Notes from last week

OvO, OvA

Minibatch vs Stochastic descent

?

Rules of Machine Learning

(source: Martin Zinkevich, Google)

Some terms

- **Instance**: The thing about which you want to make a prediction. For example, the instance might be a web page that you want to classify as either "about cats" or "not about cats".
- **Label**: An answer for a prediction task either the answer produced by a machine learning system, or the right answer supplied in training data. For example, the label for a web page might be "about cats".
- **Feature**: A property of an instance used in a prediction task. For example, a web page might have a feature "contains the word 'cat'".
- **Feature Column:** A set of related features, such as the set of all possible countries in which users might live. An example may have one or more features present in a feature column. "Feature column" is Google-specific terminology. A feature column is referred to as a "namespace" in the VW system (at Yahoo/Microsoft), or a field.
- **Example:** An instance (with its features) and a label.
- **Model:** A statistical representation of a prediction task. You train a model on examples then use the model to make predictions.
- **Metric:** A number that you care about. May or may not be directly optimized.
- **Objective:** A metric that your algorithm is trying to optimize.
- **Pipeline:** The infrastructure surrounding a machine learning algorithm. Includes gathering the data from the front end, putting it into training data files, training one or more models, and exporting the models to production.
- **Click-through Rate:** The percentage of visitors to a web page who click a link in an ad.

Overview

- Engineering is more important than ML. If it's not reliable, solid, and reproducible, the rest doesn't matter.
- Have reasonable objects
- Be as simple as possible

Start without ML

- ML needs data, you rarely start with lots of data
- Heuristics will often get you half way there
- Your first goal is just to be better than random. So identify what random looks like.

Design and Implement Metrics

- Start by measuring, otherwise you can't know how you're doing
- Measuring the first thing is the hardest
- People care less early, so less resistance
- Get historical data now. When you start to care, you'll have a baseline.

Prefer ML to complex heuristics

- It's more maintainable
- But have you tried simple heuristics?

Start with simple models and get infrastructure right

- If your pipeline is shoddy, it will be hard to do anything anyway
- The first model provides the biggest delta

- This is “hello world” territory, focus on the basics
  - getting data
  - representing data
  - identifying good vs bad
  - how to integrate model into application
- Simple features are easier to understand, debug
- Make sure you understand your data

Test infrastructure separately from ML

- Make sure the infra is testable
- Make sure the ML is encapsulated
- Test getting data into the system
- Test that features are populated correctly
- Inspect the data (if allowed)
- Compare statistics from your pipeline with other sources (if exist)
- Test moving models from training to production
- Make sure you understand your data

Heuristics become features

- Often some system already exists. It uses heuristics, produces features. Take advantage of that.
- Consider using the existing system as a sort of pre-processor, generating synthetic features.

Monitoring and alerting are important

- Understand your freshness requirements
- Do sanity checks at model export time, at deploy time

- Understand what requires an email, what requires a page, what just has to be available for inspection
- Be aware of silent failures (e.g., data source decay)
- Make sure features have owners and that it's documented who they are (and that the features are documented). Same for algorithms.

Objectives (objective function)

- Start simple: at first, many things are correlated
- Start simple: observable and easily measurable
- Avoid (at first) indirect effects (next/previous day, correlations between features)
- Don't try to use ML to measure user internal state (happiness, satisfaction)

Interpretable models are easier to debug

- Linear, logistic, and poisson regression are directly motivated by probabilistic models, so easier to reason about
- Models with objectives based on 0-1 loss, hinge loss, etc. are harder to reason about

From phase 1 to phase 2

- Phase 1 is getting a working end-to-end system
  - training data
  - metrics
  - infrastructure, pipeline
  - unit and system tests
- Phase 2 is feature engineering
  - adding and inventing new features
  - metrics mostly all rising

Launch (and iterate)

- Expect that your first model is not your last: avoid complexity that will slow you down later
- Think about how easy it is to add or remove features
- Think about how to run multiple copies in parallel
- Don't sweat the small stuff, you'll do it next iteration (next quarter)

Start with observed features

- That is, don't start with learned features
  - features from other systems (different objectives, maybe stale)
  - features you learn yourself (e.g., clustering)
- Many algorithms are non-convex, so taking their features might kill your convergence (different local minima on different runs)
- Harder to judge impact of changes
- So shoot first for good baseline

Simple feature engineering

- Consider fixed intervals (e.g., MLP) rather than variable (e.g., LSTM)
- Consider discretisation (e.g., age bands), don't worry too much about getting the banding right
- Crosses are useful, but can generate too much data, can overfit. E.g., word in query, word in document
- You can (roughly) learn as many weights as you have data
- Clean up features you are no longer using, they are technical debt

Human analysis of the system

- This is more art than science, but it's important!

- You are not a typical end user
  - You are too close to the code and to the problem
  - Confirmation bias
- Get (even pay) other people to test, they cost less than engineers
- Think about team bias: are you all white, all male, etc.
- Watch real people use the system, don't correct them, they're right, whatever they do

Comparing models

- Know how to measure delta between models (e.g., what you're working on and production)
- Make sure that comparing a model with itself says small delta
- Remember that, ultimately, you're optimising a business problem, not log loss