Production ML systems: Deployment testing

You're ready to deploy the unicorn model that predicts unicorn appearances! When deploying, your machine learning (ML) pipeline should run, update, and serve without a problem. If only deploying a model were as easy as pressing a big **Deploy** button. Unfortunately, a full machine learning system requires tests for:

- Validating input data.
- Validating feature engineering.
- Validating the quality of new model versions.
- Validating serving infrastructure.
- Testing integration between pipeline components.

Many software engineers favor test-driven development (TDD). In TDD, software engineers write tests *prior* to writing the "real" source code. However, TDD can be tricky in machine learning. For example, before training your model, you can't write a test to validate the loss. Instead, you must first discover the achievable loss during model development and *then* test new model versions against the achievable loss.

About the unicorn model

This section refers to the **unicorn model**. Here's what you need to know:

> You are using machine learning to build a classification model that predicts unicorn appearances. Your dataset details 10,000 unicorn appearances and 10,000 unicorn non-appearances. The dataset contains the location, time of day, elevation, temperature, humidity, tree cover, presence of a rainbow, and several other features.

Test model updates with reproducible training

Perhaps you want to continue improving your unicorn model. For example, suppose you do some additional feature engineering on a certain feature and then retrain the model, hoping to get better (or at least the same) results. Unfortunately, it is sometimes difficult to reproduce model training. To improve reproducibility, follow these recommendations:

- Deterministically seed the random number generator. For details, see randomization in data generation
- Initialize model components in a fixed order to ensure the components get the same random number from the random number generator on every run. ML libraries typically handle this requirement automatically.
- Take the average of several runs of the model.
- Use version control, even for preliminary iterations, so that you can pinpoint code and parameters when investigating your model or pipeline.

Even after following these guidelines, other sources of nondeterminism might still exist.

Test calls to machine learning API

How do you test updates to API calls? You could retrain your model, but that's time intensive. Instead, write a unit test to generate random input data and run a single step of gradient descent. If this step completes without errors, then any updates to the API probably haven't ruined your model.

Write integration tests for pipeline components

In an ML pipeline, changes in one component can cause errors in other components. Check that components work together by writing an **integration test** that runs the entire pipeline end-to-end.

Besides running integration tests continuously, you should run integration tests when pushing new models and new software versions. The slowness of running the entire pipeline makes continuous integration testing harder. To run integration tests faster, train on a subset of the data or with a simpler model. The details depend on your model and data. To get continuous coverage, you'd adjust your faster tests so that they run with every new version of model or software. Meanwhile, your slow tests would run continuously in the background.

Validate model quality before serving

Before pushing a new model version to production, test for the following two types of quality degradations:

- **Sudden degradation.** A bug in the new version could cause significantly lower quality. Validate new versions by checking their quality against the previous version.
- **Slow degradation.** Your test for sudden degradation might not detect a slow degradation in model quality over multiple versions. Instead, ensure your model's predictions on a validation dataset meet a fixed threshold. If your validation dataset deviates from live data, then update your validation dataset and ensure your model still meets the same quality threshold.

Validate model-infrastructure compatibility before serving

If your model is updated faster than your server, then your model could have different software dependencies from your server, potentially causing incompatibilities. Ensure that the operations used by the model are present in the server by staging the model in a sandboxed version of the server.

Key terms:

- Training-serving skew
- Z-score normalization

Help Center

Previous
When to transform data? (3 min)

Next
Monitoring pipelines (15 min)

Was this helpful?
Send feedback