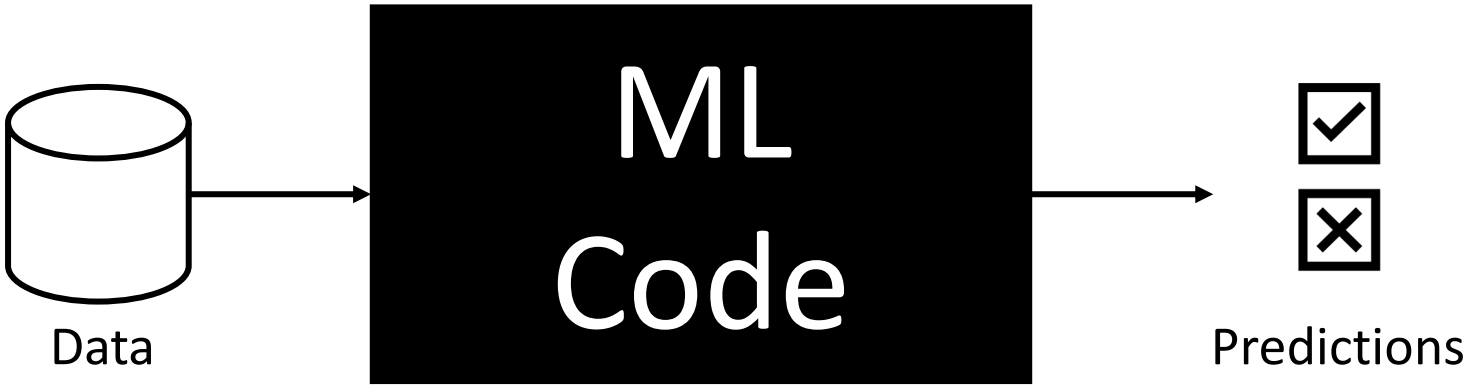
Testing Machine Learning Systems

SOEN 691: Engineering AI-based Software Systems

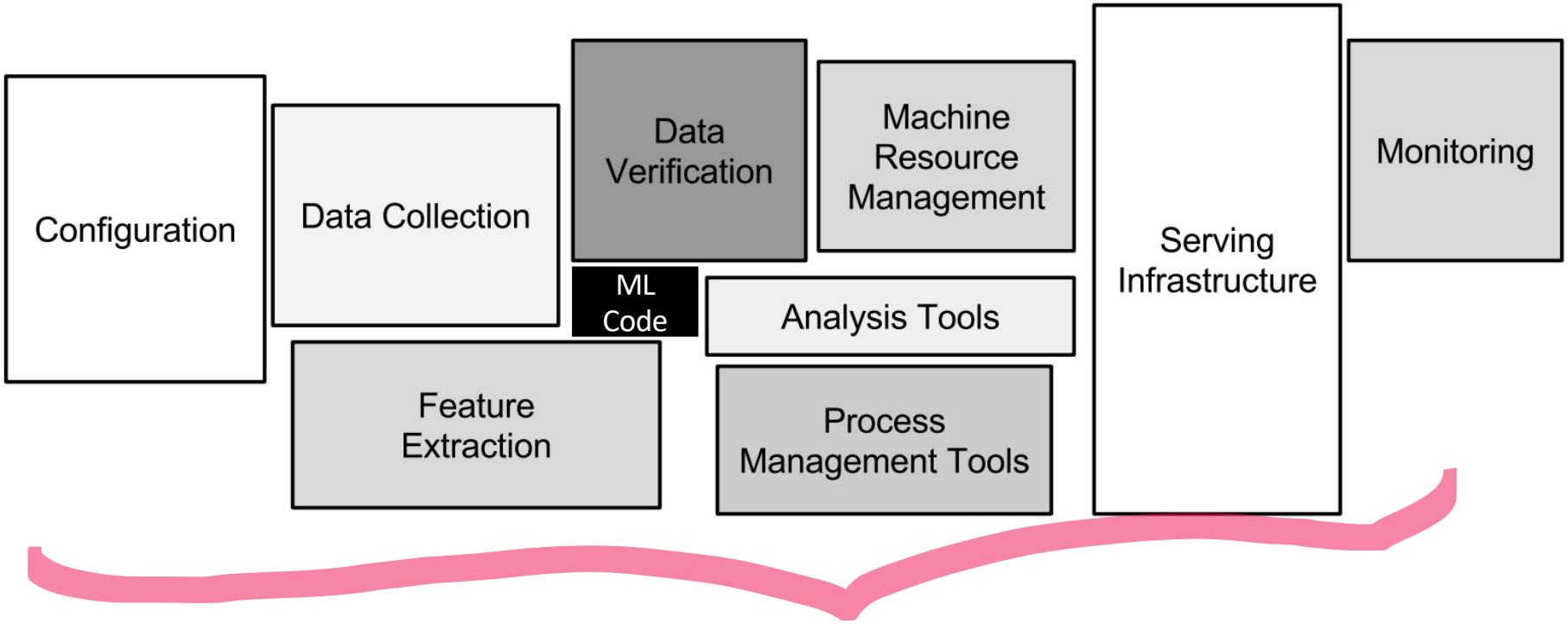
Emad Shihab, Diego Elias Costa
Concordia University

ML Code illusion



ML
Code

ML Code is just small component of a large system

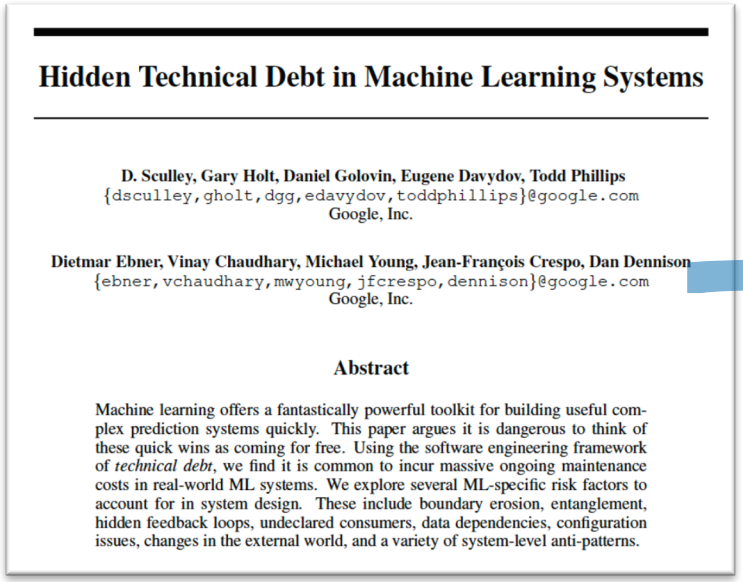


Problems may occur throughout the ML infrastructure!

Test the entire ML infrastructure

Problems and Solutions

ML Hidden Debts



Hidden Technical Debt in Machine Learning Systems

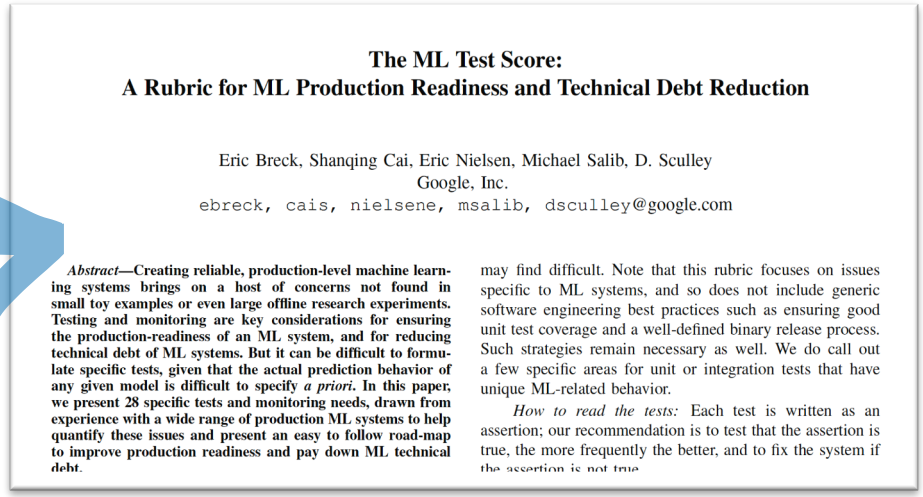
D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips
{dsculley,gholt,dgg,edavydov,toddphillips}@google.com
Google, Inc.

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

Abstract

Machine learning offers a fantastically powerful toolkit for building useful complex prediction systems quickly. This paper argues it is dangerous to think of these quick wins as coming for free. Using the software engineering framework of *technical debt*, we find it is common to incur massive ongoing maintenance costs in real-world ML systems. We explore several ML-specific risk factors to account for in system design. These include boundary erosion, entanglement, hidden feedback loops, undeclared consumers, data dependencies, configuration issues, changes in the external world, and a variety of system-level anti-patterns.

ML Tests



The ML Test Score:
A Rubric for ML Production Readiness and Technical Debt Reduction

Eric Breck, Shanqing Cai, Eric Nielsen, Michael Salib, D. Sculley
Google, Inc.
ebreck, cais, nielsene, msalib, dsculley@google.com

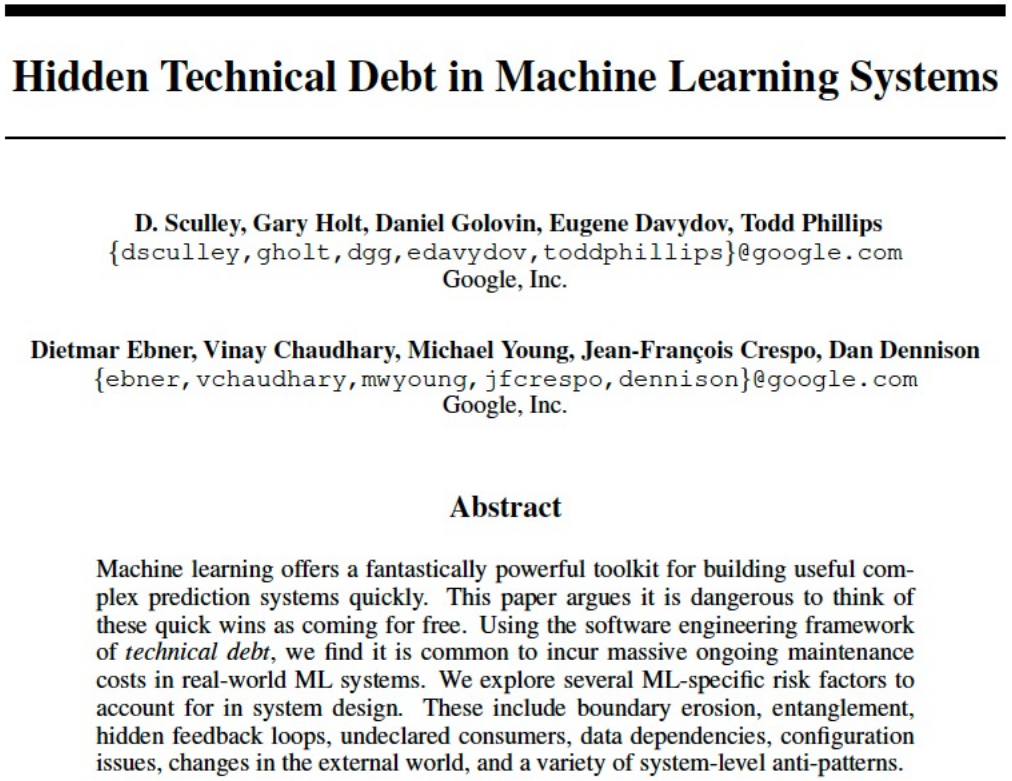
***Abstract*—Creating reliable, production-level machine learning systems brings on a host of concerns not found in small toy examples or even large offline research experiments. Testing and monitoring are key considerations for ensuring the production-readiness of an ML system, and for reducing technical debt of ML systems. But it can be difficult to formulate specific tests, given that the actual prediction behavior of any given model is difficult to specify *a priori*. In this paper, we present 28 specific tests and monitoring needs, drawn from experience with a wide range of production ML systems to help quantify these issues and present an easy to follow road-map to improve production readiness and pay down ML technical debt.**

may find difficult. Note that this rubric focuses on issues specific to ML systems, and so does not include generic software engineering best practices such as ensuring good unit test coverage and a well-defined binary release process. Such strategies remain necessary as well. We do call out a few specific areas for unit or integration tests that have unique ML-related behavior.

How to read the tests: Each test is written as an assertion; our recommendation is to test that the assertion is true, the more frequently the better, and to fix the system if the assertion is not true

Technical Debt in ML Systems

- Reports on the problems of maintaining ML systems



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,mwyoung,jfcrespo,dennison}@google.com
Google, Inc.

Abstract

Machine learning offers a fantastically powerful toolkit for building useful complex prediction systems quickly. This paper argues it is dangerous to think of these quick wins as coming for free. Using the software engineering framework of *technical debt*, we find it is common to incur massive ongoing maintenance costs in real-world ML systems. We explore several ML-specific risk factors to account for in system design. These include boundary erosion, entanglement, hidden feedback loops, undeclared consumers, data dependencies, configuration issues, changes in the external world, and a variety of system-level anti-patterns.

Study Design

- Industrial report
  - Experience from industry
  - 10 authors from Google
- Published in NeurIPS in 2015
  - Neural Information Processing Systems
- **Goal:**
  - Not to propose any new algorithm or solution
  - **Increase awareness** of the problems

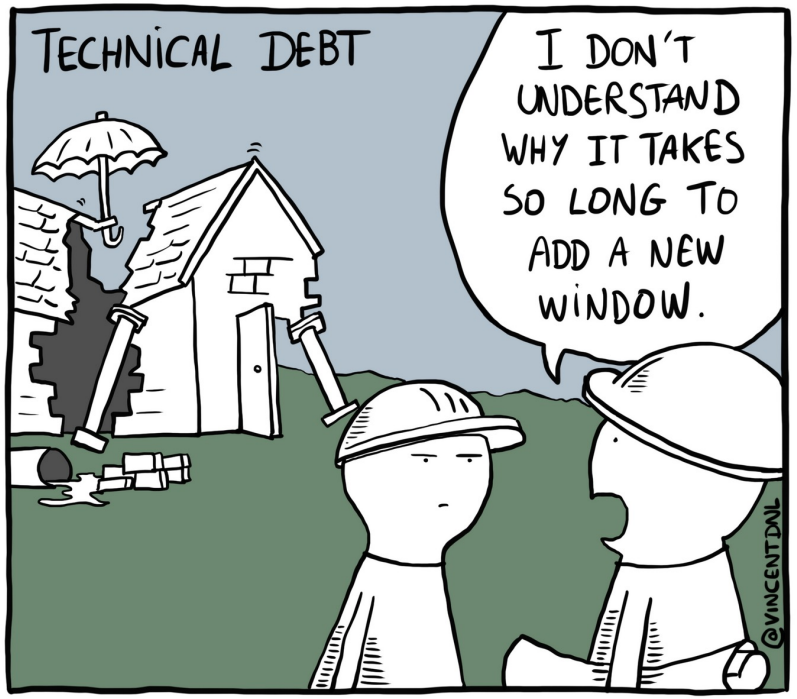
Maintaining ML Systems is Challenging

"Developing and deploying ML systems is relatively fast and cheap. **Maintaining** ML systems over time is **difficult and expensive**."

Technical Debt

- Long term costs of **moving quickly** in software engineering.

- Not all debts are bad, but they must be paid
  - Refactoring the code
  - Reducing dependencies
  - Improve tests



Technical Debt (TD) is harder in Machine Learning Systems

- Machine Learning Systems
  - All TDs of Traditional Software +
  - The TDs of Machine Learning

- Technical Debts may exist at the system level
  - More difficult than TDs at the code level
  - Traditional boundaries do not work when the data can influence ML system behavior

Paper Structure

- Complex models erode boundaries
- Data dependencies
- Feedback loops
- ML Systems antipatterns
- Configuration debt
- Changes in the external world
- Other areas of technical debts

Technical Problems!

Eroding Boundaries

- Traditional software
  - Encapsulation and struct abstraction boundaries help create maintainable code
- It is difficult to enforce strict abstract boundaries in ML systems
  - ML behavior depends on external data
  - Real world data does not fit into tidy encapsulation

Entanglement

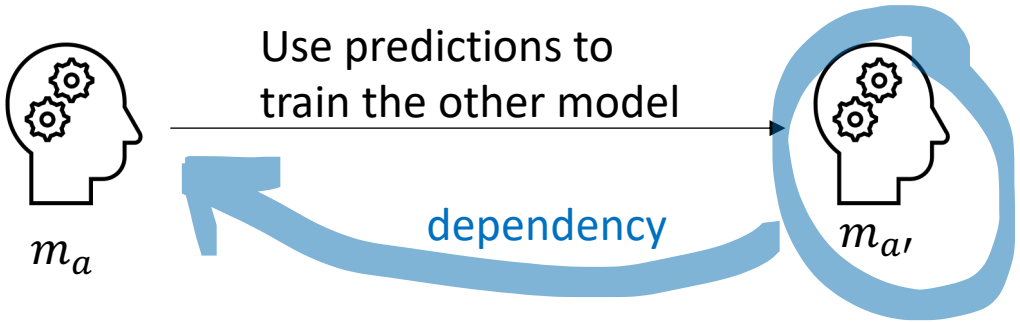
- **CACE**
  - **C**hanges **A**nything, **C**hanges **E**verything

- Adding or removing features changes how the model previously depended on other features

- Mitigation:
  - Serve ensembles trying to capture different aspects of the problem
  - Detecting changes in behavior as soon as they occur

Correction Cascades

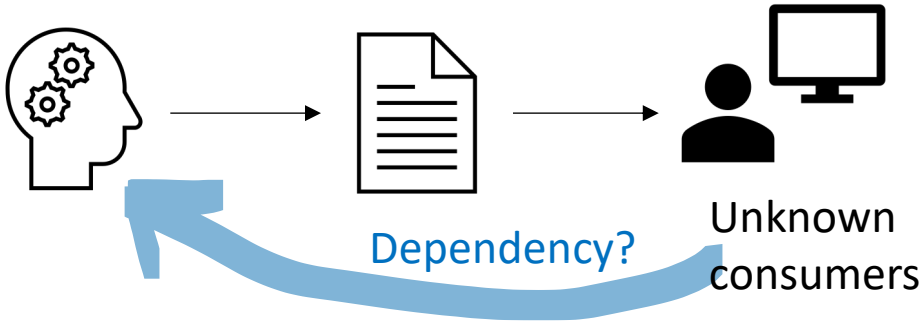
- You have a model m_a for a problem A
- You need to solve a slightly different problem A'



- Makes it more expensive to maintain
- **Mitigation:**
  - Augment m_a with new features to solve A'
  - Accept the costs of creating a completely new model

Undeclared Consumers

- Predictions of Machine learning models are made accessible internally
  - CSV files, log files



- Creates an unseen coupling between systems
- Mitigation:
  - Guard access with strict service-level agreements

Data Dependencies costs more than Code Dependencies

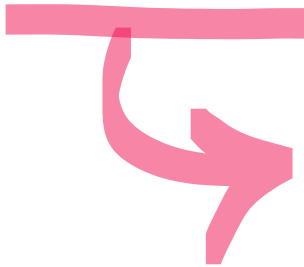
- Traditional SE: Data dependency is a key contributor to code complexity
  - Tools can help developers alleviate the issue

- In ML systems, data dependency can be equally problematic but more difficult to identify
  - No good tooling for ML data dependency

Unstable Data Dependencies

- It is common to consume data from other systems as input feature
  - What if the data is unstable?
  - Change behavior over time

- Mitigation strategy
  - Create a versioned copy of input signals



How do you maintain the versioned copies?

Underutilized Data Dependencies

- Underutilized data provide little to no benefit but can make the ML system vulnerable to changes.
- **Legacy features**:
  - Useful at some point but not anymore
- **Bundled features**:
  - Features grouped together but were not evaluated individually
- **ε-Features**:
  - negligible improvement

- Mitigation:
  - Feature importance, exhaustive leave-one-feature-out evaluation

Feedback Loops

- ML systems tend to influence their **own behavior** over time

- Direct feedback loops
  - Influence is direct and traceable through systems
- Indirect feedback loops
  - Influence comes from the response of the real world

ML-System Anti-patterns

- **Glue Code**
  - Use of genetic packages -> constrained by packages
- **Pipeline Jungle**
  - Jungle of scrapes, joins, sampling…
- **Dead Experimental Codepaths**
  - Cost of experimenting is low
  - Maintaining all experimenting pathways is hard
- **Abstraction Debt**
  - What is the right abstraction to describe data? Model? Prediction?

Common Smells in ML systems

- **Plan-Old-Data**
  - Use of raw features with no encoding or metadata
- **Multiple languages**
  - Using multiple languages often increases the cost of testing and ownership transfer
- **Prototype**
  - Evolving prototype to a solution is costly
  - Time pressure might push prototype to be the solution

Configuration Debt

- Configuration of ML systems may accumulate lots of debts
  - Data collection, data processing, sampling, models, …
- It should be easy to
  - Specify configuration
  - Visualize differences in configuration of two models
  - Detect unused or redundant settings
- It should be hard to
  - make manual errors

Dealing with changes in the External World

- Fixed thresholds in Dynamic Systems

- Monitoring and Testing
  - Unit tests are valuable but not sufficient
  - Real data changes and needs to be monitored
  - **Prediction bias**
    - Distribution of predicted labels equiv. observed labels
  - **Action limits**
    - Enforce limits to the ML system
  - **Up-stream producers**
    - Data producers should be constantly monitored

Other kinds of debts

- Data testing debt
  - Testing input data is critical
- Reproducibility debt
  - It is difficult to design reproducible real-world systems
- Process management debt
  - Dozens/hundreds of models simultaneously
  - How to update them? Assign resources? How to detect blockages in the flow of data pipelines?

Conclusion: Measuring Debt and Paying it Off

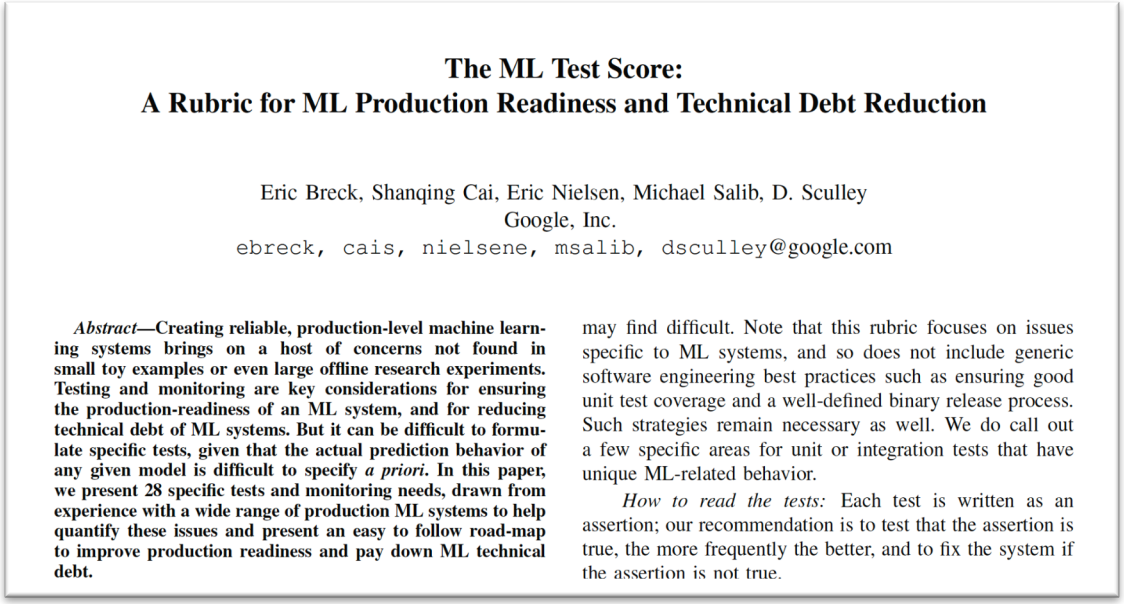
- A team is still able to move quickly is not in itself evidence of low debt or good practices
  - Full cost of debt cripples over time

- How easily can an entirely new algorithmic approach be tested at full scale?
- How precisely can the impact of a new change to the system be measured?
- How quickly can new members of the team be brought up to speed?

Open Discussion

The ML Test Score

- A framework for ML production and technical debt reduction



The ML Test Score:
A Rubric for ML Production Readiness and Technical Debt Reduction

Eric Breck, Shanqing Cai, Eric Nielsen, Michael Salib, D. Sculley
Google, Inc.
ebreck, cais, nielsene, msalib, dsculley@google.com

***Abstract*—Creating reliable, production-level machine learning systems brings on a host of concerns not found in small toy examples or even large offline research experiments. Testing and monitoring are key considerations for ensuring the production-readiness of an ML system, and for reducing technical debt of ML systems. But it can be difficult to formulate specific tests, given that the actual prediction behavior of any given model is difficult to specify *a priori*. In this paper, we present 28 specific tests and monitoring needs, drawn from experience with a wide range of production ML systems to help quantify these issues and present an easy to follow road-map to improve production readiness and pay down ML technical debt.**

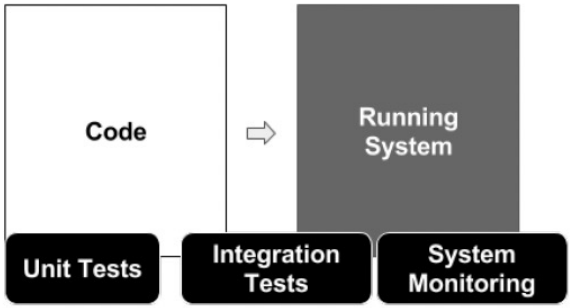
may find difficult. Note that this rubric focuses on issues specific to ML systems, and so does not include generic software engineering best practices such as ensuring good unit test coverage and a well-defined binary release process. Such strategies remain necessary as well. We do call out a few specific areas for unit or integration tests that have unique ML-related behavior.

How to read the tests: Each test is written as an assertion; our recommendation is to test that the assertion is true, the more frequently the better, and to fix the system if the assertion is not true.

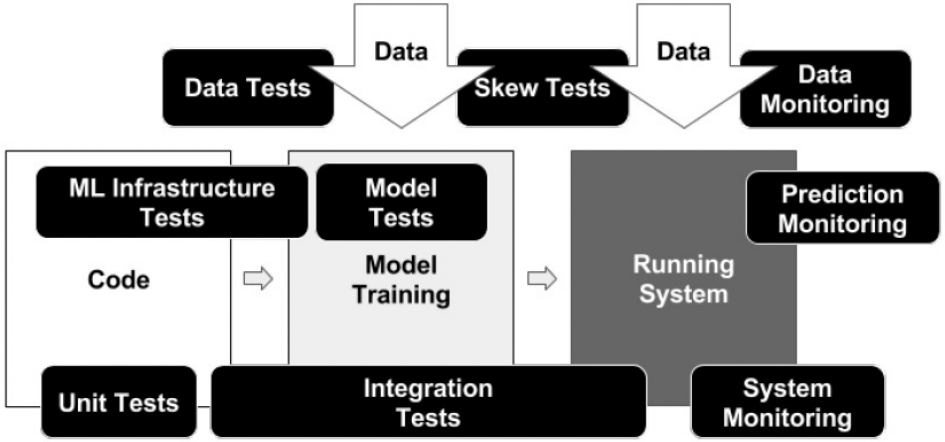
Motivation

- ML systems reliability are becoming increasingly critical
  - Testing and monitoring are important strategies for improving reliability

Traditional systems vs ML systems



Traditional System Testing and Monitoring



ML-Based System Testing and Monitoring

What should be tested?
How much is enough?

Study Design

- Industrial report
  - Google
- Published at IEEE Big Data in 2017
- Goal:
  - Rubric of 28 actionable tests
  - Scoring system to measure how ready for production a ML system is

Related work

Software testing is well studied, as is machine learning, but their intersection has been less well explored in the literature. [4] reviews testing for scientific software more generally, and cites a number of articles such as [5], who present an approach for testing ML algorithms. These ideas are a useful complement for the tests we present, which are focused on testing the use of ML in a production system rather than just the correctness of the ML algorithm per se.

Zinkevich provides extensive advice on building effective machine learning models in real world systems [6]. Those rules are complementary to this rubric, which is more concerned with determining how reliable an ML system is rather than how to build one.

Issues of surprising sources of technical debt in ML systems has been studied before [1]. It has been noted that the prior work has identified problems but been largely silent on how to address them; this paper details actionable advice drawn from practice and verified with extensive interviews with the maintainers of 36 real world systems.

Rules of machine learning – Talk in 1996

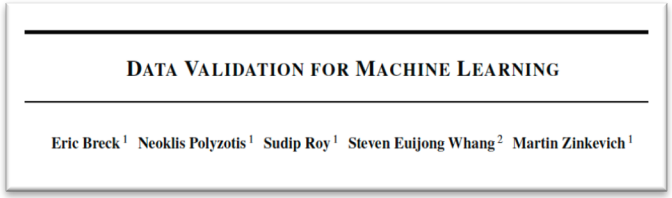
Quite short related work

Machine learning: The high interest credit card of technical debt (2014)

Categories of tests

- Tests for Features and Data
- Tests for Model Development
- Tests for ML Infrastructure
- Monitoring Tests for ML

Tests for Feature and Data



DATA VALIDATION FOR MACHINE LEARNING

Eric Breck [1] Neoklis Polyzotis [1] Sudip Roy [1] Steven Euijong Whang [2] Martin Zinkevich [1]

1	Feature expectations are captured in a schema.
2	All features are beneficial.
3	No feature's cost is too much.
4	Features adhere to meta-level requirements.
5	The data pipeline has appropriate privacy controls.
6	New features can be added quickly.
7	All input feature code is tested.

Table I
BRIEF LISTING OF THE SEVEN DATA TESTS.

#2 Are all features beneficial

- Every feature has a software engineering cost
  - Important to understand the value each feature provides in predictive power

- How?
  - Correlation coefficients
  - Training models with individual features
  - Training models with k features removed

#4 Features adhere to meta-level requirements

- Projects may impose requirements on the data coming into the system
  - Protected characteristics
    - Age, gender, race…
  - All data should come from a single source

- How?
  - **Programmatically enforce** these requirements
    - Pipeline tests
    - Model tests

#5 Data Pipeline has appropriate Privacy Control

- Data may contain sensitive information
  - Training data, test data, vocabulary files
- Teams are often aware they need to remove Personally Identifiable Information (PII)
  - Programming errors and system changes may lead to PII leakages
- How?
  - Budget sufficient time during new feature development to allow for proper handling

Testing the Model Development

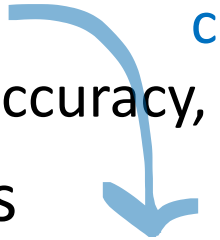
1	Model specs are reviewed and submitted.
2	Offline and online metrics correlate.
3	All hyperparameters have been tuned.
4	The impact of model staleness is known.
5	A simpler model is not better.
6	Model quality is sufficient on important data slices.
7	The model is tested for considerations of inclusion.

Baseline models

Table II
BRIEF LISTING OF THE SEVEN MODEL TESTS

#2 Offline and online metrics correlate

How ML metrics translate to user centric metrics?



- ML metrics
  - Log-loss, accuracy, prediction, AUC
- User metrics
  - Engagement, user happiness, revenue

- How?
  - Perform A/B tests to evaluate the impact of better (or intentionally worse) models

#4 Impact of Staleness

- What is model staleness?
  - A model that is not sufficiently updated
- Test the impact of an outdated model
  - Helps you determine update frequency, contingency plans
- How?
  - Conduct A/B experiments with models outdated by different range of dates

#6 Model quality is sufficient on all important data slices

- Slicing the data along certain dimensions can improve the understanding of the model
  - Global accuracy +1%, but…
    - for one country -50%
    - for young people -30%
  - Probably caused by faulty data streams…

- How?
  - Include tests with different data slices in the release process

#7 Model tested for considerations of inclusion

- ML Fairness
  - E.g., word embedding trained on news articles had learned inappropriate relations between gender and occupation
- Overlooked biases in training set influence system's behavior
- How?
  - Tests with input features x protected characteristics
  - Postprocessing to minimize disproportionate loss for certain groups

Tests for ML Infrastructure

Table III
BRIEF LISTING OF THE ML INFRASTRUCTURE TESTS

1	Training is reproducible.
2	Model specs are unit tested.
3	The ML pipeline is Integration tested.
4	Model quality is validated before serving.
5	The model is debuggable.
6	Models are canaried before serving.
7	Serving models can be rolled back.

#3 ML Pipeline is integration tested

- Complete ML pipeline
  - Collecting data -> Feature Extraction -> Model training -> Model Verification -> Deployment
- Automation tests that runs regularly and exercises the entire pipeline

- How
  - Integration tests should run continuously
  - Faster integration tests for quick feedback
    - Data and models sampled

#5 Model can be debugged

- Why is the model behaving bizarrely?
- Tools and frameworks that enable a step-by-step computation of training and inference on a **single example**
- How?
  - Through development of tools that allow users to enter an example and observe how a specific model behaves
    - TensorFlow debugger

#6 Models are canaried before serving

- Mismatch between modeling and servicing
  - Modeling code tends to be updated more frequently

- How?
  - Test if the model successfully loads into production
  - Keep both old and new model concurrently
    - New models only serve a small fraction of traffic
    - Gradually increases the load in the new model

Monitoring Tests for ML

Table IV
BRIEF LISTING OF THE SEVEN MONITORING TESTS

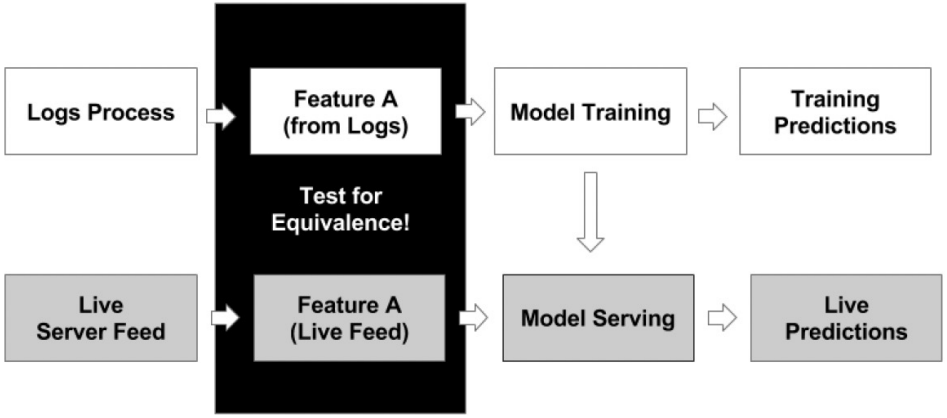
1	Dependency changes result in notification.
2	Data invariants hold for inputs.
3	Training and serving are not skewed.
4	Models are not too stale.
5	Models are numerically stable.
6	Computing performance has not regressed.
7	Prediction quality has not regressed.

#1 Dependency changes result in notification

- ML systems consume data from numerous services
  - Partial outages, version upgrades, and other changes may interrupt the stream of inputs
- Errors in the data input sources should not be silent
- How?
  - Make sure the team monitors all dependencies
    - Subscribes to announcements
    - Have a complete list of input sources

#3 Training and Test data is not skewed

Figure 3. **Monitoring for Training/Serving Skew.** It is often necessary for the same feature to be computed in different ways in different parts of the system. In such cases, we must carefully test that these different codepaths are in fact logically identical.



- How?
  - Crucial to log traffic data
  - Feature values perfectly match
  - Monitor number of features that exhibit skewness

#6 Computing performance has not regressed

- Computational performance (CPU, memory) is a **key concern** at scale
  - DNNs can be slow at training and inference time
  - Slow response time is terrible for business efficiency

- How?
  - Monitor performance
  - Slice monitored performance per data and model versions

The ML Test Score

The final score is computer as:

- 0.5 for manual tests
- 1.0 for fully automated tests
- Sum the score for each category individually
  - Feature, Model, Infrastructure, Monitoring

- Final score is the **minimum** across all categories
  - All categories are important!

How to interpret ML-Test Score

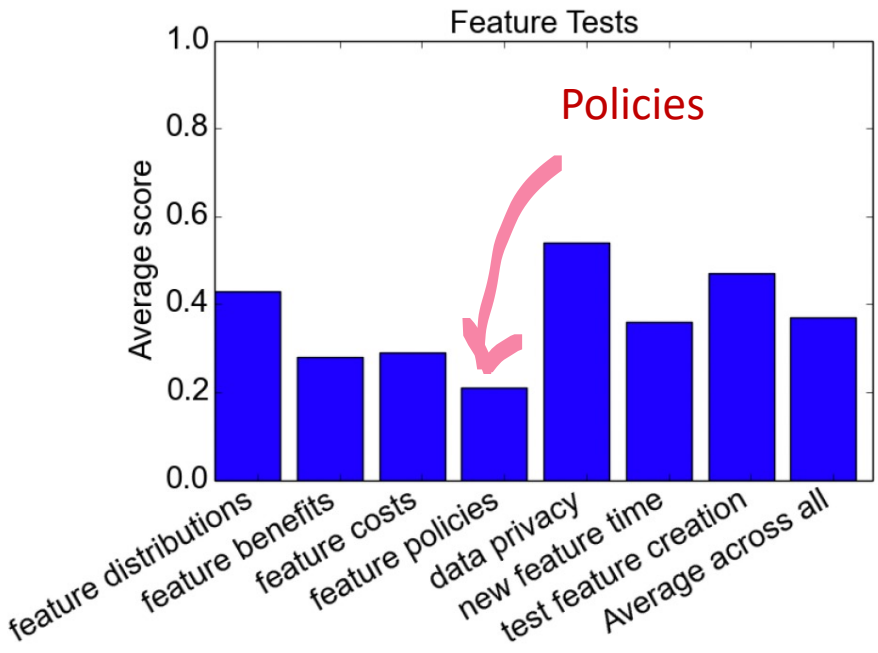
Points	Description
0	More of a research project than a productionized system.
(0,1]	Not totally untested, but it is worth considering the possibility of serious holes in reliability.
(1,2]	There's been first pass at basic productionization, but additional investment may be needed.
(2,3]	Reasonably tested, but it's possible that more of those tests and procedures may be automated.
(3,5]	Strong levels of automated testing and monitoring, appropriate for mission-critical systems.
> 5	Exceptional levels of automated testing and monitoring.

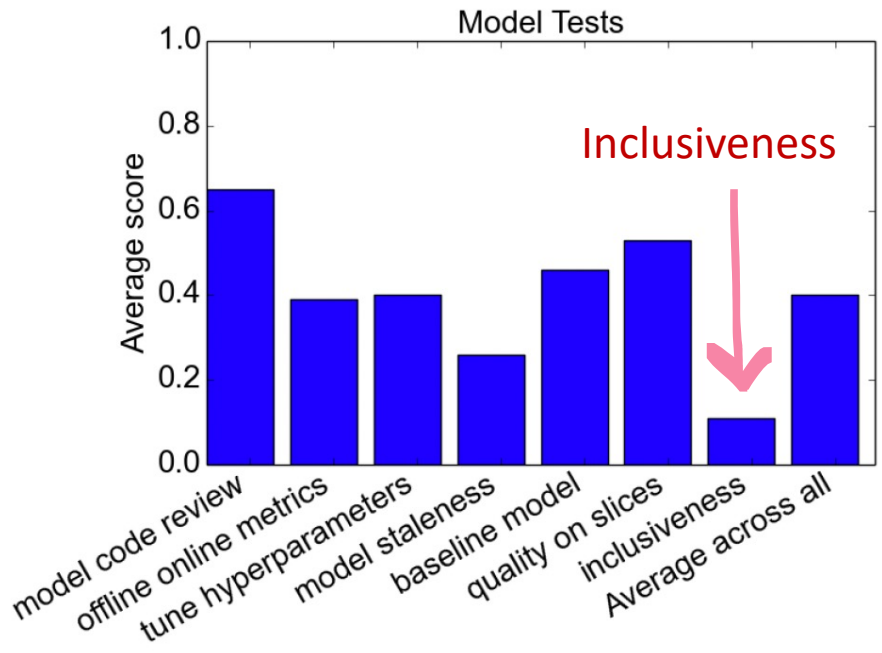
Table V

Interpreting an ML Test Score. THIS SCORE IS COMPUTED BY TAKING THE *minimum* SCORE FROM EACH OF THE FOUR TEST AREAS. NOTE THAT DIFFERENT SYSTEMS AT DIFFERENT POINTS IN THEIR DEVELOPMENT MAY REASONABLY AIM TO BE AT DIFFERENT POINTS ALONG THIS SCALE.

Applying the ML-Test Score

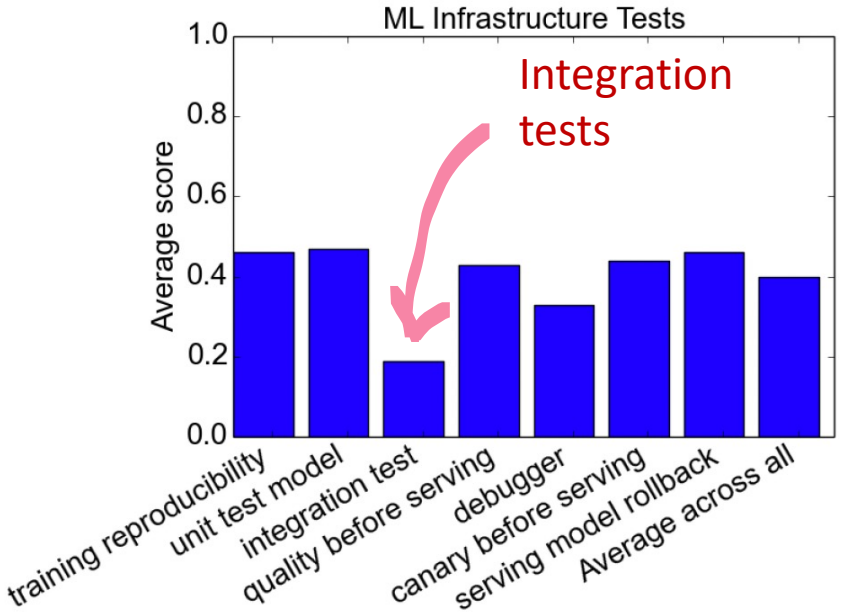
- 36 teams at Google

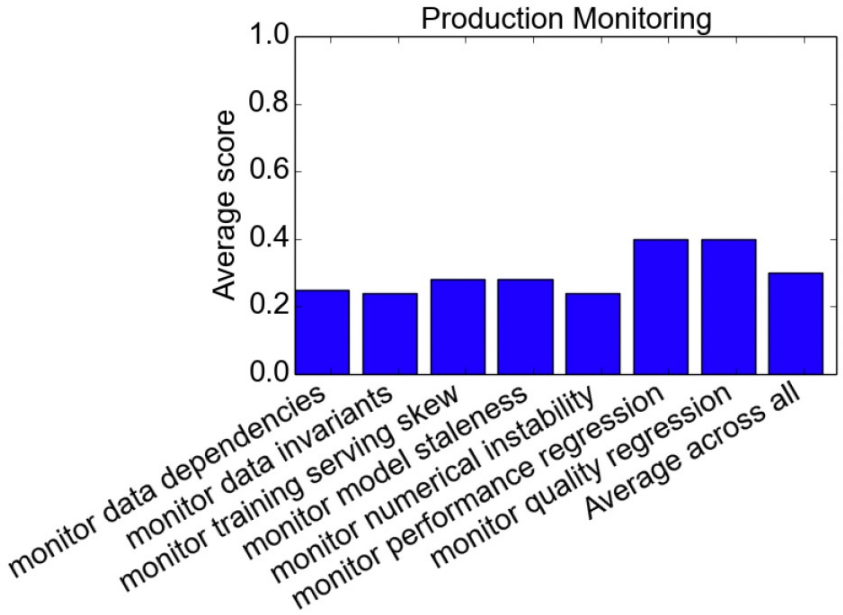




Applying the ML-Test Score

- 36 teams at Google





Open Discussion

Homework

- No homework for next class