Testing Pipelines in Production

Congratulations! You've deployed your global unicorn appearance predictor. You want your predictor to run 24x7 without a hitch. You quickly realize that you need to monitor your ML pipeline. While monitoring all your components can seem daunting, let's look at requirements and solutions.

Check for Training-Serving Skew

Training-serving skew means your input data differs between training and serving. The following table describes the two important types of skew:

Type	Definition	Example	Solution
Schema skew	Training and serving input data do not conform to the same schema.	The format or distribution of the serving data changes while your model continues to train on old data.	Use the same schema to validate training and serving data. Ensure you separately check for statistics not checked by your schema, such as the fraction of missing values
Feature skew	Engineered data differs between training and serving.	Feature engineering code differs between training and serving, producing different engineered data.	Similar to schema skew, apply the same statistical rules across training and serving engineered data. Track the number of detected skewed features, and the ratio of skewed examples per feature.

Monitor Model Age Throughout Pipeline

If the serving data evolves with time but your model isn't retrained regularly, then you will see a decline in model quality. Track the time since the model was retrained on new data and set a threshold age for alerts. Besides monitoring the model's age at serving, you should monitor the model's age throughout the pipeline to catch pipeline stalls.

Test that Model Weights and Outputs are Numerically Stable

During model training, your weights and layer outputs should not be NaN or Inf. Write tests to check for NaN and Inf values of your weights and layer outputs. Additionally, test that more than half of the outputs of a layer are not zero.

Monitor Model Performance

Your unicorn appearance predictor has been more popular than expected! You're getting lots of prediction requests and even more training data. You think that's great until you realize that your model is taking more and more memory and time to train. You decide to monitor your model's performance by following these steps:

- Track model performance by versions of code, model, and data. Such tracking lets you pinpoint the exact cause for any performance degradation.
- Test the training steps per second for a new model version against the previous version and against a fixed threshold.
- Catch memory leaks by setting a threshold for memory use.
- Monitor API response times and track their percentiles. While API response times might be outside your control, slow responses could potentially cause poor real-world metrics.
- Monitor the number of queries answered per second.

Test Quality of Live Model on Served Data

You've validated your model. But what if real-world scenarios, such as unicorn behavior, change after recording your validation data? Then the quality of your served model will degrade. However, testing quality in serving is hard because real-world data is not always labelled. If your serving data is not labelled, consider these tests:

- Generate labels using human raters (https://developers.google.com/machine-learning/data-prep/construct/collect/label-sources).
- Investigate models that show significant statistical bias in predictions. See Classification: Prediction Bias (/machine-learning/crash-course/classification/prediction-bias).
- Track real-world metrics for your model. For example, if you're classifying spam, compare your predictions to user-reported spam.
- Mitigate potential divergence between training and serving data by serving a new model version on a fraction of your queries. As your validate your new serving model, gradually switch all queries to the new version.

Using these tests, remember to monitor both sudden and slow degradation in prediction quality.

Previous
← Testing for Deployment (/machine-learning/testing-debugging/pipeline/deploying)

Next
Check Your Understanding → (/machine-learning/testing-debugging/pipeline/check-your-understanding)