

ML Advice

(Clip art day)

Chris Ré

Announcements

- Mid-quarter feedback (See Piazza Post #397)
- Midterm exam location: CEMEX (Wed May 15, 7-10pm)
- Midterm Syllabus: Upto EM algorithm (Wednesday)
- Practice Midterms: posted on Piazza (Post #414)

A magnifying glass with a black handle and a silver rim is positioned over the word "Disclaimer". The lens of the magnifying glass is focused on the letter "D", making it significantly larger than the rest of the word. The word "Disclaimer" is written in a large, bold, black sans-serif font. The background is white.

Disclaimer

- This lecture is filled with (hopefully informed) personal opinion.
- It is high level and presents some difficult, raw material.
- I tried to include ideas that people have told me were helpful to them

Phases of ML projects

- Do you really want an ML system?
- Ok, so you want to train a model. It's not working well... now what?
- Now you have to live with an ML model and its eco system...



A Running Example

- You want to build a spam detector.
- There are lots of types of spam, think of email for concreteness.
- You're tired of all that spam!

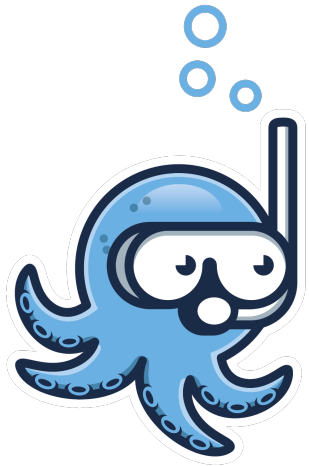


Machine learning is driven by data.

You need realistic spam and not spam!

Data is hard to get. It's critical, and you will get it wrong.

Acquiring Data



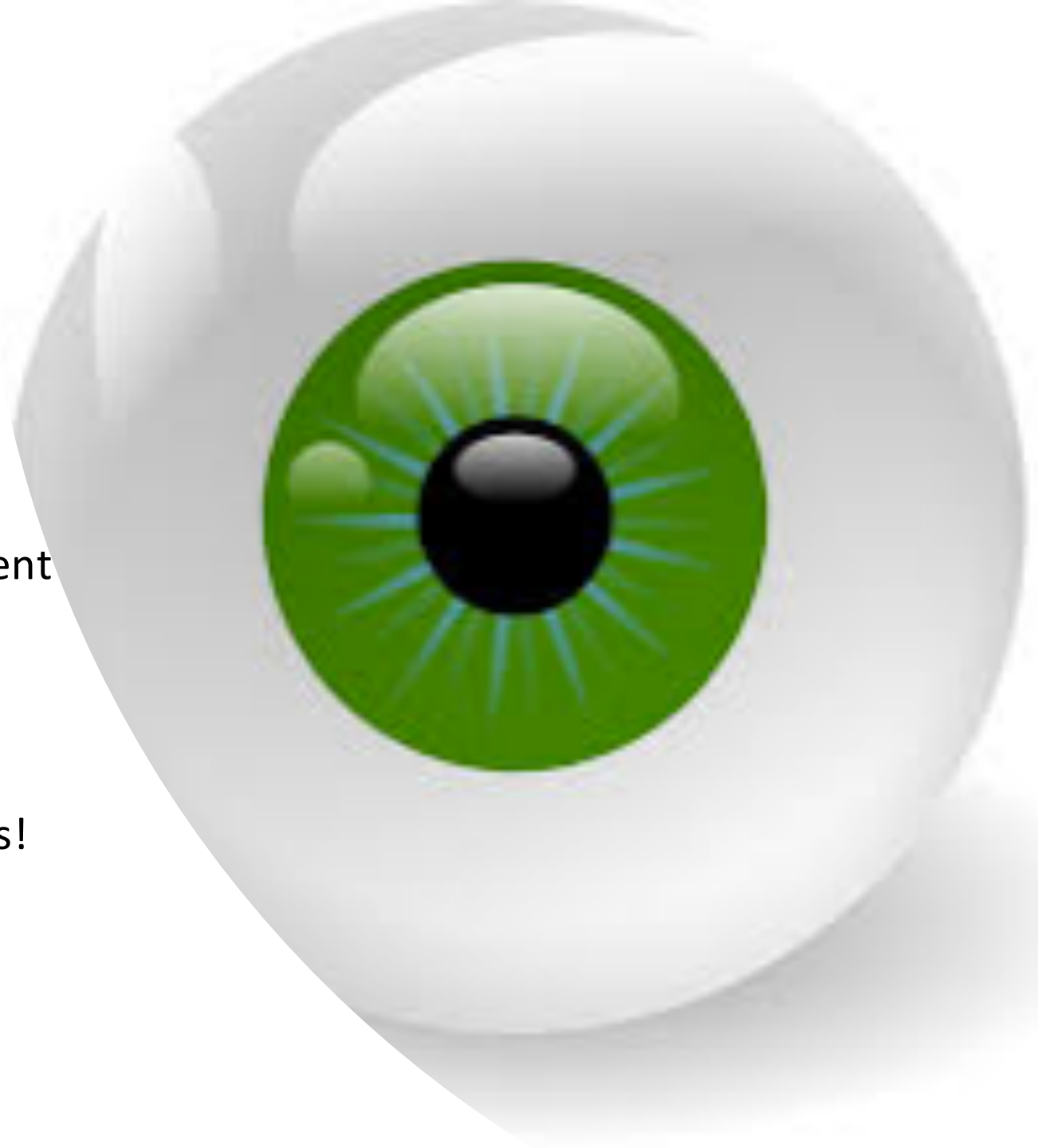
snorkel

Iterate on the collection of data. It takes time to do so.



Look at your data.

- You have some spam, look it at it!
- If needed, build tools to look at your data.
 - Slice and dice: Spam from Europe different than from Africa different from US?
 - Spam sent to .edu different than .com?
- Machine learning is iterative.
- Your data can change—there are adversaries!
 - Spam changes! (phone numbers in HT)
- “Become one with the data” – Karpathy.



Create a specification

- Machine learning doesn't obviate the need to know what you are building.
 - What is SPAM? Maybe I like ads for low low rates?
- A good specification has little ambiguity. Someone else should be able to read and implement it.
 - Critical if you want to employ graders!
 - Don't be tempted to think ML answer "is kind of ok..." this accrues debt later (next person has trouble).
 - E.g. extractor for place names, then used downstream for main location (confidence in 1st meaningless for the second)
- Your specification **must** be embodied in a **set of examples**.



A **test set** is an important part of your specification.

Simple descriptive dashboards

- Measure simple things
 - How many entities per sentence? How long are the sentences? How many verbs? Keywords per sentence.
 - Slice by time. Is your SPAM changing over time?
 - Earlier phone number change.
- You want to catch, at a glance, any changes.
 - **Story:** Economic indicators. Turned out provider had given us Spanish data...
- Model rot is real, sadly. Monitor continuously! (scoreboarding)



Class Confusion Matrices

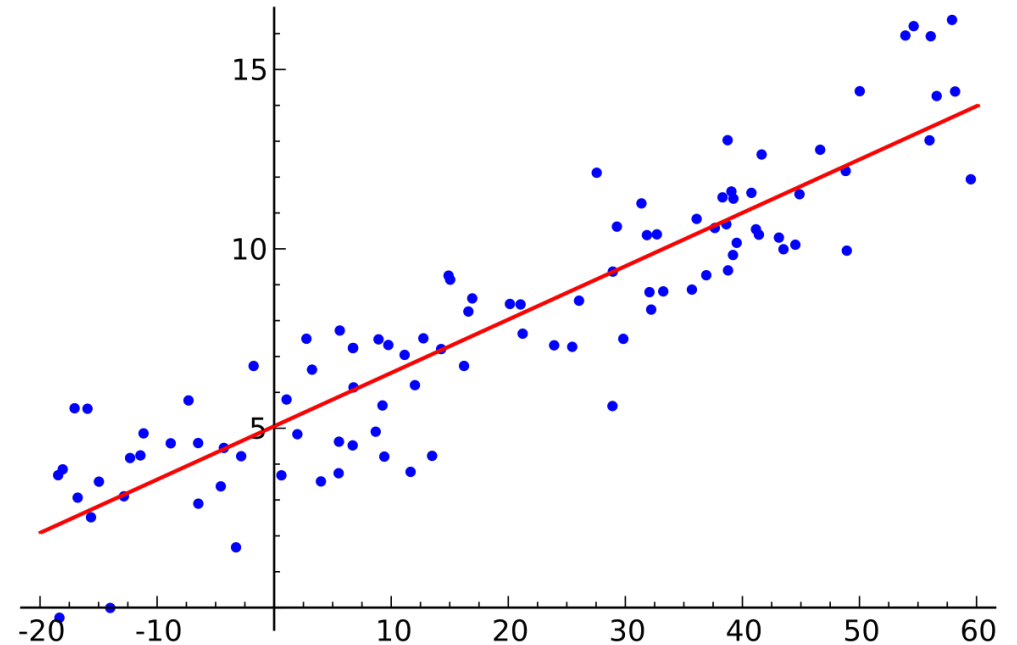
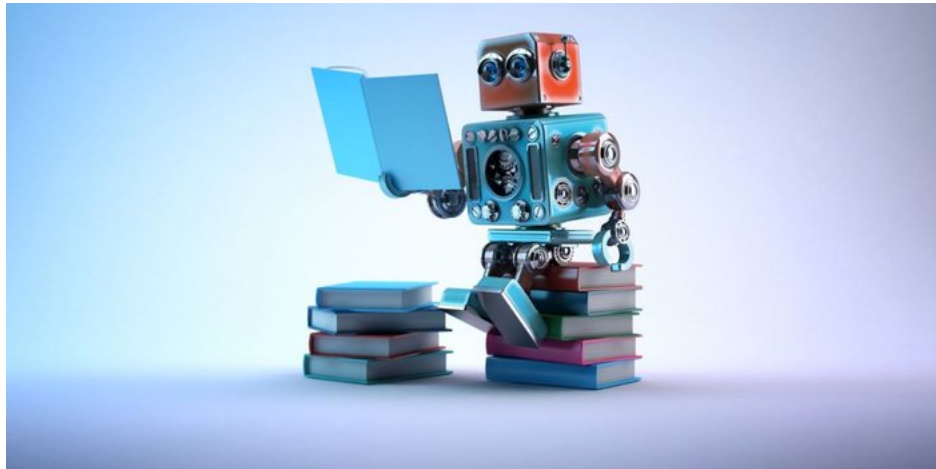
	Class LoanSpam	Class Phishing	Class Good Email
Predicts LoanSpam	1000	10	50
Predicts Phishing	45	505	30
Predicts Good Email	7	8	2000

- See at a glance, our accuracy is pretty high—but...
 - Discuss our false positive v. false negative rates?
- What would happen if we added spear phishing? Can help us debug specification!
 - Examine “top confused classes” if you have many
 - Common when building big ML models collaboratively.
 - Subtle distinctions are good--but need enough data and crispness to support them.

I have yet to see anyone get preceding steps right on first try. Build it first, and iterate!

A well running ML system is a rewritten poorly running ML system.

You want to build an ML model



Which should you build first?

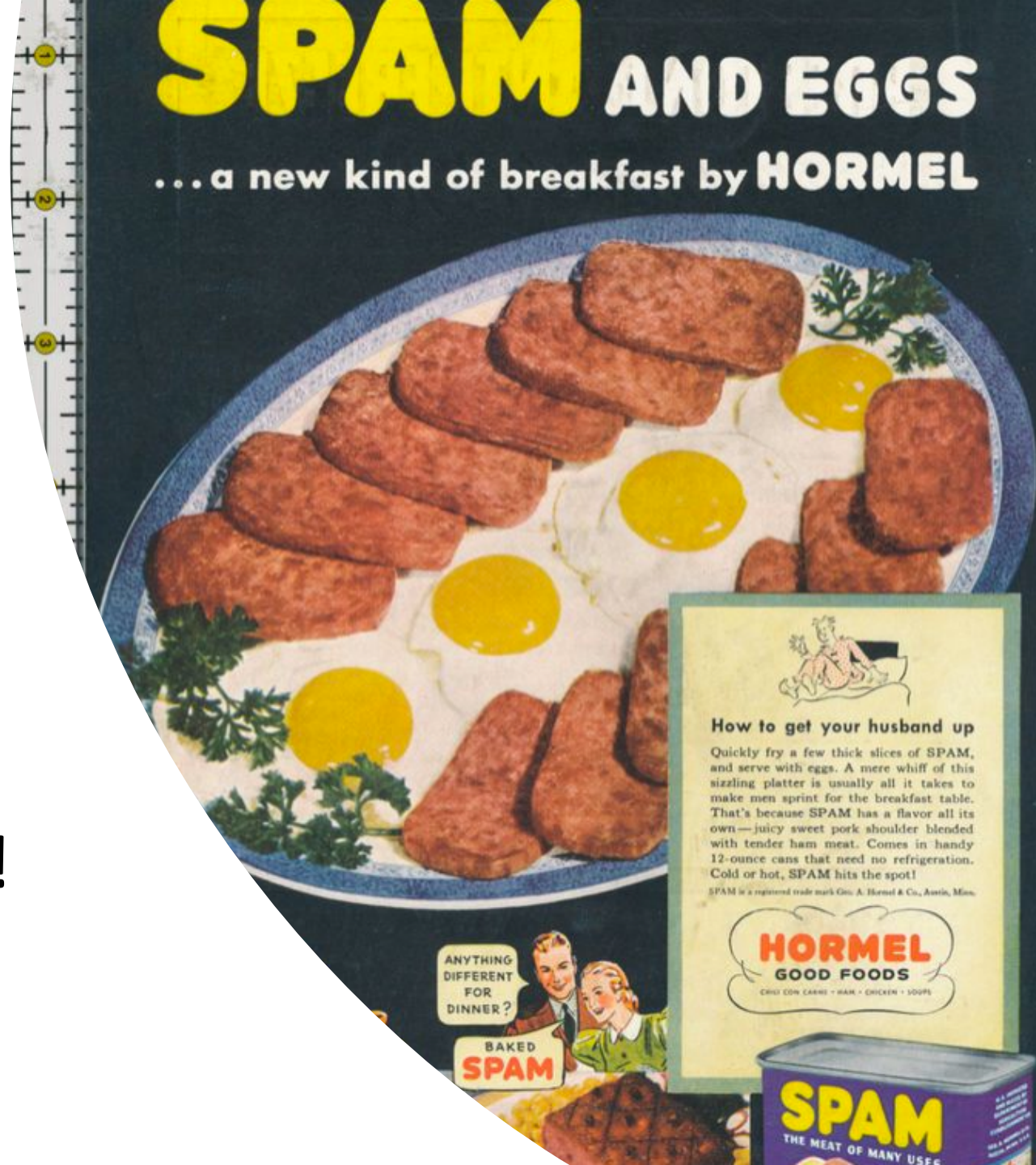
*...try simple methods first... really still debugging.
Best ML folks treat models as a way to understand.*

What to build?

- Build simplest thing first.
 - Sometimes what you have code laying around... iterate quickly!
- Linear or logistic regression w/ simple features,
 - You know it's converging, easy to setup, lots of packages that support it.
 - It runs fast! Quick iteration!
 - Features are easier to understand, add information, do error analysis.
 - Good baselines for future work
 - Many projects get good enough results here, and move on.
 - Or, more often, learn that they didn't understand the problem and refine!

Debugging Learning Algorithms

- Your goal is to build an ad spam detector.
- You run a logistic regression algorithm.
- Sadly, it's error is too high!
- What do you do?



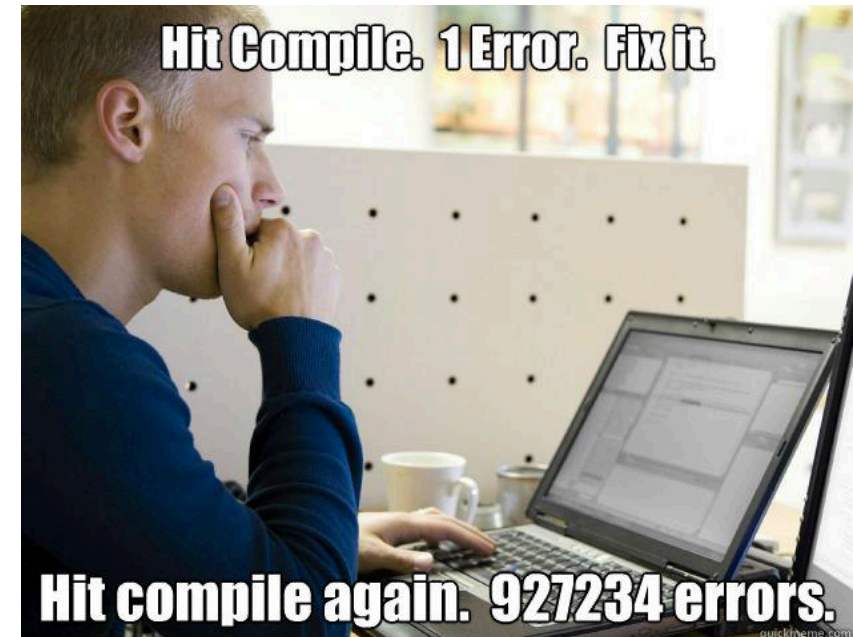
What could be wrong?

- Maybe it's the data or your features?
 - Try getting more training data.
 - Try a smaller set of features?
 - Try adding more features?
- Maybe it's the optimization algorithm?
 - Run GD a little while longer....
 - Try a different method, SGD, GD, Newton?
- Maybe it's the hyperparameters?
 - Different value of regularizer?
- Try using a different model!

**if you don't
tell me
what's
wrong, how
can i make it
right?'**

Just like compiling!

- Could hit train model, try it, and run again!
- Or you could develop **diagnostics to help you understand.**
- Recall simple metrics, these catch **data prep bugs (very nasty)**
- Bias-variance provides a set of diagnostics!



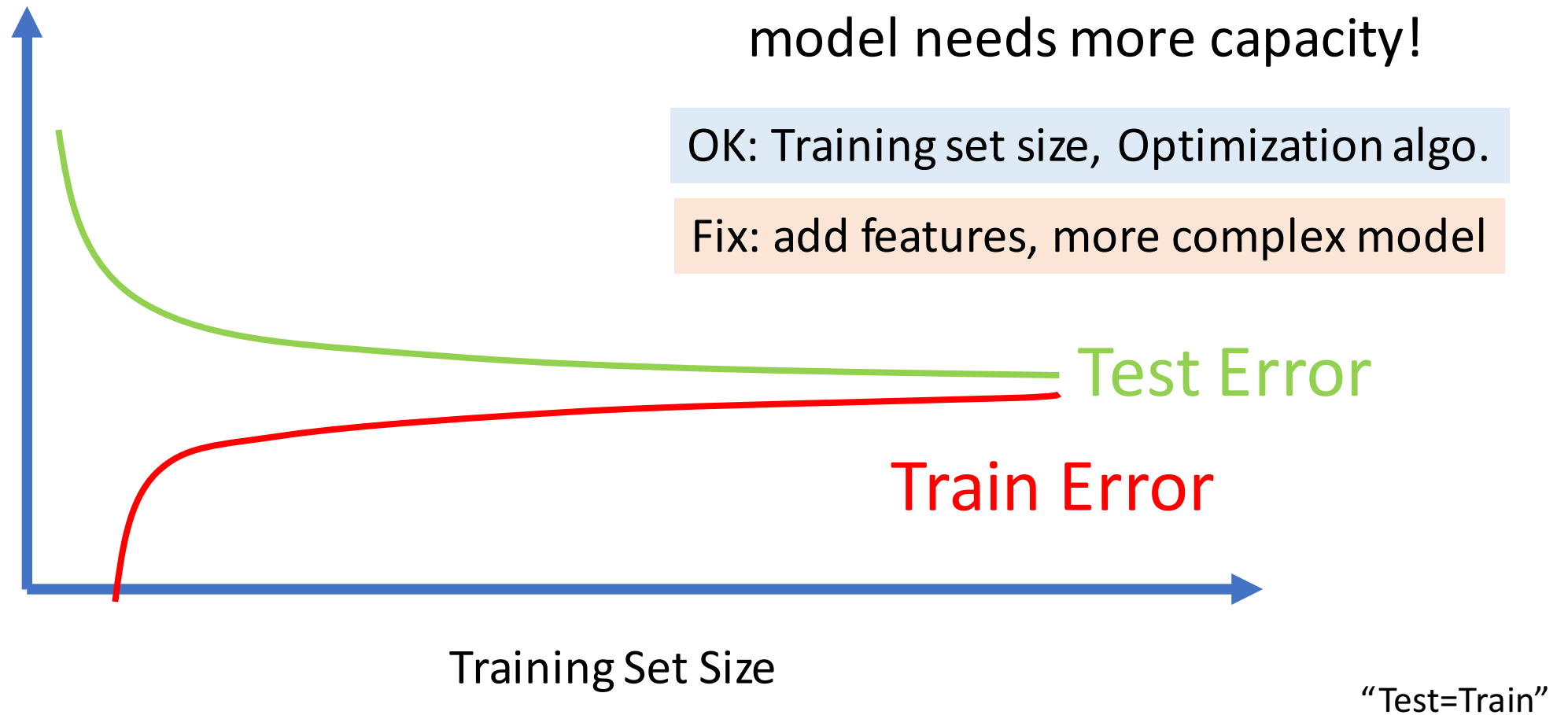
We'll cover some diagnostics that have helped us.

Diagnostic: Test versus Train Score.

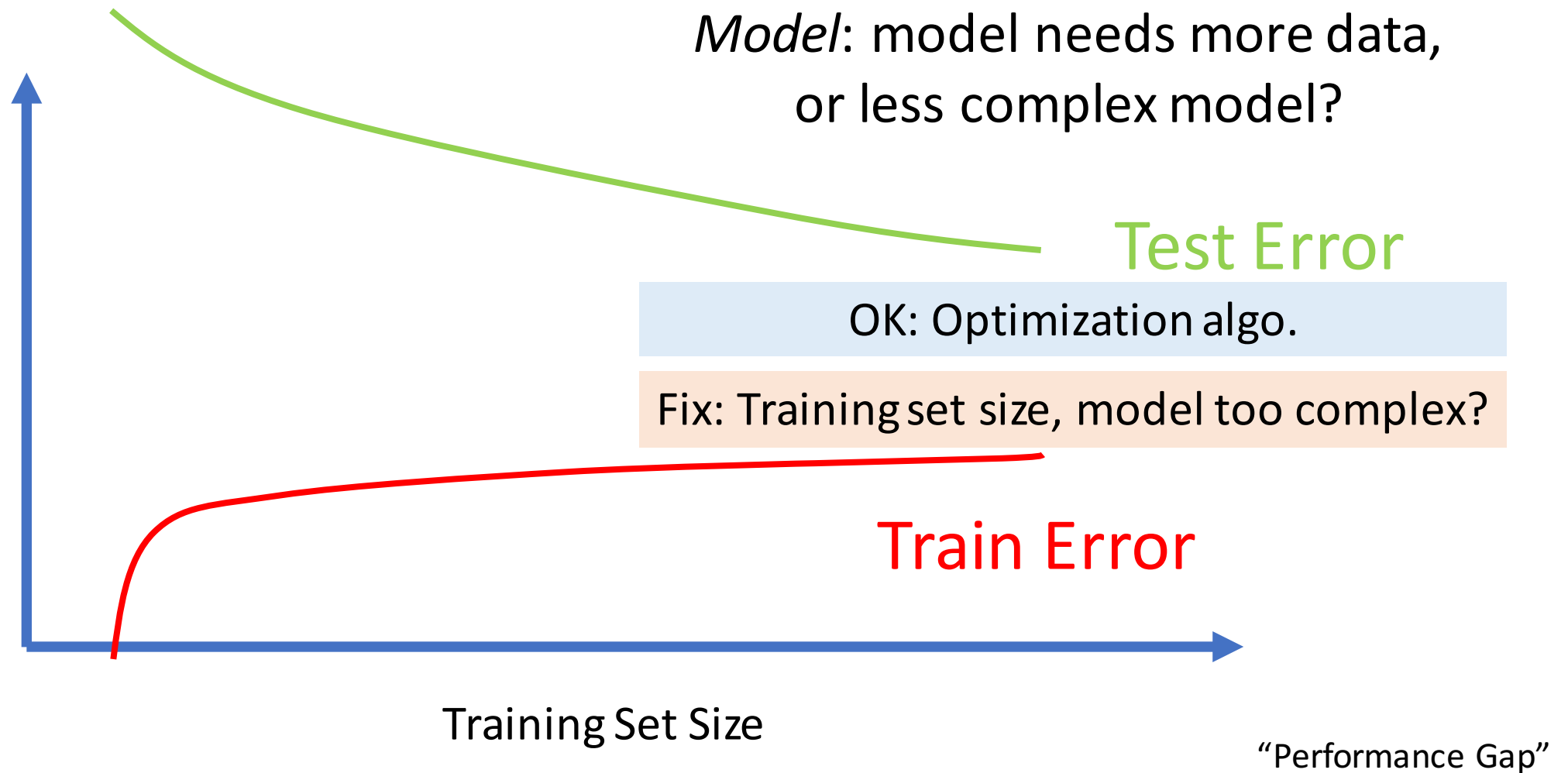
If error is too high:
model needs more capacity!

OK: Training set size, Optimization algo.

Fix: add features, more complex model



Diagnostic: Test versus Train Score.



Variance Diagnostic

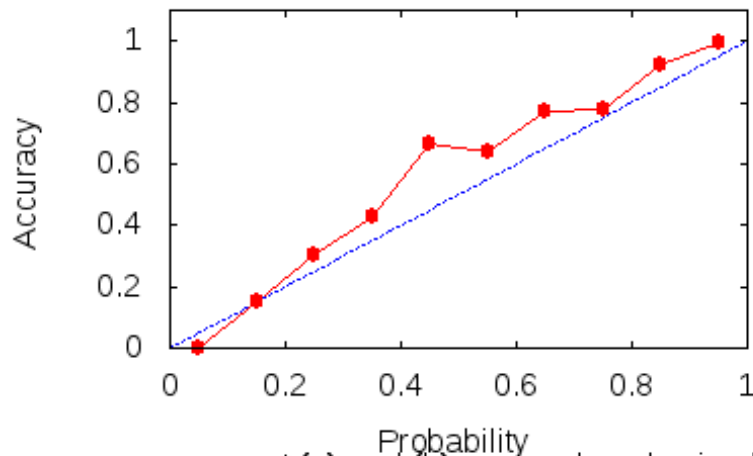
- Variance diagnostics.
 - Sample data set (k-fold cross validation)
 - Train on different folds.
- If the dev scores diff are small relative to your target error, you're OK!
 - If you're target error is 10%, and your variance $\sim 1\%$ fixing variance doesn't matter!
- If larger, too little data or algo. instability!



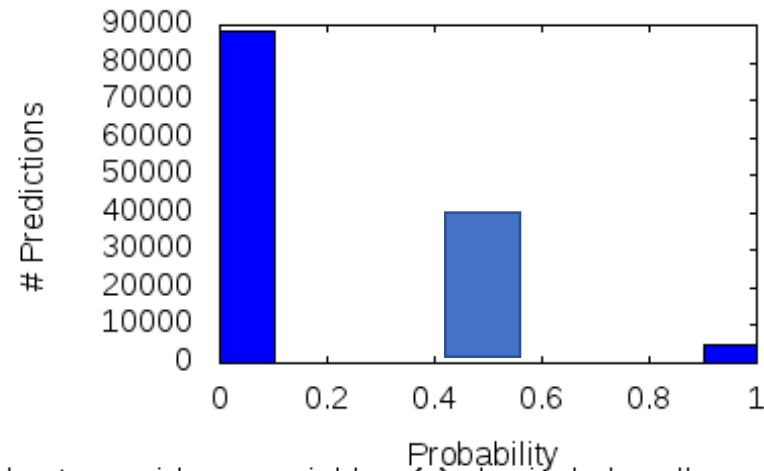
Diagnostic: Calibration Plots!

- Your spam detector uses logistic regression (or softmax last layer)

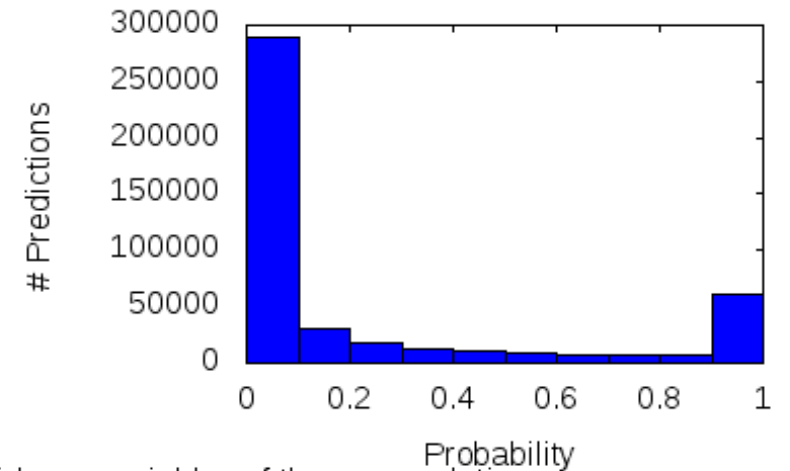
(a) Accuracy (Testing Set)



(b) # Predictions (Testing Set)



(c) # Predictions (Whole Set)



* (a) and (b) are produced using hold-out on evidence variables; (c) also includes all non-evidence variables of the same relation.

It's calibrated.

This bump means there is a lurking class!
Need more features. "Calibration Bump"

What could be wrong?

- Maybe it's the data or your features?

- Try getting more training data.
- Try a smaller set of features?
- Try adding more features?

Performance Gap

Performance Gap

Train = Test, Calibration bump

- Maybe it's the hyperparameters?

- Different value of regularizer?

- Try using a different model!

Really rough guidance

- If your test error is OK, good for now!
- Else, if $\text{train} == \text{test}$
 - Fix: you need a more complex model.
- If $\text{train} < \text{test}$ you're overfitting.
 - Fix: Regularize, less complex model
- If train oscillates wildly, you have a problem with your optimization algorithm.
- If train goes down lower with method A than method B, then prefer method A 😊



They're all just weights.

- Train another model on the same features.
 - SVM, logistic, even linear—as long as
- Suppose new model does better but you want to use the old model!
- You can plug in your new model into your old objective.
 - If loss is lower \rightarrow optimization problem!
 - If loss is higher \rightarrow model problem. (harder)
 - Examine where they differ can reveal capacity differences.



Diagnostics Summary

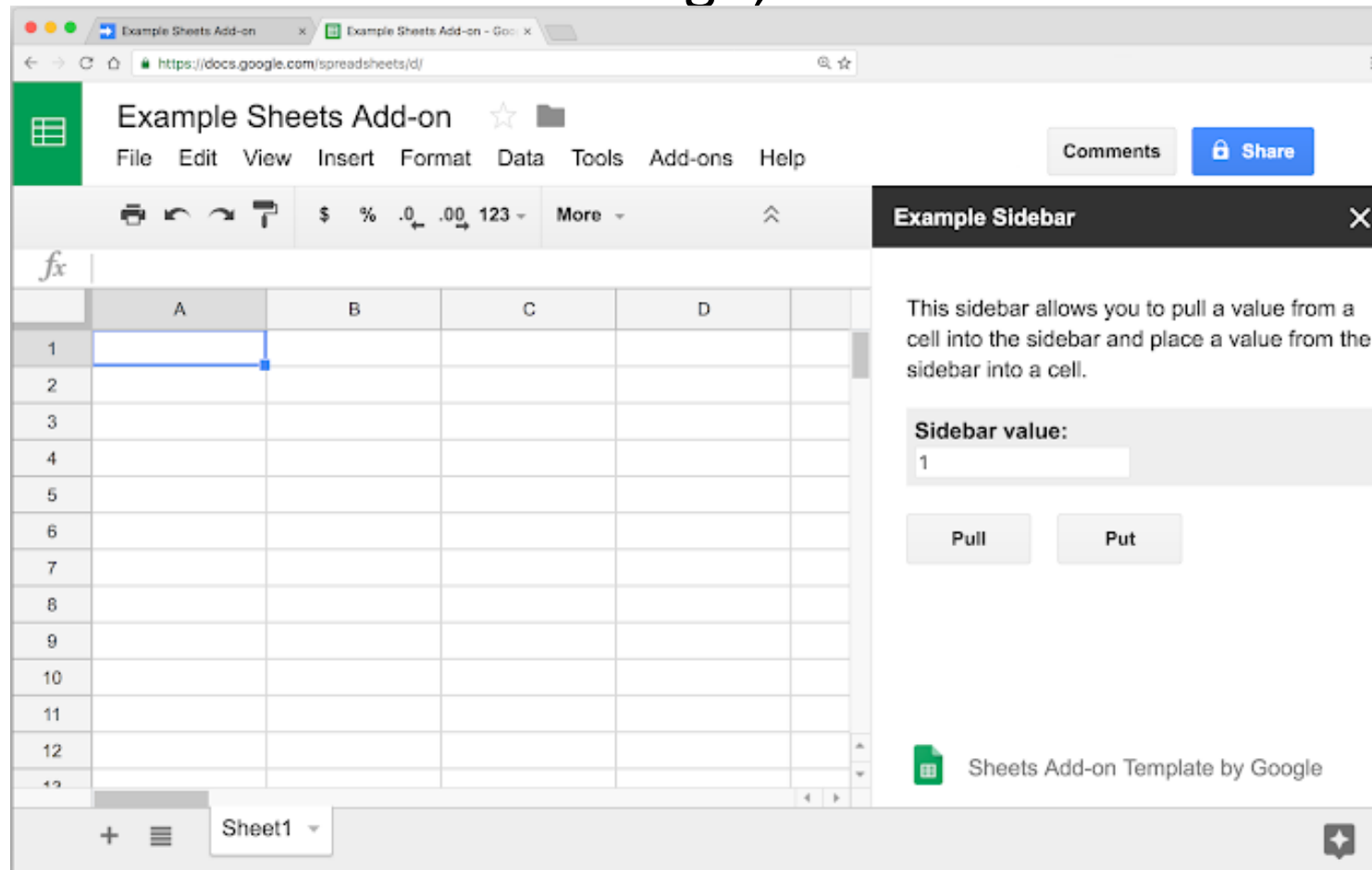
- Some I've used or seen teams use well.
- Cleverness to come up with your own.
- Think “unit testing”. It's engineering.



Ok, your model is working.
How do you improve it?

Should we add more features?

- So our train error is high, what to do now?



The screenshot shows a Google Sheets spreadsheet with a sidebar add-on titled "Example Sidebar". The sidebar contains the following text and controls:

This sidebar allows you to pull a value from a cell into the sidebar and place a value from the sidebar into a cell.

Sidebar value:

At the bottom of the sidebar, there is a small icon and the text "Sheets Add-on Template by Google". The spreadsheet itself has a grid with columns A, B, C, and D, and rows 1 through 12. The cell A1 is selected.



Labeling party!

Spoiler: It's a pipe.

- “**ground truth**” contains errors. GT was made..
 - Fix the specification..
 - Fix the data
 - At least report error bars!
- *If your error rate in GT is 3%, then your 1% change may not be meaningful.*



“This is not a pipe.”



The art of errors

- Split the error buckets into buckets such that there is some **systematic** information the model is missing.
- A good bucket for “relationship extraction”
 - “Her husband, Barack Obama,…”
 - “Her sister, Venus Williams…”
 - “His wife, Serena Williams…”
 - Aha! Missing “relationship name and appositive”
- It’s an art, if you can’t group buckets—you may be tapped out!
 - Convert high-level insight into features is an art and skill—practice it!

Slice-based Errors.

- Overall performance not as critical as important performance.
 - “Call mom” should work
 - More complex queries may be less expected.
- Record & scoreboard on these slices.
- Not all data equal! Monitor important data, but be careful how you draw statistical info here!



Selecting more labels

- It's all about sampling!
- Uniform Random Sampling
 - Advantage, you'll improve the error overall
 - Statistically meaningful.
- Importance-based sampling.
 - Can be cost effective—if your class only appears 1 of 10k times, would be expensive!
 - Pick near misses?
 - Don't use for evaluation by itself. Why?



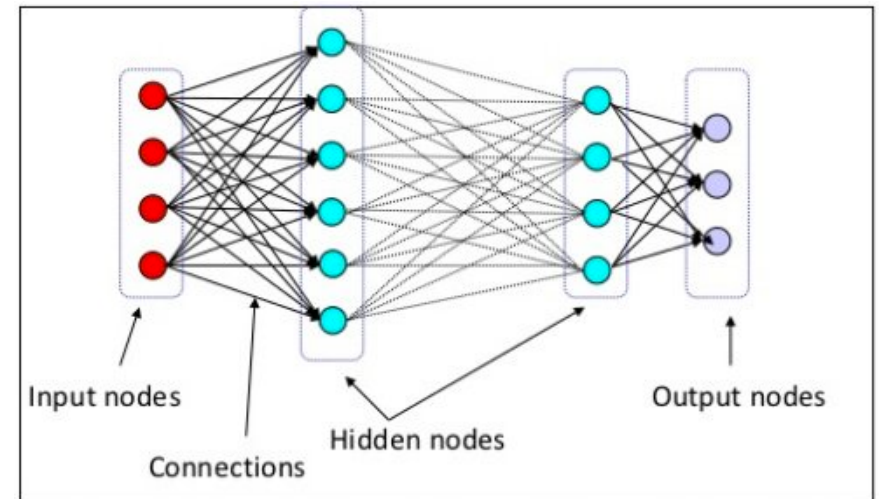


Labels Drift
(change) over time.

Automatic monitoring matters.
Adapting to customer taste change...

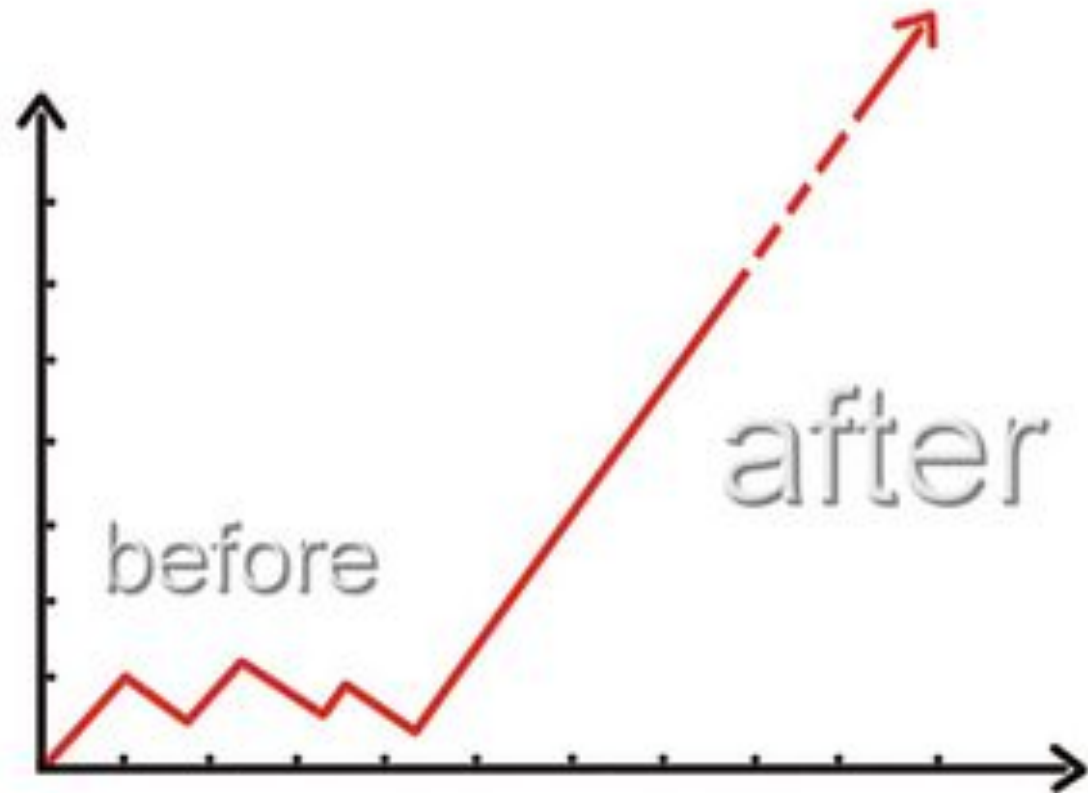
Error Analysis in the Era of Deep Learning

- Error bucketing is **still critical**.
- **Minor miracle**: you can add labels to drive model to predict the right class.
- Selecting the *right examples* is important.



Value of Baselines

- Someone will ask if your change is worth it, be prepared!
- If your fancy engine buys 0.1% but runs 1000x more slowly—you have a hard tradeoff!
- **Try to build simpler methods.** Often use deep models to “come up with features” to feed into logistic regression.



Ablation studies.

- You've built up a model, it has many different components.
- You want to know which matter and maybe which are stable!
- Remove one feature at a time!
 - NB: Adding features + baseline could overestimate overlap.
- Measure performance.
 - Critical for research.



Which features

- You derived some of the L1 technique (Lasso).
- Recall: Selects a sparse model weights..
- It enables you to select models, this changes how you build the models—often toss in many features, let it pick!
- You can freeze known good features, select among new features.



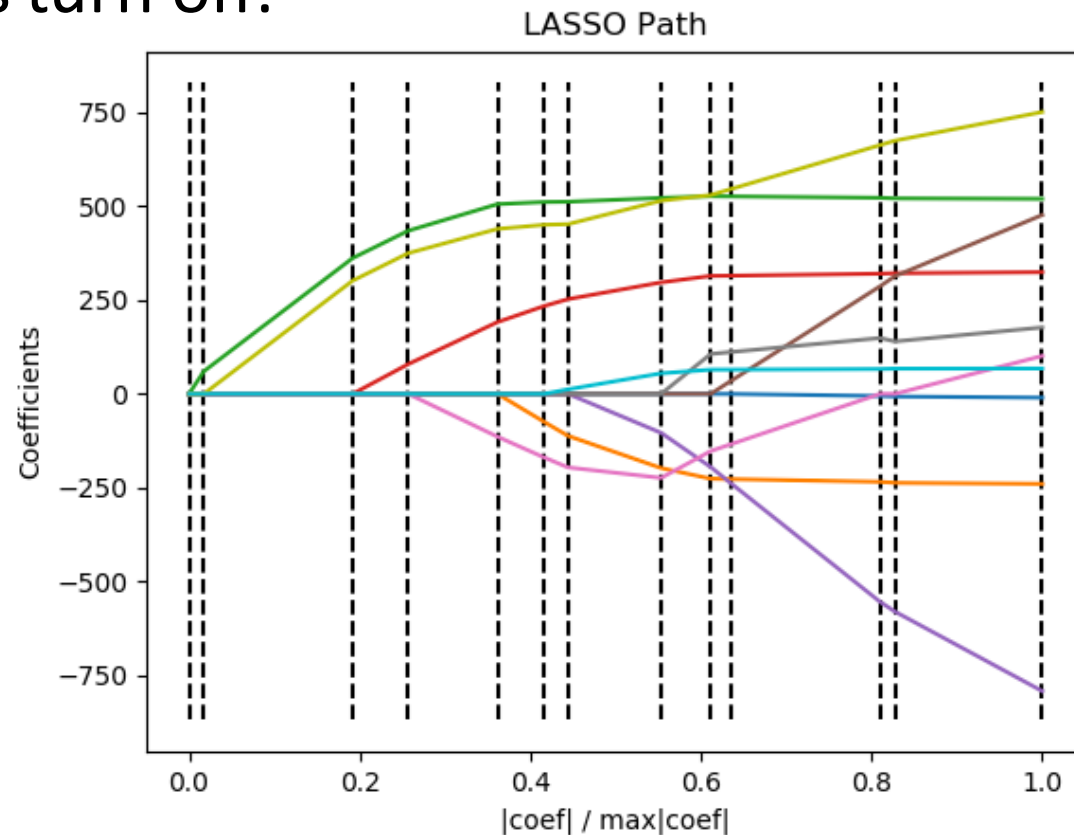
Lasso Path

Main idea: Sweep the regularization parameter for L1, train the model, see when features turn on!

LEAST ANGLE REGRESSION

BY BRADLEY EFRON,¹ TREVOR HASTIE,² IAIN JOHNSTONE³
AND ROBERT TIBSHIRANI⁴

Stanford University



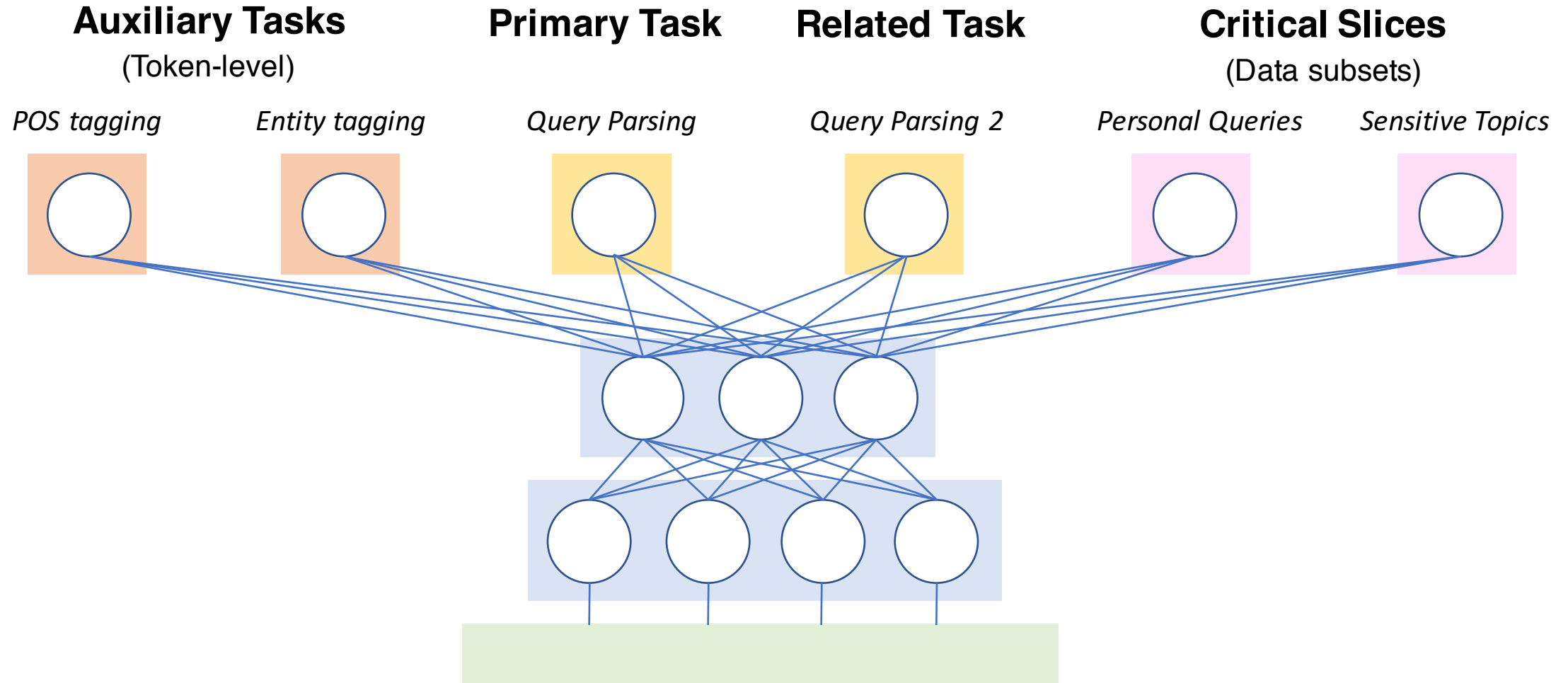
Useful to see how
valuable each feature
is: Great tool!

Last line of defense: Caches and Overrides!

- Keep in mind, ML helps you build software. It's usually not a goal in and of itself.
- ML is not infallible.
 - If you can write it easily, just do it!
 - If it makes a mistake, put it in a cache!
- Danger: you incur technical debt or you avoid fixing actual issues in your model.
- Use sparingly, but in any production system.
- Hot fixes!



Research: Massive Multi-Task Learning (MMTL)



Capitalizing on supervision at every level of granularity

Hidden technical debt of ML

Hidden Tech

Hidden Technical Debt in Machine Learning Systems

D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips
{dsculley, gholt, dgg, edavydov, toddphillips}@google.com
Google, Inc.

Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-François Crespo, Dan Dennison
{ebner, vchaudhary, mwyong, jfcrespo, dennison}@google.com
Google, Inc.

Hidden technical debt.

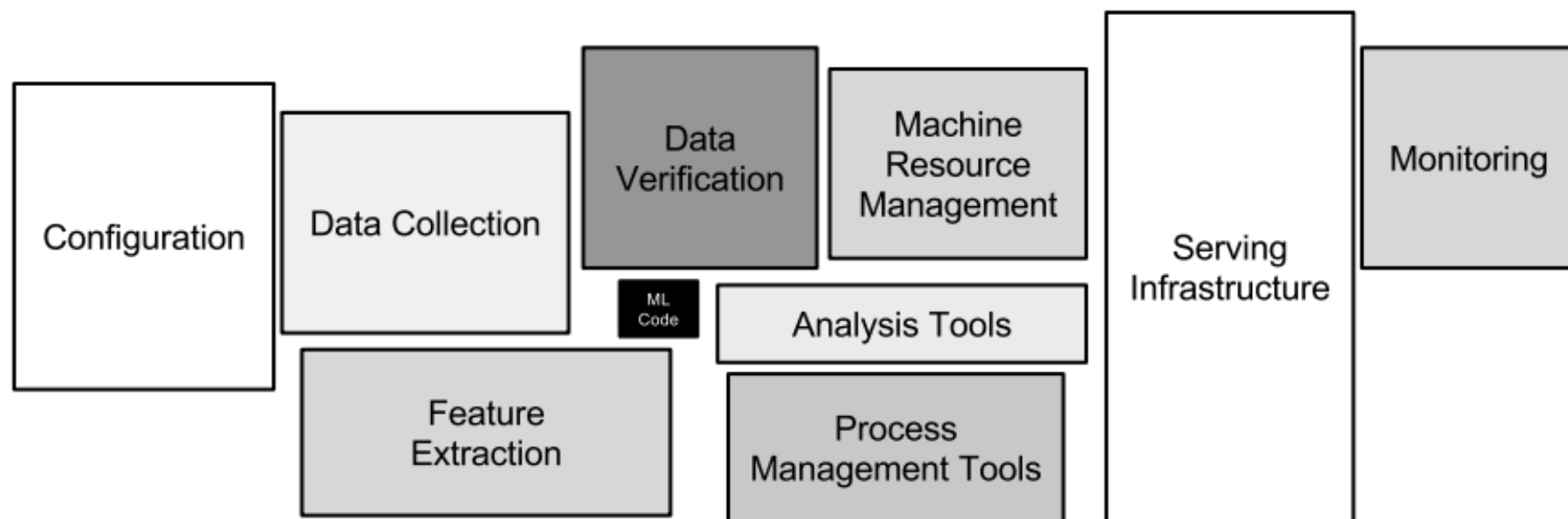


Figure 1: Only a small fraction of real-world ML systems is composed of the ML code, as shown by the small black box in the middle. The required surrounding infrastructure is vast and complex.

Code is nasty

- In conventional code, the person who wrote it usually knows why it works—but maybe no one else!
- In ML code, no one may know!



Hidden Benefit of Neural Nets

- Representation and normalization code is nasty. I've yet to see someone proud of it.
 - In a NN, you relearn it, and so don't have to maintain it.
- ML is eating software!
- This is called Software 2.0
 - Andrej Karpathy
 - Disclosure: We also work on this a lot!



Reproducibility

Reproducible, Reusable, and Robust Reinforcement Learning

Joelle Pineau

Facebook AI Research, Montreal
School of Computer Science, McGill University



Neural Information Processing Systems (NeurIPS)
December 5, 2018

Great talk! Highly recommend it
(Keynote last year—Kunle was great too!)

Reproducibility

- Your goal is to avoid fooling yourself.
 - It will be hard! You're clever!
- **Meaningless change causes a quality change:**
Random seeds shouldn't matter, but they lead to different outcomes!
- We separate train and test in an effort to not be wrong.
- No silver bullet, diligence everywhere.

HOW NOT TO
BE WRONG



THE POWER *of*
MATHEMATICAL THINKING

JORDAN
ELLENBERG

Summary

Summary

- Measure twice, cut once. Don't bash, try to setup diagnostics.
 - Ideally in code! You want to reuse these!
 - Canada has been ahead on aspects of [learning](#)
- Look at your data and your predictions. No substitute.
- ML systems are used to make it easier to write code, it's a high-interest credit card of technical debt.