@sjwhitworth

Building robust machine learning systems

Or, how to sleep well when running machine learning systems in production

ravelin.com

Me & Ravelin

- Co-founder and engineer at Ravelin - protect merchants from credit card fraud in real time
- Ingest large amounts of data about customer behaviour
- Use ML to return likelihood of fraud in milliseconds
- Clients label customers to improve system continually
- Go and Python shop
- Use other strategies in addition to ML

Fraud detection & prevention

- Merchants bear the cost of credit card fraud - can lose 1-5% of revenue with no protection, with heavy fines
- Adversarial problem - fraudsters continually trying to evade us
- If our ML messes up:
  - False positives - good customers hassled
  - False negatives - let fraudsters through
- Move quickly, but want to be sure we’re getting it right

Systems are moving from the deterministic to the probabilistic

ML learns functions from data, instead of building them

Software engineering - we aim for determinism of individual components, provably correct

Machine learning - we learn functions from a snapshot in time, with some level of predictive performance - ‘ground truth’ is approximate

How can we make machine learning systems less of a mound of complexity and tech debt, and more like normal software?

Josh Wills @josh_wills · Feb 18

Rule #1 of Hiring Data Scientists: Anyone who wants to do machine learning isn't qualified to do machine learning.

11 111 257

Josh Wills
@josh_wills

Following

@josh_wills folks who have done ML before aren't dumb enough to do it again. (Usually.)

Why is running robust ML systems hard?

- Silent failures and performance degradation
- Current actions impact future performance
- Unsure if loss function is a good proxy
- High base complexity
- Lots of supporting infrastructure needed
- Easy to mess up

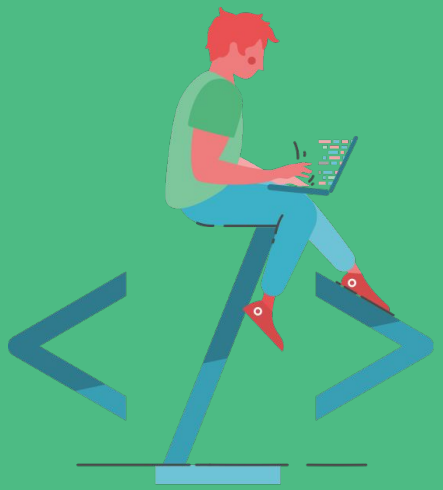
We should talk about it more

- We discuss algorithms, frameworks, papers
- We don’t really talk about designing, running, debugging, operating ML systems
- Rules of Machine Learning: Best Practices for ML Engineering (Martin Zinkevich)
- Tweet I saw in reaction: *‘Spent the last 10 years of my life learning these problems the hard way’*

Key aspects of good software

- Does what it needs to do, well
- Well tested
- Easily deployable/upgradeable
- Quick to debug problems, and avoid in future
- Audited, version controlled, automated, reproducible

Training



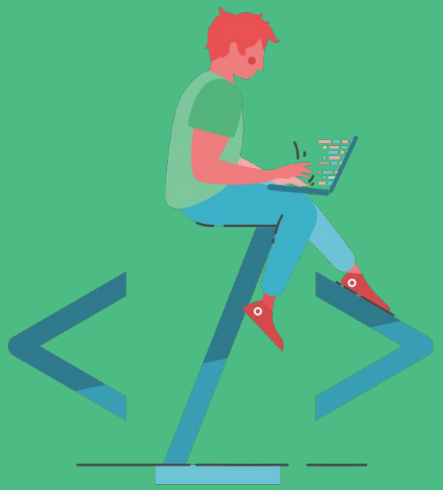
Labels & ‘truth’

- Can you trust your labels?
- Are they fuzzy or subject to feedback loops?
- Are you labels instant? Or do they suffer from a time delay?
- Implicit labels vs. explicit labels
- Your system’s actions may stop you getting labels
- Fuzzy problem, but the most important!

Data

- As a general rule, be wary when touching your training data - you're explicitly biasing
- Filtering examples out of your training data can work
- Sampling training data - sometimes computationally necessary
- Do you need a time machine?

Testing



Test your data

- Features are a function of your code, and your data. Either could be broken. Yay!
- Test invariants in your feature extraction process
- *‘This feature should monotonically increase with time per customer’*
- *‘The mean of this feature across all examples should be 1’*
- *‘This value should always be positive’*
- We wrote a small system on top of Pandas to do so
- Flushes out bugs in feature extraction, and bad data

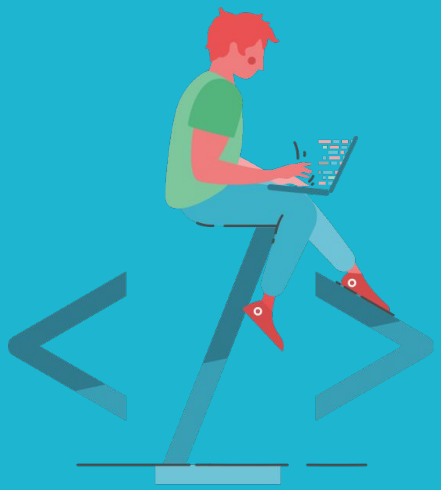
Test your models

- Build assertion set of examples that you'd be highly embarrassed to get wrong
- Don't fit to it - use as unit test
- Failures / regressions on this set indicates something has gone awfully wrong
- Never trust any big jumps in performance without verification
- Understand/cluster most misclassified examples
- Errors should be somewhat random, not systemic

Test your system + infrastructure

- Test that you get the same prediction offline that you do online
- Ensure you don’t leak information from the future
- If system is latency sensitive, add performance benchmarks to automation
- Live integration tests - be the client, send predictions, assert they’re right

Deploying



Shipping

- Ensure you share as much code as possible between offline and online
- If you have a difference between offline and online, your system will fail silently
- Thus, throwing models over the wall to developers to rebuild from scratch isn’t a great idea
- Abstract data source, to the computation of features on that data
- Pickles, PMML, weights - anything to let ML people ship quick

Rollout

- Run new model side by side with live system, in production
- Canary, release in dark mode, A/B test
- Manually inspect biggest differences online - are they sensible or spurious?

Metrics and logging

- Measure the difference in probabilities between your live + offline model. Small differences, lower risk in deploying
- Instrument (Prometheus, Datadog), and log (BigQuery / S3) everything
- Instrument feature extraction in the same way you tested offline data
- Instrument inference distribution (e.g. p99 of of probability offline should match p99 online)
- Measure whatever business metric this system is trying to improve

Debugging bad predictions

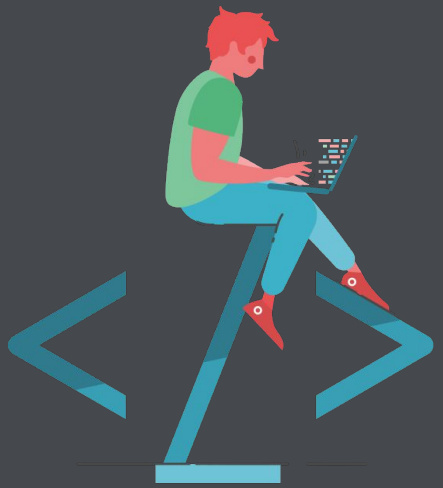
Look at the data!

- This is so obvious as to be nearly insulting
- We love to just jump in and Do Machine Learning™
- Models learn the world that you present them
- Debug when training your model, not only live predictions
- Add to the ‘obvious set’, to prevent future regressions

Make your model explain itself

- Random forests: walk the trees for bad predictions
- Neural nets: T-SNE on embeddings
- Linear models - relatively easy
- Usual suspects: difference between offline/online, information leakage, or model cheating
- Few packages to help: *LIME*, *eli5*
- *The Mythos Of Model Interpretability* - ZC Lipton

Automation



Why automate?

- Training in notebooks or shells works, until something goes wrong
- End up training on *‘data_1_final_edited_really_final (1)(2).csv’*
- Hard to replicate state for troubleshooting
- Experiments buggy and hard to replicate
- Wasting human time

Automate everything

- Choose a task manager (AirFlow/Luigi), use it
- Automate whole pipeline: *extraction -> hyperparam search -> training -> evaluation -> comparison against current system*
- Fresh models + fresh data acts as integration test
- Automation helps distill things you care about
- Team much more productive

Auditing models

- Who trained the model?
- When? What git hash?
- On what raw data?
- Which features did you use?
- With which hyperparameters?
- Bake these answers into your models so you can interact with them programmatically

Reproducibility and archiving

- Can you reproduce it?
- Control random seeds through pipeline
- Archive training stage to (cloud) storage - features, logs, hyperparameter searches, models
- A model is a snapshot of the world around it at a given time
- Taking a snapshot of that snapshot is a good idea - you'll thank me later

‘Do machine learning like the great engineer you are, not like the great machine learning expert you aren’t’ - Zinkevich

ML systems = normal software + extra paranoia

Thanks!

@sjwhitworth