Practical machine learning engineering

Curated guide for building and maintaining solid end-to-end machine learning pipelines

Author: Morales, J.A.
Last update: January 13, 2023

Contents

Introduction 3

Do we really need machine learning? 4

Start measuring before building 4

How to know if you're ready for machine learning 4

Opt for a simple model, with simple features 4

At any point, be clear about what you're testing 4

Don't drop the data, unless noted 4

Heuristics can still be handy 4

Practice 'alerting hygiene' to keep it fresh 4

Detect the defects or don't serve 4

How to hear the silent failures 5

Who owns your features? 5

All the objectives in the world 5

All the objectives in the world 5

Transparent choices mean debugging made easy 5

Do measures have the same assumptions? 5

Fail fast then iterate to fail less and faster 5

Start with the directly observable features 5

Compare measures with existing measures 5

Many but simple vs few but complex features 6

If you must aggregate the features 6

References 7

Introduction

Curated guide for building and maintaining solid end-to-end machine learning pipelines.

Do we really need machine learning?

In building a machine learning pipeline, the first question we ask is, “do we really need one?” We want to focus on the problem we’re trying to solve, even if it’s simply wanting to explore a solution or create a proof-of-concept.

Start measuring before building

There are many reasons for doing this. It provides additional information for decisions along the build process. It also provides a pulse on the data – it’ll help capture any changes before, during, and after engineering process.

How to know if you’re ready for machine learning

One sign the project is ready for machine learning is when heuristics begin to get too complex. Simple heuristics work but complicated heuristics can get messy and will hurt more in the long run. At this point, consider machine learning instead. The trifecta you want: unwieldy heuristics + enough data + clear objective = start machine learning.

Opt for a simple model, with simple features

Keeping the first model simple allows you to more efficiently get the infrastructure right. It makes it easier to build up, maintain, and troubleshoot the data and the model.

At any point, be clear about what you’re testing

Test the model, infrastructure, and their integration separately. This helps isolate problems as well as package, prioritize, and plan work.

Don’t drop the data, unless noted

It’s not uncommon to duplicate an existing pipeline as a starting point for a new build. Caveat: the old pipeline might have dropped data that we need in the new pipeline. Make sure it’s noted. This also applies to any filtering and significant data transformations.

Heuristics can still be handy

If we have pre-existing heuristics that already provide good insight or could potentially help us understand better, we can try to use them by turning them into features.

Practice ‘alerting hygiene’ to keep it fresh

Get the model out fast – and keep it fresh. Know the freshness requirements of the system and build them into actionable alerts and dashboards.

Detect the defects or don’t serve

It’s obvious but it’s important to mention. Promote to higher environments only after testing, fixing, and/or noting bugs. Even if it’s a POC project. Yes, release that MVP. But. Don’t serve to the user if you haven’t done your detecting.

How to hear the silent failures

Compared to other system failures, silent errors are tricky to catch. Many mechanisms are in place to catch the loud and noticeable issues. To reduce silent failures however, it's helpful to keep statistics and manually inspect the data on occasion.

Who owns your features?

Give feature columns owners. In development, we often hear that the documentation is in the code. However, it can be time consuming to piece together the undisclosed or changing significance of data. At critical failure points, in production, this could make or break the solution.

All the objectives in the world

It’s great to check all the objectives but sometimes it’s impractical to work through them all. One way to prioritize is to look at what’s optimizable. We can work to increase this score.

All the objectives in the world

What’s your criteria for measuring proxy objectives – the things that we can directly evaluate in lieu of the real objectives? Ideally, proxy objectives are simple, observable, and attributable.

Transparent choices mean debugging made easy

Starting with an interpretable model makes troubleshooting easier. We can re-trace our steps and understand differences more clearly.

Do measures have the same assumptions?

For example, quality ranking assumes that data is posted in good faith. However, spam-filtering assumes that some aren’t. We can arrange the tasks so that they strengthen each other’s purpose. More importantly, we want to be clear about the assumptions. At some point, we may need to call on these assumptions to make important decisions.

Fail fast then iterate to fail less and faster

Plan to launch quickly. Try not to add things that will slow down early and future releases.

Start with the directly observable features

Instead of relying early on learned features from the get go, begin by looking at the directly observed and reported features. They’re simpler, faster to implement, and easier to troubleshoot.

Compare measures with existing measures

If the model is going to be a part of a family of systems, can we leverage the other systems to check the goodness of the model, the data, and the hypotheses?

Many but simple vs few but complex features

If possible, use very specific features. You can aggregate later. Normalize now.

If you must aggregate the features

Combine and modify existing features to create new features in human-understandable ways. Don't forget to note the transformation that you made.

…more coming soon.

References

Zinkevich, M. (n.d.) 'Rules of machine learning: Best practices for ML engineering', Google. Available at: https://developers.google.com/machine-learning/guides/rules-of-ml/ (Accessed: 01 April 2019).