

# ML Advice (Clipart day)

Chris Ré

*with help from Alex Ratner*

# Disclaimer

- This lecture is filled **with personal opinion** informed by building production and clinical prototypes (and research too!).
- It is high level and presents some difficult, raw material.
- **Improve over time:** include ideas folks have told me were helpful to them.

# Disclaimer



- Will describe errors that I and collaborators have made or pointed out.
- **Goal is NOT** to cast aspersions but to get to better practice.
  - Many of these folks are my intellectual idols.
  - The worst errors are my own!

# 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.



7 Steps of ML Systems.

# The 7 steps Overview



- Step 1: Acquire Data
- Step 2: Look at your data\* -- after every step.
- Step 3: Create train/dev/test splits
- Step 4: Create/Refine a specification
- Step 5: Build model (simplest that works!)
- Step 6: Measurement
- Step 7: Repeat.

Step 1: Acquire Data

# You need **realistic** spam (and not spam).

---

- Ideal data sampled from the data your SPAM product will be run on.
- Ideal not always available.
  - **Cold-start.** Feature doesn't exist yet!
  - Legal/ethical issues to look at data.
- You will get it wrong on 1<sup>st</sup> try.



# Data Artifacts are hard.

Any model may pick out unintended signal.  
Modern, deep models may pick out *more* unintended signal.



**Upshot:**  
Picked up on  
*mascara*

Kuehlkamp et al. *Gender-from-Iris or Gender from-Mascara*

Hidden data artifacts are very challenging!

Step 2: Look at the data

# Look at your data.

---

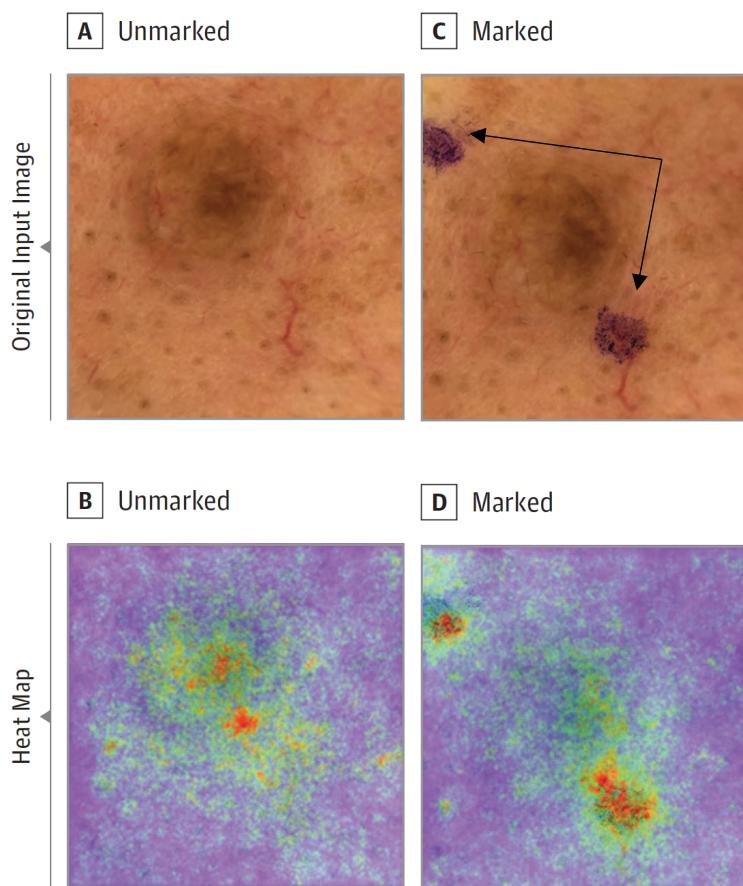
- You have some spam, look it at it!
- If needed, build tools to look at your data.
  - Spam from Europe different than from Africa? from US?
  - Spam to .edu different than .com?
- “Become one with the data” – Karpathy.
- Do this at every stage!



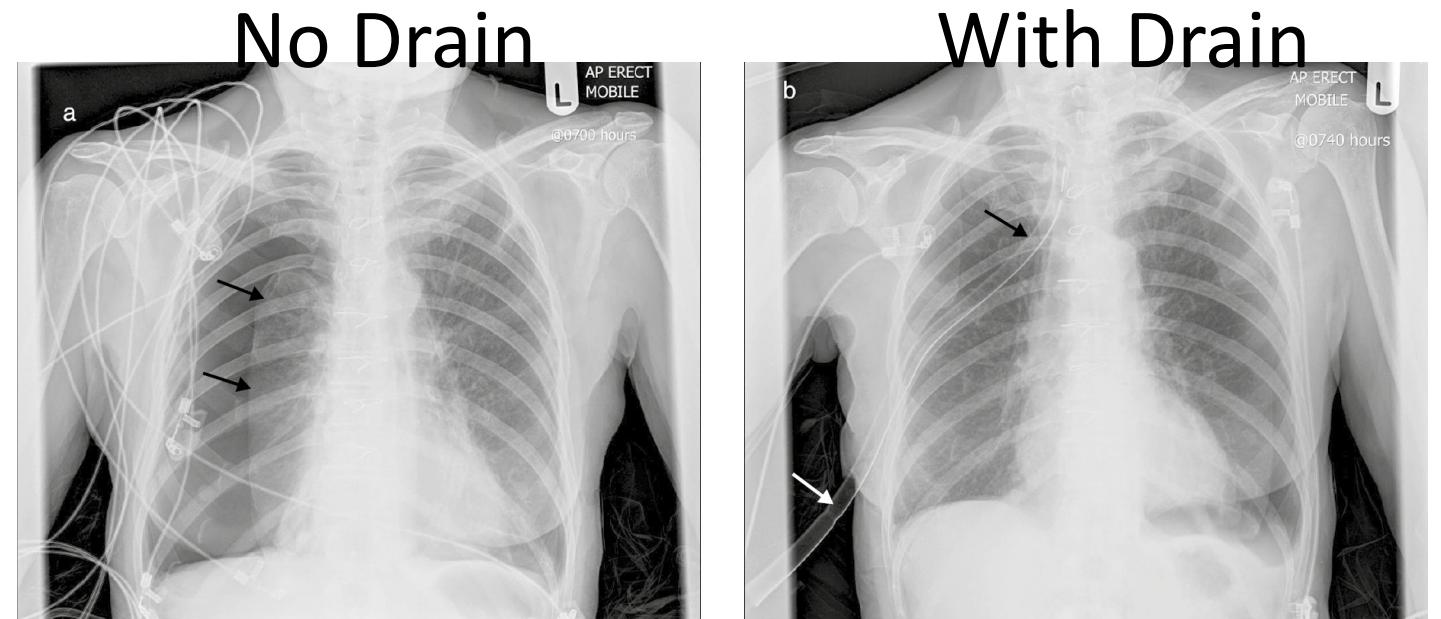
Have the right people look at your data

# Expertise is often required!

## Melanoma Recognition (Surgical Marks)



## Pneumonia Detection (Chest Drains)



Pneumothorax has 0.94 AUC—with chest drains—but 0.77 without... ***Chest drain means already treated!***

### Image Credits

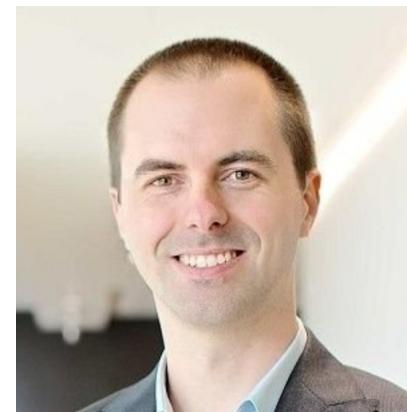
Valchanov, Kamen, Nicola Jones, and Charles W. Hogue, eds. *Core Topics in Cardiothoracic Critical Care*. Cambridge University Press, 2018.  
Winkler, Julia K., et al. "Association between surgical skin markings in dermoscopic images and diagnostic performance of a deep learning convolutional neural network for melanoma recognition." *JAMA dermatology* (2019).

<https://lukeoakdenrayner.wordpress.com/2019/10/14/improving-medical-ai-safety-by-addressing-hidden-stratification/>

For more detail see...



Gustavo Carneiro (Adelaide)



Luke Oakden-Rayner (Adelaide)

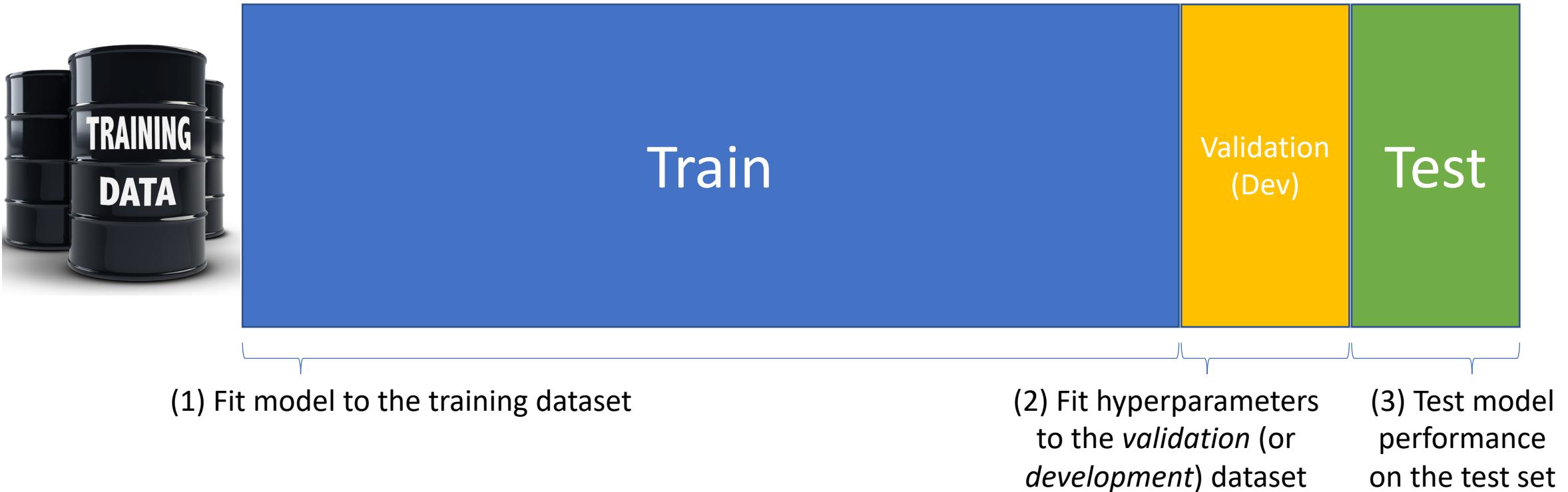


Jared Dunnmon (Stanford)

## Hidden Stratification Causes Clinically Meaningful Failures in Machine Learning for Medical Imaging

## Step 3: Train/Dev/Test Split

# Partitioning Data: Train, Test, and Validation



Critical to avoid leakage / adaptive overfitting

# What makes a good split?

- **Ideal:** Train, test, & dev randomly sampled
  - Allows us to say train quality is approximately test quality.
- But, test is a **proxy** for the real world!
  - We'll talk more about this later...
- **Challenge:** Leakage.
  - (Nearly) same example in train and dev.
  - Causes performance to be overstated!
    - Eg., same senders in train and test?



# Random isn't the only goal.

**Given:** 50 company stock price for 100 days

- Triples of the form (Company, day, price.)

**Goal:** predict stock prices (Ahead of time for \$)

What happens if you **randomly** split into halves?

- Train contains { (GOOG, day 10, 126), (GOOG, day 12, 1275) }
- Test contains { (GOOG, day 11, 1270) }

**Better split:** Use first 50 days to predict last 50 days?

- More later...



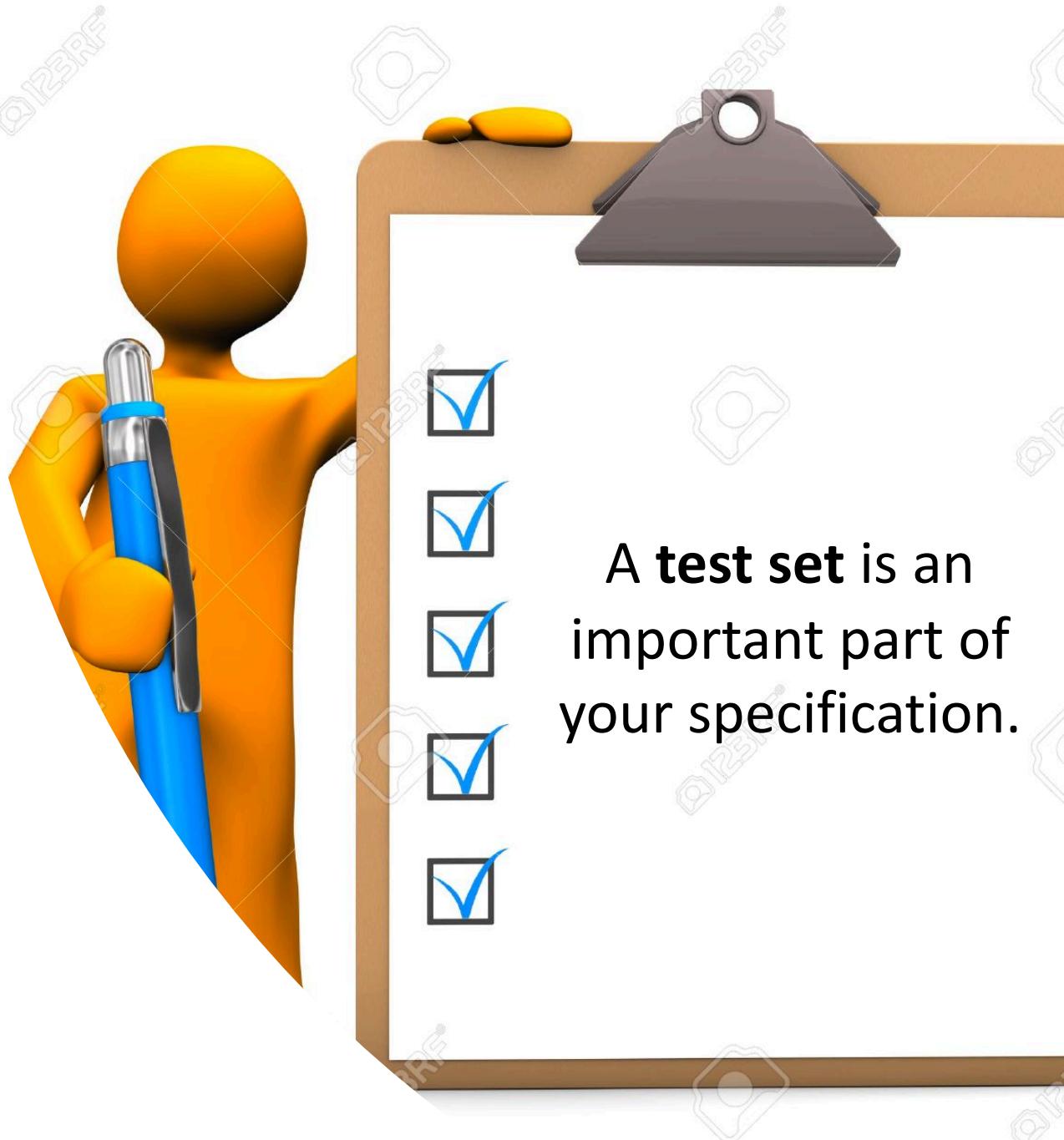
***Our split doesn't reflect  
our predictive goal!***

Step 4: Create a specification

# 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.
  - What level of expertise is required to understand it?
- Your specification **must** be embodied in a set of examples. **A test set!**



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

# Quick and Dirty Test: Inner Annotator Agreement

---

- You write down your notion of SPAM. Select N randomly selected examples. Give to three different annotators.
  - **Inter-annotator agreement.** How often do annotators agree?
- Let's say annotators agree 95% of the time—great!
  - **The meaningless accuracy problem.** *Can you build a product with greater than 95% accuracy against this spec?*
- Examine the spec & disagreement cases.
  - Train annotators or change spec?
  - If humans can't agree, the machine is going to have trouble...
- Do you train the people or the machine? (serious Q)



Subtle  
Problem:  
*Consistency in  
test sets*

Consider following protocol.

- Every day, sample data
- Send examples to crowd
- Get grades back

# Subtle Problem: Spec creep



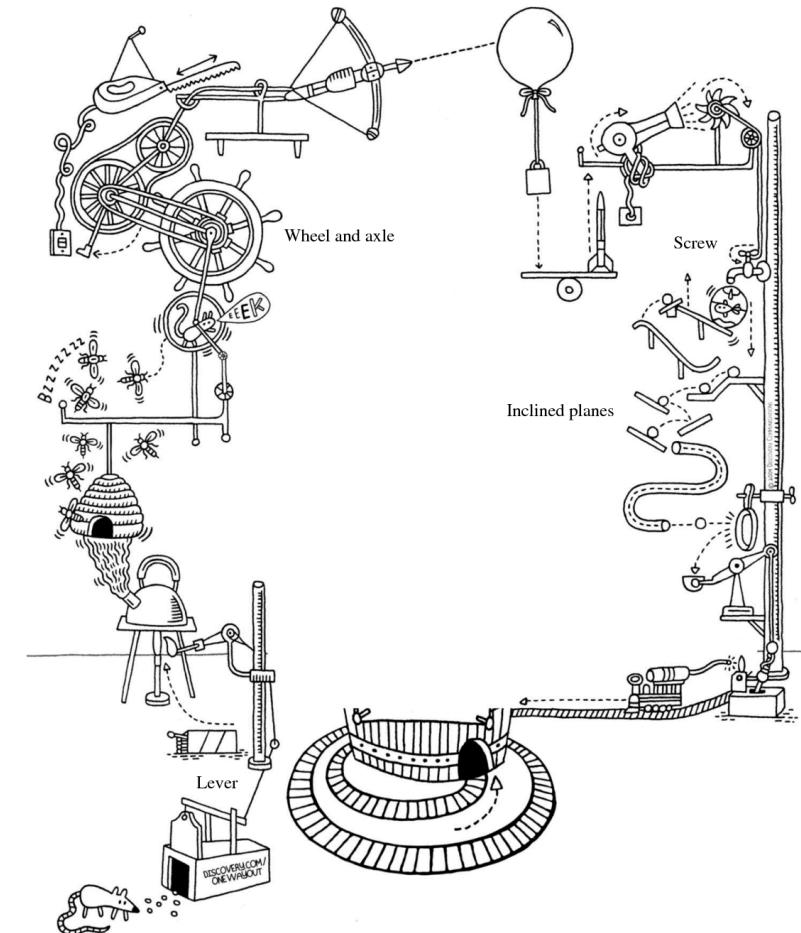
- **Warning:** Tempted when answer is “*pretty good*” to say yes—for classification this must be precise or you accrue rot.
  - **Label rot—not debt** hard to find once there! You should clean it up!
- **Spec:** Unsolicited drug ads should be marked drug spam.
- Spam for vitamins comes in—and is successfully filtered.
  - This answer is marked as drugs and filtered.
  - Answer is useful, but allowing it may cause **scope creep** (are vitamins drugs?)
  - If you yes, revise the specification!

Step 5: Implement simplest possible model

# Keep it simple!

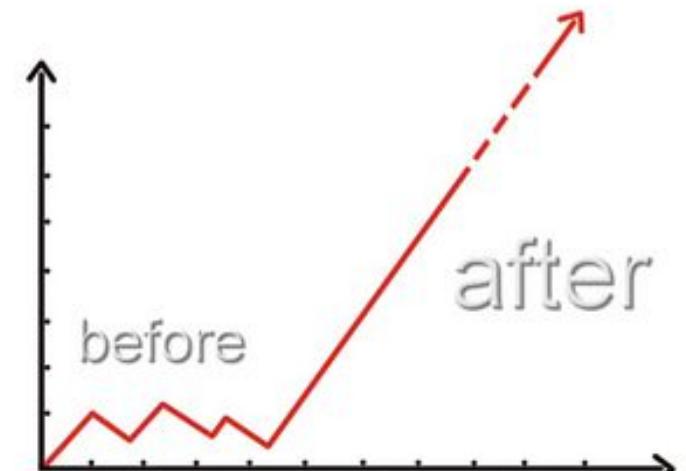
- If python code can get 90% accuracy, use it!
- If python code can get 100% accuracy, use it!
- ML shines on a class of problems:
  - can be **precisely specified** but
  - Writing down program is prohibitively hard
- **Avoid getting bogged down in new models, use them to understand the data!**
  - Models can be tools (scaffolding), not just end goal.

Rube Goldberg Design Project



# 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... or needs 1000x more data...
- *Build simpler methods **even after** fancy models.* Often use deep models to “come up with features” – by looking at output!
  - Models can be a tool to understand data!



# Ablation studies.

- You've built up a model, it has many different components.
  - Which matter?
  - which are stable?
- Remove one feature at a time!
  - *Adding* features + baseline could overestimate overlap. How?
- Measure performance.
  - Critical for research!



Step 6: Measure the output!

# Simple descriptive dashboards

---

- **Challenge:** Don't make the same mistake twice!
  - Measure end-to-end quality metrics.
- **Challenge:** catch new mistakes, asap.
  - Harder!
  - 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?



# Avoid Known mistakes Slice-based Monitoring.

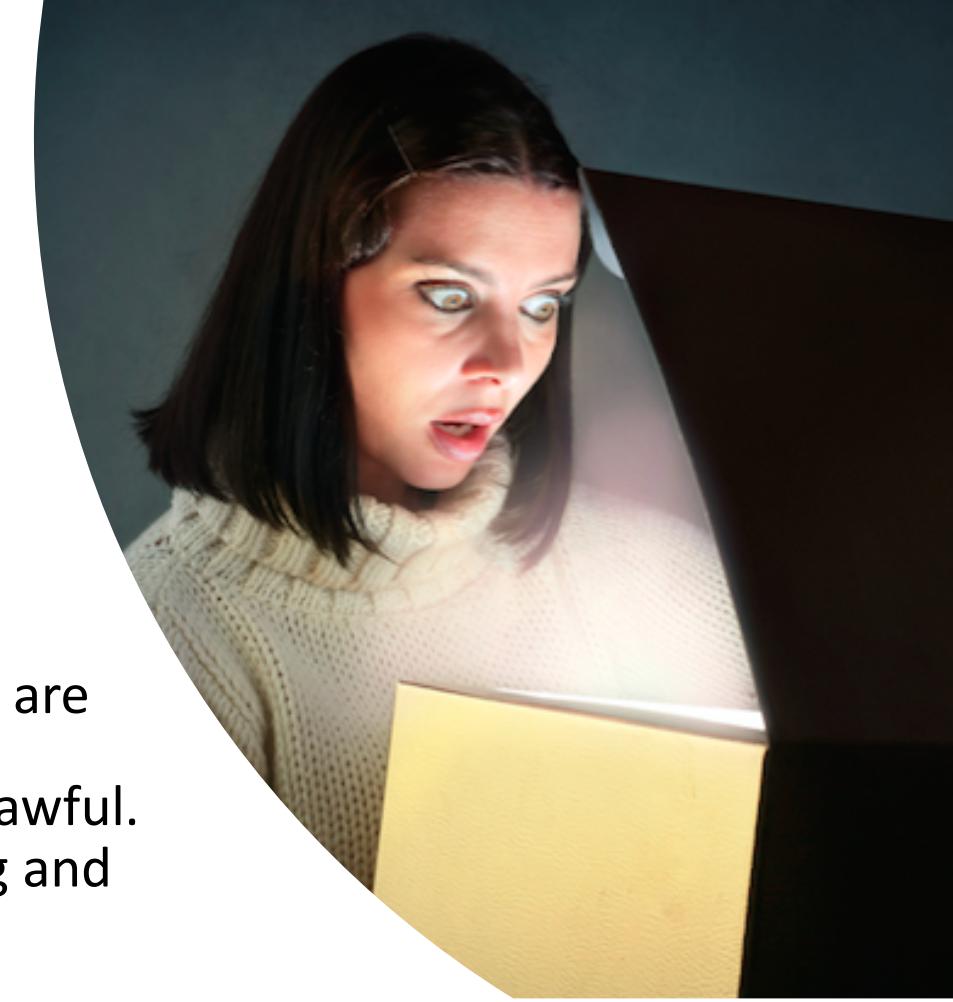
- Overall performance may not be as critical as important “slice”.
  - “Call mom” should work
  - More complex queries may be less expected.
- Record & scoreboard on these slices.
- Your monitoring should have support for fine-grained reporting!



# Challenge: Avoid *unknown* mistakes

---

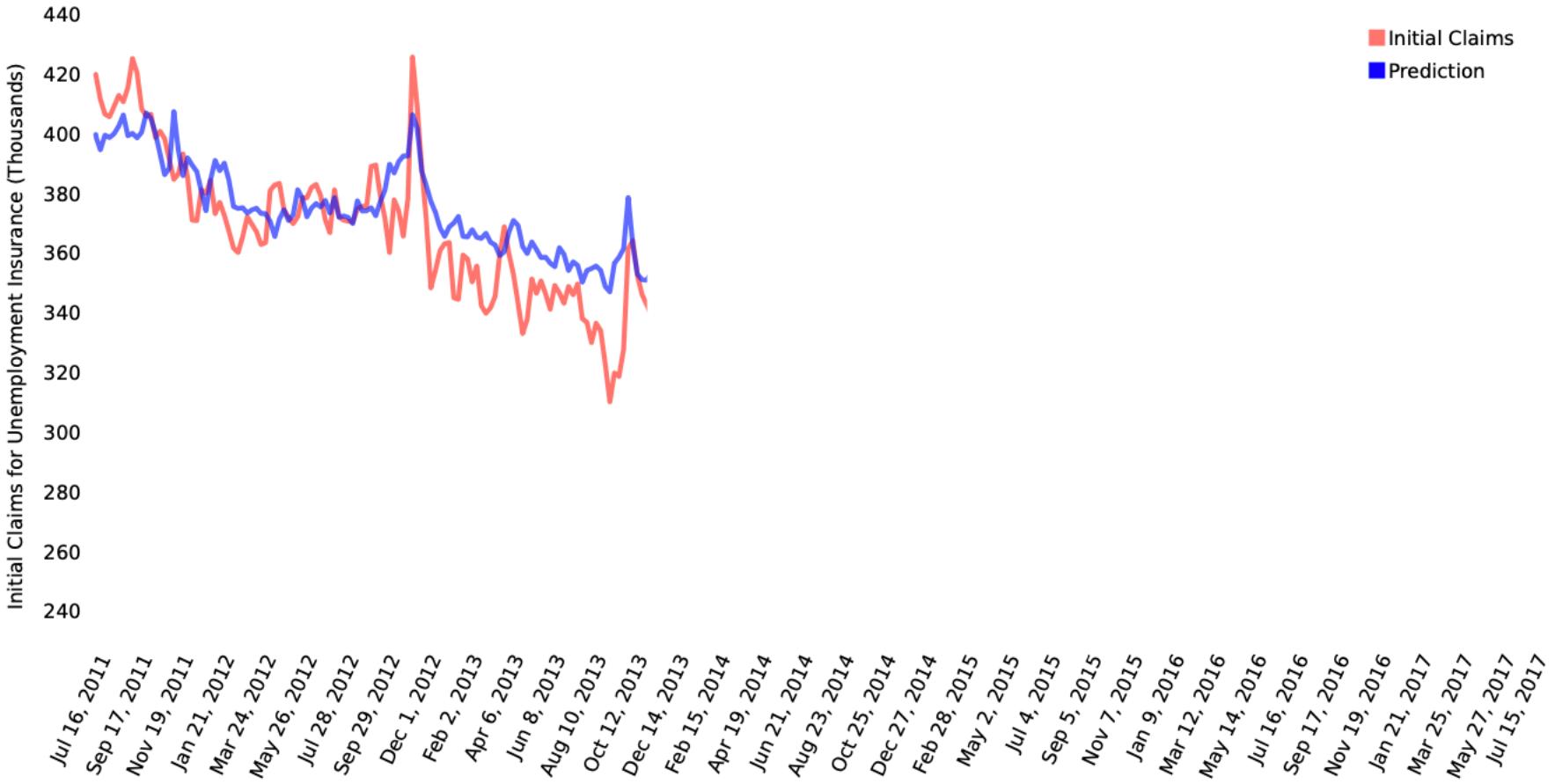
- Distribution shift is a real issue.
  - **Popularity Shift/Cold-start.**
    - You release a feature, queries that weren't popular are now **very popular**.
    - Old score says "we're great", but felt experience is awful.
    - **Remedy:** Hopefully, you knew this slice was coming and you monitored it proactively.
  - **Input Shift.** Your input changes in some way.
    - Much harder to catch in my experience...
    - More next!



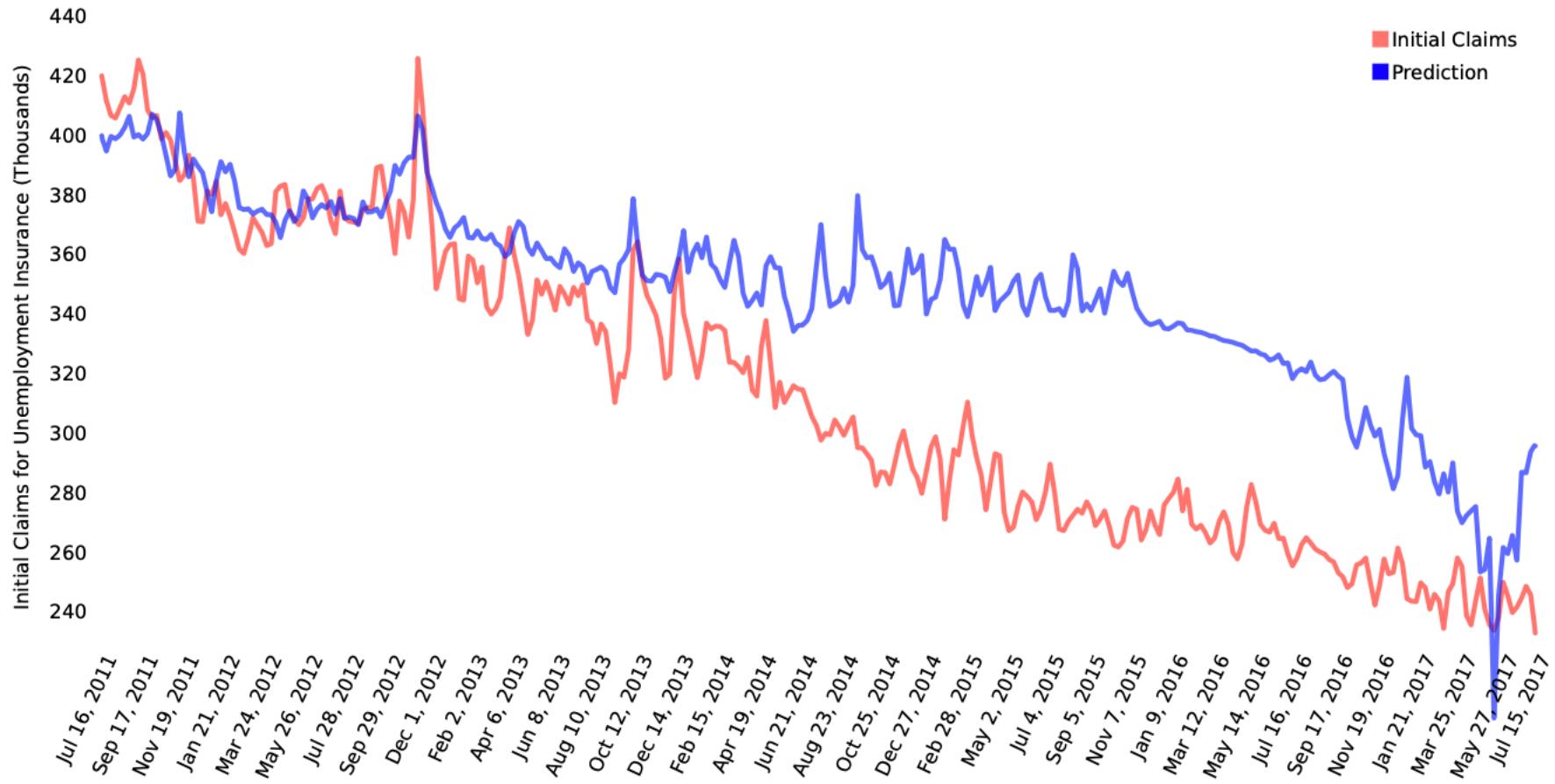
This is incredibly hard! No ideal solutions in industry...

*I've gotten this **very** wrong...*

## Predicting unemployment claims from Twitter (at time of publication)



## Predicting unemployment claims from Twitter (post-publication)



What went wrong?!?

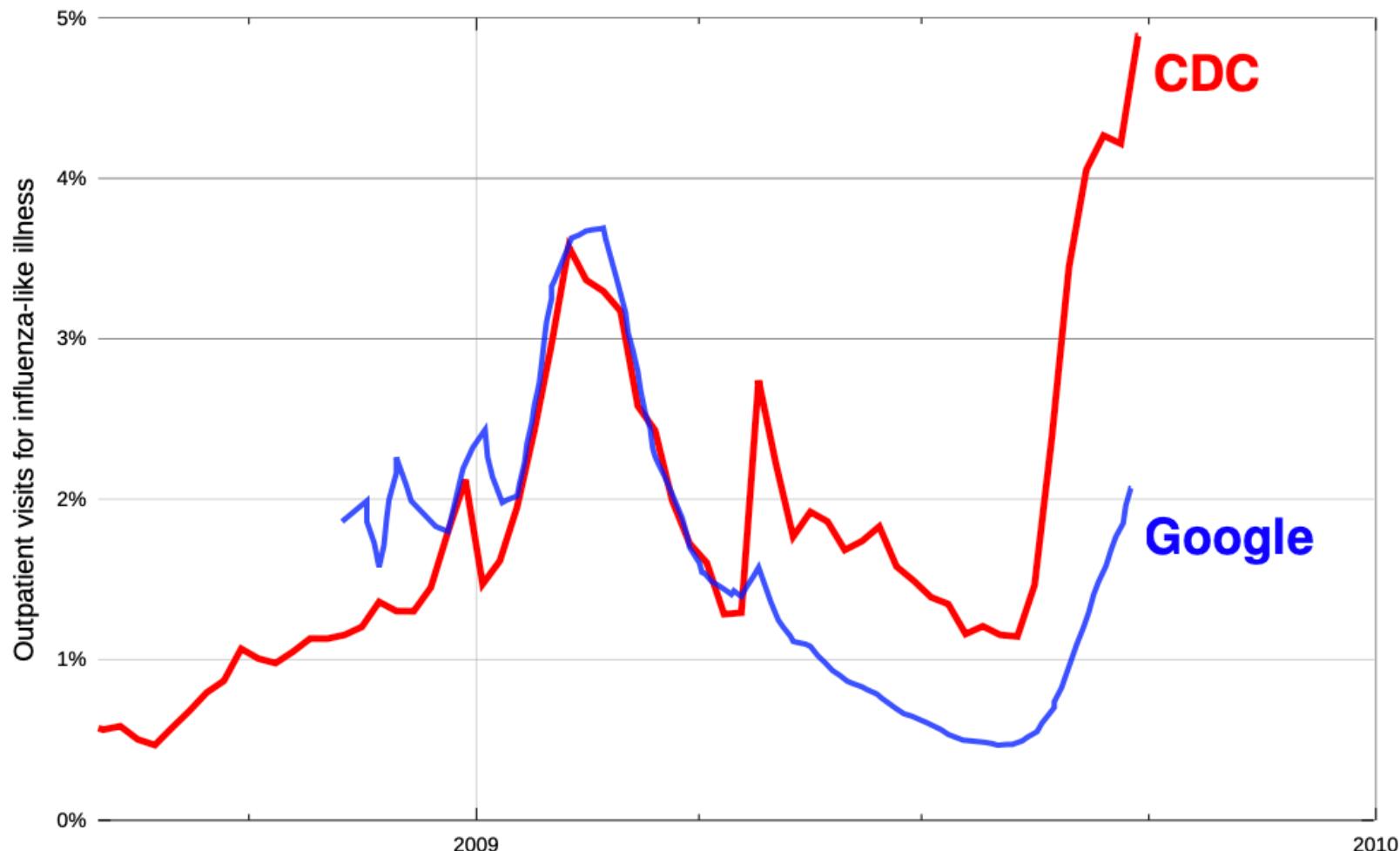
# Postmortem: Three identified issues—many remain!

---



- Twitter changed the text filtering model reflecting business priorities.
  - This is **common** in most systems! How do we control for it?
  - *Cope retraining—we imperfectly understand substrate and it can change!*
- Model sensitive to outliers “**Gray swans**”
  - Mined keywords like “*pink slip*” were used as part of an advertising campaign.
  - *Median regularization to deal with feature spikes.*
- Economic reality changed in 2014
  - Losing a job meant you quickly got a new job—stale more quickly.
  - *Used PCA to help diagnose changes...*

## Performance in the first year





Labels & Input Drift  
(change) over time.

---

Automatic monitoring matters.

Adapting to customer taste change.

This is **REALLY** hard

Step 7: Repeat! (And Look at your data!)

# The 7 steps recap

---



- Step 1: Acquire Data
- Step 2: Look at your data\* -- after every step.
- Step 3: Create train/dev/test splits
- Step 4: Create/Refine a specification
- Step 5: Build model (simplest that works!)
- Step 6: Measurement
- Step 7: Repeat.

I have yet to see anyone get preceding steps right on first try. *Build quickly and iterate.*

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

# More detailed version

- Specification Challenges
- Model Training and Performance Diagnostics
- Monitoring Challenges
- Issues in Shipping to Production/Use

# Specification

# Types of Errors in Specification

- Class schema issues
  - Two distinguishable classes merged as one
  - One class split into two now indistinguishable class
- Unknowable class.
  - Information to distinguish two different cases is not available to the model
  - More common than you would think!
- Unrealized structure between classes
  - For example, fine-grained errors may not count as much as coarse-grained errors.
- Test set label variance
  - If this (e.g. inter-annotator disagreement) is > the error deltas being tested for, nonsensical!
- Change between test set versions
  - Test sets need to be regularly “refreshed”- need to watch for changes between versions!
  - More later...

# Types of Errors in Specification

- Class schema issues
  - Two distinguishable classes merged as one
  - One class split into two now indistinguishable class
- Unknowable class.
  - Information to distinguish two different cases is not available to the model
  - More common than you would think!
- Unrealized structure between classes
  - For example, fine-grained errors may not count as much as coarse-grained errors.
- Test set label variance
  - If this (e.g. inter-annotator disagreement) is > the error deltas being tested for, nonsensical!
- Change between test set versions
  - Test sets need to be regularly “refreshed” - need to watch for changes between versions!

# 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 (look at diagonal)—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. (duplicate names for a concept)
  - Subtle distinctions are good!
    - **Crisp.** If they manifest differently in data and we can define this difference.
    - AND we have enough data.

# Types of Errors in Specification

- Class schema issues
  - Two distinguishable classes merged as one
  - One class split into two now indistinguishable class
- Unknowable class.
  - Information to distinguish two different cases is not available to the model
  - More common than you would think!
- Unrealized structure between classes
  - For example, fine-grained errors may not count as much as coarse-grained errors.
- Test set label variance
  - If this (e.g. inter-annotator disagreement) is > the error deltas being tested for, nonsensical!
- Change between test set versions
  - Test sets need to be regularly “refreshed” - need to watch for changes between versions!

# Spoiler: It's a pipe. *(Your Ground Truth Contains Errors)*

- “ground truth” is constructed.
  - Fix the specification..
  - Fix the data
  - *It is a curated resource!*
- **Measure Error!** If your error rate in GT is 3%, then 1% change may not be meaningful.



“This is not a pipe.”

# Look at the data! Nothing fancy.

- Simple tool....

A screenshot of a Google Sheets interface. The title bar says "Example Sheets Add-on". The menu bar includes File, Edit, View, Insert, Format, Data, Tools, Add-ons, and Help. A sidebar titled "Example Sidebar" is open, containing the text: "This sidebar allows you to pull a value from a cell into the sidebar and place a value from the sidebar into a cell." Below this is a "Sidebar value:" input field with the value "1", and two buttons: "Pull" and "Put". The main spreadsheet area shows a single row with columns A, B, C, and D. Cell A1 contains the value "1". The bottom right corner of the spreadsheet has a link: "Sheets Add-on Template by Google".



Labeling party!

# 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!

# Selecting more labels

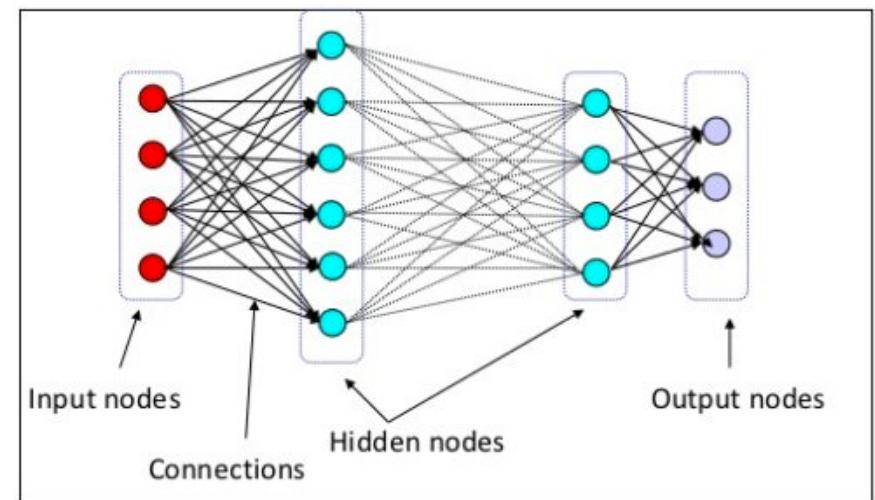
---

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



# Error Analysis in the Era of Deep Learning

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



# Types of Errors in Specification

- Class schema issues
  - Two distinguishable classes merged as one
  - One class split into two now indistinguishable class
- Unknowable class.
  - Information to distinguish two different cases is not available to the model
  - More common than you would think!
- Unrealized structure between classes
  - For example, fine-grained errors may not count as much as coarse-grained errors.
- Test set label variance
  - If this (e.g. inter-annotator disagreement) is > the error deltas being tested for, nonsensical!
- Change between test set versions
  - Test sets need to be regularly “refreshed” - need to watch for changes between versions!

# Do CIFAR-10 Classifiers Generalize to CIFAR-10?

Benjamin Recht  
UC Berkeley

Rebecca Roelofs  
UC Berkeley

Ludwig Schmidt  
MIT

Vaishaal Shankar  
UC Berkeley

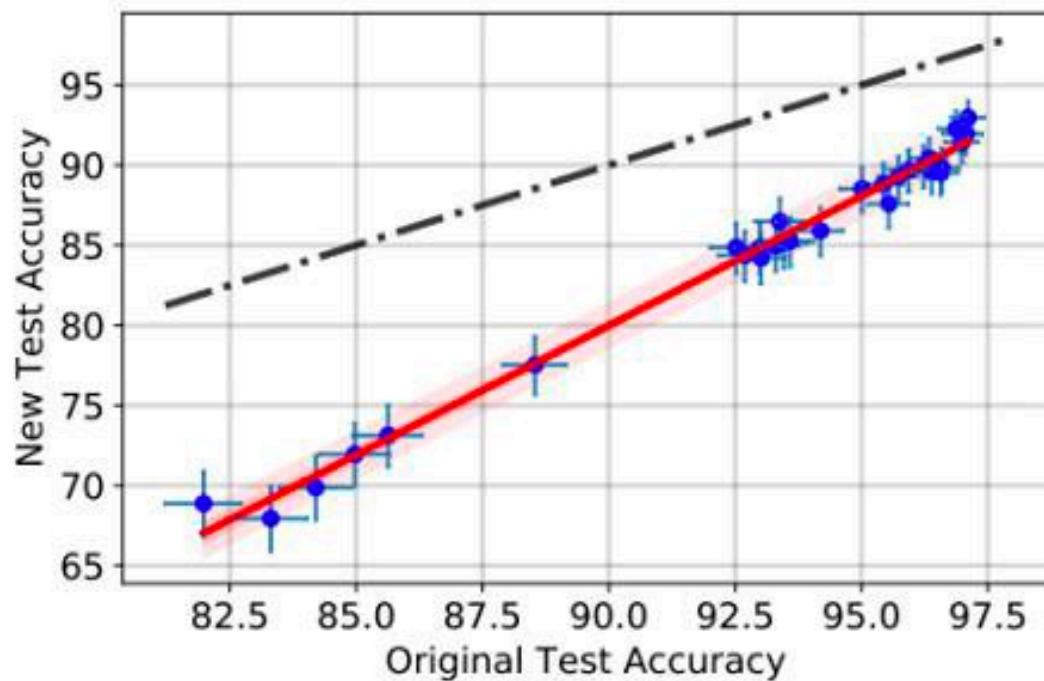
June 4, 2018

# Do ImageNet Classifiers Generalize to ImageNet?

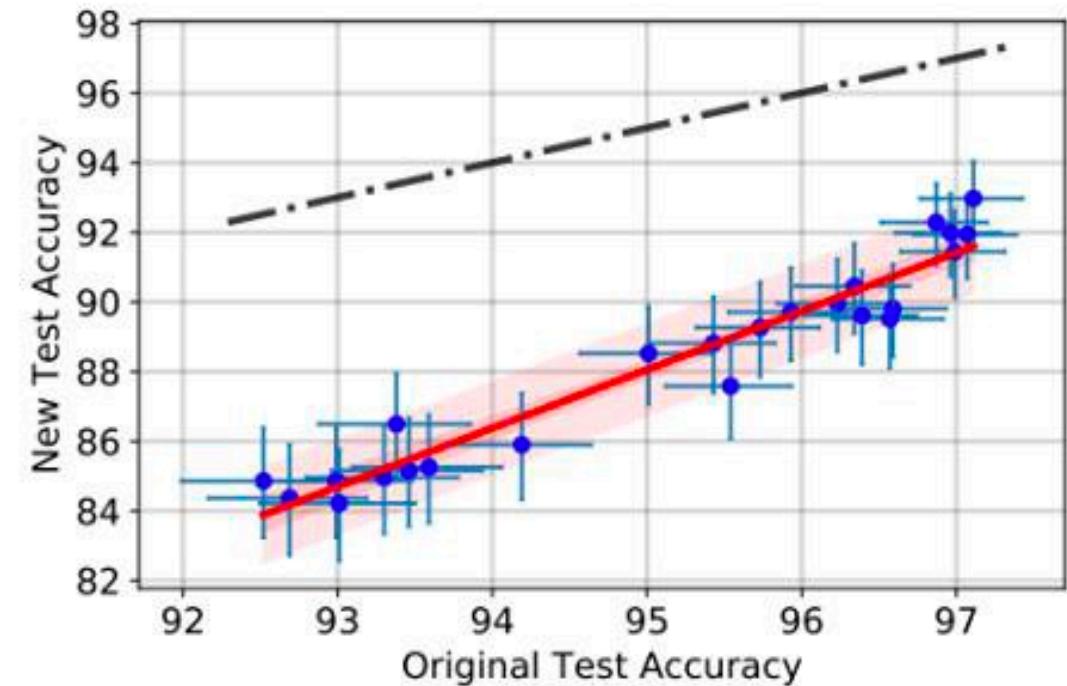
What if we sampled a new test set according  
to the same specification as the original?

*Expect the same outcome!*

# No real evidence of adaptive overfitting!



(a) All models

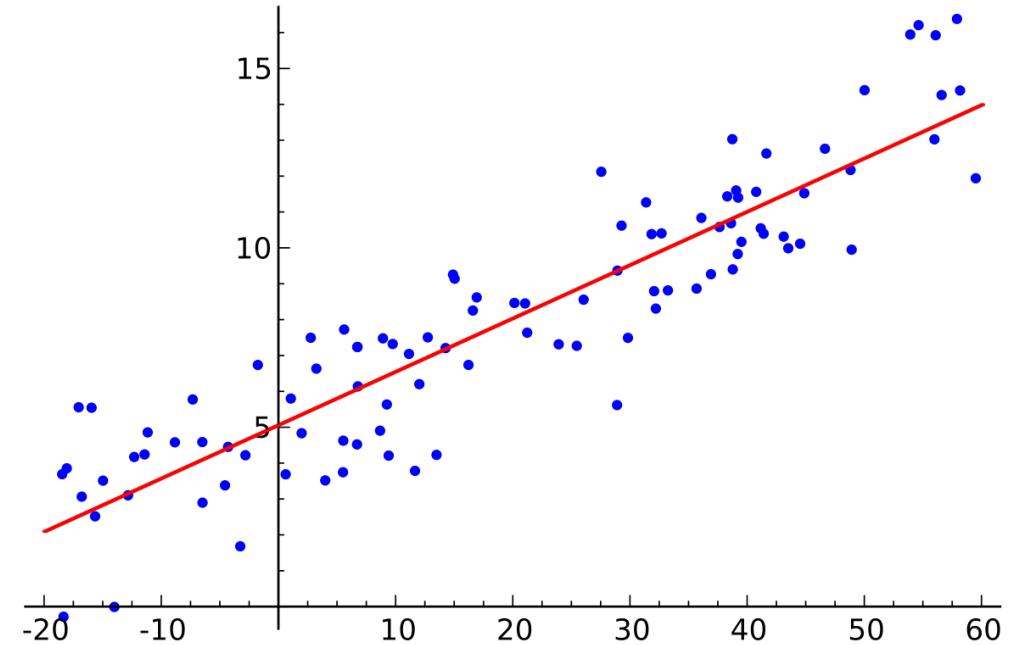
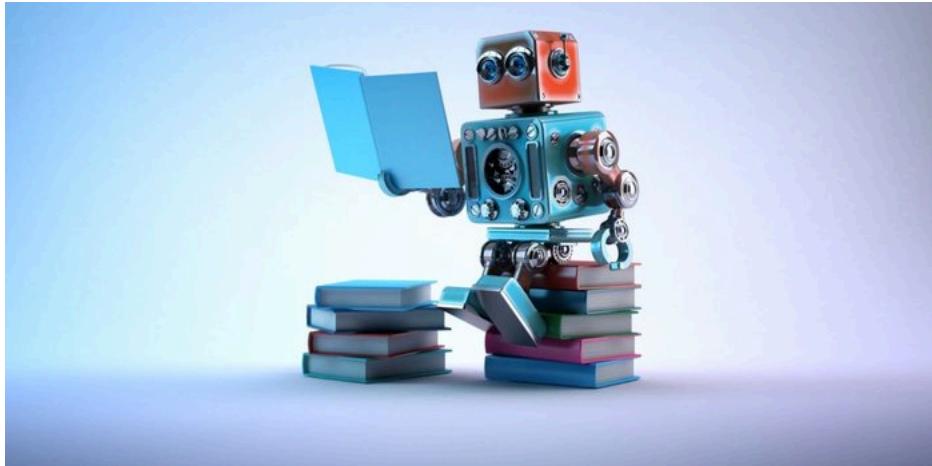


(b) High accuracy models

But specification and distribution shift hit you! Same on ImageNet

# Model Diagnostics

# 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/model 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, its 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?'**

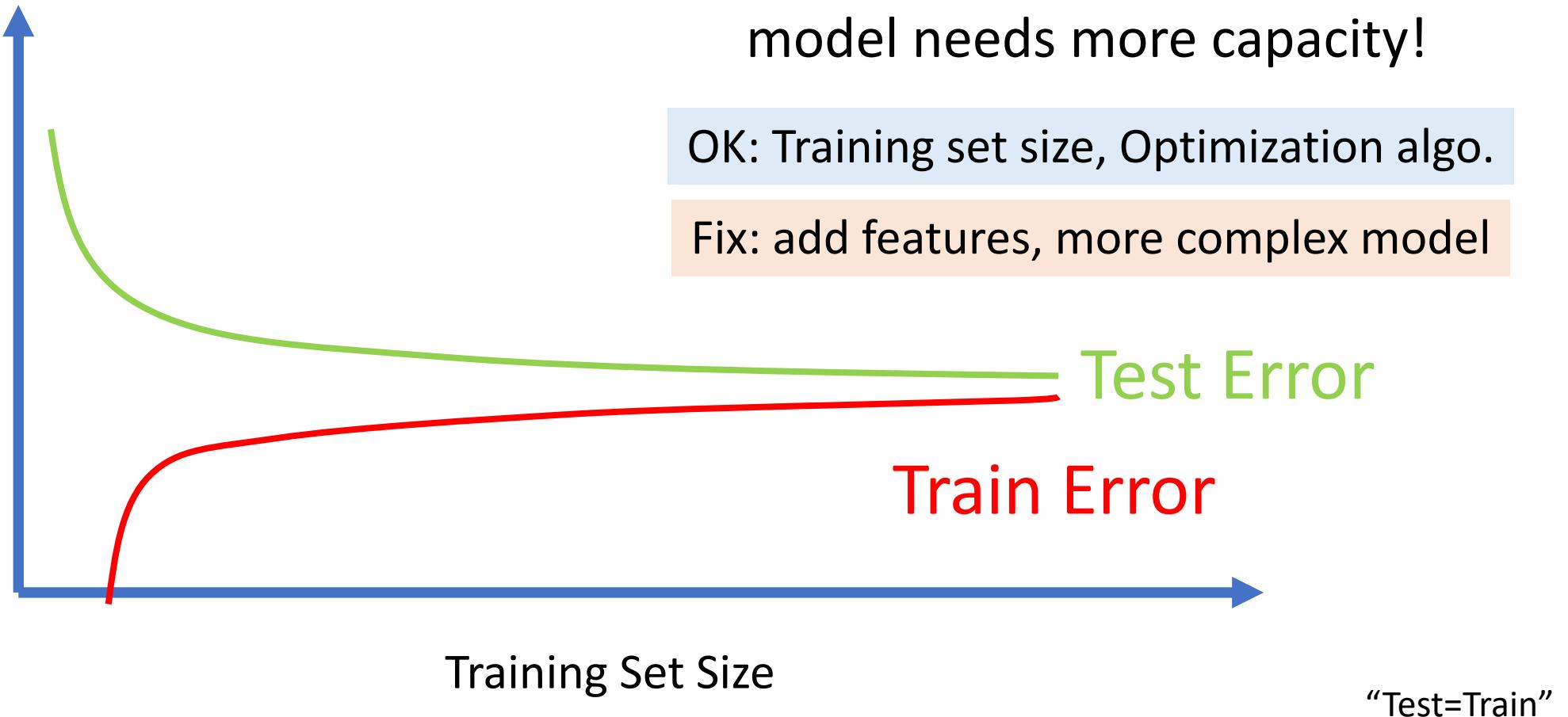
# Just like compiling!

- 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!

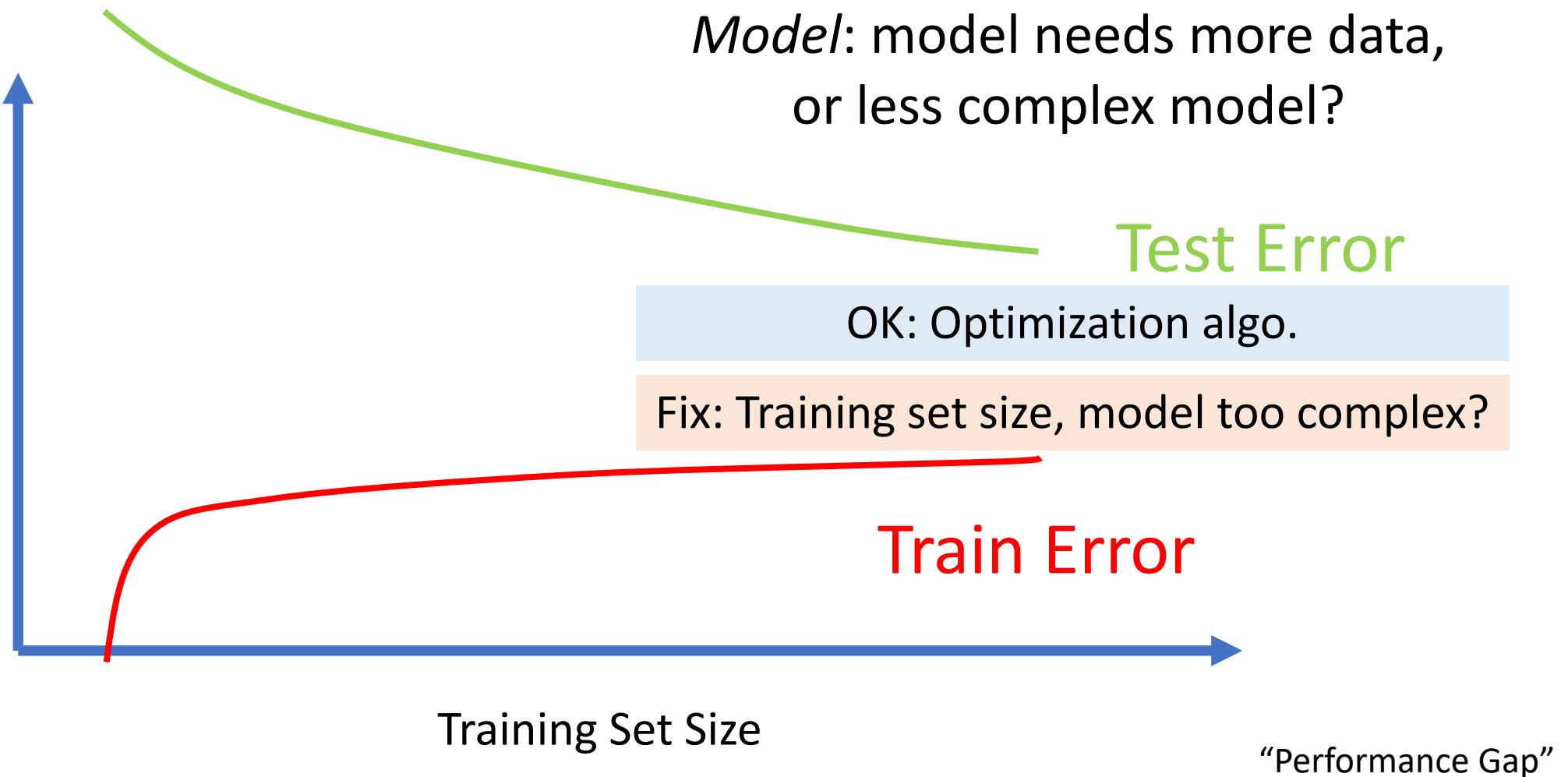


We'll cover some diagnostics that have helped us.

# Diagnostic: Test versus Train Score.



# Diagnostic: Test versus Train Score.



# Variance Diagnostic

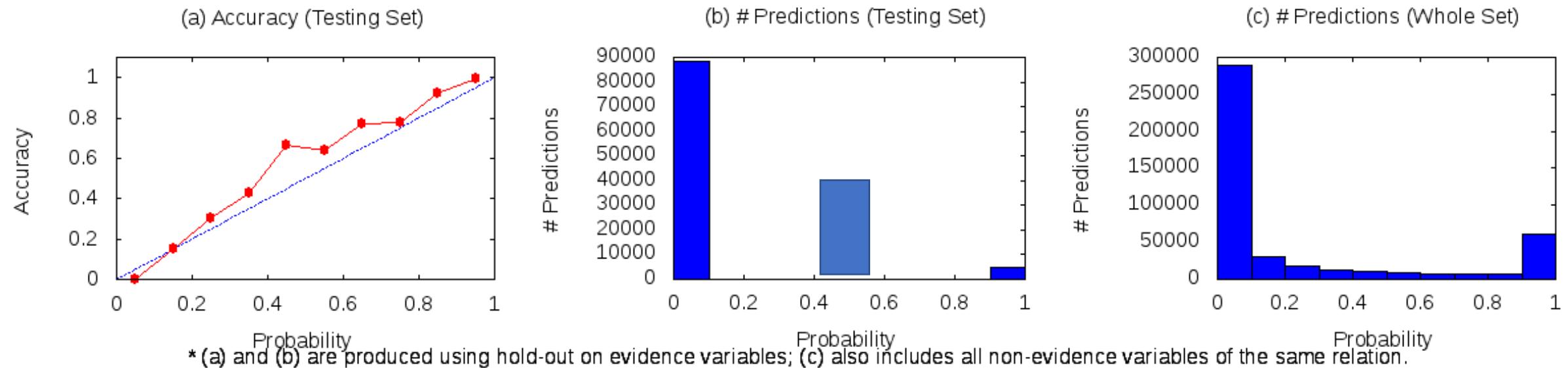
---

- 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 your 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)



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?
- Maybe it's the hyperparameters?
  - Different value of regularizer?
- Try using a different model!

Performance Gap

Performance Gap

Train = Test, Calibration bump

# 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 → optimization problem!
  - If loss is higher -> 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.



# Advanced Techniques and Recent Studies

# Selecting 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.



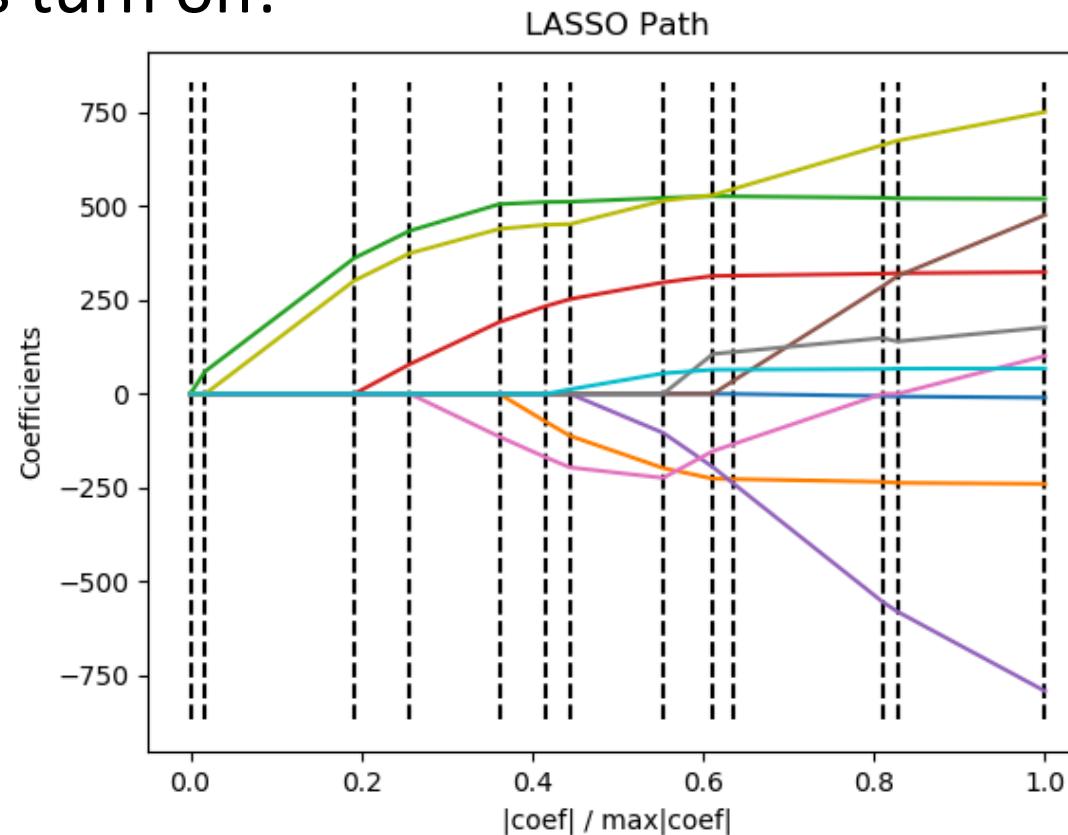
# Lasso Path

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

## LEAST ANGLE REGRESSION

BY BRADLEY EFRON,<sup>1</sup> TREVOR HASTIE,<sup>2</sup> IAIN JOHNSTONE<sup>3</sup>  
AND ROBERT TIBSHIRANI<sup>4</sup>

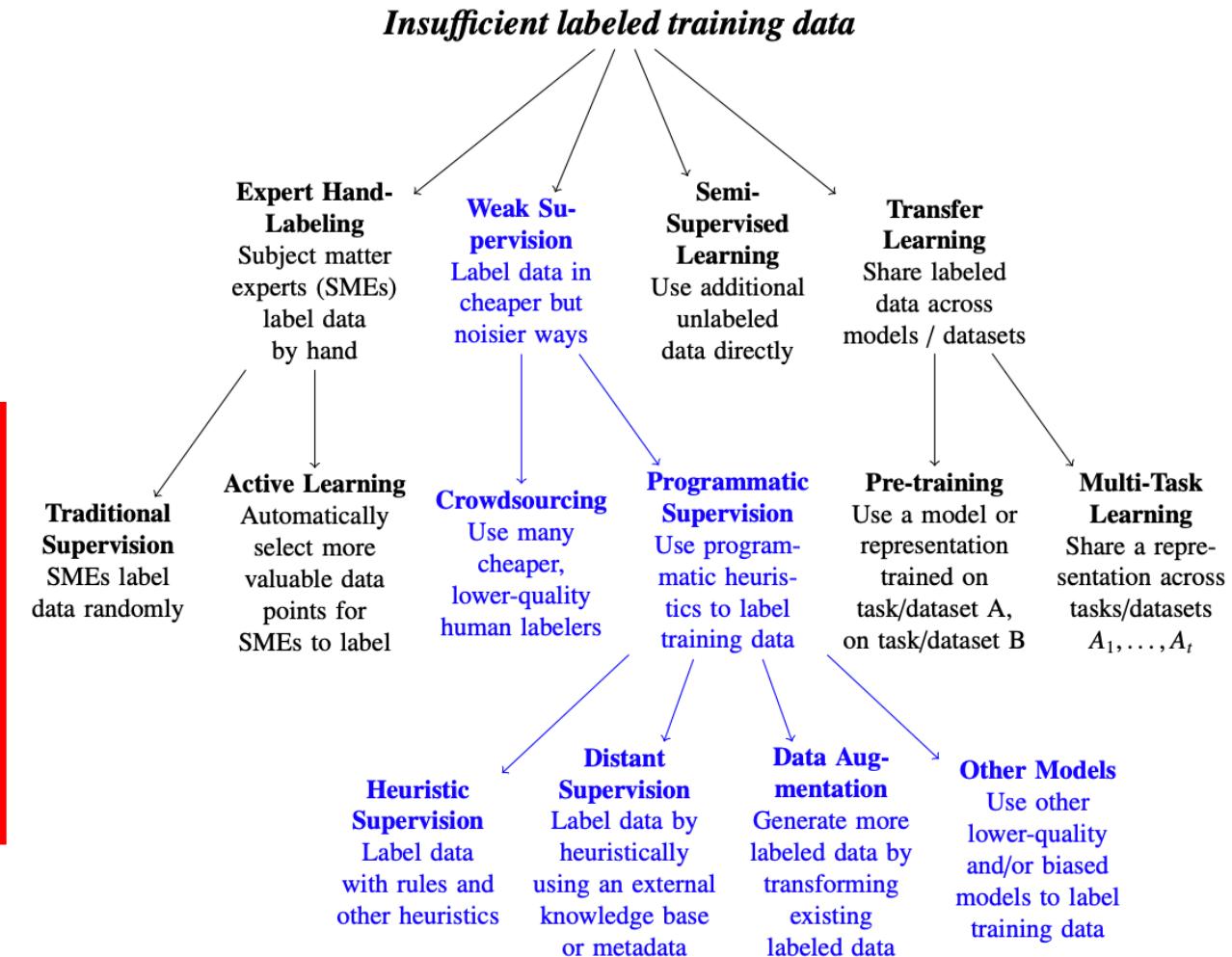
*Stanford University*



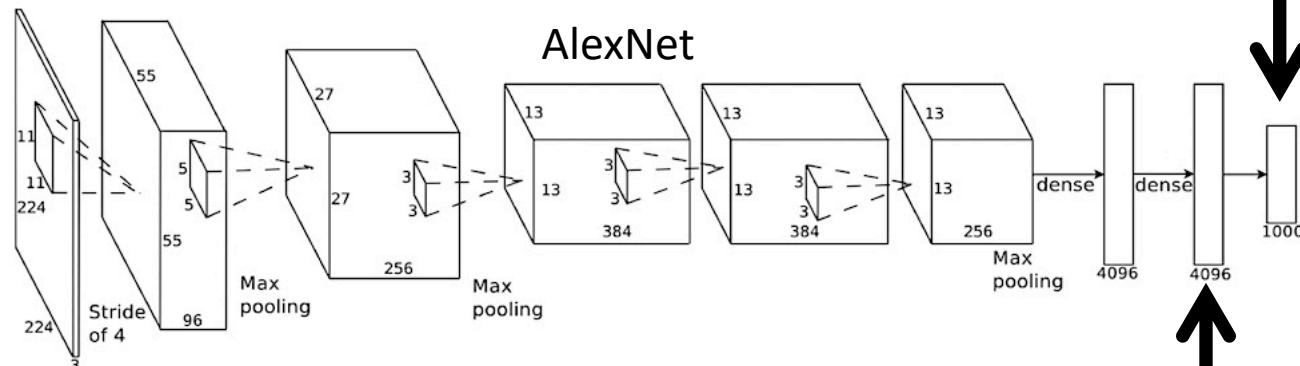
Useful to see how valuable each feature is: Great tool!

# Various techniques for limited labeled data

- **Active learning:** Select points to label more intelligently
- **Semi-supervised learning:** Use unlabeled data as well
- **Transfer learning:** Transfer from one training dataset to a new task
- **Weak supervision:** Label data in cheaper, higher-level ways



# Transfer learning: Basic Idea



1000 classes in ImageNet, each component likelihood of class

Cat has highest value.  
(index 763)

**Challenge:** Want to classify data into new classes—don't have 1M (picture, label) pairs. *Not enough data!*

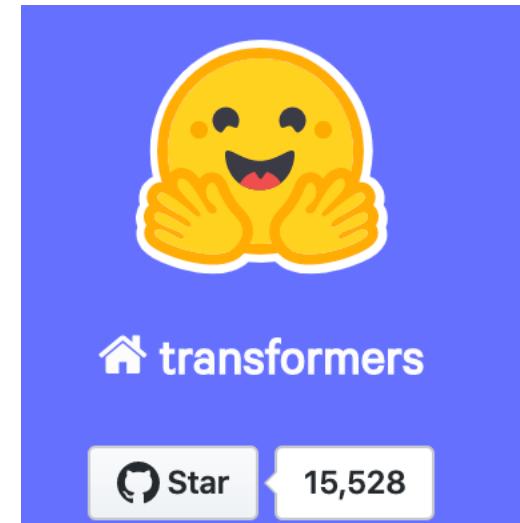
**Transfer learning Idea:** *Can we just replace the last layer with our classes, and just retrain that part?*

A vector "representing" the image, features for "logistic regression".

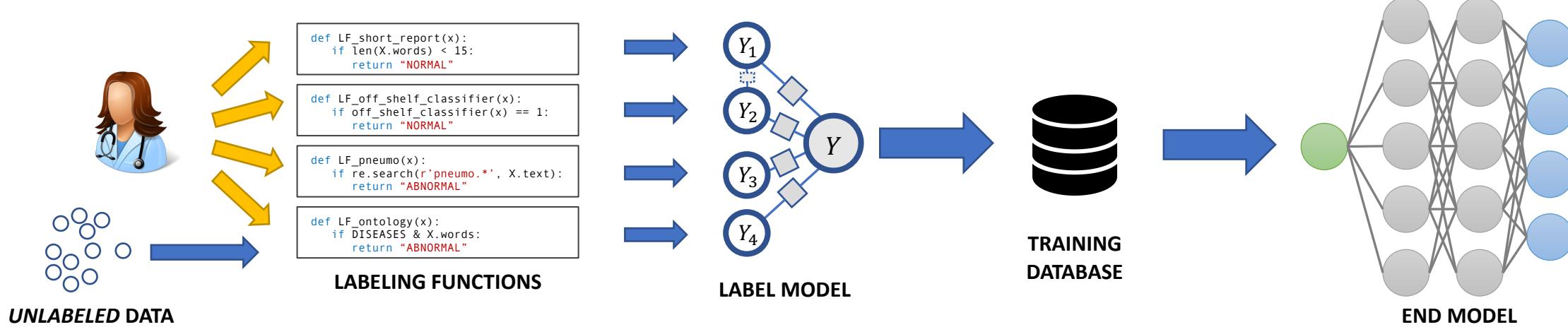
**Abstractly:** A model can be viewed as a function from image to a vector.

# Transfer learning for language

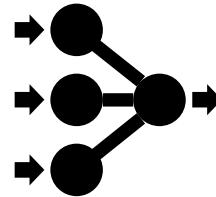
- Train (huge) models offline for language.
  - ELMO – UW, Allen
  - BERT -- Google
  - GPT-2 -- OpenAI
  - Roberta – Facebook
  - XLNet – Google/CMU
- Use trained representation (same last layer)—and simple refinement.
  - A great library with tutorials.
  - Outstanding way to get started with little data.
- **Key question:** If you have enough data for your task, pretraining shouldn't help.
  - Where is the cross over point? What is the performance cost?



# Weak Supervision Ex: Snorkel



 **Users write**  
*labeling functions*  
to heuristically  
label data



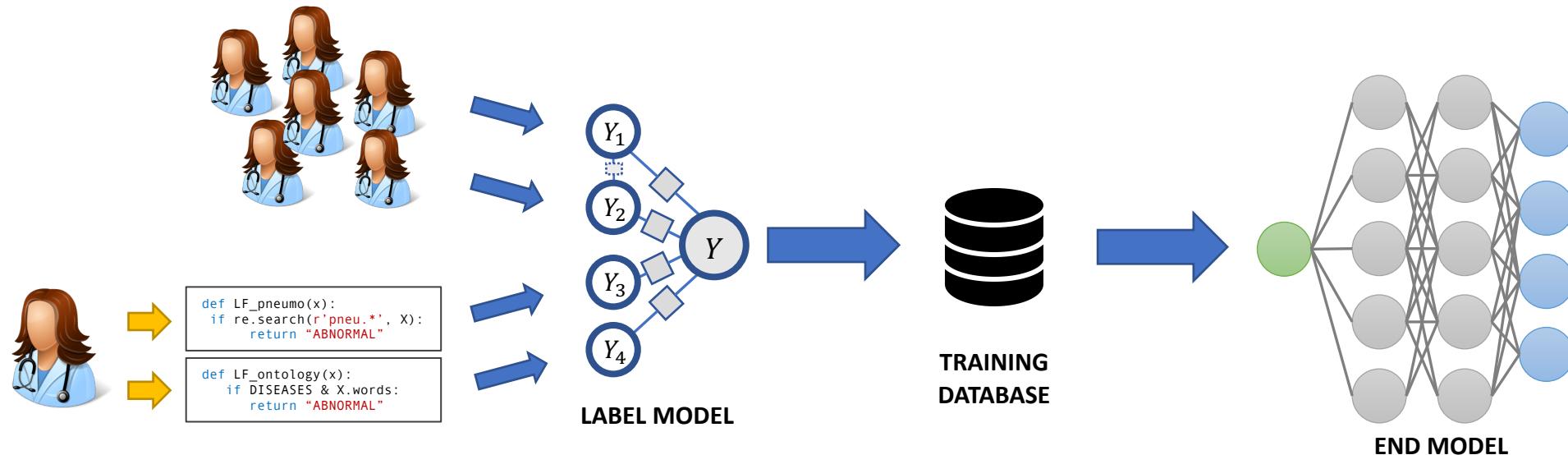
**Snorkel**  
*cleans and*  
*combines* the  
LF labels



**The resulting**  
training database  
used to train an  
ML model

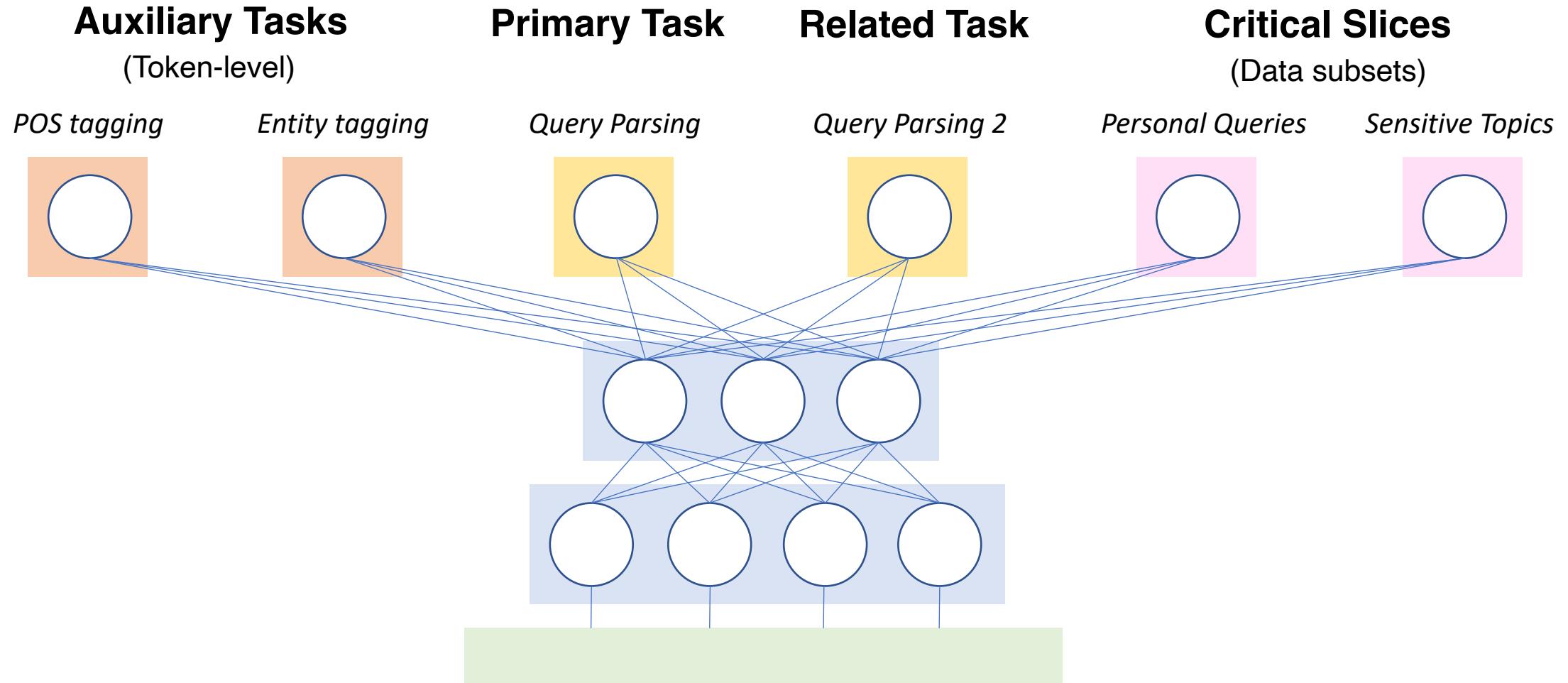
**Note: No hand-labeled training data!**

# Can subsume noisy “crowd” labeling as well



**“Hybrid” approaches use programmatic +  
human supervision**

# Massive Multi-Task Learning (MMTL)



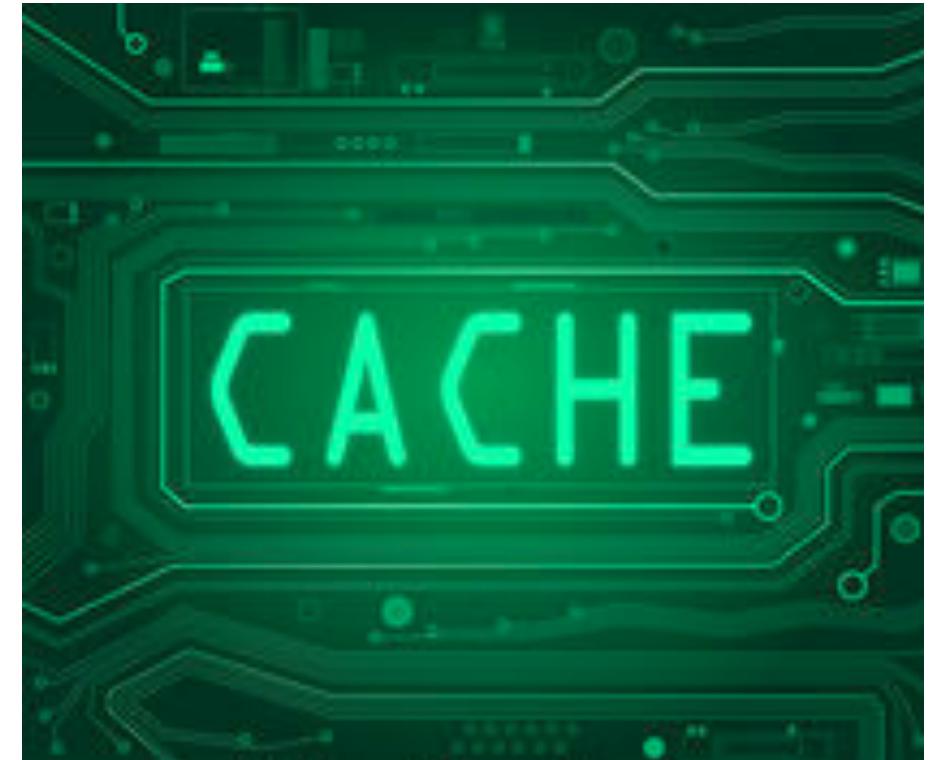
**Capitalizing on supervision at every level of granularity**

# Production Issues

- New course this year (CS329)! See Piazza!
- Folks running a seminar and workshop this year (DAWN)

# 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 used in most production systems.
- Hot fixes!



# Hidden Tech

---

## **Hidden Technical Debt in Machine Learning Systems**

---

**D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips**

{dsculley, gholt, dg, edavydov, toddphillips}@google.com

Google, Inc.

**Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-François Crespo, Dan Dennison**

{ebner, vchaudhary, mwyoung, jfcrespo, dennison}@google.com

Google, Inc.

# Hidden technical debt.

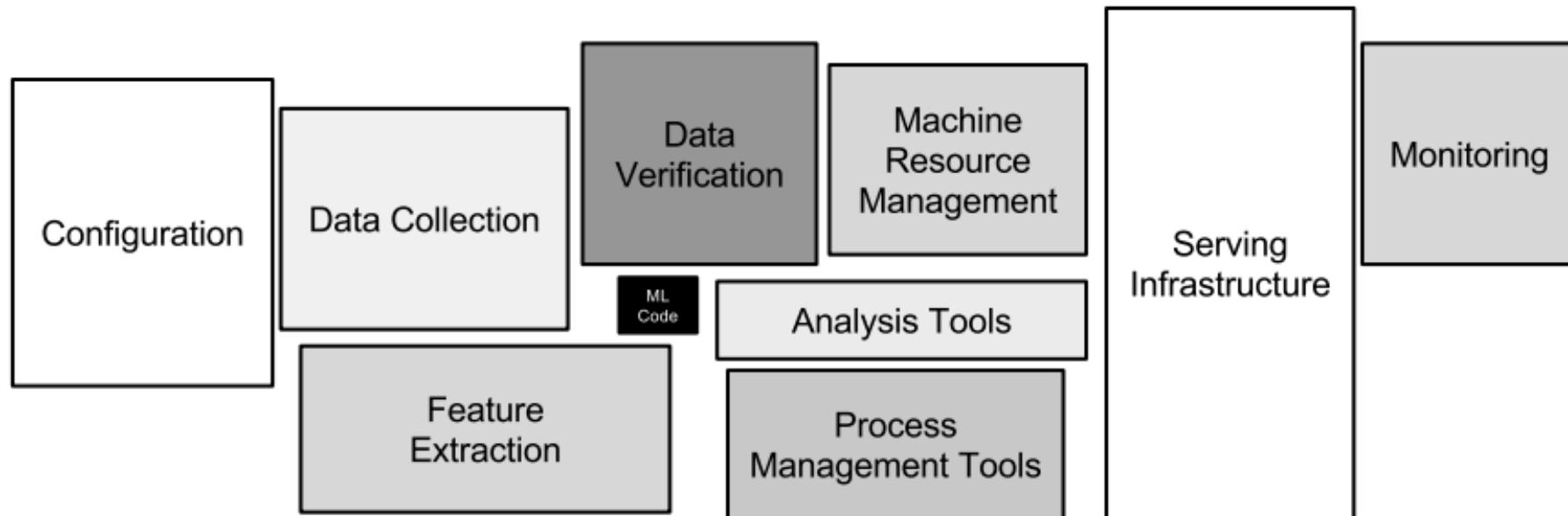


Figure 1: Only a small fraction of real-world ML systems is composed of the ML code, as shown by the small black box in the middle. The required surrounding infrastructure is vast and complex.

# 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!



# Overton: A Data System for Monitoring and Improving Machine-Learned Products

Christopher Ré  
Apple

Feng Niu  
Apple

Pallavi Gudipati  
Apple

Charles Srisuwananukorn  
Apple

Raising abstraction and focus on monitoring  
rather than building models.

# 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!
- 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*."