Machine Learning to the rescue

Jan Macháček

May 28, 2018

Abstract

Machine learning to the rescue!... the first question is "to the rescue of what?"; immediately followed by "when is it indeed rescued?". The answers to these questions are crucial; luckily, the software engineering process is quite used to asking and answering these questions. Careful project analysis and inception, followed by continous integration and continous deployment in development (supported by adequate tests); all overseen by systematic project governance leads to successful software projects. This paper's proposition is that machine learning projects that are to apply established machine learning approaches and algorithms are no different than any other software project; and they must follow all practices of software engineering.

1 Machine Learning to the rescue!

Before embarking on a machine learning project, the first question is "to the rescue of what?"; immediately followed by "when is it indeed rescued?". The answers to these questions are crucial; luckily, the software engineering process is quite used to asking and answering these questions. Careful project analysis and inception, followed by continous integration and continous deployment in development (supported by adequate tests); all overseen by systematic project governance leads to successful software projects.

This paper's proposition is that "business" projects that use machine learning are no different than any other software project; and that all practices of software engineering have to be applied to the machine learning subsystems.

1. Versioned, testable; continously tested and sanity-checked analytics (BI)
2. Any BI query can be answered under 10 minutes
3. Monitoring on the BI environment to identify queries that use normalised data
4. The results of the BI queries that humans process define what ML should solve
5. Ingestion components decoupled from the rest of the system
6. Versioned, testable; continously tested and sanity-checked data sets
7. Pre-computed "return constant" model
8. Versioned, testable; continously tested and sanity-checked model storage with training and validation data set references
9. Model deployer and "debugger"

But where is the ML that builds the model? That's the code that the engineering teams need to build to replace item 6.

Once the first four steps are known, the engineering teams can implement the remaining steps of the pipeline. If the system that is to take advantage of ML is event-based, the event delivery mechanism provides the decoupling, resulting in architecture shown in Figure 1.

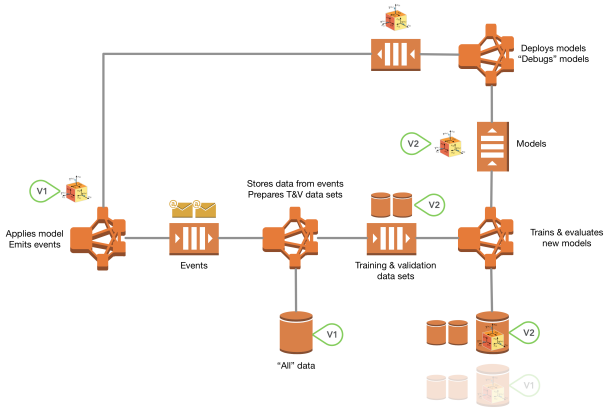

Figure 1: ML pipeline in event-based system

If the front-end system is not event-sourced, the ingestion must be decoupled using a read-only replica of the live data. Notice in Figure 2 the flow of the data: the data is pushed into the read-only replica in the first step to allow the front-end system to control the load on its data store; from the read-only replica, the data is pulled into the ML data store.

Regardless of the approach used (or even if a hybrid approach is deployed), the entire system has to be aware of any back-pressure.

The ML team maintains the tooling for the pipeline, consults on the best models, researches, ...; but the product teams (that ultimately work on the service that *uses* the

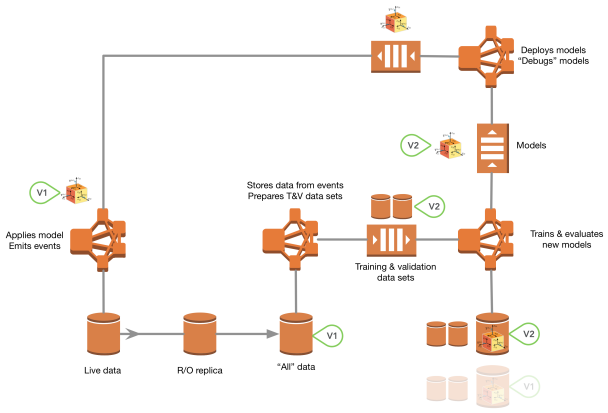

Figure 2: ML pipeline in non event-based system

model) have the first dibs on implementing the model. Successful implementation of this strategy means that anyone can implement a new model (even if only to just see what will happen!), train it, debug it, and deploy it all within a single day. All the mechanics of data ingestion, storage, versioning; runtime of training a model, evaluation, storage, versioning; debugging and deploying; and the usage is all implemented.

In this sense, the machine learning code is just like any other ordinary code; it is subject to all the high engineering standards and safeguards.