Monitoring and Maintenance

Michael Cooper

CS 329S – Machine Learning Systems Design

Lecture 13, March 1, 2021

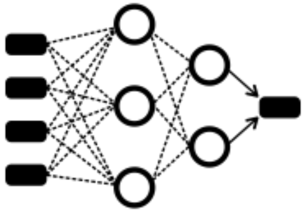
Logistics

- Mid-quarter evaluation! https://forms.gle/R8vpmu9mCziupiYR9
- Piazza + course staff email have a much shorter response time than to Chip's personal email
- Feel free to email us for 1:1s

How can ML systems fail?

Pre-Mortem Breakout Exercise

- 5 minutes
- About to deploy a mobile phone app that allows users to take picture of food and tells them what food category it is.
- Great performance on test set.
- If the system fails, what might be the most likely cause?



Model Failure

Deliberate System Abuse

Resource Overload

How can ML systems fail?

Excessive Latency

Poor Security

Downtime/Crashing

How can ML systems fail?

How ML Breaks

Categories of Failure

Nineteen ways of thinking about how things break

- Process orchestration issues
- Overloaded backends
- Temporary failure to join with expected data
- CPU failures
- Cache invalidation bugs
- Changes to the distribution of examples that we are generating inference on
- Config changes pushed out of order
- Suboptimal data structure used
- Challenges assigning work between clusters
- Example training strategy resulted in unexpected ordering
- ML hyperparameters adjusted on the fly
- Configuration change not properly canaried or validated
- Client made incorrect assumption about model providing inference
- Inference takes too long
- Incorrect assert() in code
- Labels weren't available/mostly correct at the time the model wished to visit the example
- Embeddings interpreted in the wrong embedding-space
- QA/Test jobs incorrectly communicating with prod backends
- Failed to provision necessary resources (bandwidth, ram, CPU)

Google

How ML Breaks

ML vs. Not-ML

Would this kind of failure have happened in a non-ML pipeline?

ML:

- Changes to the distribution of examples.
- Problems with selection and processing of training data: either sampling wrong, re-visiting the same data, skipping data, etc.
- Hyperparameters
- Mismatch in embedding interpretation
- Training on mislabeled data

Not ML:

- Dependency failure (other than data)
- Deployment failure (out of order, wrong target, wrong binaries, etc.)
- CPU failures
- Inefficient data structure

Google

Hidden Technical Debt

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.

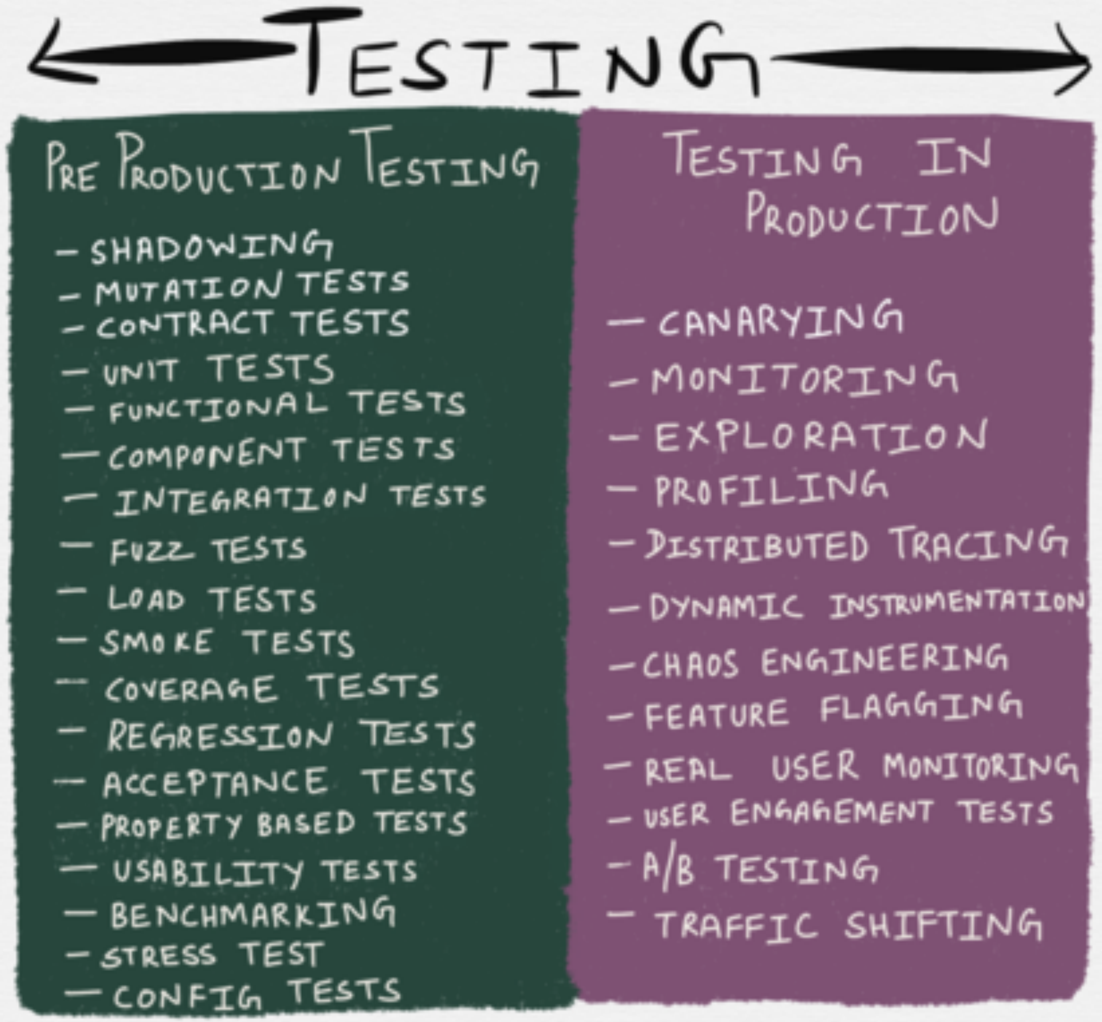
Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-François Crespo, Dan Dennison
{ebner,vchaudhary,mwyoung,jfcrespo,dennison}@google.com
Google, Inc.

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.

Source: https://papers.nips.cc/paper/2015/file/86df7dcfd896fcaf2674f757a2463eba-Paper.pdf

Monitoring: tracking statistics about a ML system to understand its environment and behavior.



Monitoring is post-production testing.
(link)

Maintenance: updating a deployed machine learning system to improve performance or correct for failure.

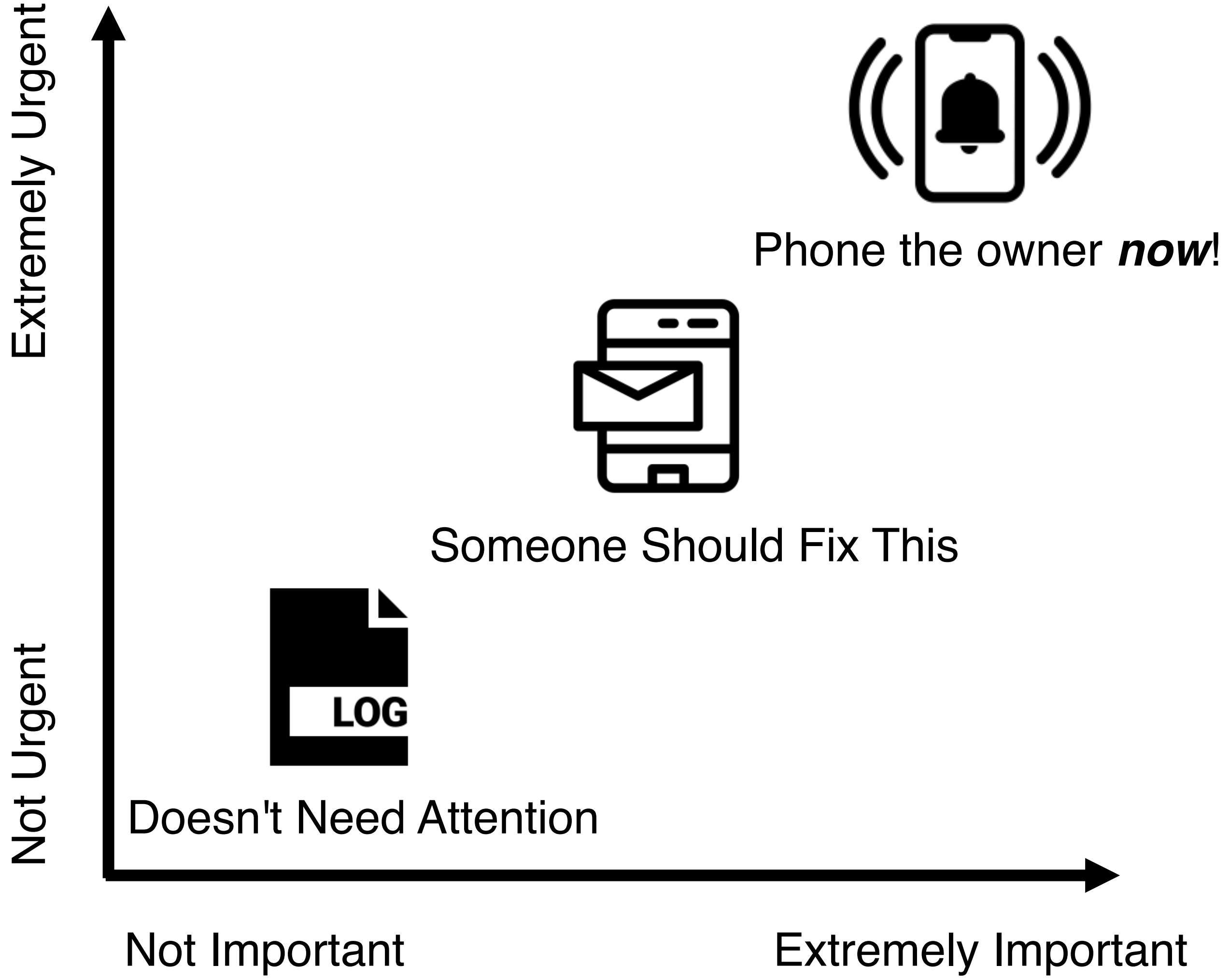
Monitoring and Maintenance: collection of techniques and protocols for managing a deployed ML system to detect and correct system failures or improve overall system performance.

Outline

- Monitoring
  - Monitoring Overview
  - Monitoring System Infrastructure
  - Monitoring Data Pipelines
  - Monitoring Model Performance
- Maintenance
  - Guide to Releasing a New Model

Monitoring Overview

Something Happened! What Do I Do?



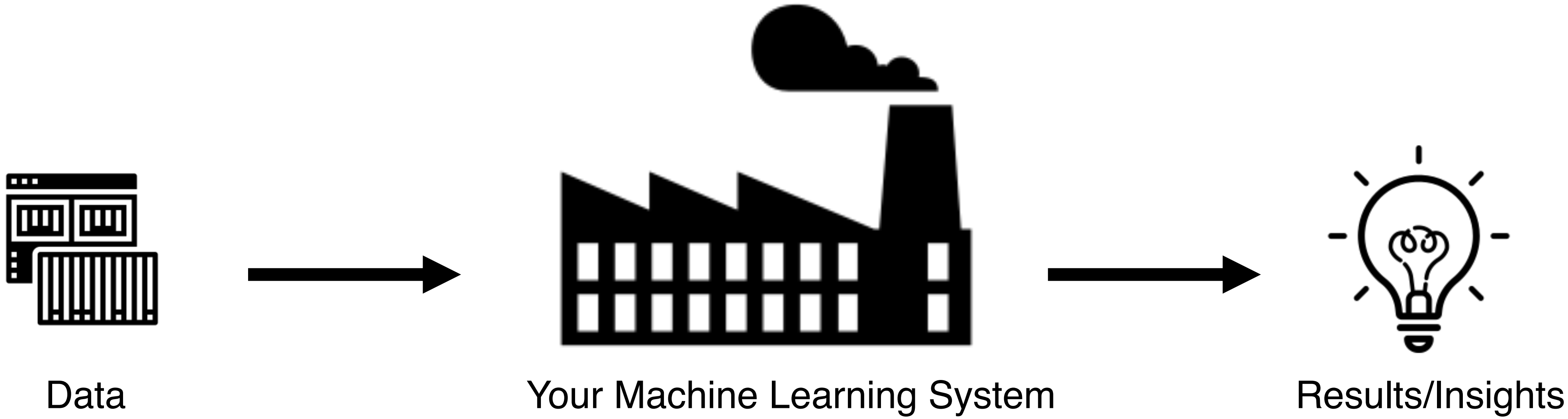
Outline

- Monitoring
    - ~~Monitoring Overview~~
    - Monitoring System Infrastructure
    - Monitoring Data Pipelines
    - Monitoring Model Performance
- Maintenance
    - Guide to Releasing a New Model

Monitoring System Infrastructure

Securing the Foundations!

The Factory Analogy

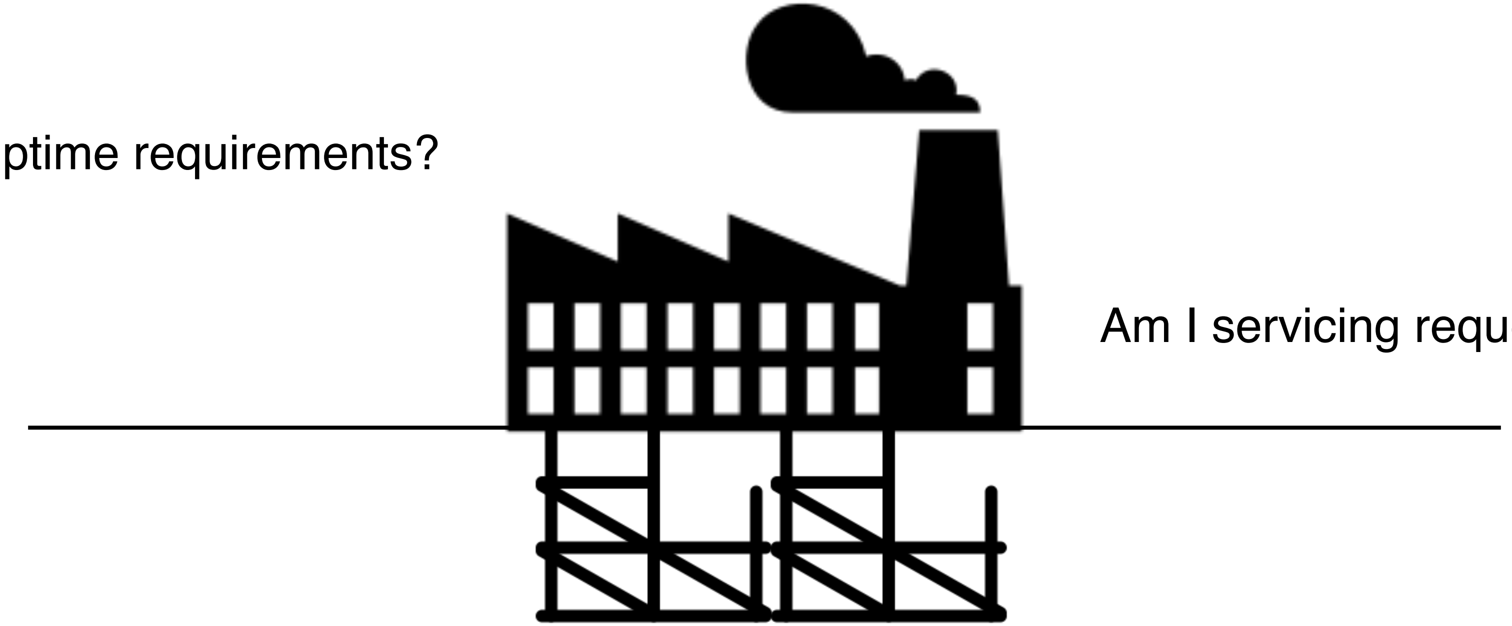


Monitoring System Infrastructure

Is the foundation strong?

Monitoring System Infrastructure

Am I meeting uptime requirements?

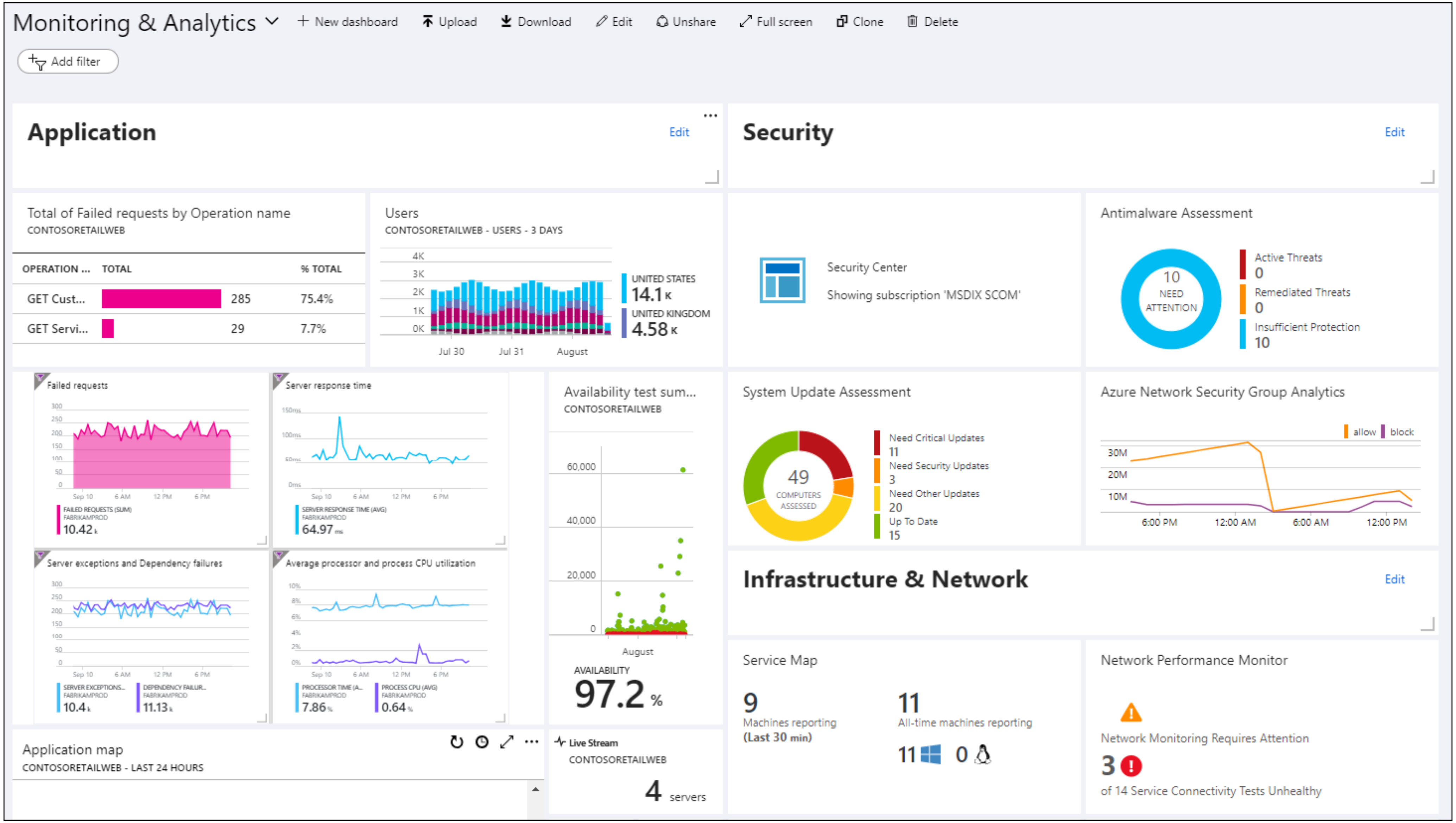


Am I servicing requests quickly enough?

Am I prepared if a code dependency changes?

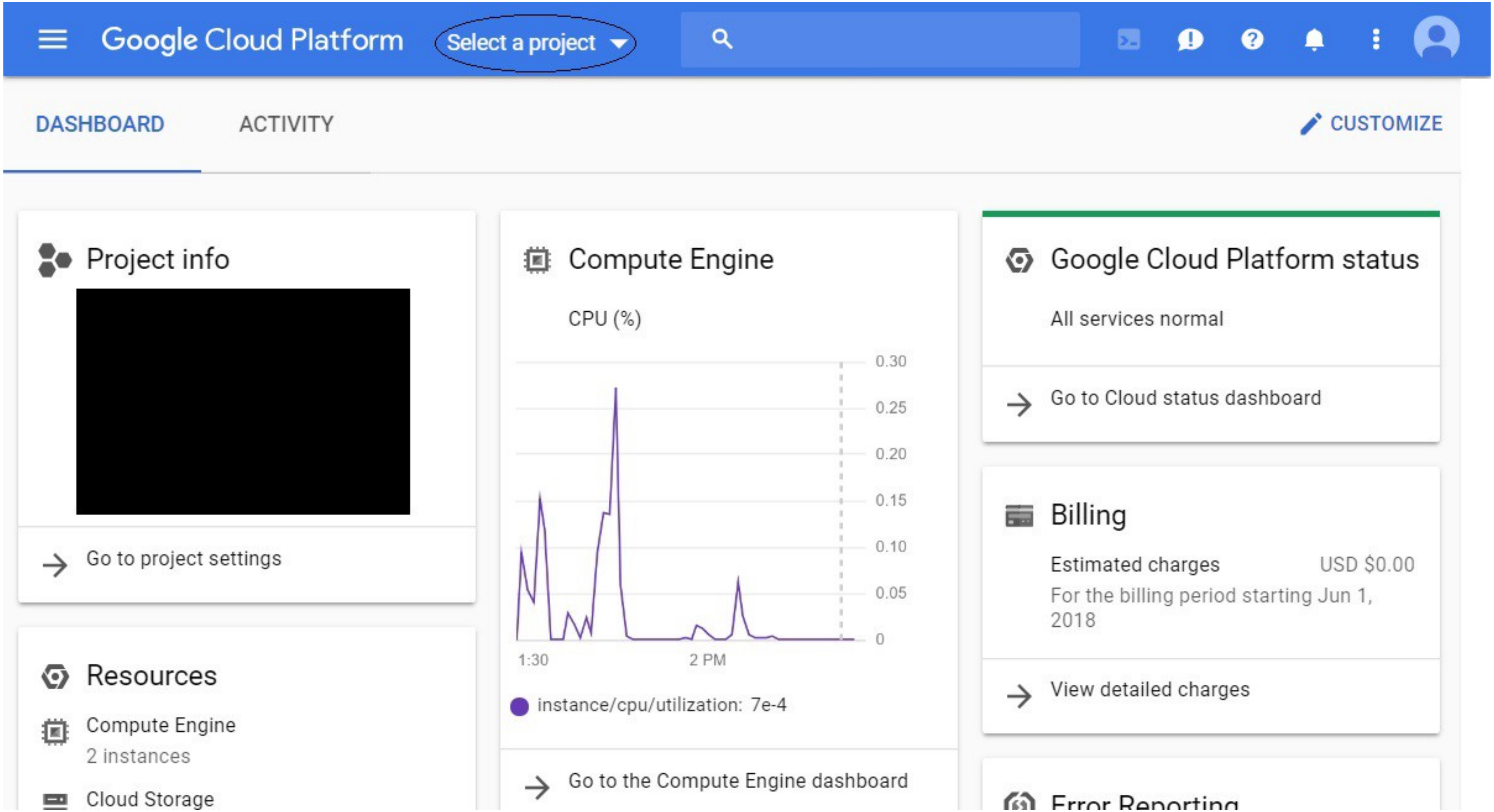
Am I making reasonable demands on my system's resources?

Likely Done For You: Monitoring Compute Resources



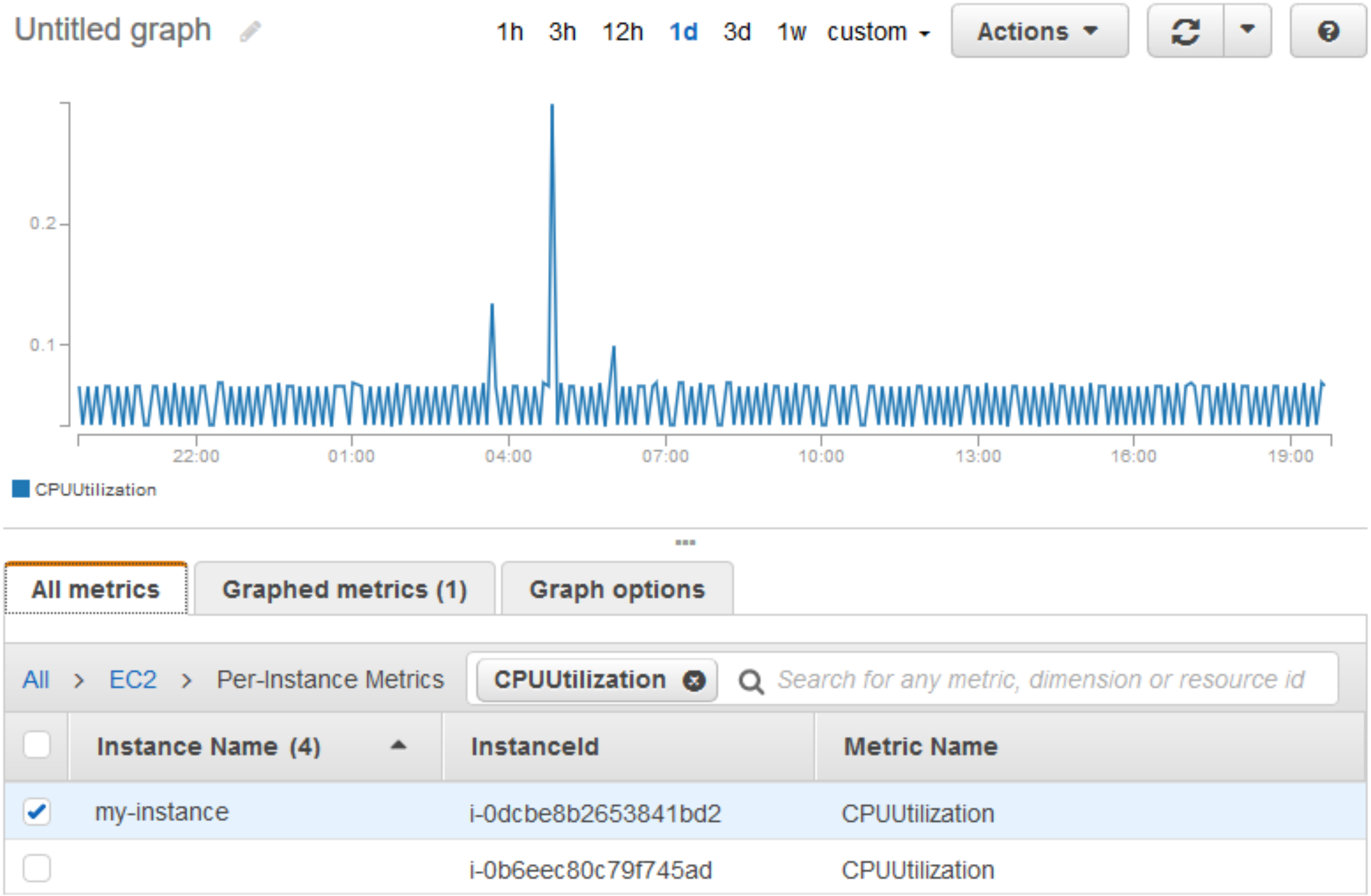
Azure Monitor

Source: https://docs.microsoft.com/en-us/azure/azure-monitor/overview



Google Cloud Dashboard

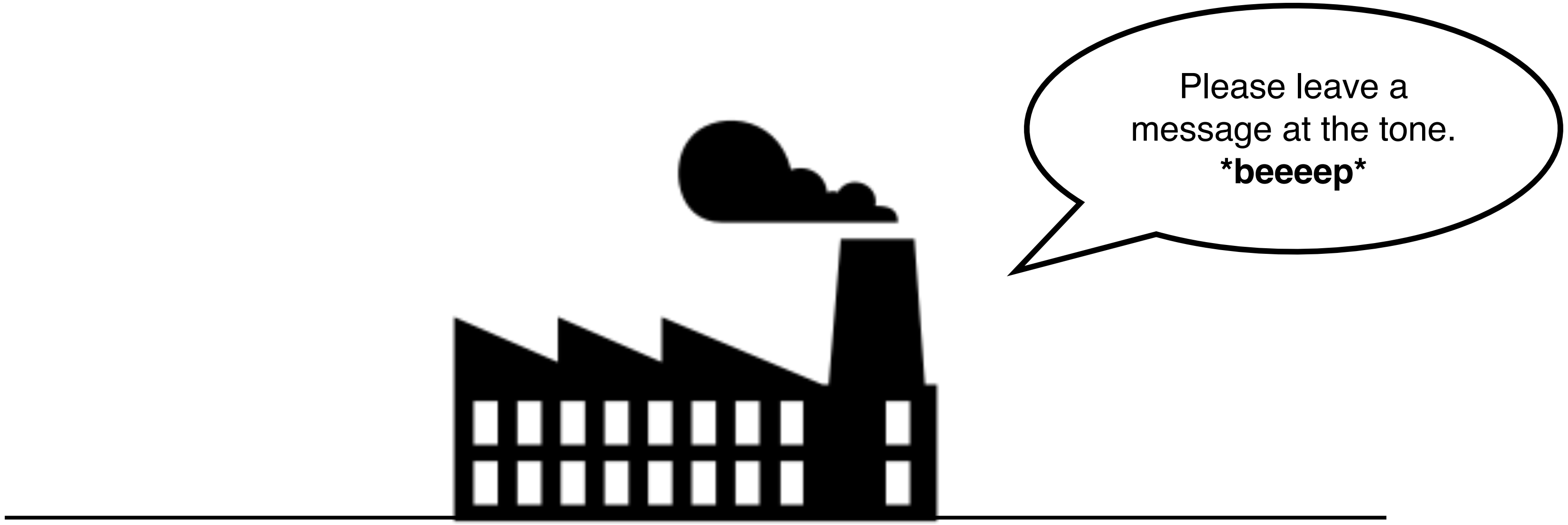
Source: https://medium.com/@piyushgoel997/setting-up-a-google-cloud-instance-for-deep-learning-

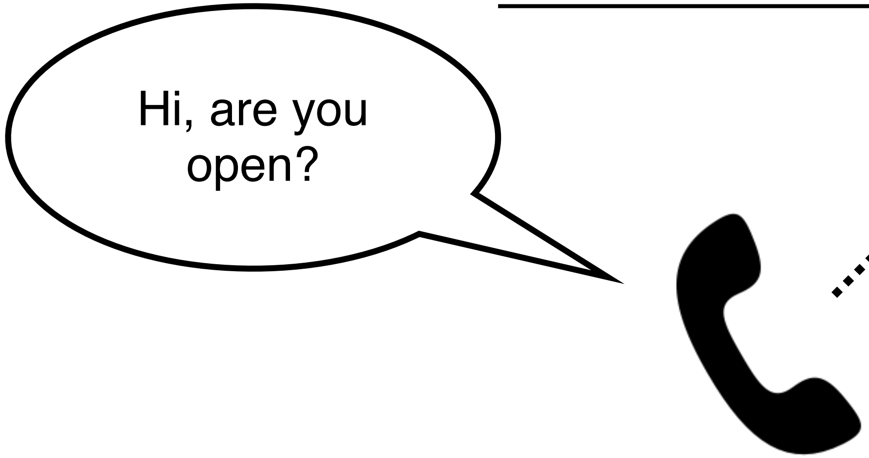


Amazon Cloudwatch

Source:

The Low-Hanging Fruit: Uptime and Latency

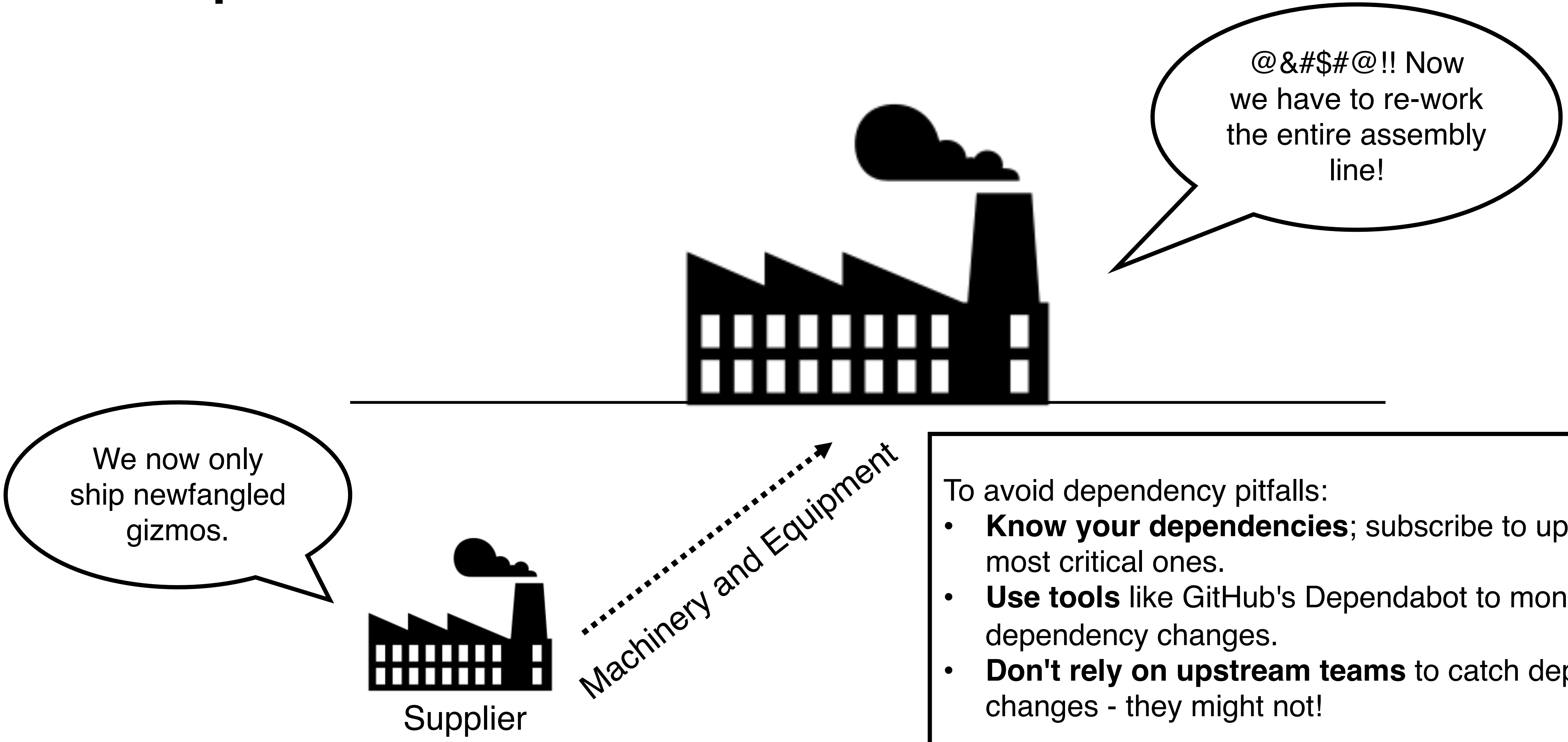




To check uptime and latency, query the application:

- **Query responses** (e.g. 200 vs. 404) can be indicative of downtime.
- **Timing decorators** within the application can calculate and log latency associated with incoming requests.

Getting Trickier: Depending on Dependencies



To avoid dependency pitfalls:

- **Know your dependencies**; subscribe to updates for the most critical ones.
- **Use tools** like GitHub's Dependabot to monitor and track dependency changes.
- **Don't rely on upstream teams** to catch dependency changes - they might not!

Dependabot: An Overview

How it works

1

Dependabot checks for updates

Dependabot pulls down your dependency files and looks for any outdated or insecure requirements.

2

Dependabot opens pull requests

If any of your dependencies are out-of-date, Dependabot opens individual pull requests to update each one.

3

You review and merge

You check that your tests pass, scan the included changelog and release notes, then hit merge with confidence.

Source: https://dependabot.com/#how-it-works

Logging: A Brief Detour

Logging is Boring.

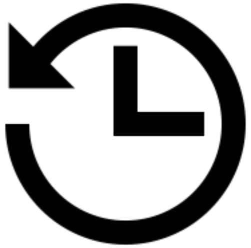
Until something bad happens!

```
2021-02-26 20:43:20,606:INFO: Average reward: -797.83 +/- 1
2021-02-26 20:43:42,931:INFO: Average reward: -1711.11 +/-
2021-02-26 20:44:06,407:INFO: Average reward: -597.93 +/- 1
2021-02-26 20:44:30,793:INFO: Average reward: -526.05 +/- 1
2021-02-26 20:44:54,532:INFO: Average reward: -520.63 +/- 9
2021-02-26 20:45:16,747:INFO: Average reward: -490.96 +/- 8
2021-02-26 20:45:39,303:INFO: Average reward: -481.73 +/- 1
2021-02-26 20:46:02,442:INFO: Average reward: -436.28 +/- 9
2021-02-26 20:46:25,954:INFO: Average reward: -397.38 +/- 8
2021-02-26 20:46:47,760:INFO: Average reward: -381.99 +/- 9
2021-02-26 20:47:09,469:INFO: Average reward: -360.61 +/- 6
2021-02-26 20:47:31,090:INFO: Average reward: -311.84 +/- 9
2021-02-26 20:47:52,830:INFO: Average reward: -309.95 +/- 9
2021-02-26 20:48:14,528:INFO: Average reward: -302.66 +/- 7
2021-02-26 20:48:36,235:INFO: Average reward: -250.24 +/- 8
2021-02-26 20:48:57,937:INFO: Average reward: -248.25 +/- 7
2021-02-26 20:49:19,612:INFO: Average reward: -236.26 +/- 8
2021-02-26 20:49:41,453:INFO: Average reward: -205.36 +/- 8
2021-02-26 20:50:03,778:INFO: Average reward: -200.46 +/- 6
2021-02-26 20:50:25,548:INFO: Average reward: -191.66 +/- 7
2021-02-26 20:50:47,243:INFO: Average reward: -166.35 +/- 7
2021-02-26 20:51:09,027:INFO: Average reward: -179.22 +/- 6
2021-02-26 20:51:31,219:INFO: Average reward: -172.91 +/- 7
2021-02-26 20:51:53,212:INFO: Average reward: -156.50 +/- 5
2021-02-26 20:52:16,895:INFO: Average reward: -138.76 +/- 6
```


```
2021-02-26 20:43:20,606:INFO: Average reward: -797.83 +/- 1
2021-02-26 20:43:42,931:INFO: Average reward: -1711.11 +/-
2021-02-26 20:44:06,407:INFO: Average reward: -597.93 +/- 1
2021-02-26 20:44:30,793:INFO: Average reward: -526.05 +/- 1
2021-02-26 20:44:54,532:INFO: Average reward: -520.63 +/- 9
2021-02-26 20:45:16,747:INFO: Average reward: -490.96 +/- 8
2021-02-26 20:45:39,303:INFO: Average reward: -481.73 +/- 1
2021-02-26 20:46:02,442:INFO: Average reward: -436.28 +/- 9
2021-02-26 20:46:25,954:INFO: Average reward: -397.38 +/- 8
2021-02-26 20:46:47,760:INFO: Average reward: -381.99 +/- 9
2021-02-26 20:47:09,469:INFO: Average reward: -360.61 +/- 6
2021-02-26 20:47:31,090:INFO: Average reward: -311.84 +/- 9
2021-02-26 20:47:52,830:INFO: Average reward: -309.95 +/- 9
2021-02-26 20:48:14,528:INFO: Average reward: -302.66 +/- 7
2021-02-26 20:48:36,235:INFO: Average reward: -250.24 +/- 8
2021-02-26 20:48:36,235:INFO: Average reward: -250.24 +/- 8
```

Why Does Logging Matter?

- When things break, you *must* be able to look over what happened.



"Those who cannot remember the past are condemned to repeat it."

Auditing and Compliance Requirements

A Logging Tutorial in 6 Lines of Code

```
1  import logging
2  logging.basicConfig(filename="example.log",
           encoding="utf-8", level=logging.DEBUG)
3  logging.debug("This message should go to the log file")
4  logging.info("So should this")
5  logging.warning("And this, too")
6  logging.error("And non-ASCII stuff, too, like Øresund and Malmö")
```

File: example.log

```
DEBUG:root:This message should go to the log file
INFO:root:So should this
WARNING:root:And this, too
ERROR:root:And non-ASCII stuff, too, like Øresund and Malmö
```

Source: https://docs.python.org/3/howto/logging.html

Logger object

Logs for different services/applications

- Sent to different destinations
- Easier to search/filter

```
import logging
model_logger = logging.get_logger("Modeling service")
model_logger.debug(f"Input {input}")
model_logger.info("Start training …")
model_logger.warning("Sequence longer than 512 tokens. You might want to truncate it.")
model_logger.error(f"File {filename} doesn't exist")
model_logger.critical("99% memory used. Will likely be OOM.")
```

Logger object: levels

Logs for different services/application s

- Sent to different destinations
- Easier to search/filter

More
severe

```
import logging
model_logger = logging.get_logger("Modeling service")
model_logger.debug(f"Input {input}")
model_logger.info("Start training …")
model_logger.warning("Sequence longer than 512 tokens. You might want to truncate it.")
model_logger.error(f"File {filename} doesn't exist")
model_logger.critical("99% memory used. Will likely be OOM.")
```

Logger object: levels

Set level of severeness:

- Logs less severe won't be shown

More severe

```
import logging
model_logger = logging.get_logger("Modeling service")
model_logger.setLevel("WARNING")
model_logger.debug(f"Input {input}")
model_logger.info("Start training …")
model_logger.warning("Sequence longer than 512 tokens. You might want to truncate it.")
model_logger.error(f"File {filename} doesn't exist")
model_logger.critical("99% memory used. Will likely be OOM.")
```

What Should You Be Logging?

Output prediction

Current model hyperparameters.

Which model you're running

How long model has been in production (date/time).

Date/time

More than you might think.

Number of requests served by the model.

True label (if available)

Historical data used as training data.

Input data

Min/max/average serving times.

Threshold to convert probabilities to binary.

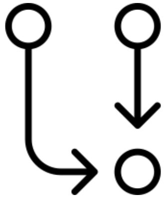
Source: https://eugeneyan.com/writing/practical-guide-to-maintaining-machine-learning/

Outline

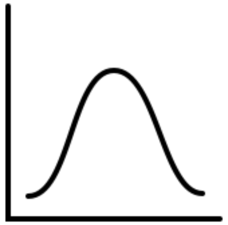
- Monitoring
  - ~~Monitoring Overview~~
  - ~~Monitoring System Infrastructure~~
  - Monitoring Data Pipelines
  - Monitoring Model Performance
- Maintenance
  - Guide to Releasing a New Model

Monitoring Data Pipelines

Data Validation



Data Dependencies



Data Distribution

Data Validation Starts with UX.

Please Enter Your Age

Please Enter Your Age

I'm 23 years old.

```
age = float(input("Please Enter Your Age: "))
```

Please Enter Your Age

I'm 23 years old.

Please enter a valid age.

Please Enter Your Age

NaN

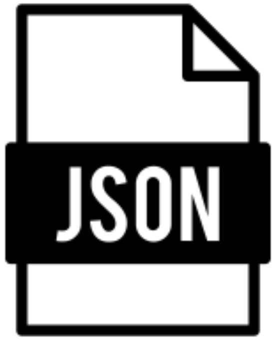
```
age = float(input("Please Enter Your Age: "))
```

Please Enter Your Age

NaN

Please enter a valid age.

The Low Hanging Fruit: Basic Data Validation



Correct Data Structure

No NaNs in the Data

Please Enter Your Age

Realistically, ages fall between 0 and 130.

Please Enter Your Age

-3.1415926525

Please Enter Your Age

-3.1415926525

Please enter a valid age.

Please Enter Your Age

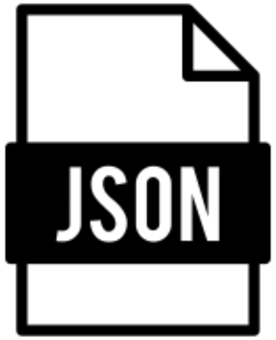
329

Please Enter Your Age

329

Please enter a valid age.

The Low Hanging Fruit: Basic Data Validation



Correct Data Structure

No NaNs in the Data

Basic Semantic Validation

Tools like pydantic are awesome for this!

A Little Trickier: Statistical Guarantees on Dataset Shift

- How can I know whether I'm experiencing dataset shift?

T-Test

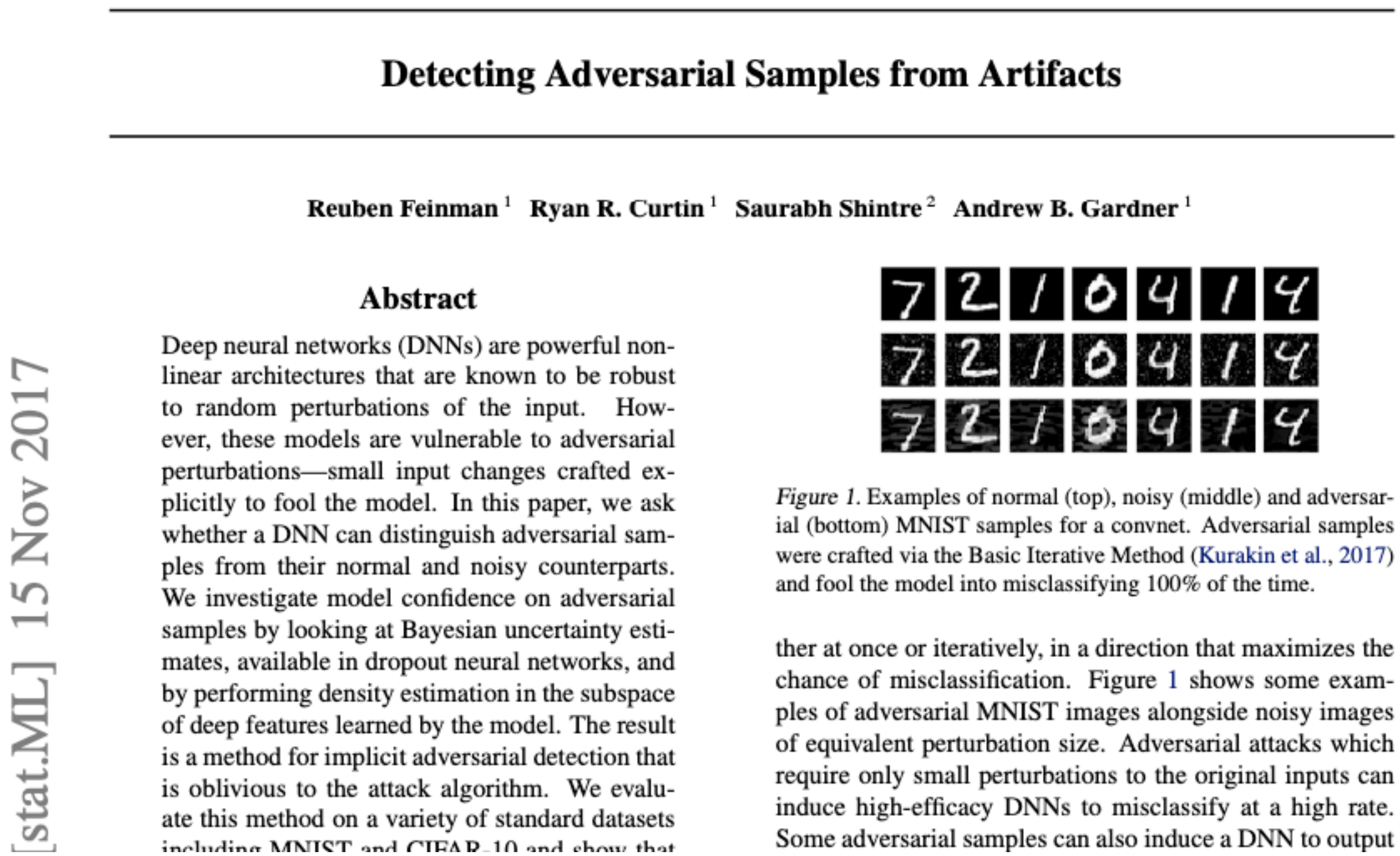
```
scipy.stats.ttest_ind(a, b, axis=0,
equal_var=True, nan_policy='propagate'
, alternative='two-sided')
```

ANOVA

```
scipy.stats.f_oneway(*args, axis=0)
```

- Each test returns a statistic (either t, or F), and a p-value.
- Assumptions
  - Normally distributed data
  - IID samples
  - Homogeneity of variance

Very Fancy: Statistical Data Validation



Detecting Adversarial Samples from Artifacts

Reuben Feinman [1] **Ryan R. Curtin** [1] **Saurabh Shintre** [2] **Andrew B. Gardner** [1]

Abstract

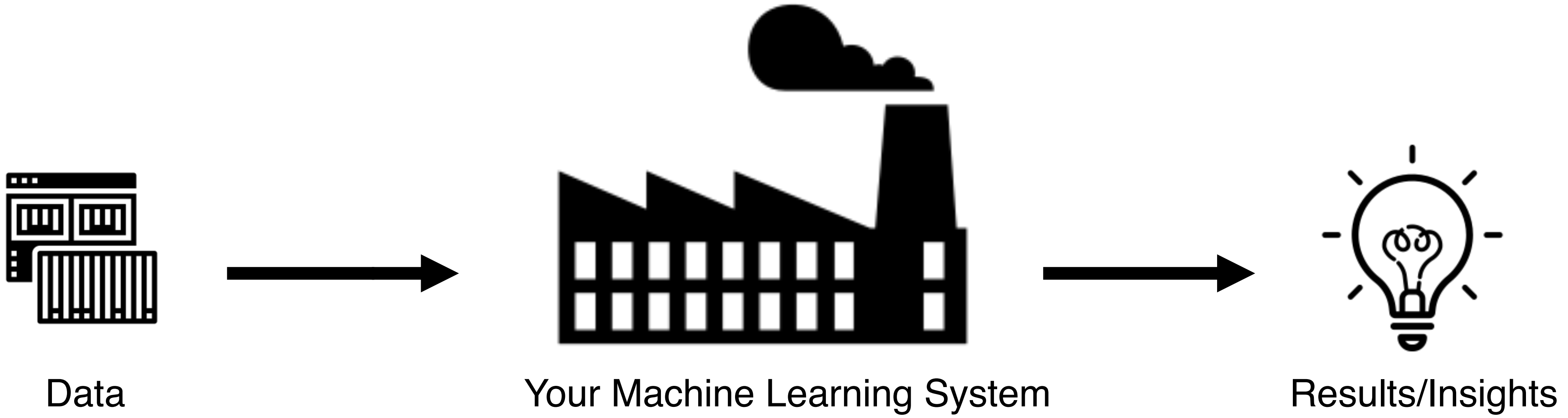
Deep neural networks (DNNs) are powerful non-linear architectures that are known to be robust to random perturbations of the input. However, these models are vulnerable to adversarial perturbations—small input changes crafted explicitly to fool the model. In this paper, we ask whether a DNN can distinguish adversarial samples from their normal and noisy counterparts. We investigate model confidence on adversarial samples by looking at Bayesian uncertainty estimates, available in dropout neural networks, and by performing density estimation in the subspace of deep features learned by the model. The result is a method for implicit adversarial detection that is oblivious to the attack algorithm. We evaluate this method on a variety of standard datasets including MNIST and CIFAR-10 and show that

Figure 1. Examples of normal (top), noisy (middle) and adversarial (bottom) MNIST samples for a convnet. Adversarial samples were crafted via the Basic Iterative Method (Kurakin et al., 2017) and fool the model into misclassifying 100% of the time.

ther at once or iteratively, in a direction that maximizes the chance of misclassification. Figure 1 shows some examples of adversarial MNIST images alongside noisy images of equivalent perturbation size. Adversarial attacks which require only small perturbations to the original inputs can induce high-efficacy DNNs to misclassify at a high rate. Some adversarial samples can also induce a DNN to output

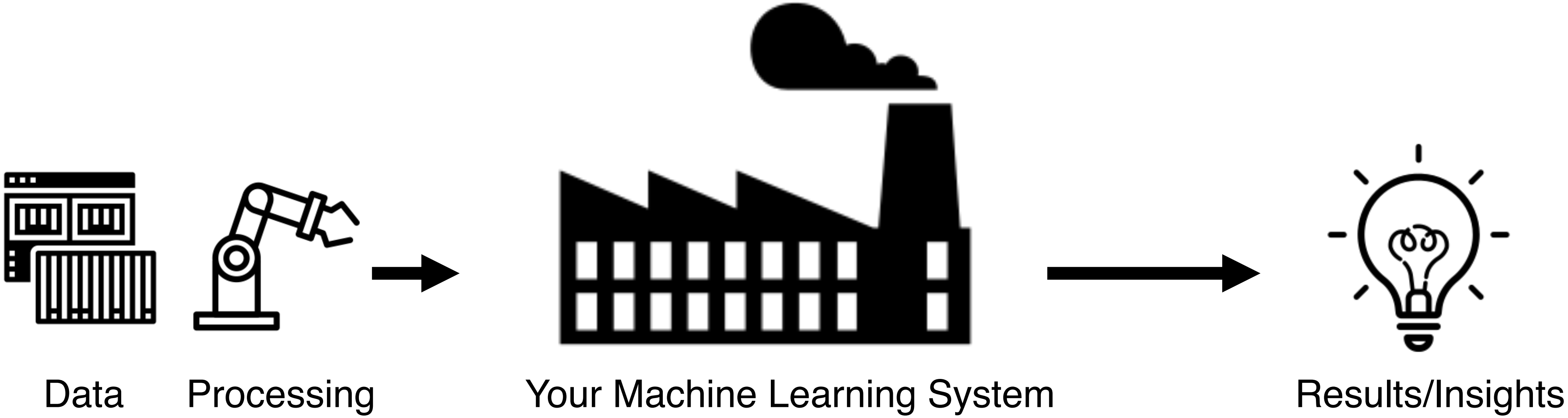
Source: https://arxiv.org/pdf/1703.00410.pdf

Track Data Dependencies for Instability



Source: https://papers.nips.cc/paper/2015/file/86df7dcfd896fcaf2674f757a2463eba-Paper.pdf

Managing Unstable Data Dependencies



Problem: **instability in input**.

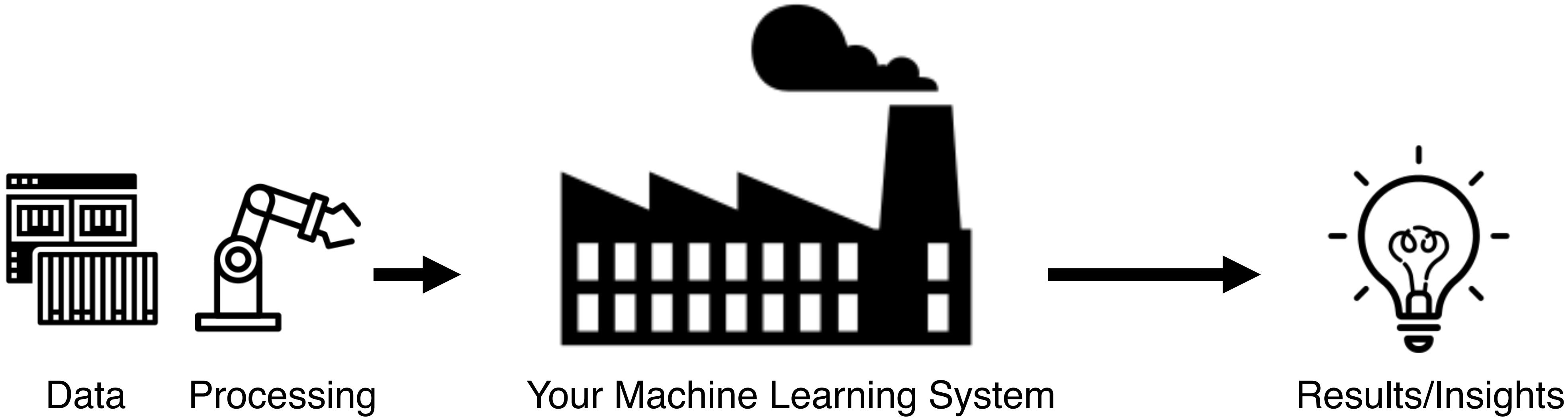
- **Explicit** – pre-processing procedure might change.
- **Implicit** – signal comes from ML model that changes over time.

Solution: **input system versioning**.

- Retain a frozen version of the processing model, so that it cannot change.
- Vet updates to the frozen version so that your system is ready for changes.

Source: https://papers.nips.cc/paper/2015/file/86df7dcfd896fcaf2674f757a2463eba-

Managing Underutilized Data Dependencies



Problem: **some input signals don't improve the model.**

- Legacy features, bundled features, correlated features, .
- Make ML system unnecessarily vulnerable.

Solution: **data dependency pruning**.

- Detect by leave-one-out evaluations.

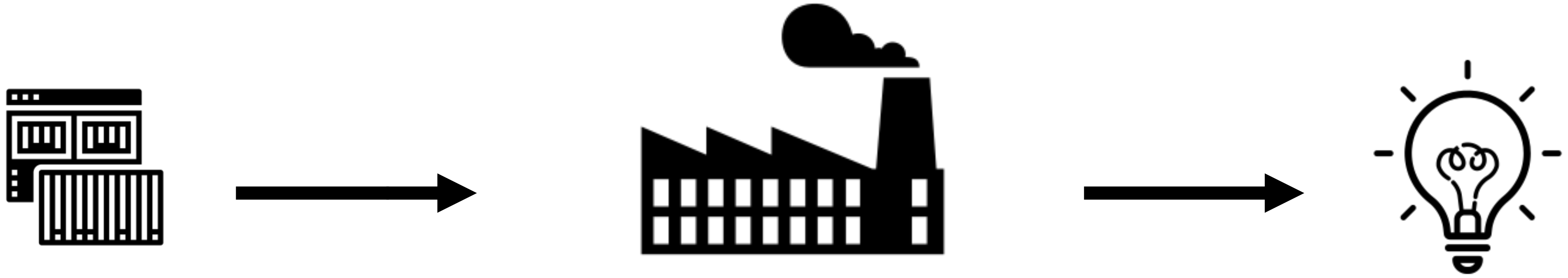
Source: https://papers.nips.cc/paper/2015/file/86df7dcfd896fcaf2674f757a2463eba-

Outline

- Monitoring
  - ~~Monitoring Overview~~
  - ~~Monitoring System Infrastructure~~
  - ~~Monitoring Data Pipelines~~
  - Monitoring Model Performance
- Maintenance
  - Guide to Releasing a New Model

Monitoring Model Performance

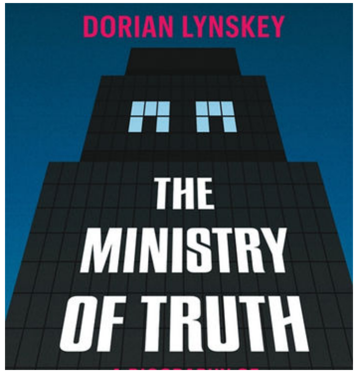
The Basics: Monitoring Output Performance Statistics



Data

Your Machine Learning System

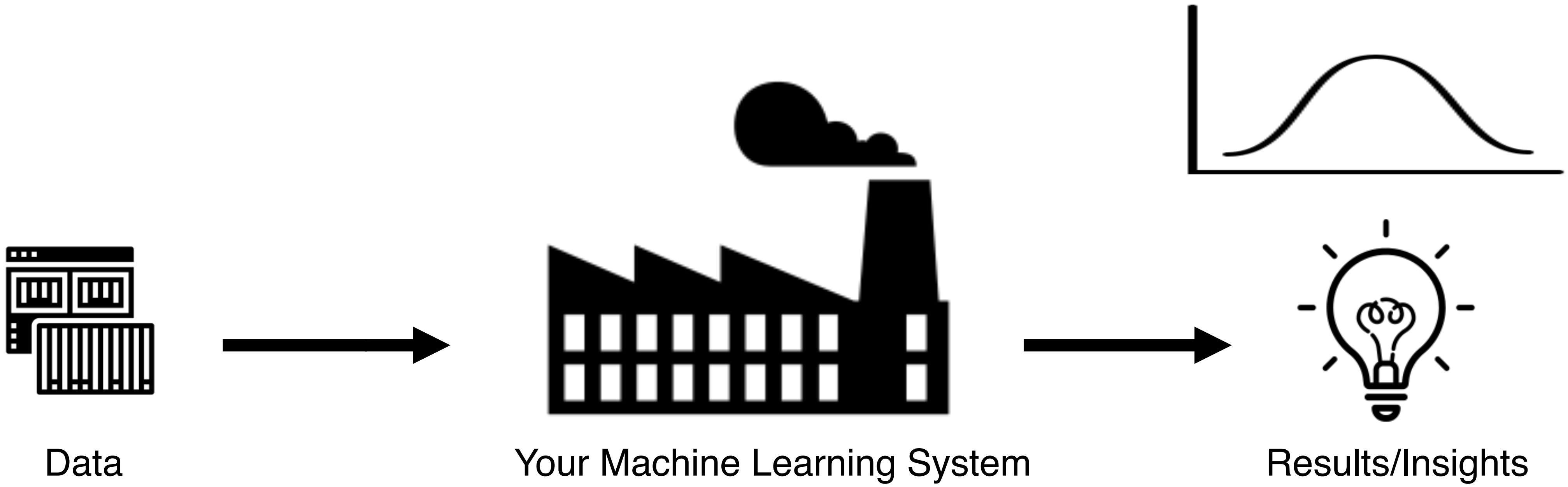
Results/Insights



Ground Truth

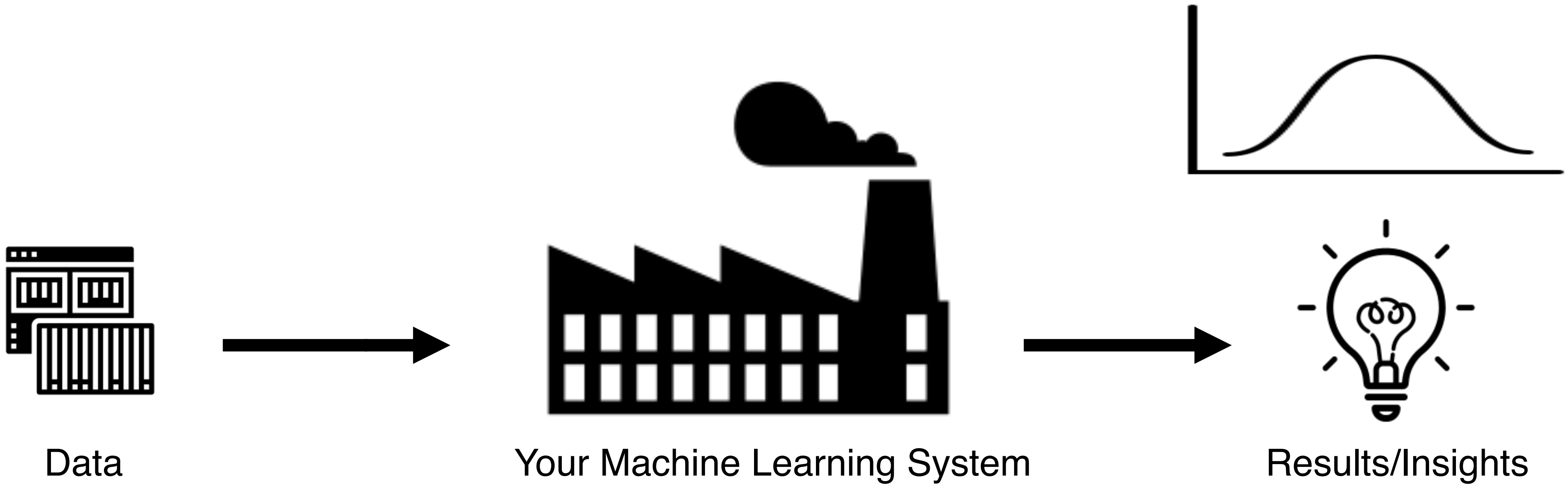
Is my model getting the right answer?

The Basics: Monitoring the Output Distribution



Didn't we just talk about distribution shift?

The Basics: Monitoring the Output Distribution



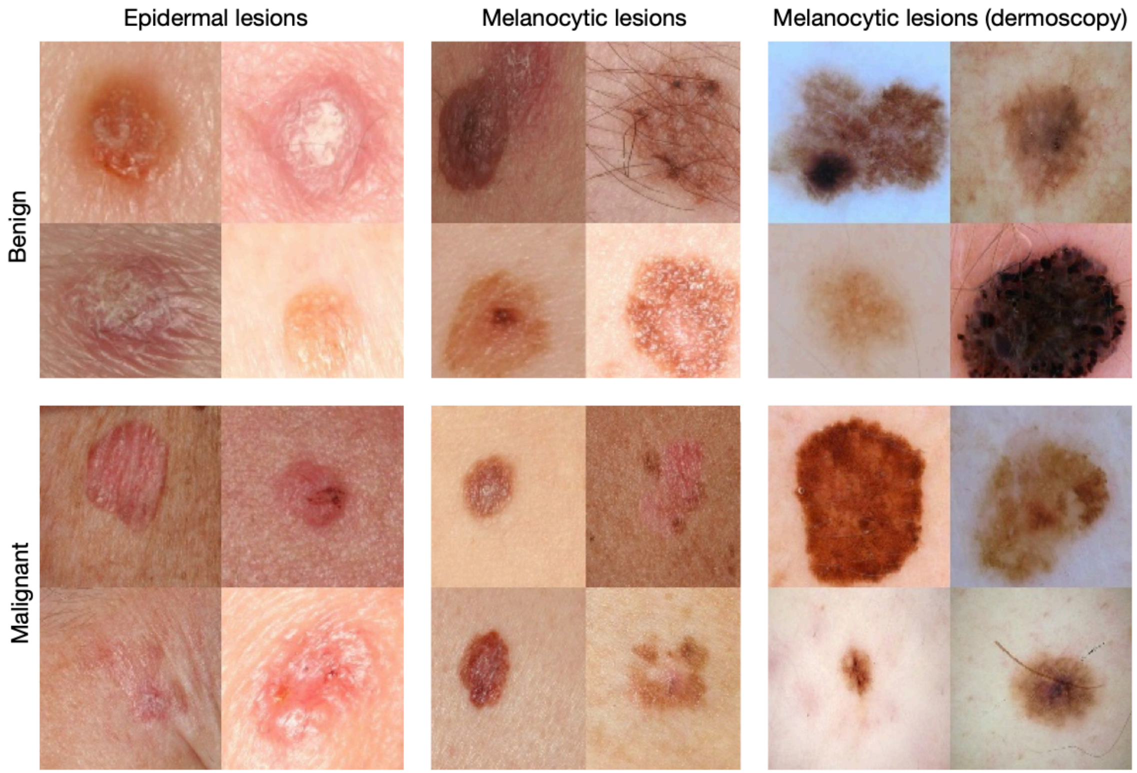
Monitor "distance" between label distribution with ground-truth distribution.

Monitor changes in the label distribution if you have an established data flywheel/retraining procedure.

Getting More Advanced: Model Auditing and Interpretability

Source: https://www.nature.com/articles/nature21056.pdf

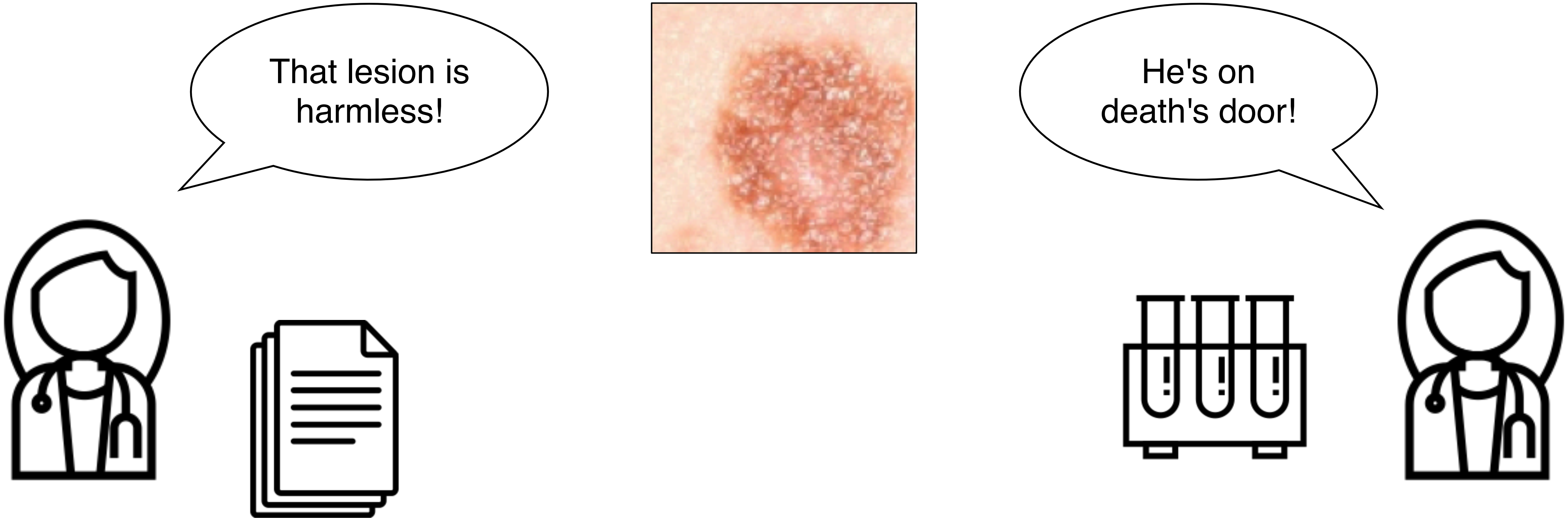
Getting More Advanced: Model Auditing and Interpretability



Is this skin lesion benign or malignant?

Source: https://www.nature.com/articles/nature21056.pdf

Getting More Advanced: Model Auditing and Interpretability



Source: https://www.nature.com/articles/nature21056.pdf

Reasoning Matters.

Can We Obtain *Explanations* from Models?

Source: https://www.nature.com/articles/nature21056.pdf

A Trivial Case: Linear Models

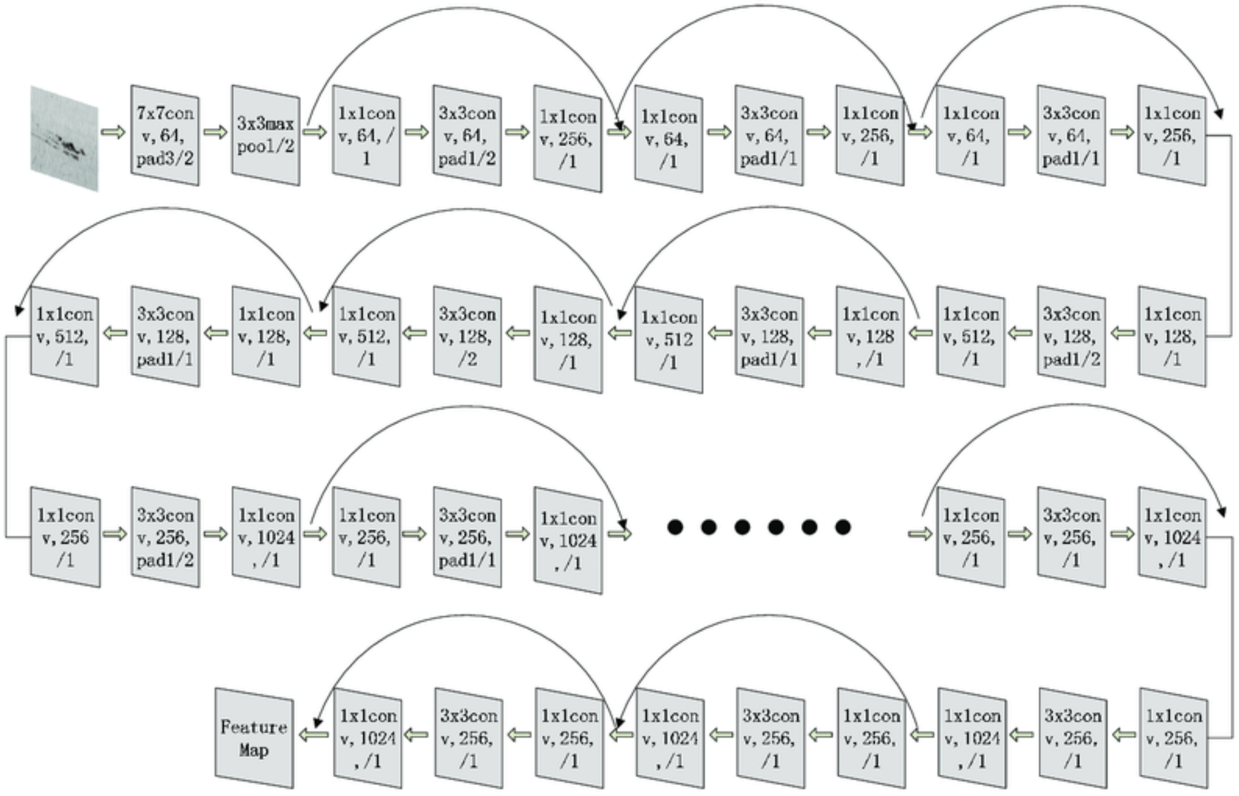
The **colour** of the lesion led us to believe it was benign.

$$y = \sum w_i \phi_i(x)$$

The **size** indicates it might be malignant.

A Less Trivial Case: Deep Models

Huh? How do I read that?



Beats me, Bob. Let's go get a drink.

Source: https://www.researchgate.net/publication/338087022_Object_Detection_and_Classification_of_Metal_Polishing_Shaft_Surface_Defects_Based_on_Convolutional_Neural_Network_Deep_Learning

“Why Should I Trust You?” Explaining the Predictions of Any Classifier

Marco Tulio Ribeiro
University of Washington
Seattle, WA 98105, USA
marcotcr@cs.uw.edu

Sameer Singh
University of Washington
Seattle, WA 98105, USA
sameer@cs.uw.edu

Carlos Guestrin
University of Washington
Seattle, WA 98105, USA
guestrin@cs.uw.edu

ABSTRACT

Despite widespread adoption, machine learning models remain mostly black boxes. Understanding the reasons behind predictions is, however, quite important in assessing *trust*, which is fundamental if one plans to take action based on a prediction, or when choosing whether to deploy a new model. Such understanding also provides insights into the model, which can be used to transform an untrustworthy model or prediction into a trustworthy one.

In this work, we propose LIME, a novel explanation technique that explains the predictions of *any* classifier in an interpretable and faithful manner, by learning an interpretable model locally around the prediction. We also propose a

how much the human understands a model’s behaviour, as opposed to seeing it as a black box.

Determining trust in individual predictions is an important problem when the model is used for decision making. When using machine learning for medical diagnosis [6] or terrorism detection, for example, predictions cannot be acted upon on blind faith, as the consequences may be catastrophic.

Apart from trusting individual predictions, there is also a need to evaluate the model as a whole before deploying it “in the wild”. To make this decision, users need to be confident that the model will perform well on real-world data, according to the metrics of interest. Currently, models are evaluated using accuracy metrics on an available validation dataset. However, real-world data is often significantly different, and

Source: https://arxiv.org/pdf/1602.04938.pdf

LIME: An Overview

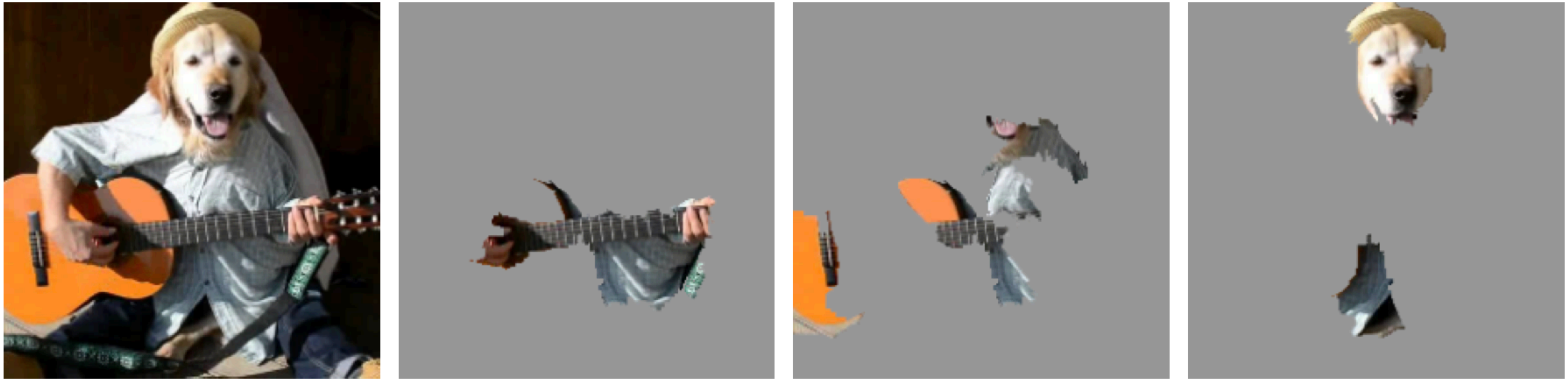
- Learn an interpretable model locally around the prediction.
- *Local surrogate* model: interpretable model, must be good local approximation, does not need to be good global approximation.

$$\xi(x) = \arg\min_{g \in G} L(f, g, \pi_x) + \Omega(g)$$

- f is opaque model; g is interpretable model; $\pi_x(z)$ is a proximity measure between x, z; $\Omega(g)$ measure of complexity.

Source: https://arxiv.org/pdf/1602.04938.pdf

LIME: An Overview



(a) Original Image (b) Explaining *Electric guitar* (c) Explaining *Acoustic guitar* (d) Explaining *Labrador*

Figure 4: Explaining an image classification prediction made by Google's Inception neural network. The top 3 classes predicted are "Electric Guitar" ($p = 0.32$), "Acoustic guitar" ($p = 0.24$) and "Labrador" ($p = 0.21$)

Source: https://arxiv.org/pdf/1602.04938.pdf

LIME: An Overview

(a) Husky classified as wolf (b) Explanation

Source: https://arxiv.org/pdf/1602.04938.pdf

SHAP: Bringing it All Together

A Unified Approach to Interpreting Model Predictions

Scott M. Lundberg
Paul G. Allen School of Computer Science
University of Washington
Seattle, WA 98105
slund1@cs.washington.edu

Su-In Lee
Paul G. Allen School of Computer Science
Department of Genome Sciences
University of Washington
Seattle, WA 98105
suinlee@cs.washington.edu

Abstract

Understanding why a model makes a certain prediction can be as crucial as the prediction's accuracy in many applications. However, the highest accuracy for large modern datasets is often achieved by complex models that even experts struggle to interpret, such as ensemble or deep learning models, creating a tension between *accuracy* and *interpretability*. In response, various methods have recently been

Source: https://arxiv.org/pdf/1705.07874.pdf

SHAP: An Overview

- Additive Feature Attribution Methods.

$$g(z') = \phi_0 + \sum_{i=1}^{M} \phi_i z_i'$$

- Required Properties:
    - Local accuracy (estimator is faithful to the local model).
    - Missingness (constrain features to zero if they have no impact).
    - Consistency
- Only one possible explanation model g is an additive feature attribution model with these requirements.

Source: https://arxiv.org/pdf/1705.07874.pdf

SHAP: An Overview

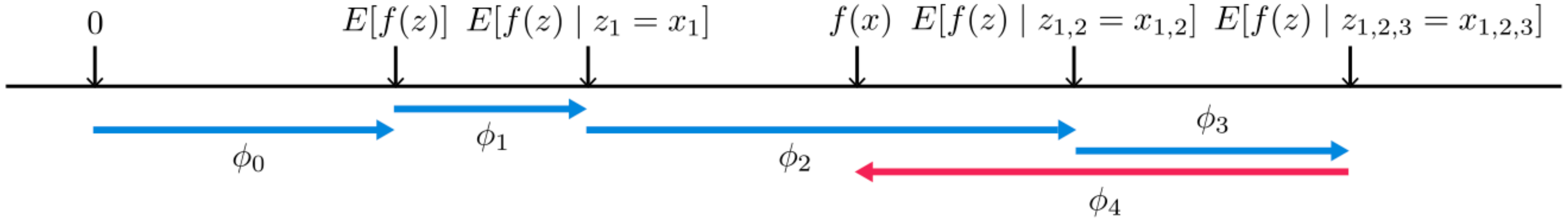
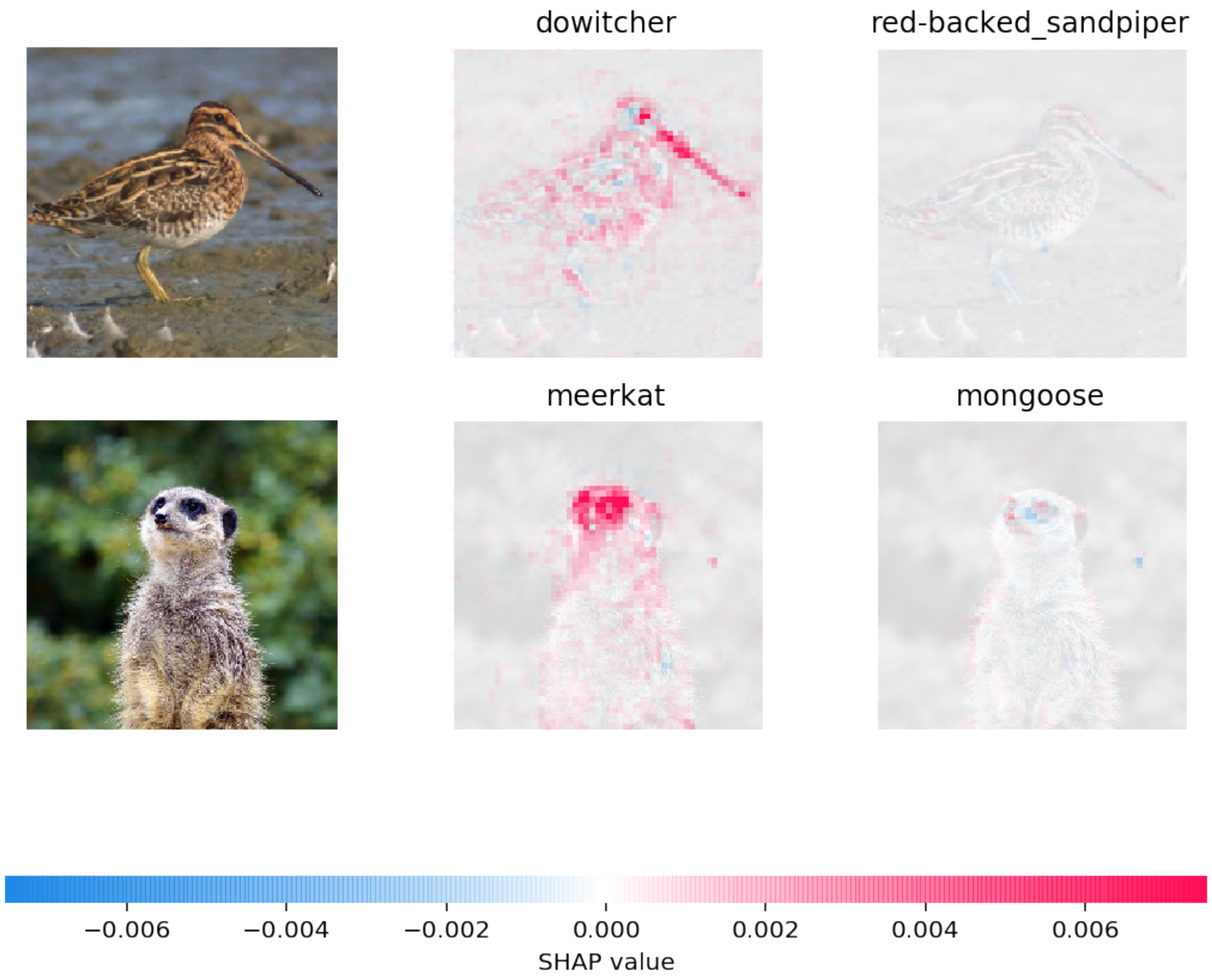


Figure 1: SHAP (SHapley Additive exPlanation) values attribute to each feature the change in the expected model prediction when conditioning on that feature. They explain how to get from the base value $E[f(z)]$ that would be predicted if we did not know any features to the current output $f(x)$. This diagram shows a single ordering. When the model is non-linear or the input features are

Source: https://arxiv.org/pdf/1705.07874.pdf

SHAP: An Overview



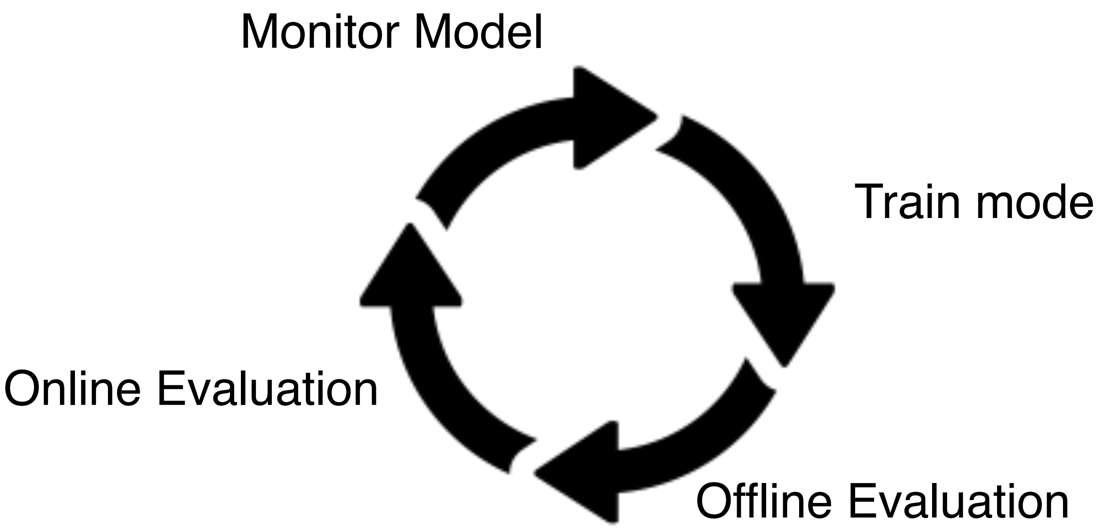
Source: https://github.com/slundberg/shap

Outline

- Monitoring
    - ~~Monitoring Overview~~
    - ~~Monitoring System Infrastructure~~
    - ~~Monitoring Data Pipelines~~
    - ~~Monitoring Model Performance~~
- Maintenance
    - Guide to Releasing a New Model

The Maintenance Cycle

The Maintenance Cycle

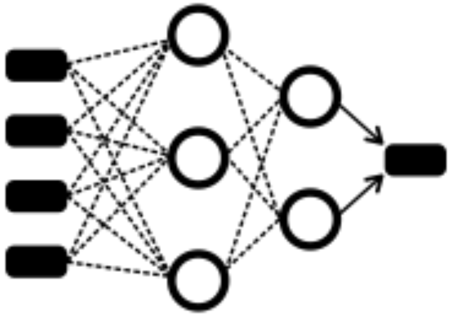


Performance Validation

Latency concerns?

More powerful/expressive architecture?

Dataset drift /
Poor slice performance?



Compute concerns?

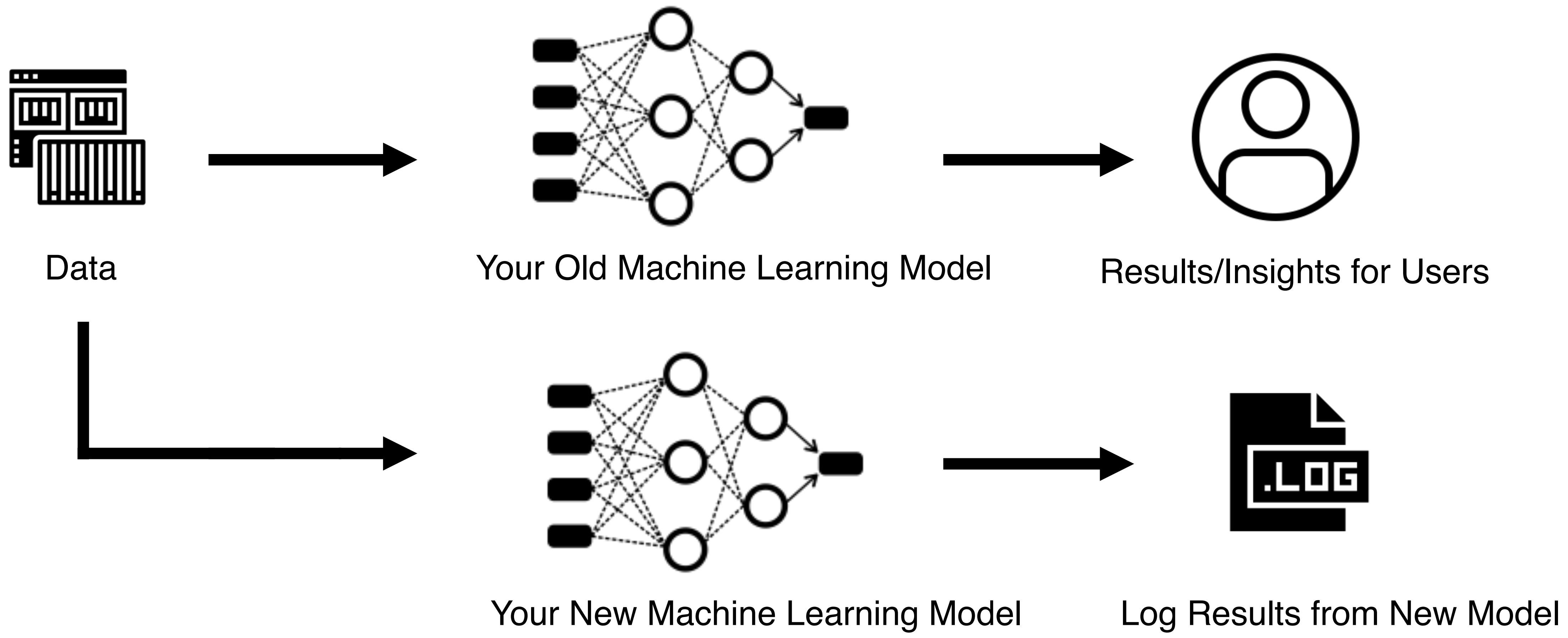
Why are you deploying a new model?

Judiciously choose your validation hold-out set to meet your objectives.

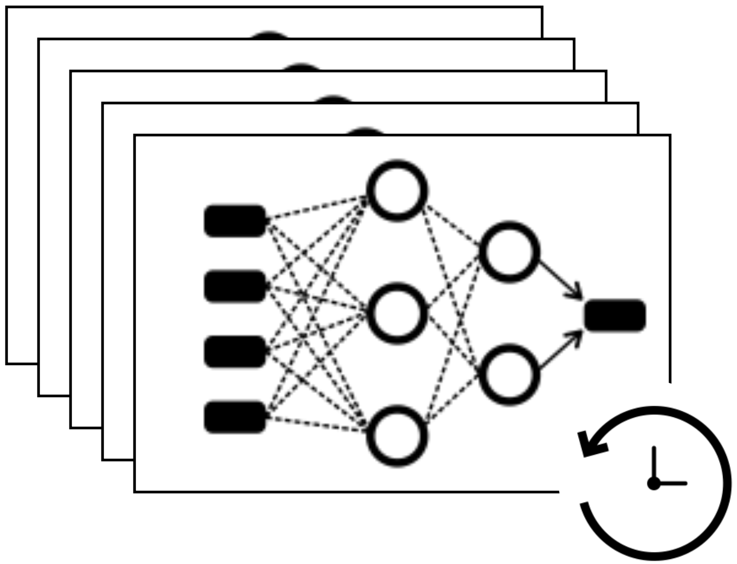
Source: https://eugeneyan.com/writing/practical-guide-to-maintaining-machine-learning/

Shadow Release

Latency?
Stability?
Error Rate?
False Positives/Negatives?
Precision/Recall?
etc.



Monitor Model Health



Build in rollback capacity if anything goes wrong.

Actively pursue simplicity.

Outline

- Monitoring
  - ~~Monitoring Overview~~
  - ~~Monitoring System Infrastructure~~
  - ~~Monitoring Data Pipelines~~
  - ~~Monitoring Model Performance~~
- Maintenance
  - ~~Guide to Releasing a New Model~~

Dashboard Demo

Prizes!

1

2

9

4

5

3

6

7

8

10

Thank You!