slice can lead to a substantial loss of economic value. Today this means that ML engineers must spend a lot of their time writing SQL queries and manually dissecting the model until a solution emerges. There is also a natural tendency to look at explainability as a shortcut. While explainability can often help you understand the problem, it is important to have other tools in your arsenal—particularly ML performance tracing—to get to the bottom of issues.

To recap, the fundamentals of ML performance tracing are:

1. **Compare to something you know;**
2. **Go beyond averages into slices of data; and**
3. **Root cause and resolve**

Ultimately, knowing what information to seek and having good tools that surface the information quickly—and in an easily digestible way—can save many hours, dollars, and customer relationships.

To implement full-stack ML observability with performance tracing with Arize, **sign up for an account** today.

For the latest on ML observability best practices and tips, **Sign up** for our monthly newsletter The Drift.