BITS Pilani
Pilani | Dubai | Goa | Hyderabad

ML Production Monitoring

Pravin Y Pawar

Adapted from “Reliable Machine Learning”
By Cathy Chen

The basics

- Monitoring, at the most basic level, provides data about how your systems are performing
  - data is made storable, accessible, and displayable in some reasonable way

- Observability is an attribute of software, meaning that when correctly written, the emitted monitoring data
  - usually extended or expanded in some way, with labeling or tagging
  - can be used to correctly infer behavior of system

- Obviously, monitoring is hugely important in and of itself,
  - but an offshoot of monitoring is absolutely crucial: alerting
  - A useful simplification is that when things go wrong, humans are alerted to fix them
  - defining the conditions for "things going wrong," and being able to reliably notify the responsible folks that

What Does It Look Like?

- To do monitoring,
    - must have a monitoring system
    - as well as systems to be monitored (called the target systems)

- Today, target systems
    - emit metrics a series, typically of numbers, with an identifying name
    - which are then collected by the monitoring system
    - and transformed in various ways,
        - often via aggregation (producing a sum or a rate across multiple instances or machines)
        - or decoration (adding, say, event details onto the same data)
    - aggregated metrics are used for system analysis, debugging, and the alerting

Example

Web Server

- Web server emits a metric of the total number of requests it received
  - The monitoring system will obtain these metrics, usually via push or pull,
    - which refers to whether the metrics get pulled from the target systems or get pushed from them
  - These metrics are then collated, stored,
  - and perhaps processed in some way, generally as a time series

- Different monitoring systems will make different choices about how to receive, store, process,
  - but the data is generally queryable and often there's a graphical way to plot the monitoring data
  - to take advantage of our visual comparison hardware (eyes, retinas, optic nerves, and so on) to figure out what's actually happening

Problems with ML Production Monitoring

- ML model development is still in its infancy!
  - tools are immature, conceptual frameworks are underdeveloped
  - discipline is in short supply, as everyone scrambles to get some kind of model—any kind of model!

- The pressure to ship is real and has real effects!

- In particular, model development,
  - which is inherently hard because it involves reconciling a wide array of conflicting concerns
  - gets harder because that urgency forces developers and data science folks to focus on those hard problems
  - ignore the wider picture

- That wider picture often involves questions around monitoring and observability!

Difficulties of Development Versus Serving

The first problem is that effectively simulating production in development is extremely hard

- Even if separate environments dedicated to that task (like test, staging, and so on.)

- Primarily Because of
- the wide variety of possible serving architectures
  - model pools, shared libraries, edge devices, etc., with the associated infrastructure running on)
- the invocation of the predictions
  - in development often invoke prediction methods directly or with a relatively small amount of code between developer and the model for velocity reasons
  - Running in production also generally means don't have the ability to manipulate input, logging level, processing, and so on
    - leading to huge difficulties in debugging, reproducing problematic configurations, etc.
- data in testing is not necessarily distributed like the data the model encounters in production
  - as always for ML, data distribution really matters

Difficulties of Development Versus Serving(2)

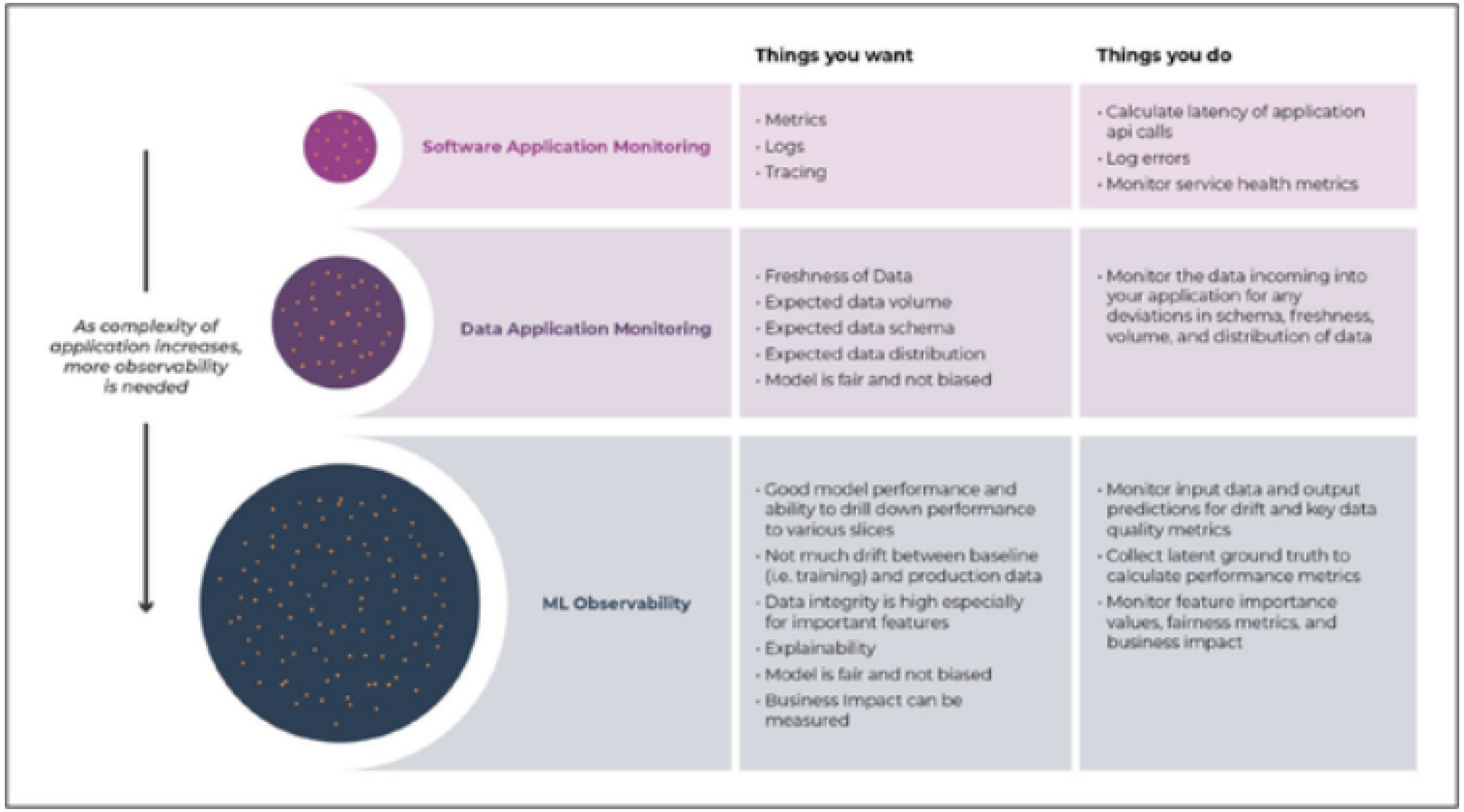
The second problem is about mature practices

- In conventional software delivery,
  - the industry has a good handle on work practices to improve throughput, reliability, and developmental velocity
  - most important is grouped concepts of continuous integration / continuous deployment (CI/CD), unit tests, small changes

- Unfortunately, today w\missing this equivalent of CI/CD for model development
  - not yet converged onto a good set of (telemetry-related, or otherwise) tools for model training and validation

- Expect this will improve over time as
  - existing tools (such as MLflow and Kubeflow) gain traction
  - vendors incorporate more of these concerns into their platforms
  - and the mindset of holistic, or whole-lifecycle monitoring gains more acceptance

Observability layers and system requirements



Observability layers and system requirements

Reasons for Continual ML Observability—in Production

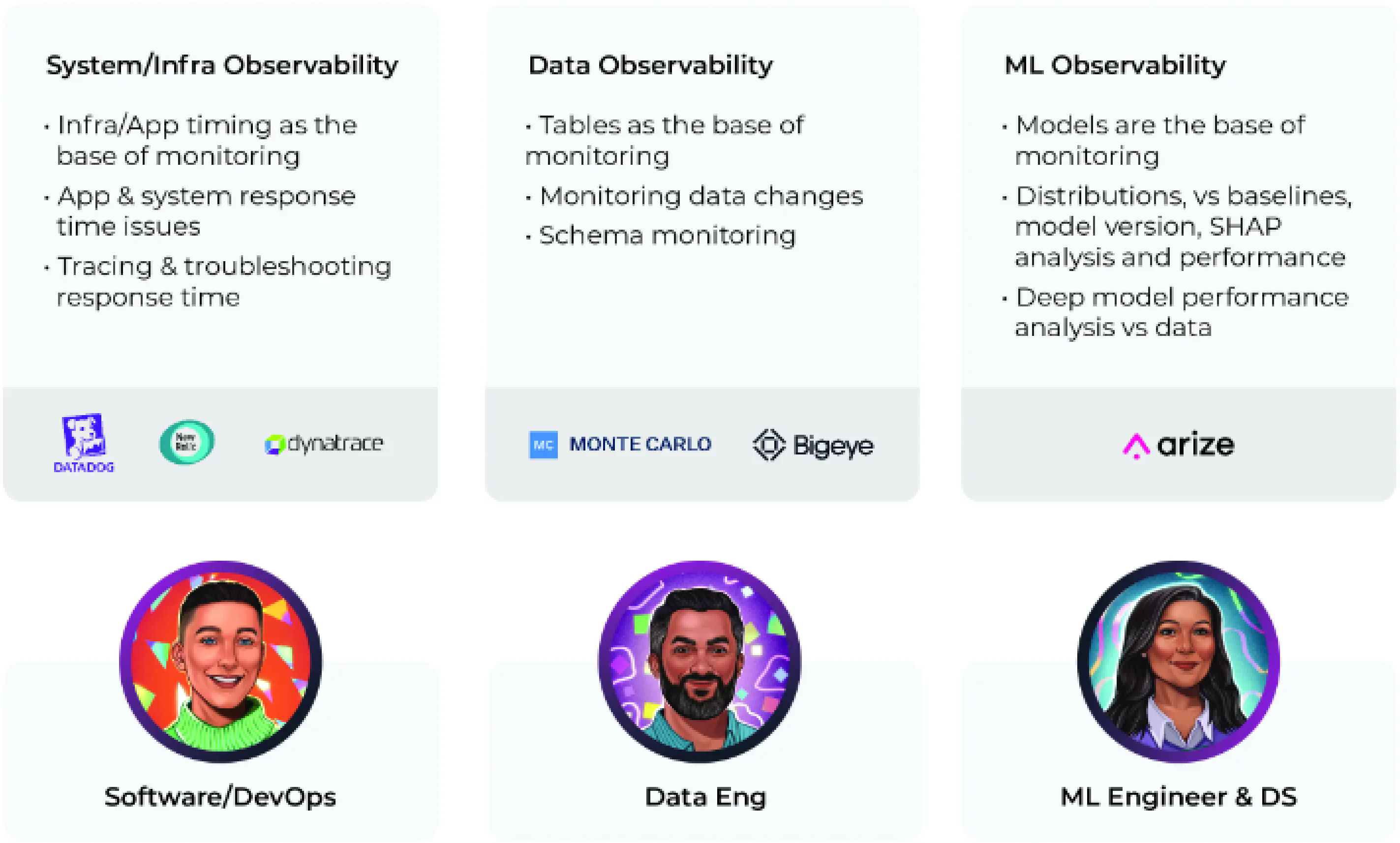
- Observability data from models is absolutely fundamental to business
    - both tactical operations and strategic insights

- One example - connection between latency and online sales
    - In 2008, Amazon discovered that each additional 100 ms of latency lost 1% of sales,and also the converse
    - Similar results have been confirmed by Akamai Technologies, Google, Zalando, and others

- Without observability, there would be no way to have discovered this effect,
    - and certainly no way to know for sure that either making it better or worse!

- Ultimately, observability data is business outcome data
    - In the era of ML, this happily allows not just to detect and respond to outages,
    - but also to understand incredibly important things that are happening to business

ML observability in context



[arize](#)

Key Components of Observability

System vs 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 model's predictions. · Mutable by prediction ID and dataset.
Metrics · Measured values of system performance. · Metrics comprise a set of attributes (i.e. 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 the 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.

Thank You!

In our next session: