Testing for Deploying Machine Learning Models

You're ready to deploy! If only deploying a model were as easy as pressing a big red button. When deploying, you want your pipeline to run, update, and serve without a hitch. These needs lead to the requirements and solutions discussed on this page.

Test Model Updates with Reproducible Training

No doubt you want to continue improving your unicorn appearance predictor. Say you refactor the feature engineering code for the "time of day" feature. How do you test that the code is correct? You decide to train your model again and see if you get the same result. Oh no, you find that your model training is not reproducible. Determined to continue predicting unicorn appearances, you investigate further. You find that you can achieve reproducibility by following these steps:

- Deterministically seed the random number generator (RNG). For details, see randomization in data generation (https://developers.google.com/machine-learning/data-prep/construct/sampling-splitting/randomization) from the Data Preparation and Feature Engineering in ML course.
- Initialize model components in a fixed order to ensure the components get the same random number from the RNG on every run. ML libraries typically handle this requirement automatically.
- Average 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 taking these steps, you could have other sources of non-determinism.

Testing Model Updates to Specs and API calls

After updating your model to Unicorn Predictor 2.0, you need to test the new model for algorithmic correctness, along with any changes to API calls. Let's discuss how.

Testing API Calls

How do you test updates to API calls? Sure, 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. You want the step to complete without runtime errors.

Testing for Algorithmic Correctness

A model must not only predict correctly, but do so because it is algorithmically correct, not lucky. For example, if 99% of emails aren't spam, then classifying all email as not spam gets 99% accuracy through chance. Therefore, you need to check your model for algorithmic correctness. Follow these steps:

- Train your model for a few iterations and verify that the loss decreases.
- Train your algorithm without regularization. If your model is complex enough, it will memorize the training data and your training loss will be close to 0.
- Test specific subcomputations of your algorithm. For example, you can test that a part of your RNN runs once per element of the input data.

ꓭecause training a model takes so long, you might try training a model only partially before comparing its
' against a previously trained model. However, avoid testing partially-trained models because the test is ha
ıin and interpret.

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 a test that runs the entire pipeline end-to-end. Such a test is called an **integration test**.

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 these two types of degradations in quality:

- 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-Infra Compatibility before Serving

If your model is updated faster than your server, then your model will 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.

←
Previous
Overview (/machine-learning/testing-debugging/pipeline/overview)

Next
Testing in Production (/machine-learning/testing-debugging/pipeline/production) →