MODEL QUALITY

Christian Kaestner

Required reading:

- Hulten, Geoff. "Building Intelligent Systems: A Guide to Machine Learning Engineering." Apress, 2018, Chapter 19 (Evaluating Intelligence).
- Ribeiro, Marco Tulio, Tongshuang Wu, Carlos Guestrin, and Sameer Singh. "Beyond Accuracy: Behavioral Testing of NLP Models with CheckList." In Proceedings ACL, p. 4902–4912. (2020).

LEARNING GOALS

- Select a suitable metric to evaluate prediction accuracy of a model and to compare multiple models
- Select a suitable baseline when evaluating model accuracy
- Explain how software testing differs from measuring prediction accuracy of a model
- Curate validation datasets for assessing model quality, covering subpopulations and capabilities as needed
- Use invariants to check partial model properties with automated testing
- Avoid common pitfalls in evaluating model quality
- Select and deploy automated infrastructure to evaluate and monitor model quality

MODEL QUALITY

FIRST PART: MEASURING PREDICTION ACCURACY

the data scientist's perspective

SECOND PART: WHAT IS CORRECTNESS ANYWAY?

the role and lack of specifications, validation vs verification

THIRD PART: LEARNING FROM SOFTWARE TESTING

unit testing, test case curation, invariants, test case generation

testing in production (next week)

MODEL QUALITY VS SYSTEM QUALITY

PREDICTION ACCURACY OF A MODEL

- *model:* $X \rightarrow Y$

- *validation data (tests?):* sets of (X, Y) pairs indicating desired outcomes for select inputs

For our discussion: any form of model, including machine learning models, scientific models, hardcoded heuristics, composed models, ...

COMPARING MODELS

Compare two models (same or different implementation/learning technology) for the same task:

- Which one supports the system goals better?
- Which one makes fewer important mistakes?
- Which one is easier to operate?
- Which one is better overall?
- Is either one good enough?

ML ALGORITHM QUALITY VS MODEL QUALITY VS DATA QUALITY VS SYSTEM QUALITY

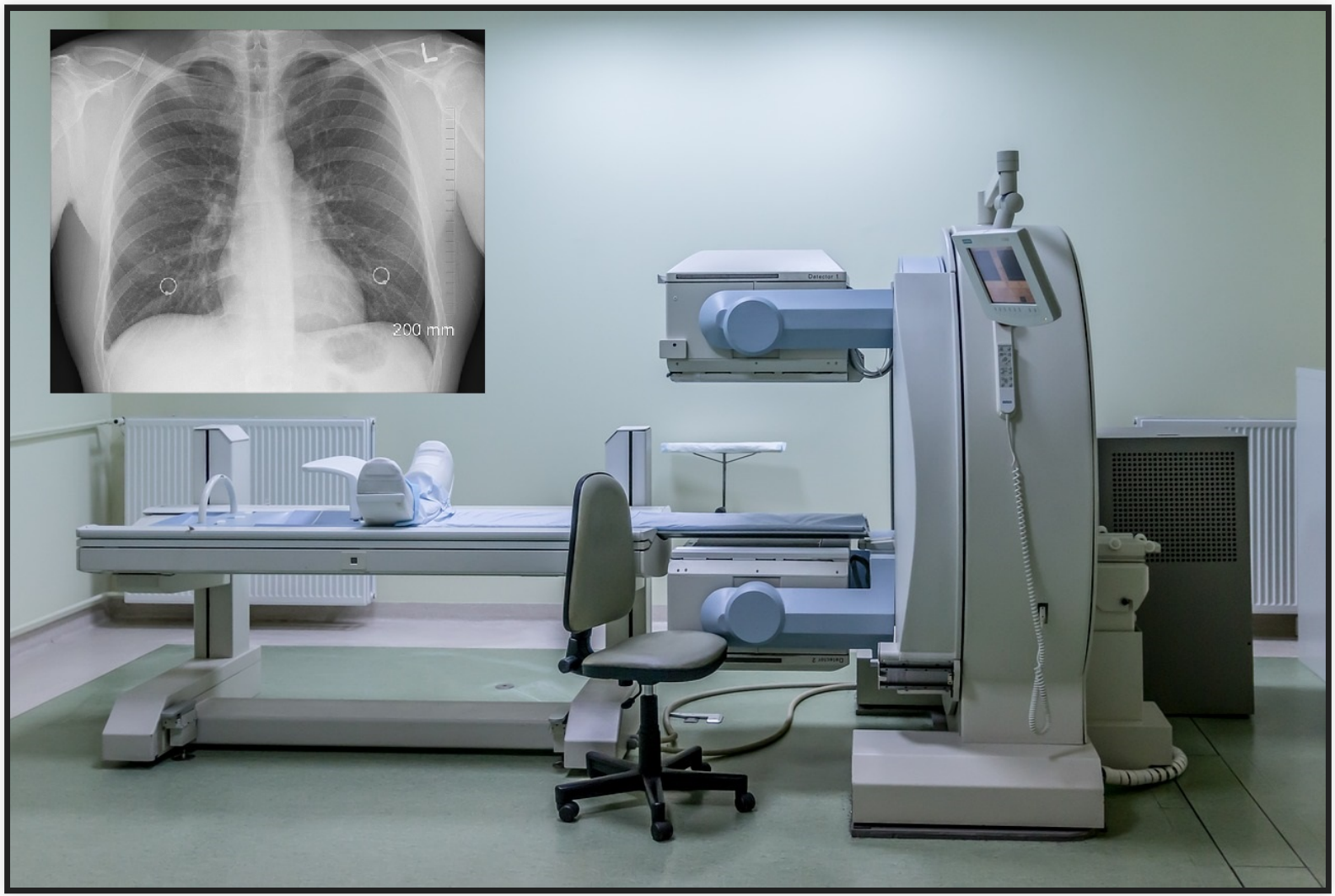
Todays focus is on the quality of the produced *model*, not the algorithm used to learn the model or the data used to train the model

i.e. assuming *Decision Tree Algorithm* and feature extraction are correctly implemented (according to specification), is the model learned from data any good?

The model is just one component of the entire system.

Focus on measuring quality, not debugging the source of quality problems (e.g., in data, in feature extraction, in learning, in infrastructure)

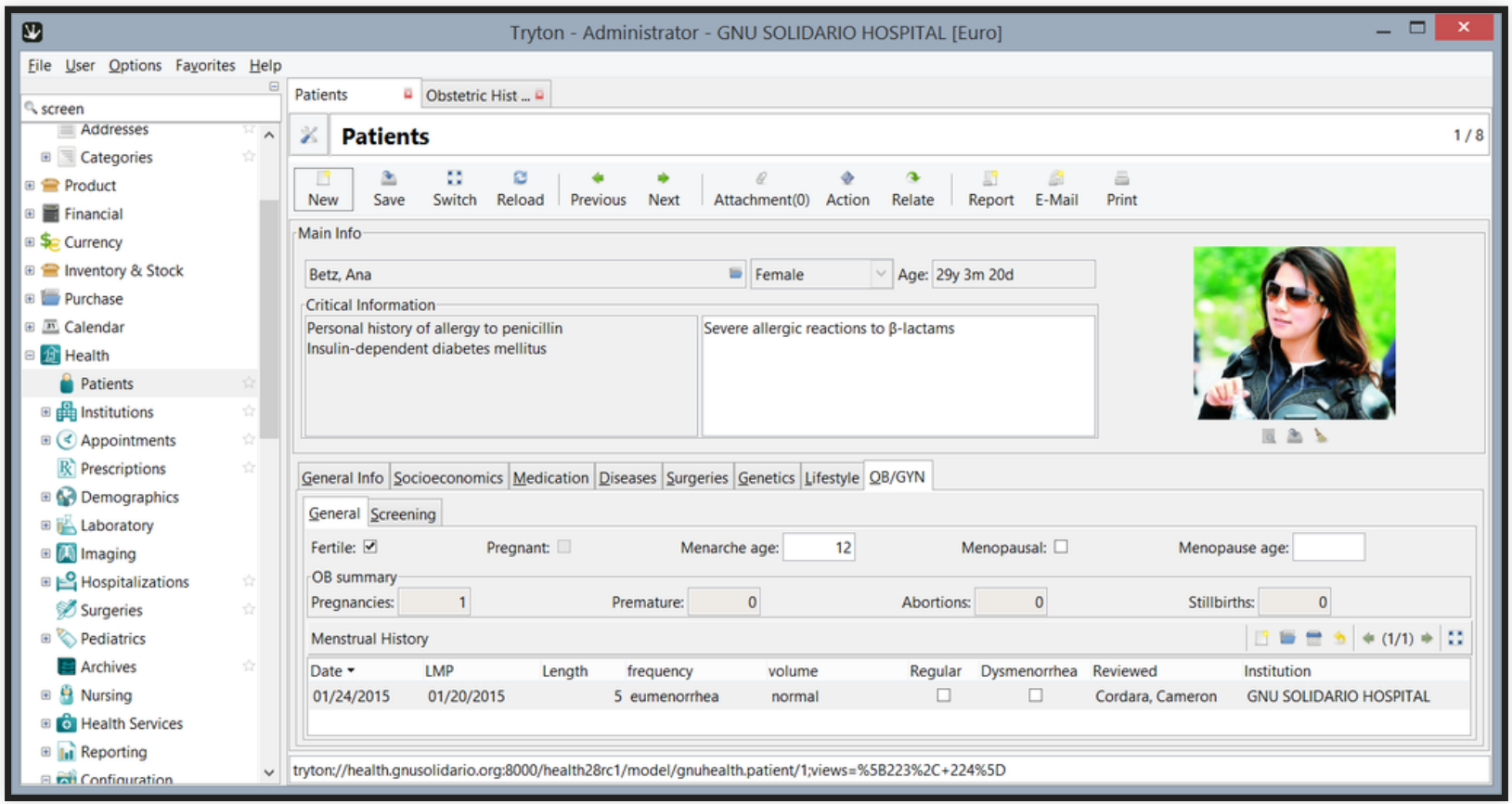
CASE STUDY: CANCER DETECTION



Speaker notes

Application to be used in hospitals to screen for cancer, both as routine preventative measure and in cases of specific suspicions. Supposed to work together with physicians, not replace.

THE MODEL IS PART OF A SYSTEM IN AN ENVIRONMENT



MANY QUALITIES

Prediction accuracy of a model is important

But many other quality matters when building a system:

- Model size
- Inference time
- User interaction model
- Kinds of mistakes made
- How the system deals with mistakes
- Ability to incrementally learn
- Safety, security, fairness, privacy
- Explainability

Today: Narrow focus on prediction accuracy of the model

ON TERMINOLOGY: PERFORMANCE

In machine learning, "performance" typically refers to accuracy

"this model performs better" = it produces more accurate results

Be aware of ambiguity across communities (see also: performance in arts, job performance, company performance, performance test (bar exam) in law, software/hardware/network performance)

- When speaking of "**time**", be explicit: "learning time", "inference time", "latency", ...
- When speaking of model **accuracy** use "prediction accuracy", ...

MEASURING PREDICTION ACCURACY FOR CLASSIFICATION TASKS

(The Data Scientists Toolbox)

CONFUSION/ERROR MATRIX

	Actually Grade 5 Cancer	Actually Grade 3 Cancer	Actually Benign
Model predicts Grade 5 Cancer	**10**	6	2
Model predicts Grade 3 Cancer	3	**24**	10
Model predicts Benign	5	22	**82**

$$\text{accuracy} = \frac{\text{correct predictions}}{\text{all predictions}}$$

$$\text{Example's accuracy} = \frac{10+24+82}{10+6+2+3+24+10+5+22+82} = .707$$

```
def accuracy(model, xs, ys):
  count = length(xs)
  countCorrect = 0
  for i in 1..count:
    predicted = model(xs[i])
    if predicted == ys[i]:
      countCorrect += 1
  return countCorrect / count
```

IS 99% ACCURACY GOOD?

IS 99% ACCURACY GOOD?

-> depends on problem; can be excellent, good, mediocre, terrible

10% accuracy can be good on some tasks (information retrieval)

Always compare to a base rate!

$$\text{Reduction in error} = \frac{(1 - accuracy_{\text{baseline}}) - (1 - accuracy_f)}{1 - accuracy_{\text{baseline}}}$$

- from 99.9% to 99.99% accuracy = 90% reduction in error
- from 50% to 75% accuracy = 50% reduction in error

BASELINES?

Suitable baselines for cancer prediction? For recidivism?

Speaker notes

Many forms of baseline possible, many obvious: Random, all true, all false, repeat last observation, simple heuristics, simpler model

TYPES OF MISTAKES

Two-class problem of predicting event A:

	Actually A	**Actually not A**
AI predicts A	True Positive (TP)	False Positive (FP)
AI predicts not A	False Negative (FN)	True Negative (TN)

True positives and true negatives: correct prediction

False negatives: wrong prediction, miss, Type II error

False positives: wrong prediction, false alarm, Type I error

MULTI-CLASS PROBLEMS VS TWO-CLASS PROBLEM

	Actually Grade 5 Cancer	Actually Grade 3 Cancer	Actually Benign
Model predicts Grade 5 Cancer	**10**	6	2
Model predicts Grade 3 Cancer	3	**24**	10
Model predicts Benign	5	22	**82**

MULTI-CLASS PROBLEMS VS TWO-CLASS PROBLEM

	Actually Grade 5 Cancer	Actually Grade 3 Cancer	Actually Benign
Model predicts Grade 5 Cancer	**10**	6	2
Model predicts Grade 3 Cancer	3	**24**	10
Model predicts Benign	5	22	**82**

	Act. Grade 5 Cancer	Act. not Grade 5 Cancer
Model predicts Grade 5 Cancer	10	8
Model predicts not Grade 5 Cancer	8	138

Speaker notes

Individual false positive/negative classifications can be derived by focusing on a single value in a confusion matrix. False positives/recall/etc are always considered with regard to a single specific outcome.

CONSIDER THE BASELINE PROBABILITY

Predicting unlikely events -- 1 in 2000 has cancer (stats)

Random predictor

	Cancer	No c.
Cancer pred.	3	4998
No cancer pred.	2	4997

.5 accuracy

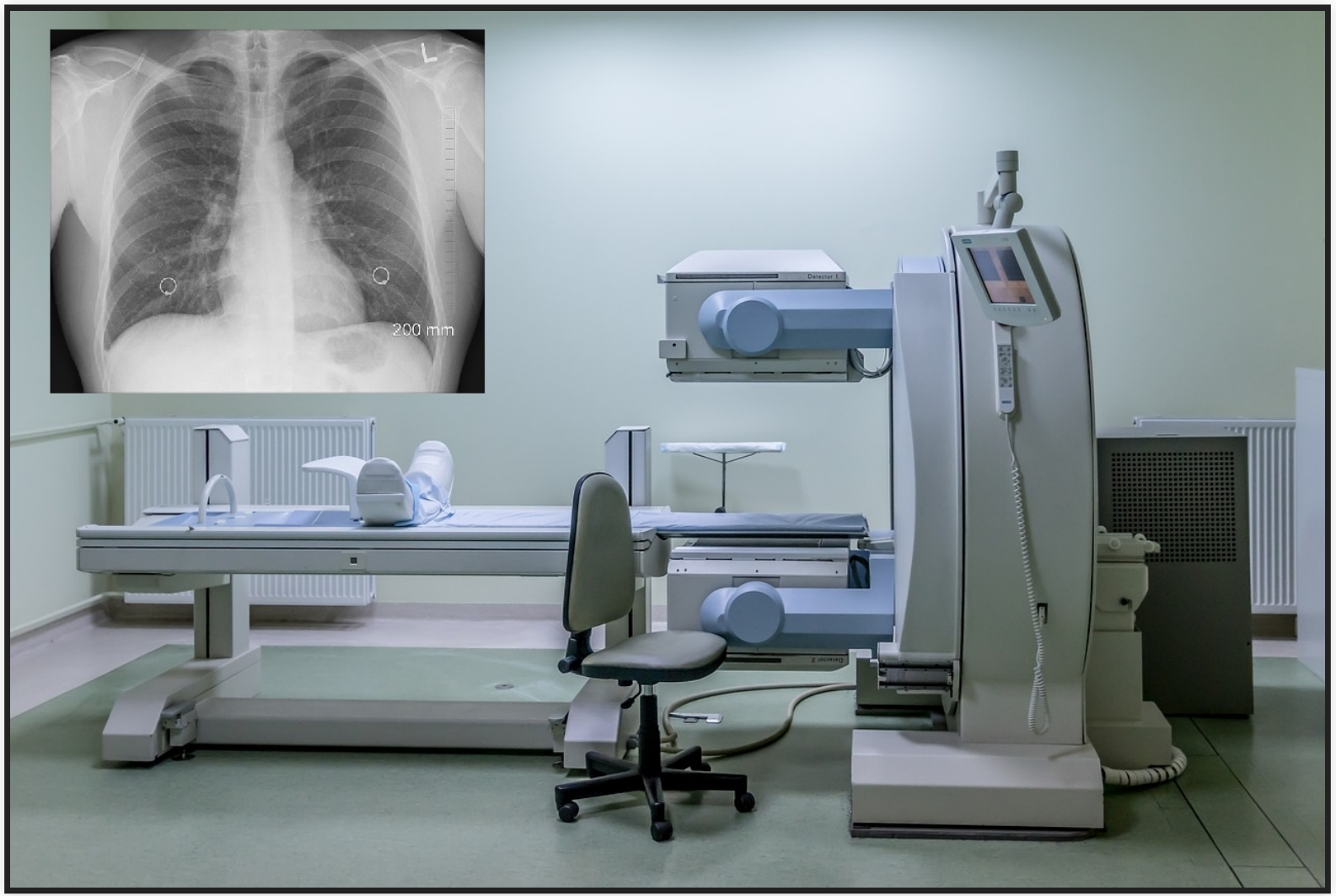
Never cancer predictor

	Cancer	No c.
Cancer pred.	0	0
No cancer pred.	5	9995

.999 accuracy

See also Bayesian statistics

TYPES OF MISTAKES IN IDENTIFYING CANCER?



MEASURES

Measuring success of correct classifications (or missing results):

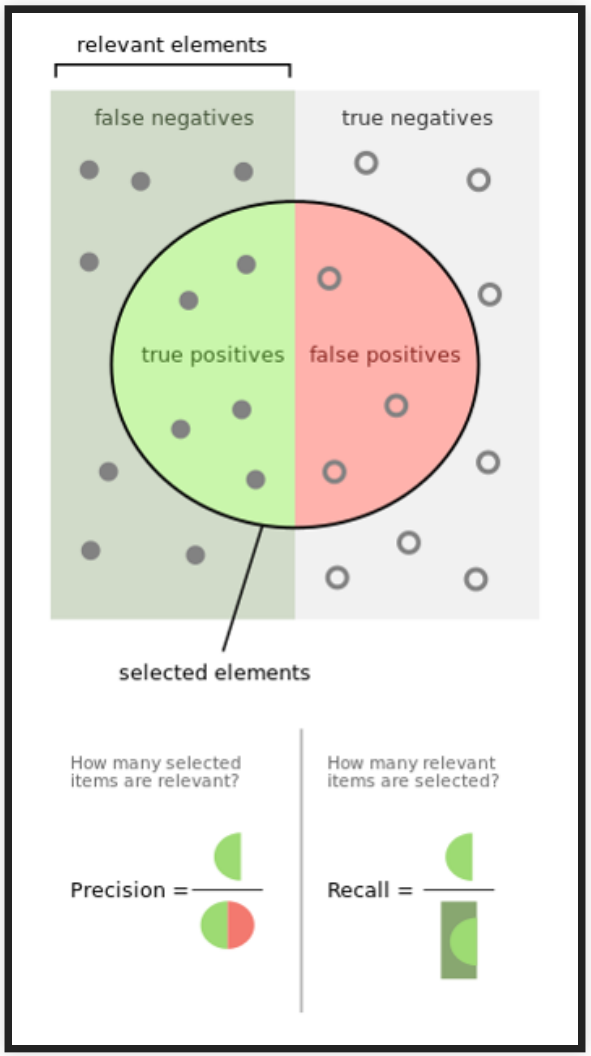
- Recall = TP/(TP+FN)
  - aka true positive rate, hit rate, sensitivity; *higher is better*
- False negative rate = FN/(TP+FN) = 1 - recall
  - aka miss rate; *lower is better*

Measuring rate of false classifications (or noise):

- Precision = TP/(TP+FP)
  - aka positive predictive value; *higher is better*
- False positive rate = FP/(FP+TN)
  - aka fall-out; *lower is better*

Combined measure (harmonic mean):

$$\text{F1 score} = 2\frac{recall * precision}{recall + precision}$$



FALSE POSITIVES AND FALSE NEGATIVES EQUALLY BAD?

Consider:

- Recognizing cancer
- Suggesting products to buy on e-commerce site
- Identifying human trafficking at the border
- Predicting high demand for ride sharing services
- Predicting recidivism chance
- Approving loan applications

No answer vs wrong answer?

EXTREME CLASSIFIERS

- Identifies every instance as negative (e.g., no cancer):
  - 0% recall (finds none of the cancer cases)
  - 100% false negative rate (misses all actual cancer cases)
  - undefined precision (no false predictions, but no predictions at all)
  - 0% false positive rate (never reports false cancer warnings)
- Identifies every instance as positive (e.g., has cancer):
  - 100% recall (finds all instances of cancer)
  - 0% false negative rate (does not miss any cancer cases)
  - low precision (also reports cancer for all noncancer cases)
  - 100% false positive rate (all noncancer cases reported as warnings)

CONSIDER THE BASELINE PROBABILITY

Predicting unlikely events -- 1 in 2000 has cancer (stats)

Random predictor

	Cancer	No c.
Cancer pred.	3	4998
No cancer pred.	2	4997

.5 accuracy, .6 recall, 0.001 precision

Never cancer predictor

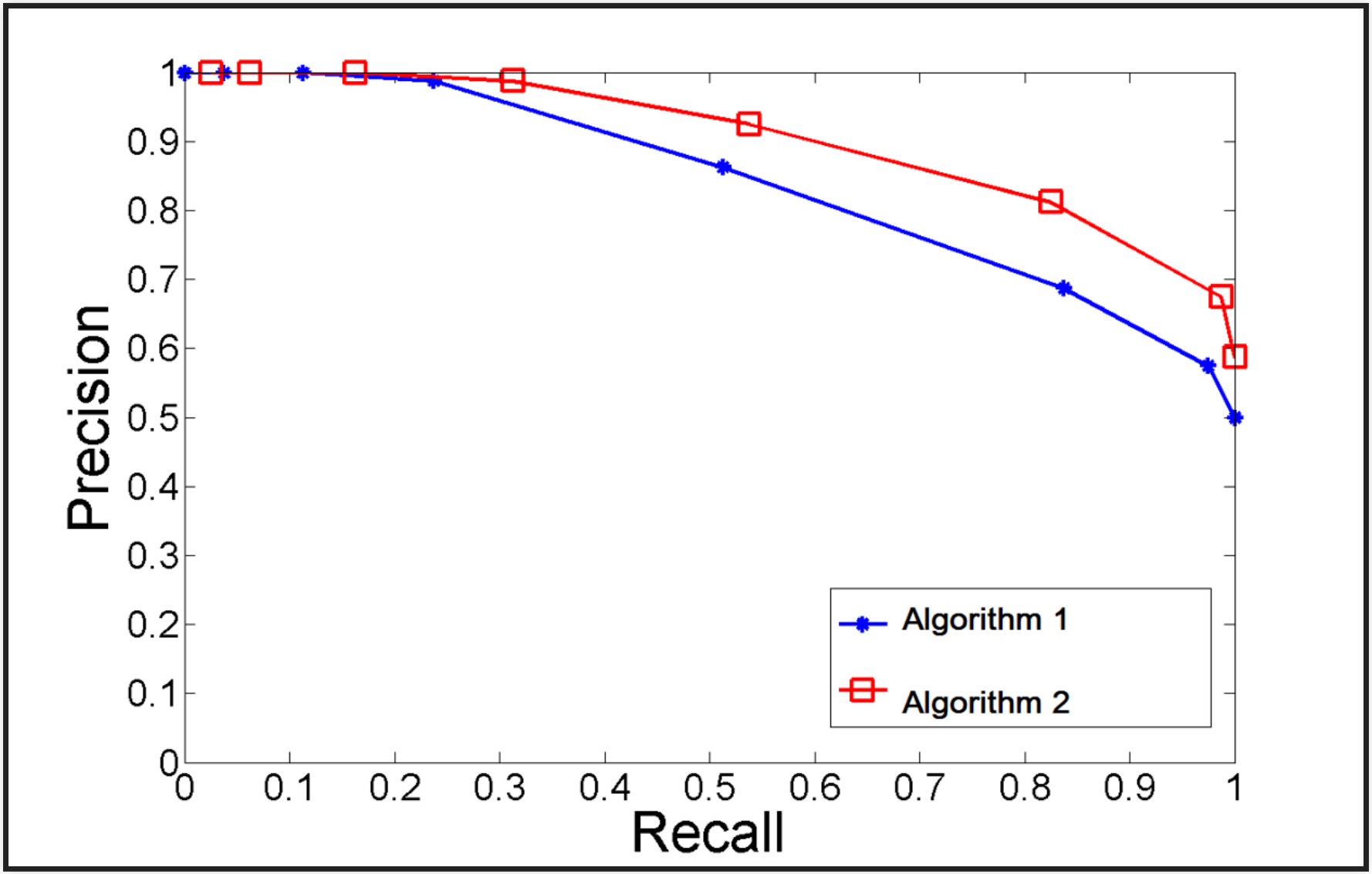
	Cancer	No c.
Cancer pred.	0	0
No cancer pred.	5	9995

.999 accuracy, 0 recall, .999 precision

See also Bayesian statistics

AREA UNDER THE CURVE

Turning numeric prediction into classification with threshold ("operating point")


Precision
Recall
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
Algorithm 1
Algorithm 2

Speaker notes

The plot shows the recall precision/tradeoff at different thresholds (the thresholds are not shown explicitly). Curves closer to the top-right corner are better considering all possible thresholds. Typically, the area under the curve is measured to have a single number for comparison.

MORE ACCURACY MEASURES FOR CLASSIFICATION PROBLEMS

- Lift
- Break even point
- F1 measure, etc
- Log loss (for class probabilities)
- Cohen's kappa, Gini coefficient (improvement over random)

MEASURING PREDICTION ACCURACY FOR REGRESSION AND RANKING TASKS

(The Data Scientists Toolbox)

CONFUSION MATRIX FOR REGRESSION TASKS?

Rooms	Crime Rate	...	Predicted Price	Actual Price
3	.01	...	230k	250k
4	.01	...	530k	498k
2	.03	...	210k	211k
2	.02	...	219k	210k

Speaker notes

Confusion Matrix does not work, need a different way of measuring accuracy that can distinguish "pretty good" from "far off" predictions

COMPARING PREDICTED AND EXPECTED OUTCOMES

Mean Absolute Percentage Error

MAPE =

$$\frac{1}{n}\sum_{t=1}^{n}\left|\frac{A_t - F_t}{A_t}\right|$$

(A_t actual outcome, F_t predicted outcome, for row t)

Compute relative prediction error per row, average over all rows

Rooms	Crime Rate	...	Predicted Price	Actual Price
3	.01	...	230k	250k
4	.01	...	530k	498k
2	.03	...	210k	211k
2	.02	...	219k	210k

MAPE =

$$\frac{1}{4}(20/250 + 32/498 + 1/211 + 9/210) =$$

$$\frac{1}{4}(0.08 + 0.064 + 0.005 + 0.043) = 0.048$$

OTHER MEASURES FOR REGRESSION MODELS

- Mean Absolute Error (MAE) = $\frac{1}{n}\sum_{t=1}^{n}\left|A_t - F_t\right|$
- Mean Squared Error (MSE) = $\frac{1}{n}\sum_{t=1}^{n}\left(A_t - F_t\right)^2$
- Root Mean Square Error (RMSE) = $\sqrt{\frac{\sum_{t=1}^{n}(A_t - F_t)^2}{n}}$
- R^2 = percentage of variance explained by model
- ...

EVALUATING RANKINGS

Ordered list of results, true results should be ranked high

Common in information retrieval (e.g., search engines) and recommendations

Mean Average Precision

MAP@K = precision in first K results

Averaged over many queries

Rank	Product	Correct?
1	Juggling clubs	true
2	Bowling pins	false
3	Juggling balls	false
4	Board games	true
5	Wine	false
6	Audiobook	true

MAP@1 = 1, MAP@2 = 0.5, MAP@3 = 0.33, ...

OTHER RANKING MEASURES

- Mean Reciprocal Rank (MRR) (average rank for first correct prediction)
- Average precision (concentration of results in highest ranked predictions)
- MAR@K (recall)
- Coverage (percentage of items ever recommended)
- Personalization (how similar predictions are for different users/queries)
- Discounted cumulative gain
- ...

Speaker notes

Good discussion of tradeoffs at https://medium.com/swlh/rank-aware-recsys-evaluation-metrics-5191bba16832

MODEL QUALITY IN NATURAL LANGUAGE PROCESSING?

Highly problem dependent:

- Classify text into positive or negative -> classification problem
- Determine truth of a statement -> classification problem
- Translation and summarization -> comparing sequences (e.g ngrams) to human results with specialized metrics, e.g. BLEU and ROUGE
- Modeling text -> how well its probabilities match actual text, e.g., likelyhood or perplexity

ALWAYS COMPARE AGAINST BASELINES!

Accuracy measures in isolation are difficult to interpret

Report baseline results, reduction in error

Example: Baselines for house price prediction? Baseline for shopping recommendations?

MEASURING GENERALIZATION

THE LEGEND OF THE FAILED TANK DETECTOR

Speaker notes

Widely shared story, authenticity not clear: AI research team tried to train image recognition to identify tanks hidden in forests, trained on images of tanks in forests and images of same or similar forests without tanks. The model could clearly separate the learned pictures, but would perform poorly on other pictures.

Turns out the pictures with tanks were taken on a sunny day whereas the other pictures were taken on a cloudy day. The model picked up on the brightness of the picture rather than the presence of a tank, which worked great for the training set, but did not generalize.

Pictures: https://pixabay.com/photos/lost-places-panzer-wreck-metal-3907364/, https://pixabay.com/photos/forest-dark-woods-trail-path-1031022/

OVERFITTING IN CANCER DETECTION?

SEPARATE TRAINING AND VALIDATION DATA

Always test for generalization on *unseen* validation data

Accuracy on training data (or similar measure) used during learning to find model parameters

```
train_xs, train_ys, valid_xs, valid_ys = split(all_xs, all_ys)
model = learn(train_xs, train_ys)

accuracy_train = accuracy(model, train_xs, train_ys)
accuracy_valid = accuracy(model, valid_xs, valid_ys)
```

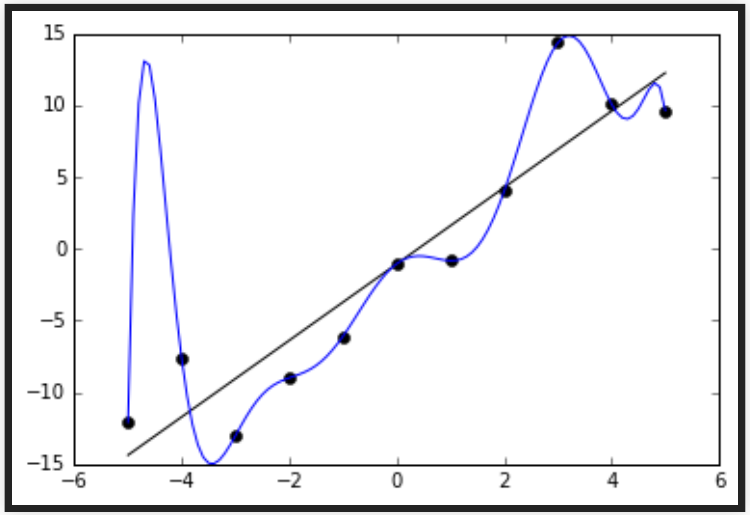
accuracy_train >> accuracy_valid = sign of overfitting

OVERFITTING/UNDERFITTING

Overfitting: Model learned exactly for the input data, but does not generalize to unseen data (e.g., exact memorization)

Underfitting: Model makes very general observations but poorly fits to data (e.g., brightness in picture)

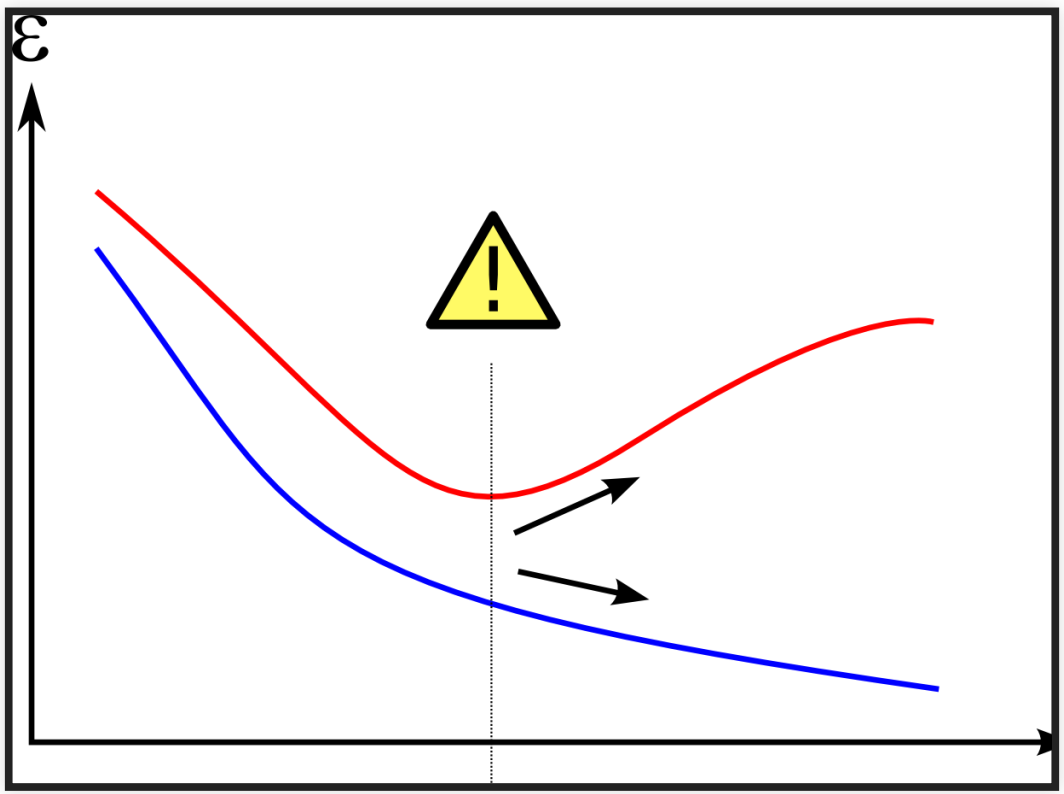
Typically adjust degrees of freedom during model learning to balance between overfitting and underfitting: can better learn the training data with more freedom (more complex models); but with too much freedom, will memorize details of the training data rather than generalizing



(CC SA 4.0 by Ghiles)

DETECTING OVERFITTING

Change hyperparameter to detect training accuracy (blue)/validation accuracy (red) at different degrees of freedom



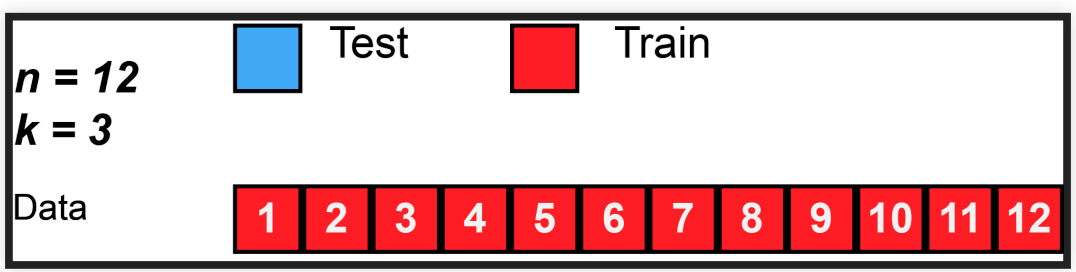
Speaker notes

Overfitting is recognizable when performance of the evaluation set decreases.

Demo: Show how trees at different depth first improve accuracy on both sets and at some point reduce validation accuracy with small improvements in training accuracy

CROSSVALIDATION

- Motivation
  - Evaluate accuracy on different training and validation splits
  - Evaluate with small amounts of validation data
- Method: Repeated partitioning of data into train and validation data, train and evaluate model on each partition, average results
- Many split strategies, including
  - leave-one-out: evaluate on each datapoint using all other data for training
  - k-fold: k equal-sized partitions, evaluate on each training on others
  - repeated random sub-sampling (Monte Carlo)



(Graphic CC MBanuelos22 BY-SA 4.0)

PRODUCTION DATA -- THE ULTIMATE UNSEEN VALIDATION DATA

more next week

SEPARATE TRAINING, VALIDATION AND TEST DATA

Often a model is "tuned" manually or automatically on a validation set (hyperparameter optimization)

In this case, we can overfit on the validation set, separate test set is needed for final evaluation

```
train_xs, train_ys, valid_xs, valid_ys, test_xs, test_ys =
            split(all_xs, all_ys)

best_model = null
best_model_accuracy = 0
for (hyperparameters in candidate_hyperparameters)
  candidate_model = learn(train_xs, train_ys, hyperparameter)
  model_accuracy = accuracy(model, valid_xs, valid_ys)
  if (model_accuracy > best_model_accuracy)
    best_model = candidate_model
    best_model_accuracy = model_accuracy

accuracy_test = accuracy(model, test_xs, test_ys)
```

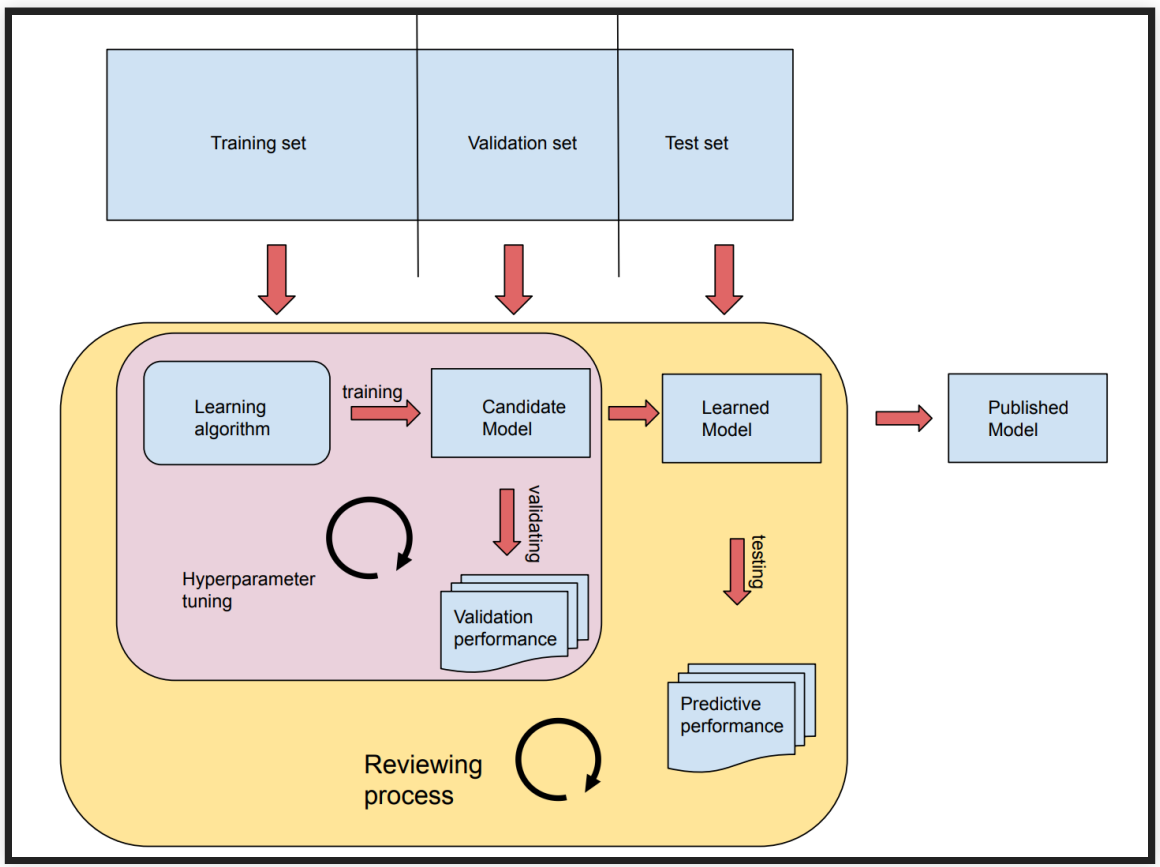
ON TERMINOLOGY

- The decisions in a model (weights, coefficients) are called *model parameter* of the model, their values are usually learned from the data
    - To a software engineer, these are *constants* in an algorithm, or configuration options
- The parameters to the learning algorithm that are not the data are called *model hyperparameters*
    - To a software engineer, these are *parameters* to the learning algorithm, similar to compiler options

```
// max_depth and min_support are hyperparameters
def learn_decision_tree(data, max_depth, min_support): Model =
  ...

// A, B, C are model parameters of model f
def f(outlook, temperature, humidity, windy) =
   if A==outlook
      return B*temperature + C*windy > 10
```

COMMON PITFALLS OF EVALUATING MODEL QUALITY

ACADEMIC ESCALATION: OVERFITTING ON BENCHMARKS

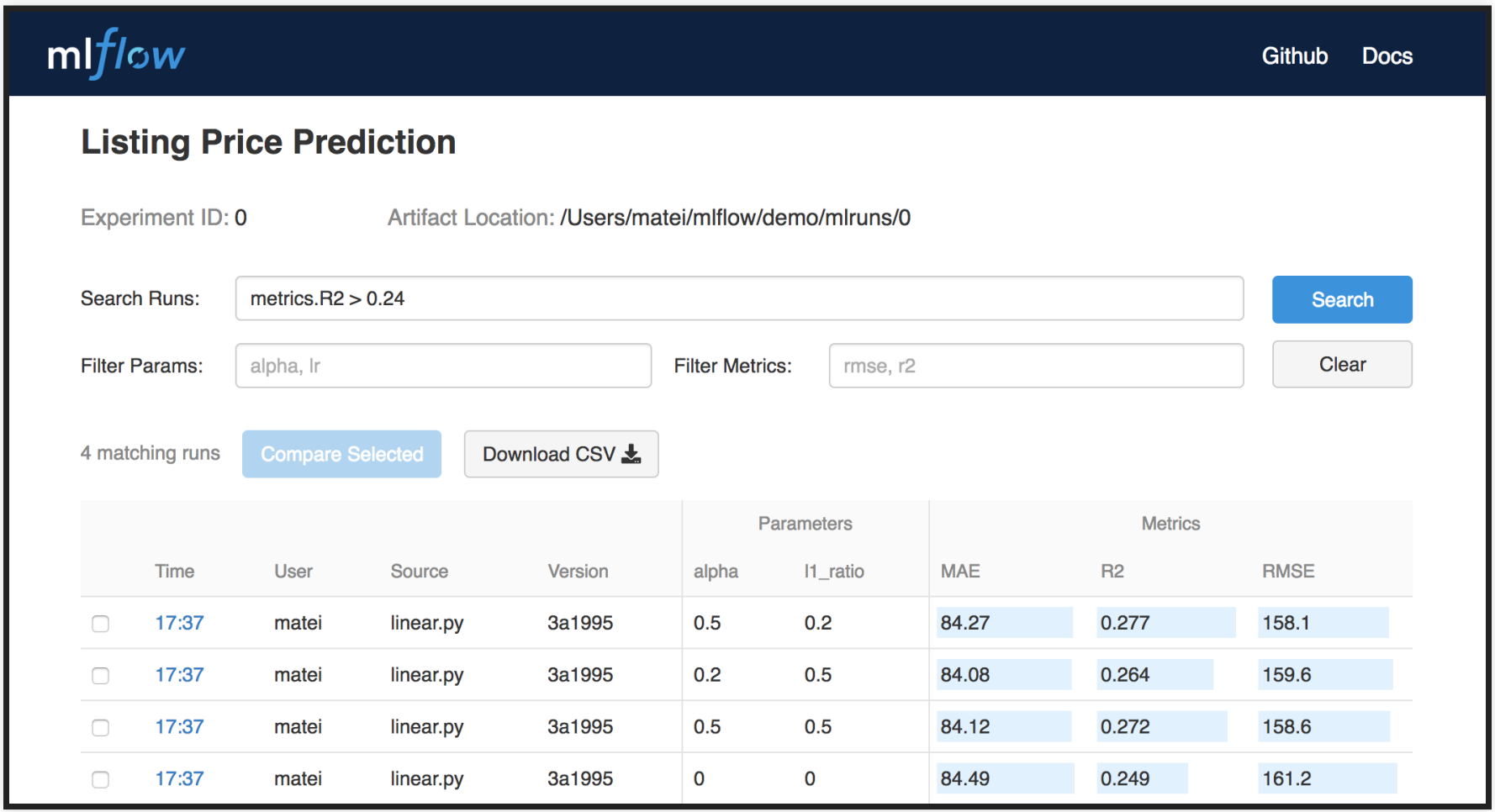


(Figure by Andrea Passerini)

Speaker notes

If many researchers publish best results on the same benchmark, collectively they perform "hyperparameter optimization" on the test set

OVERFITTING IN CONTINUOUS EXPERIMENTATION SYSTEMS



OVERFITTING IN CONTINUOUS EXPERIMENTATION SYSTEMS

- Use of test sets to compare (hyperparameter-tuned) models in dashboards
  - -> danger of overfitting
- Need fresh test data regularly
- Statistical techniques to approximate the needed amount of test data and the needed rotation

Recommended reading: Renggli, Cedric, Bojan Karlaš, Bolin Ding, Feng Liu, Kevin Schawinski, Wentao Wu, and Ce Zhang. "Continuous integration of machine learning models with ease.ml/ci: Towards a rigorous yet practical treatment." arXiv preprint arXiv:1903.00278 (2019).

EVALUATING ON TRAINING DATA

- surprisingly common in practice
- by accident, incorrect split -- or intentional using all data for training
- overlap between multiple datasets used
- tuning on validation data (e.g., crossvalidation) without separate testing data

- Results in overfitting and misleading accuracy measures

LABEL LEAKAGE

Label or close correlates included in inputs

Examples:

- Input "interview conducted" in turnover prediction encodes human judgement
- Input "has bank account" associates with predicting whether somebody will open one

Is this a problem or a good thing?

EVALUATION DATA REPRESENTATIVE?

Assumption: Independently, randomly drawn from same distribution as training data

Does this distribution correspond to distribution in practice?

USING MISLEADING QUALITY MEASURES

- using accuracy, when false positives are more harmful than false negatives
- comparing area under the curve, rather than relevant thresholds
- averaging over all populations, ignoring different results for subpopulations or different risks for certain predictions
- accuracy results on old static test data, when production data has shifted
- results on tiny validation sets
- reporting results without baseline
- ...

INDEPENDENCE OF DATA: TEMPORAL

Attempt to predict the stock price development for different companies based on twitter posts

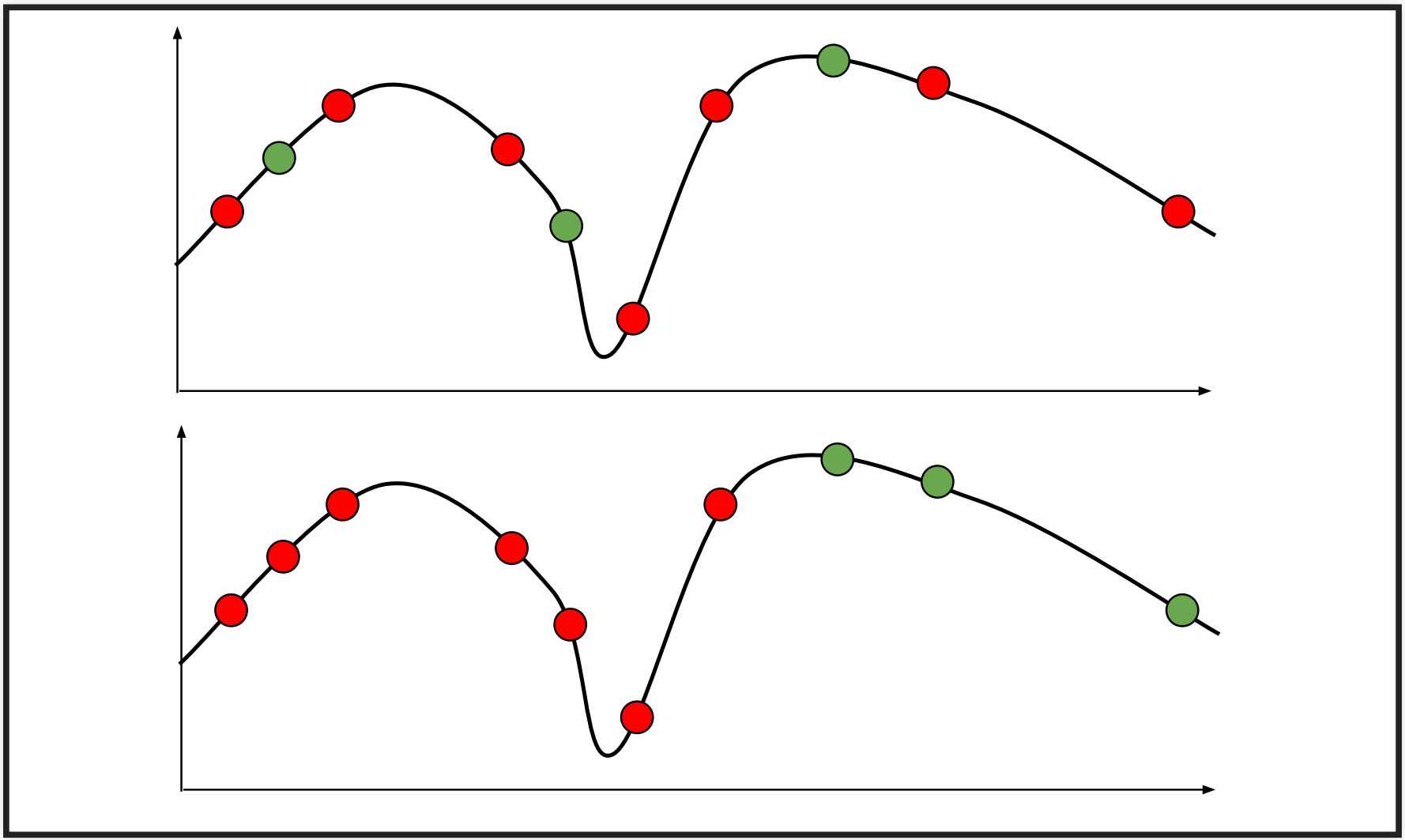
Data: stock prices of 1000 companies over 4 years and twitter mentions of those companies

Problems of random train--validation split?

Speaker notes

The model will be evaluated on past stock prices knowing the future prices of the companies in the training set. Even if we split by companies, we could observe general future trends in the economy during training

INDEPENDENCE OF DATA: TEMPORAL



Speaker notes

The curve is the real trend, red points are training data, green points are validation data. If validation data is randomly selected, it is much easier to predict, because the trends around it are known.

INDEPENDENCE OF DATA: RELATED DATAPOINTS

Kaggle competition on detecting distracted drivers

Relation of datapoints may not be in the data (e.g., driver)

https://www.fast.ai/2017/11/13/validation-sets/

Speaker notes

Many potential subtle and less subtle problems:

- Sales from same user
- Pictures taken on same day

DATA DEPENDENCE AND LABEL LEAKAGE IN CANCER CASE STUDY?

SUMMARY: COMMON PITFALLS

- Test data not representative
- Misleading accuracy metrics
- Evaluating on training or validation data
- Dependence between training and test data
- Label leakage
- Overfitting on test data through repeated evaluations

How to avoid? Ensure as part of process?

WHAT IS CORRECTNESS ANYWAY?

specifications, bugs, fit

EVALUATING A COMPONENT'S FUNCTIONAL CORRECTNESS

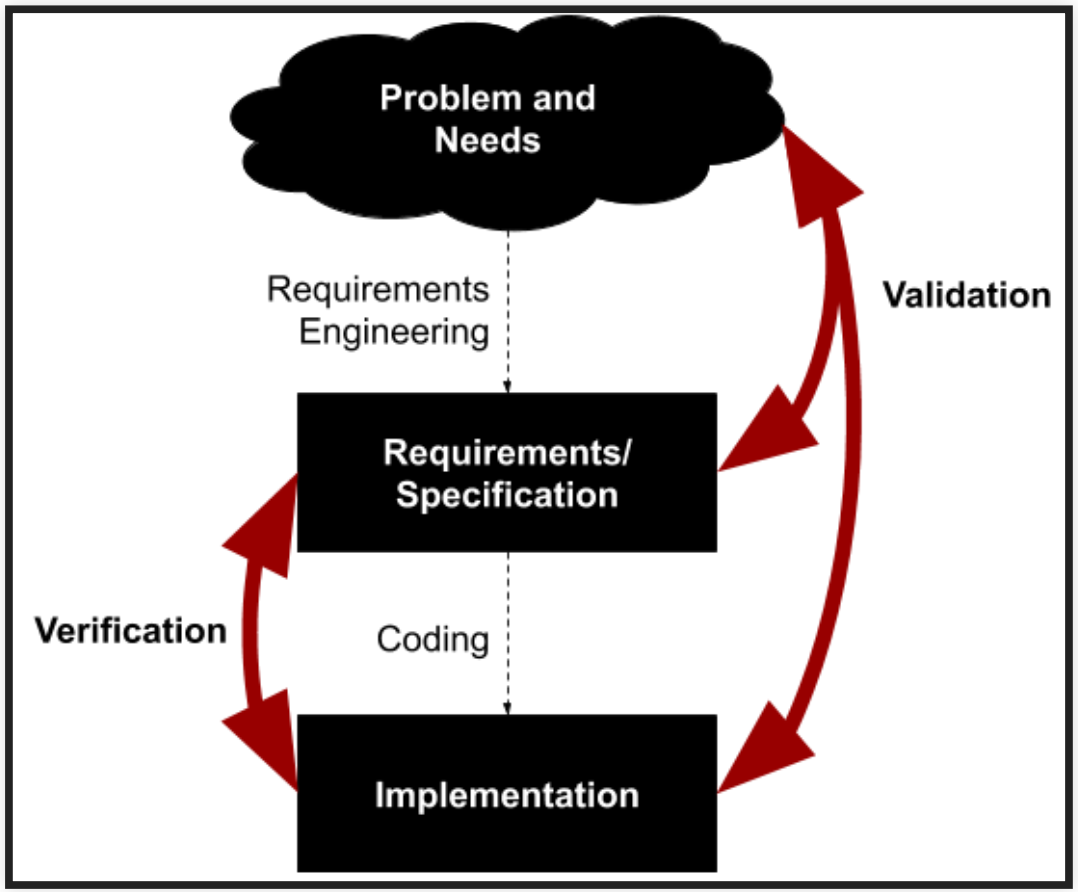
Given a specification, do outputs match inputs?

```
/**
 * compute deductions based on provided adjusted
 * gross income and expenses in customer data.
 *
 * see tax code 26 U.S. Code A.1.B, PART VI
 */
float computeDeductions(float agi, Expenses expenses);
```

Each mismatch is considered a bug, should to be fixed*.

(*=not every bug is economical to fix, may accept some known bugs)

VALIDATION VS VERIFICATION



VALIDATION PROBLEM: CORRECT BUT USELESS?

- Correctly implemented to specification, but specifications are wrong
- Building the wrong system, not what user needs
- Ignoring assumptions about how the system is used

Example: Compute deductions with last year's tax code

Other examples?

WRONG SPECIFICATIONS: ARIANE 5

Software was working as specified, within the specified parameters. Inputs exceeded specified parameters.

STRICT CORRECTNESS ASSUMPTION

- Specification determines which outputs are correct/wrong
- Not "pretty good", "95% accurate", or "correct for 98% of all users"
- A single wrong result indicates a bug in the system

```
/**
 * compute deductions based on provided adjusted
 * gross income and expenses in customer data.
 *
 * see tax code 26 U.S. Code A.1.B, PART VI
 */
float computeDeductions(float agi, Expenses expenses);
```

Speaker notes

A single wrong tax prediction would be a bug. No tolerance of occasional wrong predictions, approximations, nondeterminism.

IDEALLY FORMAL SPECIFICATIONS

Formal verification possible, proving that implementation matches specification.

```
/*@ public normal_behavior
  @ ensures (\forall int j; j >= 0 && j < a.length;
  @                          \result = a[j]);
  @*/
public static /*@ pure @*/ int max(int[] a);
```

In practice, typically informal, textual and "incomplete" specifications, but still enabling analyzing inputs-output correspondence

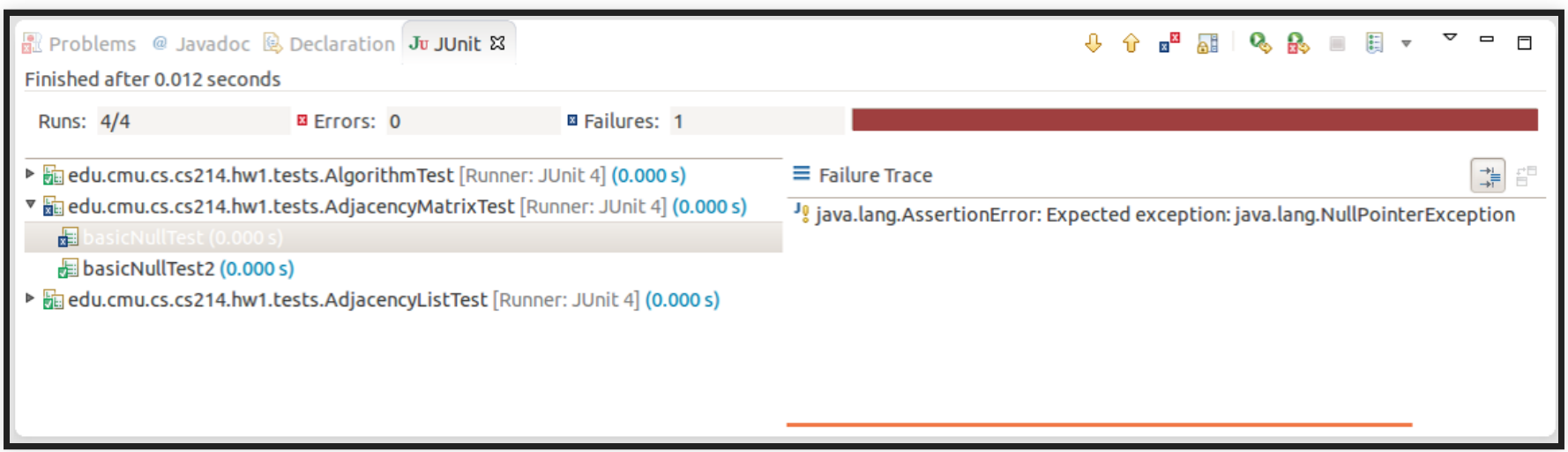
COMMON PRACTICE: TESTING

- Verification technique comparing program behavior to specification
- Provide select inputs, expect correct outputs (according to specification)
- Failing test case reveals bug
- No guarantee to find all bugs

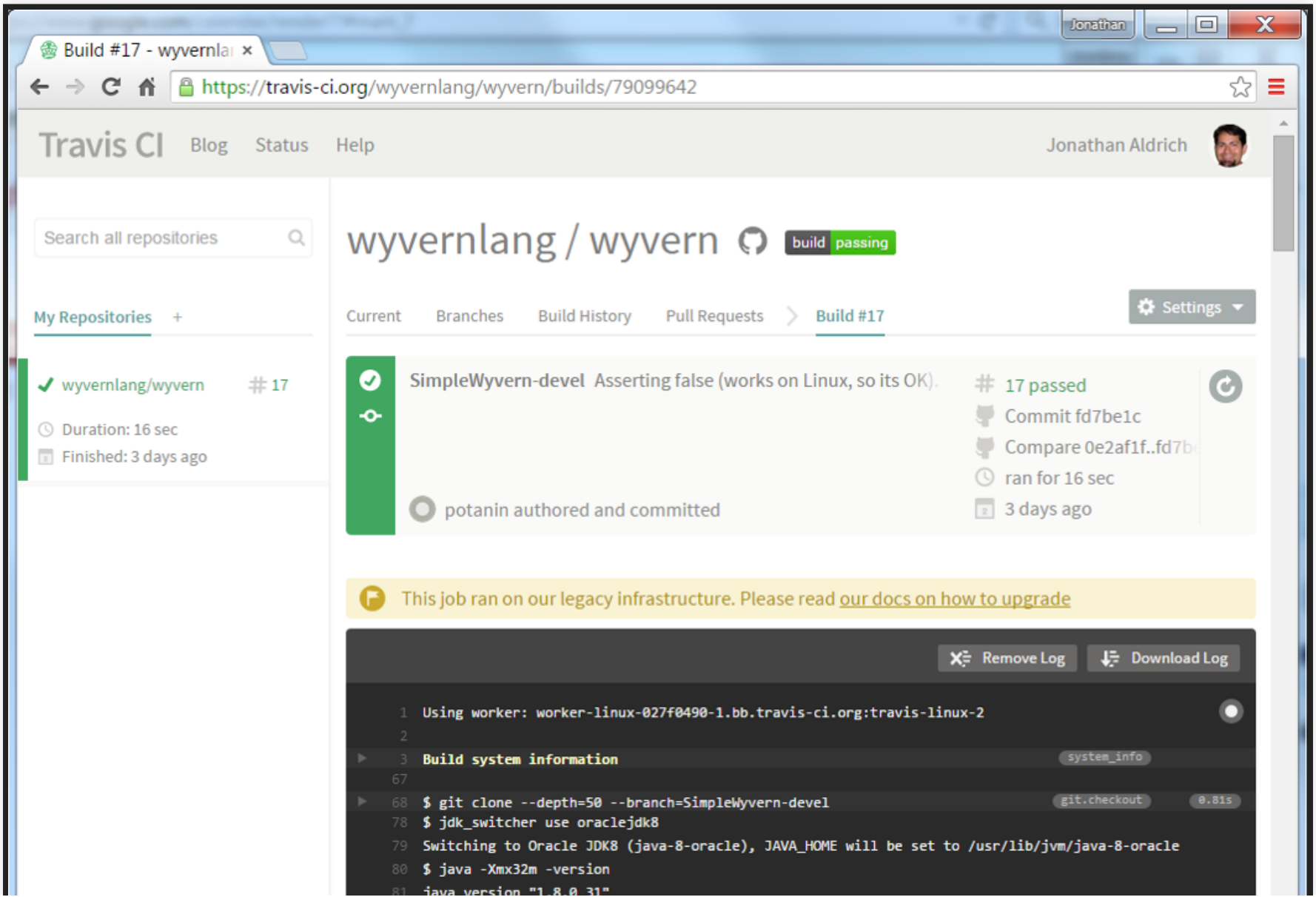
```
// returns the sum of two arguments
int add(int a, int b) { ... }

@Test
void testAddition_2_2() {
  assertEquals(4, add(2, 2));
}
@Test
void testAddition_1_2() {
  assertEquals(3, add(1, 2));
}
```

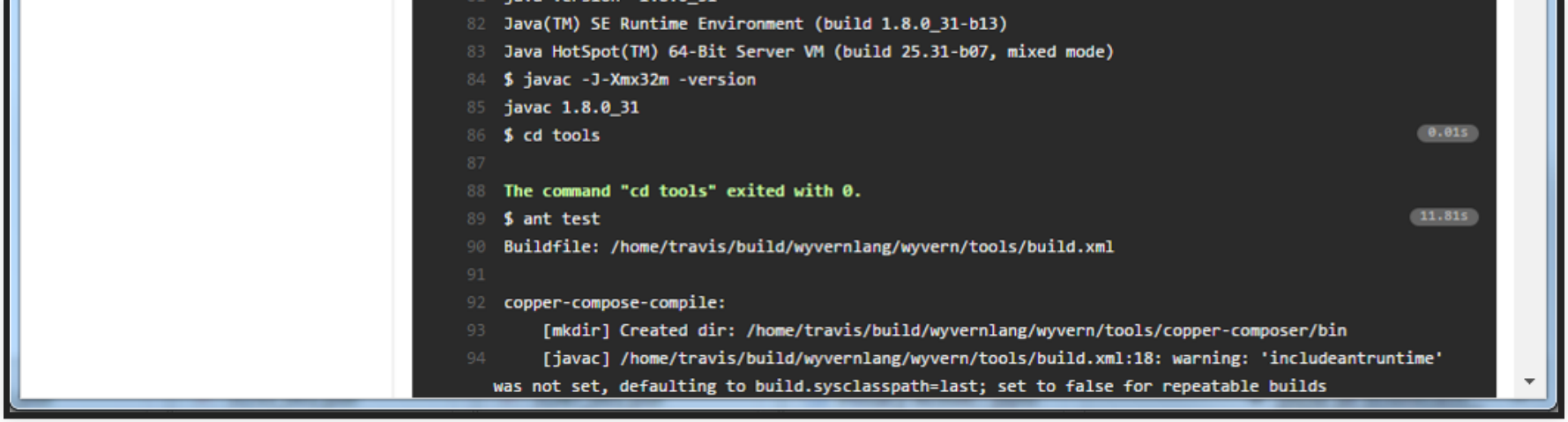
TEST AUTOMATION

```
@Test
public void testSanityTest(){
    //setup
    Graph g1 = new AdjacencyListGraph(10);
    Vertex s1 = new Vertex("A");
    Vertex s2 = new Vertex("B");
    //check expected behavior
    assertEquals(true, g1.addVertex(s1));
    assertEquals(true, g1.addVertex(s2));
    assertEquals(true, g1.addEdge(s1, s2));
    assertEquals(s2, g1.getNeighbors(s1)[0]);
}
```



CONTINUOUS INTEGRATION




82 Java(TM) SE Runtime Environment (build 1.8.0_31-b13)
83 Java HotSpot(TM) 64-Bit Server VM (build 25.31-b07, mixed mode)
84 $ javac -J-Xmx32m -version
85 javac 1.8.0_31
86 $ cd tools
0.01s
87
88 The command "cd tools" exited with 0.
89 $ ant test
11.81s
90 Buildfile: /home/travis/build/wyvernlang/wyvern/tools/build.xml
91
92 copper-compose-compile:
93 [mkdir] Created dir: /home/travis/build/wyvernlang/wyvern/tools/copper-composer/bin
94 [javac] /home/travis/build/wyvernlang/wyvern/tools/build.xml:18: warning: 'includeantruntime'
was not set, defaulting to build.sysclasspath=last; set to false for repeatable builds

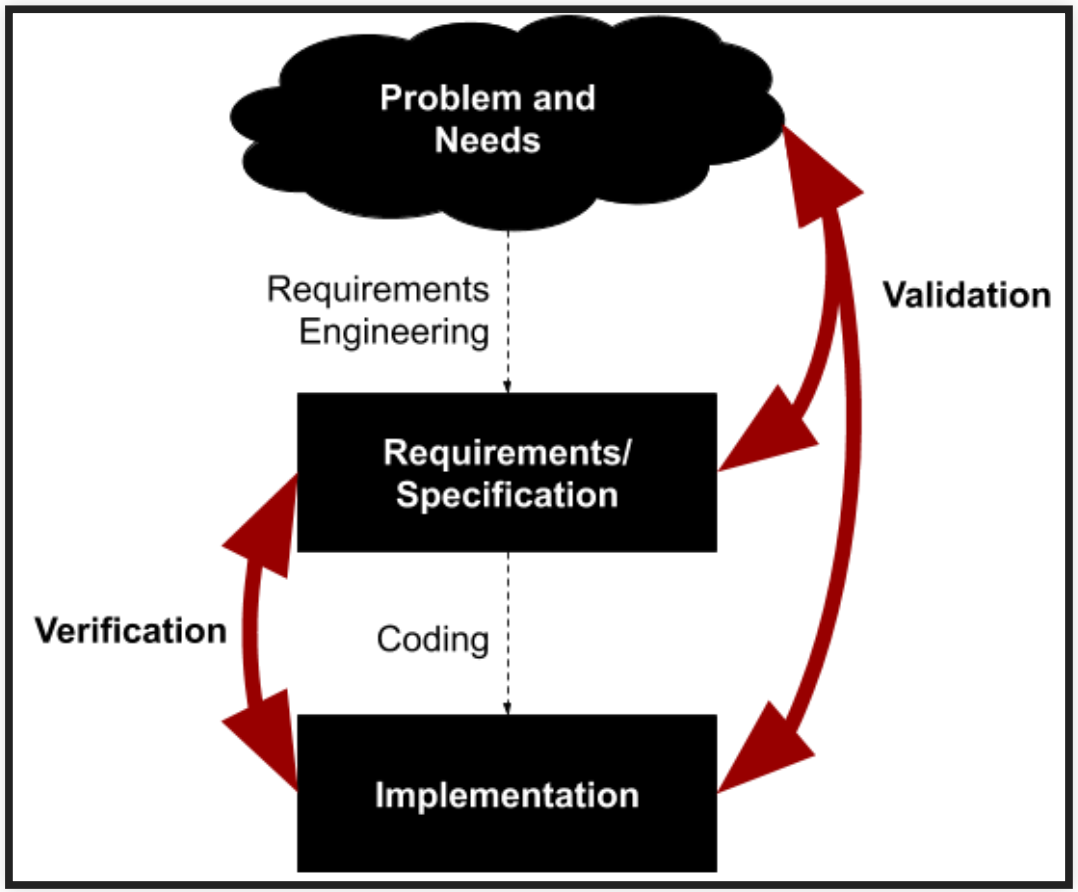
SOFTWARE TESTING

"Testing shows the presence, not the absence of bugs" -- Edsger W. Dijkstra 1969

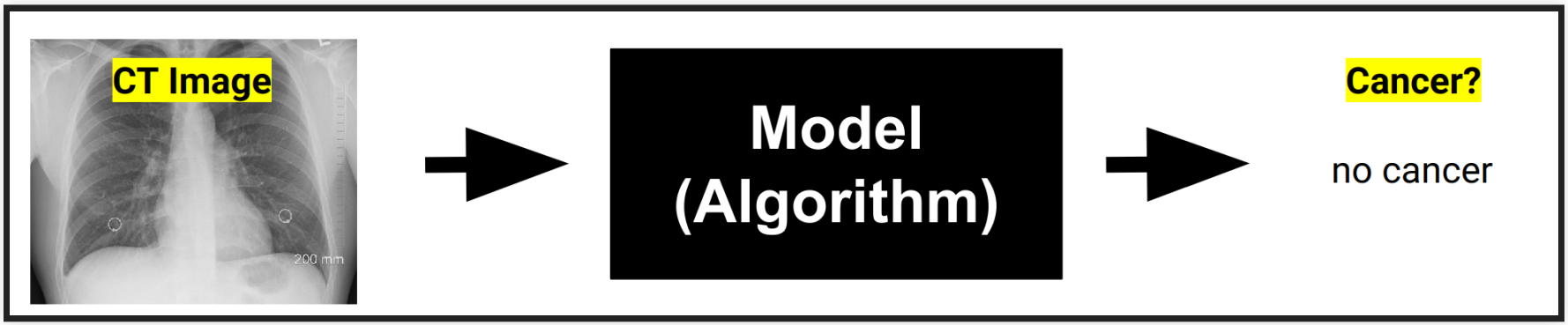
Software testing can be applied to many qualities:

- Functional errors
- Performance errors
- Buffer overflows
- Usability errors
- Robustness errors
- Hardware errors
- API usage errors

VALIDATION VS VERIFICATION



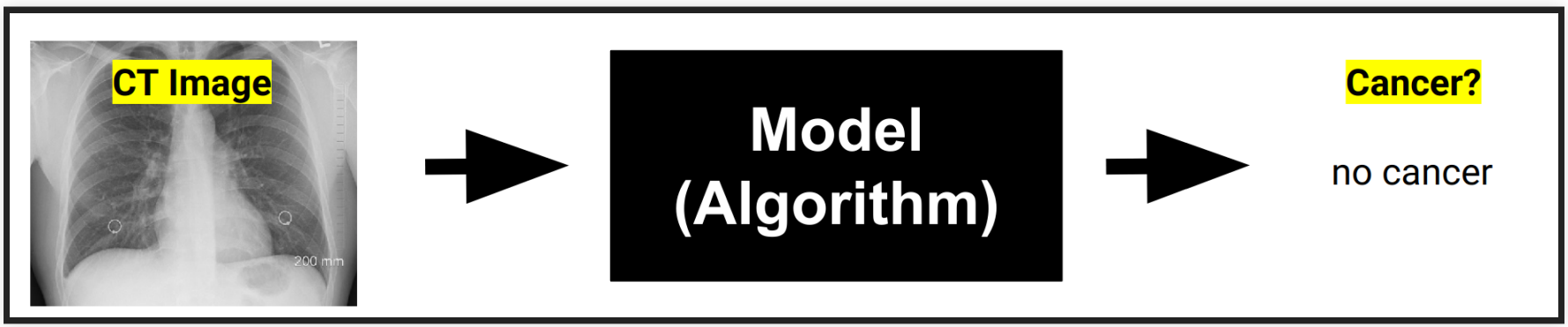
HOW TO EVALUATE PREDICTION TASKS?




```
/**
  ????
*/
boolean hasCancer(Image scan);
```

NO SPECIFICATION!

We use ML precisely because we do not have a specification (too complex, rules unknown)



No specification that could tell us for any input whether the output is correct

Intuitions, ideas, goals, "implicit specifications", but nothing we can write down!

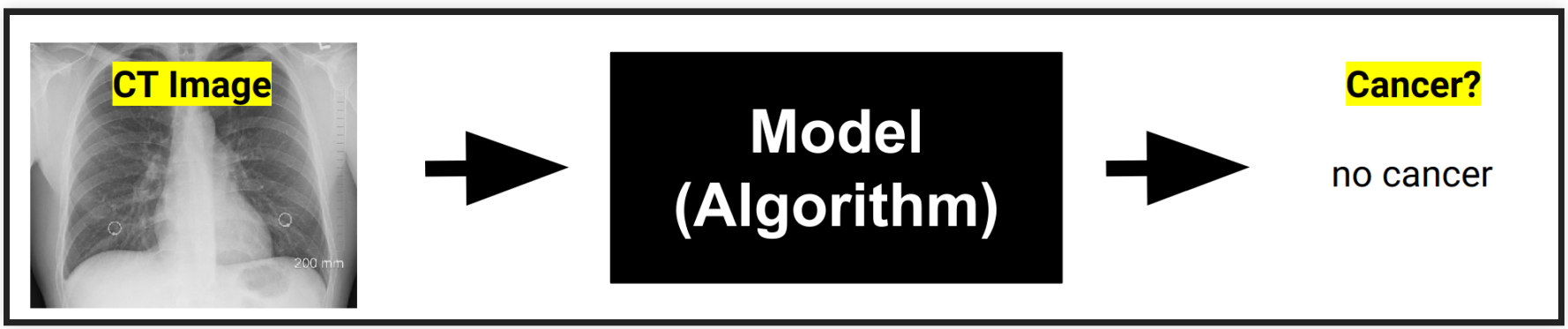
We are usually okay with some wrong predictions

TESTING A MACHINE LEARNING MODEL?

```
// detects cancer in an image
boolean hasCancer(Image scan);

@Test
void testPatient1() {
  assertEquals(loadImage("patient1.jpg"), false);
}
@Test
void testPatient2() {
  assertEquals(loadImage("patient2.jpg"), false);
}
```

WEAK CORRECTNESS ASSUMPTIONS

- Often no reliable ground truth (e.g. human judgment and disagreement)
- Examples, but no rules
- Accepting that mistakes will happen, hopefully not to frequently; "95% accuracy" may be pretty good
- More confident for data similar to training data



All models are approximations. Assumptions, whether implied or clearly stated, are never exactly true. ***All models are wrong, but some models are useful****. So the question you need to ask is not "Is the model true?" (it never is) but "Is the model good enough for this particular application?"*
-- George Box

See also https://en.wikipedia.org/wiki/All_models_are_wrong

NON-ML EXAMPLE: NEWTON'S LAWS OF MOTION

2nd law: "the rate of change of momentum of a body over time is directly proportional to the force applied, and occurs in the same direction as the applied force" $\boldsymbol{F} = \frac{d\boldsymbol{p}}{dt}$

"Newton's laws were verified by experiment and observation for over 200 years, and they are excellent approximations at the scales and speeds of everyday life."

Do not generalize for very small scales, very high speeds, or in very strong gravitational fields. Do not explain semiconductor, GPS errors, superconductivity, ... Those require general relativity and quantum field theory.

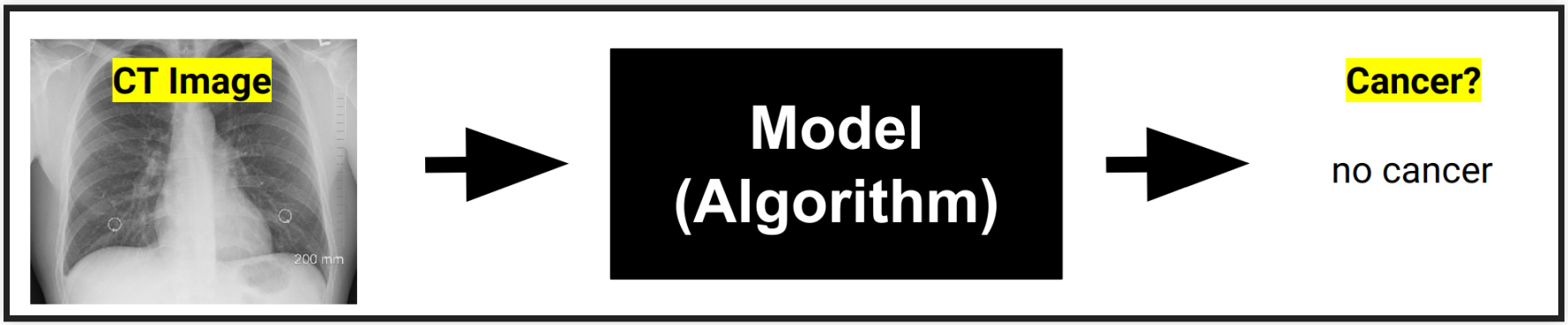
Further readings: https://en.wikipedia.org/wiki/Newton%27s_laws_of_motion

"Since all models are wrong the scientist cannot obtain a "correct" one by excessive elaboration. On the contrary following William of Occam he should seek an economical description of natural phenomena." -- George Box, 1976

"Since all models are wrong the scientist must be alert to what is importantly wrong. It is inappropriate to be concerned about mice when there are tigers abroad." -- George Box, 1976

See also https://en.wikipedia.org/wiki/All_models_are_wrong

FIND BETTER MODELS?



We are looking for models that better **fit** the problem

No specification of "correctness"

A single wrong prediction is (usually) not problem, many wrong predictions might be.

TYPICAL ACCURACY EVALUATION

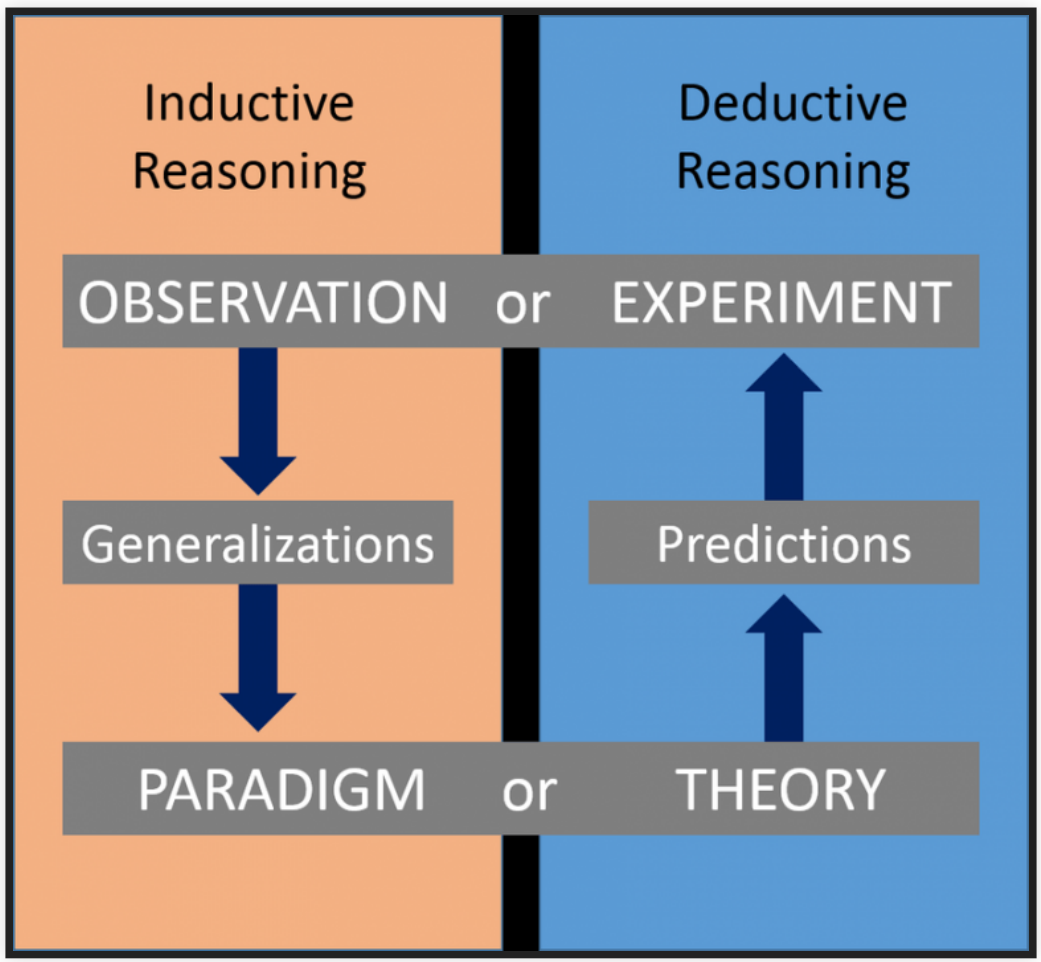
Given example data, evaluate how well the model fits that data

```
def accuracy(model, xs, ys):
  count = length(xs)
  countCorrect = 0
  for i in 1..count:
    predicted = model(xs[i])
    if predicted == ys[i]:
      countCorrect += 1
  return countCorrect / count
```

Like testing, only sample inputs.

Unlike traditional software do not expect "correctness"

EXCURSION: DEDUCTIVE VS INDUCTIVE REASONING



(Daniel Miessler, CC SA 2.0)

DEDUCTIVE REASONING

- Combining logical statements following agreed upon rules to form new statements
- Proving theorems from axioms
- From general to the particular
- *mathy reasoning, eg. proof that π is irrational*

- Formal methods, classic rule-based AI systems, expert systems

INDUCTIVE REASONING

- Constructing axioms from observations
- Strong evidence suggests a rule
- From particular to the general
- *sciency reasoning, eg. finding laws of nature*

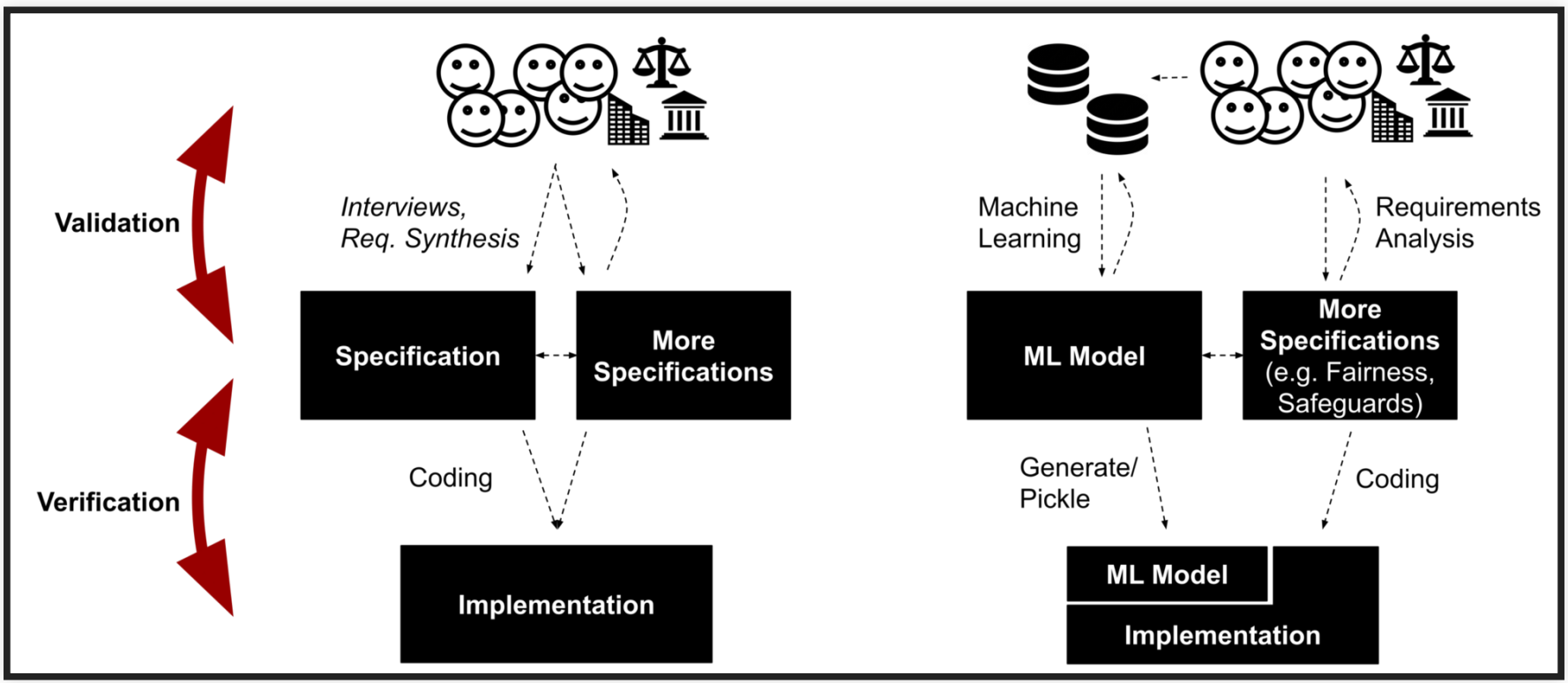
- Most modern machine learning systems, statistical learning

MACHINE LEARNING MODELS FIT, OR NOT

- A model is learned from given data in given procedure
  - The learning process is typically not a correctness concern
  - The model itself is generated, typically no implementation issues
- Is the data representative? Sufficient? High quality?
- Does the model "learn" meaningful concepts?

- **Is the model useful for a problem?** Does it *fit*?
- Do model predictions *usually* fit the users' expectations?
- Is the model *consistent* with other requirements? (e.g., fairness, robustness)

MY PET THEORY: MACHINE LEARNING IS REQUIREMENTS ENGINEERING



Long version: https://medium.com/@ckaestne/machine-learning-is-requirements-engineering-8957aee55ef4

ON TERMINOLOGY

- Avoid term *model bug*, no agreement, no standardization
- *Performance* or *accuracy* or *fit* are better fitting terms than *correct* for model quality
- Careful with the term *testing* for measuring *prediction accuracy*, be aware of "correctness" connotations
- *Verification/validation* analogy may help frame thinking, but will likely be confusing to most without longer explanation

EXCURSION: PERFORMANCE TESTING?

- Performance tests are not precise (measurement noise)
  - Averaging over repeated executions *of the same test*
  - Commonly using diverse benchmarks, i.e., *multiple inputs*
  - Need to control environment (hardware)
- No precise specification
  - Regression tests
  - Benchmarking as open-ended comparison, or tracking results over time

```
@Test(timeout=100)
public void testCompute() {
   expensiveComputation(...);
}
```

Is this a better analogy for model quality?

EXCURSION: PERFORMANCE TESTING IS POOR ANALOGY

- Performance specifications tend to be weak
  - no precise expectations
  - partial specifications
- Performance specifications are probabilistic, because algorithm behavior is nondeterministic
  - "90% of executions shall terminate in less than 1s"
  - but: *repetitions of same program, capturing nondeterminism in program*
- ML models are usually deterministic
  - accuracy measured across multiple inputs, not repeated evaluation of same input

CURATING VALIDATION DATA

(Learning from Software Testing)

BREAKOUT DISCUSSION

Write a few tests for the following program:

```
def nextDate(year: Int, month: Int, day: Int) = ...
```

for example

```
assert nextDate(2021, 2, 8) == (2021, 2, 9);
```

Discuss how you select tests. Discuss how many tests you need to feel confident.

DEFINING SOFTWARE TESTING

- Program *p* with specification *s*
- Test consists of
  - Controlled environment
  - Test call, test inputs
  - Expected behavior/output (oracle)

```
assertEquals(4, add(2, 2));
assertEquals(??, factorPrime(15485863));
```

Testing is complete but unsound: Cannot guarantee the absence of bugs

HOW TO CREATE TEST CASES?

```
def nextDate(year: Int, month: Int, day: Int) = ...
```

Speaker notes

Can focus on specification (and concepts in the domain, such as leap days and month lengths) or can focus on implementation

Will not randomly sample from distribution of all days

SOFTWARE TEST CASE DESIGN

- Opportunistic/exploratory testing: Add some unit tests, without much planning
- Black-box testing: Derive test cases from specifications
    - Boundary value analysis
    - Equivalence classes
    - Combinatorial testing
    - Random testing
- White-box testing: Derive test cases to cover implementation paths
    - Line coverage, branch coverage
    - Control-flow, data-flow testing, MCDC, ...

- Test execution usually automated, but can be manual too
- Automated generation from specifications or code possible

EXAMPLE: BOUNDARY VALUE TESTING

- Analyze the specification, not the implementation!
- Key Insight: Errors often occur at the boundaries of a variable value
- For each variable select (1) minimum, (2) min+1, (3) medium, (4) max-1, and (5) maximum; possibly also invalid values min-1, max+1

- Example: nextDate(2015, 6, 13) = (2015, 6, 14)
    - **Boundaries?**

EXAMPLE: EQUIVALENCE CLASSES

- Idea: Typically many values behave similarly, but some groups of values are different
- Equivalence classes derived from specifications (e.g., cases, input ranges, error conditions, fault models)
- Example nextDate(2015, 6, 13)
    - leap years, month with 28/30/31 days, days 1-28, 29, 30, 31
- Pick 1 value from each group, combine groups from all variables

EXERCISE

```
/**
 * Compute the price of a bus ride:
 *   * Children under 2 ride for free, children under 18 and
 *     senior citizen over 65 pay half, all others pay the
 *     full fare of $3.
 *   * On weekdays, between 7am and 9am and between 4pm and
*      7pm a peak surcharge of $1.5 is added.
 *   * Short trips under 5min during off-peak time are free.
 */
def busTicketPrice(age: Int,
                   datetime: LocalDateTime,
                   rideTime: Int)
```

suggest test cases based on boundary value analysis and equivalence class testing

SELECTING VALIDATION DATA FOR MODEL QUALITY?

VALIDATION DATA REPRESENTATIVE?

- Validation data should reflect usage data
- Be aware of data drift (face recognition during pandemic, new patterns in credit card fraud detection)
- "*Out of distribution*" predictions often low quality (it may even be worth to detect out of distribution data in production, more later)

(note, similar to requirements validation: did we hear all/representative stakeholders)

NOT ALL INPUTS ARE EQUAL

"Call mom" "What's the weather tomorrow?" "Add asafetida to my shopping list"

NOT ALL INPUTS ARE EQUAL

There Is a Racial Divide in Speech-Recognition Systems, Researchers Say: Technology from Amazon, Apple, Google, IBM and Microsoft misidentified 35 percent of words from people who were black. White people fared much better. -- NYTimes March 2020

Tweet

NOT ALL INPUTS ARE EQUAL

some random mistakes vs rare but biased mistakes?

- A system to detect when somebody is at the door that never works for people under 5ft (1.52m)
- A spam filter that deletes alerts from banks

Consider separate evaluations for important subpopulations; monitor mistakes in production

IDENTIFY IMPORTANT INPUTS

Curate Validation Data for Specific Problems and Subpopulations:

- *Regression testing:* Validation dataset for important inputs ("call mom") -- expect very high accuracy -- closest equivalent to **unit tests**
- *Uniformness/fairness testing:* Separate validation dataset for different subpopulations (e.g., accents) -- expect comparable accuracy
- *Setting goals:* Validation datasets for challenging cases or stretch goals -- accept lower accuracy

Derive from requirements, experts, user feedback, expected problems etc. Think *blackbox testing*.

IMPORTANT INPUT GROUPS FOR CANCER DETECTION?

INPUT PARTITIONING

- Guide testing by identifying groups and analyzing accuracy of subgroups
  - Often for fairness: gender, country, age groups, ...
  - Possibly based on business requirements or cost of mistakes
- Slice test data by population criteria, also evaluate interactions
- Identifies problems and plan mitigations, e.g., enhance with more data for subgroup or reduce confidence

Example: Testing sentiment classifier on IMDB reviews: Similar accuracy across genres? Across movie ages? Across review length?

Good reading: Barash, Guy, Eitan Farchi, Ilan Jayaraman, Orna Raz, Rachel Tzoref-Brill, and Marcel Zalmanovici. "Bridging the gap between ML solutions and their business requirements using feature interactions." In Proc. Symposium on the Foundations of Software Engineering, pp. 1048-1058. 2019.

INPUT PARTITIONING EXAMPLE

DECADE	SUPPORT	ACC
1910s	38	78.94
1930s	338	87.87
1990s	3007	90.95
2000s	6192	91.40

Input divided by movie age. Notice low accuracy, but also low support (i.e., little validation data), for old movies.

MAIN_GENRE	RAT_CAT	LEN_CAT	SUPPORT	ACC
Mystery	OK	long	11	72.72
Fantasy	OK	short	36	77.77
Crime	OK	long	100	81.00
Comedy	GOOD	long	55	96.36

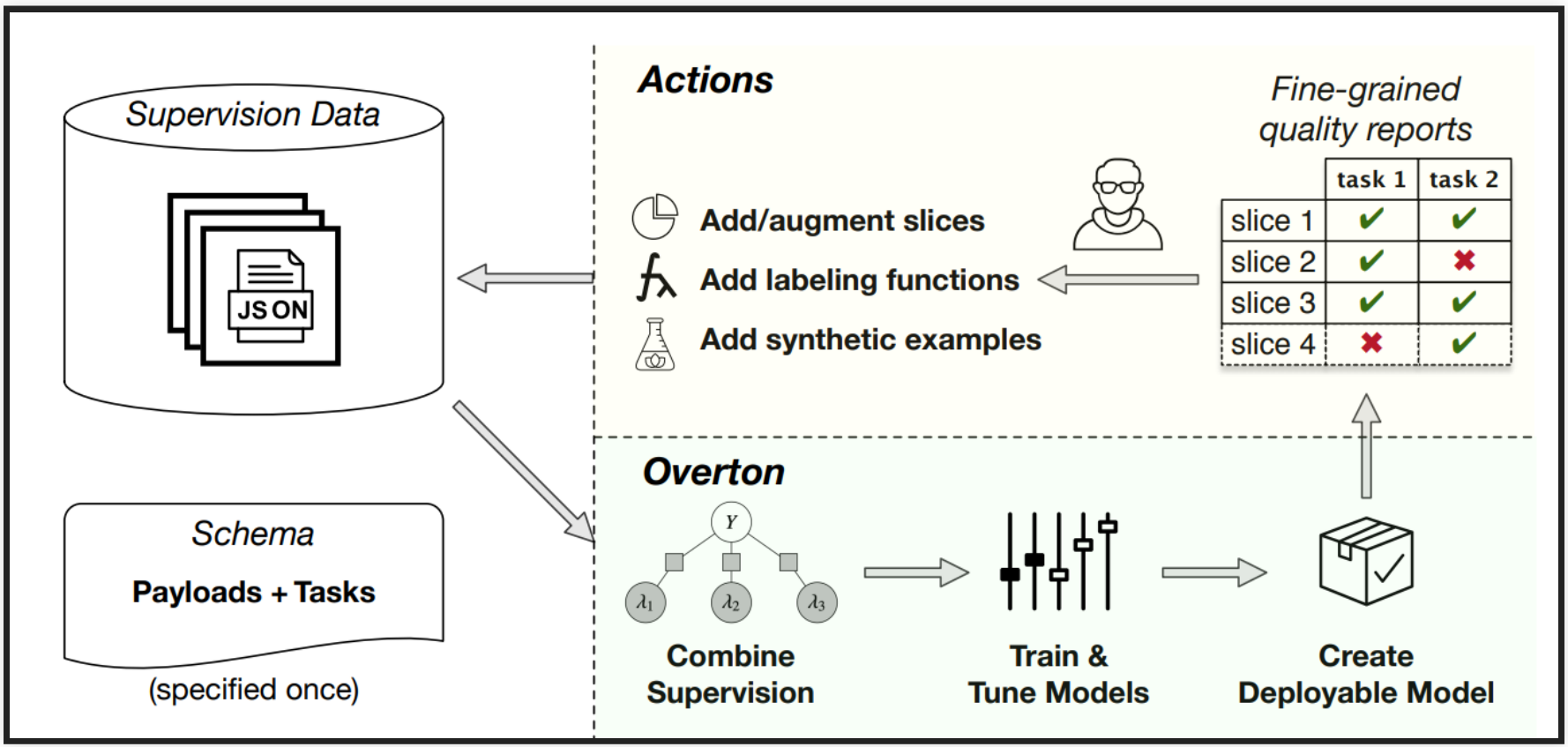
Input divided by genre, rating, and length. Accuracy differs, but also amount of test data used ("support") differs, highlighting low confidence areas.

Source: Barash, Guy, Eitan Farchi, Ilan Jayaraman, Orna Raz, Rachel Tzoref-Brill, and Marcel Zalmanovici. "Bridging the gap between ML solutions and their business requirements using feature interactions." In Proc. Symposium on the Foundations of Software Engineering, pp. 1048-1058. 2019.

INPUT PARTITIONING DISCUSSION

How to slice evaluation data for cancer detection?

EXAMPLE: MODEL IMPROVEMENT AT APPLE (OVERTON)



Ré, Christopher, Feng Niu, Pallavi Gudipati, and Charles Srisuwananukorn. "Overton: A Data System for Monitoring and Improving Machine-Learned Products." arXiv preprint arXiv:1909.05372 (2019).

EXAMPLE: MODEL IMPROVEMENT AT APPLE (OVERTON)

- Focus engineers on creating training and validation data, not on model search (AutoML)
- Flexible infrastructure to slice telemetry data to identify underperforming subpopulations -> focus on creating better training data (better, more labels, in semi-supervised learning setting)

Ré, Christopher, Feng Niu, Pallavi Gudipati, and Charles Srisuwananukorn. "Overton: A Data System for Monitoring and Improving Machine-Learned Products." arXiv preprint arXiv:1909.05372 (2019).

TESTING CAPABILITIES ("STRESS TESTING")

Even without specifications, are there "concepts" or "capabilities" the model should learn?

Example capabilities of sentiment analysis:

- Handle *negation*
- Robustness to *typos*
- Ignore synonyms and abbreviations
- Person and location names are irrelevant
- Ignore gender
- ...

For each capability create specific test set (multiple examples) -- manually or following patterns

Ribeiro, Marco Tulio, Tongshuang Wu, Carlos Guestrin, and Sameer Singh. "Beyond Accuracy: Behavioral Testing of NLP Models with CheckList." In Proceedings ACL, p. 4902–4912. (2020).

TESTING CAPABILITIES ("STRESS TESTING")

Robust.	***INV:*** Add randomly generated URLs and handles to tweets	9.6	13.4	24.8	11.4	7.4	@JetBlue that selfie was extreme. @pi9QDK INV @united stuck because staff took a break? Not happy 1K.... https://t.co/PWK1jb INV
	INV: Swap one character with its neighbor (typo)	5.6	10.2	10.4	5.2	3.8	@JetBlue → @JeBtlue I cri INV @SouthwestAir no thanks → thakns INV
NER	***INV:*** Switching locations should not change predictions	7.0	20.8	14.8	7.6	6.4	@JetBlue I want you guys to be the first to fly to # Cuba → Canada... INV @VirginAmerica I miss the #nerdbird in San Jose → Denver INV
	INV: Switching person names should not change predictions	2.4	15.1	9.1	6.6	2.4	...Airport agents were horrendous. Sharon → Erin was your saviour INV @united 8602947, Jon → Sean at http://t.co/58tuTgli0D, thanks. INV

From: Ribeiro, Marco Tulio, Tongshuang Wu, Carlos Guestrin, and Sameer Singh. "Beyond Accuracy: Behavioral Testing of NLP Models with CheckList." In Proceedings ACL, p. 4902–4912. (2020).

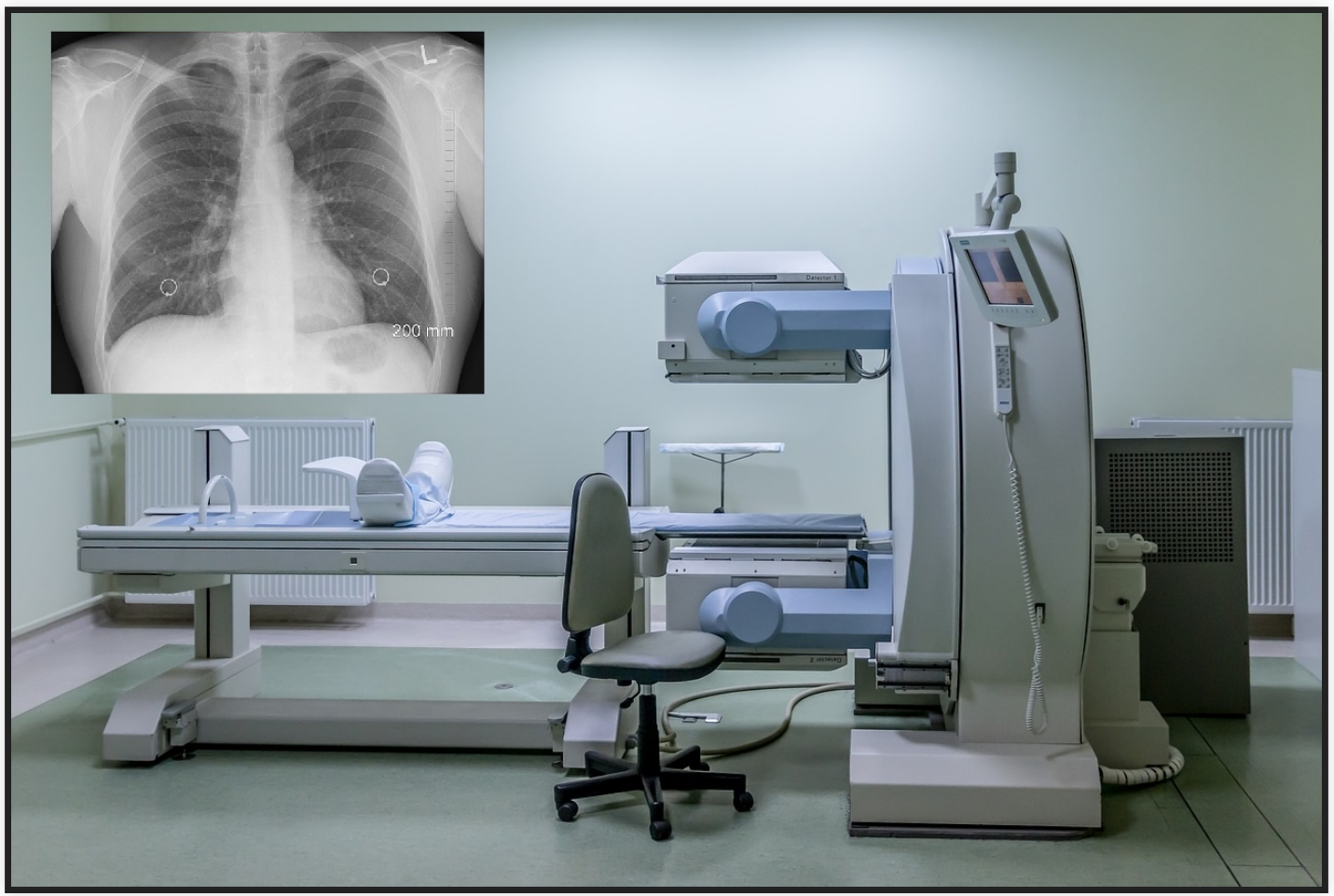
TESTING CAPABILITIES ("STRESS TESTING")

Negation	***MFT:*** Negated negative should be positive or neutral	18.8	54.2	29.4	13.2	2.6	The food is not poor. pos or neutral It isn't a lousy customer service. pos or neutral
	MFT: Negated neutral should still be neutral	40.4	39.6	74.2	98.4	95.4	This aircraft is not private. neutral This is not an international flight. neutral
	MFT: Negation of negative at the end, should be pos. or neut.	100.0	90.4	100.0	84.8	7.2	I thought the plane would be awful, but it wasn't. pos or neutral I thought I would dislike that plane, but I didn't. pos or neutral
	MFT: Negated positive with neutral content in the middle	98.4	100.0	100.0	74.0	30.2	I wouldn't say, given it's a Tuesday, that this pilot was great. neg I don't think, given my history with airplanes, that this is an amazing staff. neg
	MFT: Author sentiment is more important than of others	45.4	62.4	68.0	38.8	30.0	Some people think you are excellent, but I think you are nasty. neg Some people hate you, but I think you are exceptional. pos

From: Ribeiro, Marco Tulio, Tongshuang Wu, Carlos Guestrin, and Sameer Singh. "Beyond Accuracy: Behavioral Testing of NLP Models with CheckList." In Proceedings ACL, p. 4902–4912. (2020).

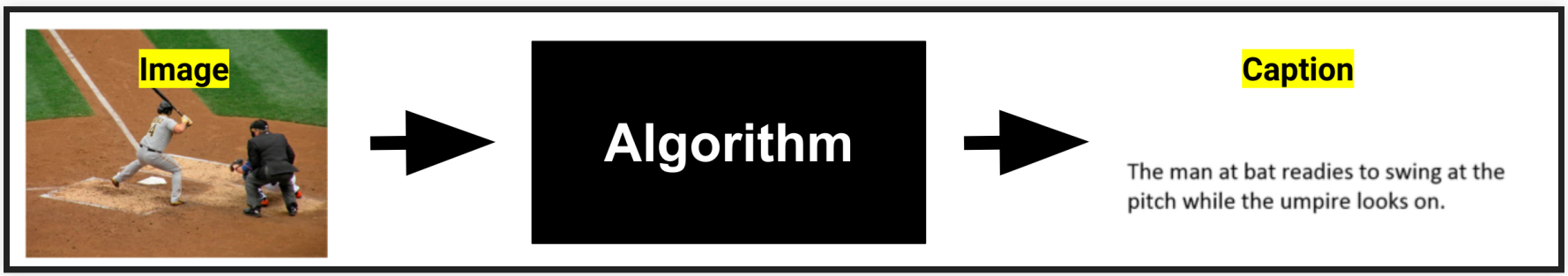
EXAMPLE OF CAPABILITIES

What could be capabilities of the cancer detector?



EXAMPLE OF CAPABILITIES

What could be capabilities of image captioning system?



IS IT FAIR TO EXPECT GENERALIZATION BEYOND TRAINING DISTRIBUTION?

For example, shall a cancer detector generalize to other hospitals? Shall image captioning generalize to describing pictures of star formations?

Speaker notes

We wouldn't test a first year elementary school student on high-school math. This would be "out of the training distribution"

GENERALIZATION BEYOND TRAINING DISTRIBUTION?

- Typically training and validation data from same distribution (e.g., crossvalidation)
- Many models can achieve similar accuracy
- Models that learn "right" abstractions possibly indistinguishable from models that use shortcuts
    - see tank detection example
    - Can we guide the model towards "right" abstractions?
- Some models generalize better to other distributions not used in training
    - e.g., cancer images from other hospitals, from other populations
    - Drift and attacks, ...

See discussion in D'Amour, Alexander, Katherine Heller, Dan Moldovan, Ben Adlam, Babak Alipanahi, Alex Beutel, Christina Chen et al. "Underspecification presents challenges for credibility in modern machine learning." arXiv preprint arXiv:2011.03395 (2020).

TESTING CAPABILITIES MAY HELP WITH GENERALIZATION

- Capabilities are "partial specifications", given beyond training data
- Encode domain knowledge of the problem
- Testing for capabilities helps to distinguish models that use intended abstractions
- May help find models that generalize better

See discussion in D'Amour, Alexander, Katherine Heller, Dan Moldovan, Ben Adlam, Babak Alipanahi, Alex Beutel, Christina Chen et al. "Underspecification presents challenges for credibility in modern machine learning." arXiv preprint arXiv:2011.03395 (2020).

<!--

HOW MUCH VALIDATION DATA?

- Problem dependent
- Statistics can give confidence interval for results
    - e.g. Sample Size Calculator: 384 samples needed for ±5% confidence interval (95% conf. level; 1M population)
- Experience and heuristics. Example: Hulten's heuristics for stable problems:
    - 10s is too small
    - 100s sanity check
    - 1000s usually good
    - 10000s probably overkill
    - Reserve 1000s recent data points for evaluation (or 10%, whichever is more)
    - Reserve 100s for important subpopulations -->

SUMMARY: BLACK-BOX TESTING TECHNIQUES AS INSPIRATION

- Boundary value analysis
- Partition testing & equivalence classes
- Combinatorial testing
- Decision tables

Use to identify datasets for **subpopulations** and **capabilities**, not individual tests.

ON TERMINOLOGY

- Test data curation is emerging as a very recent concept for testing ML components
- No consistent terminology
    - "Testing capabilities" in checklist paper
    - "Stress testing" in some others (but stress testing has a very different meaning in software testing: robustness to overload)
- Software engineering concepts translate, but names not adopted in ML community
    - specification-based testing, black-box testing
    - equivalence class testing, boundary-value analysis

AUTOMATED (RANDOM) TESTING AND INVARIANTS

(if it wasn't for that darn oracle problem)

RANDOM TEST INPUT GENERATION IS EASY

```
@Test
void testNextDate() {
  nextDate(488867101, 1448338253, -997372169)
  nextDate(2105943235, 1952752454, 302127018)
  nextDate(1710531330, -127789508, 1325394033)
  nextDate(-1512900479, -439066240, 889256112)
  nextDate(1853057333, 1794684858, 1709074700)
  nextDate(-1421091610, 151976321, 1490975862)
  nextDate(-2002947810, 680830113, -1482415172)
  nextDate(-1907427993, 1003016151, -2120265967)
}
```

But is it useful?

RANDOMLY GENERATING "REALISTIC" INPUTS IS POSSIBLE

```
@Test
void testNextDate() {
  nextDate(2010, 8, 20)
  nextDate(2024, 7, 15)
  nextDate(2011, 10, 27)
  nextDate(2024, 5, 4)
  nextDate(2013, 8, 27)
  nextDate(2010, 2, 30)
}
```

But how do we know whether the computation is correct?

AUTOMATED MODEL VALIDATION DATA GENERATION?

```
@Test
void testCancerPrediction() {
  cancerModel.predict(generateRandomImage())
  cancerModel.predict(generateRandomImage())
  cancerModel.predict(generateRandomImage())
}
```

- Realistic inputs?
- But how do we get labels?

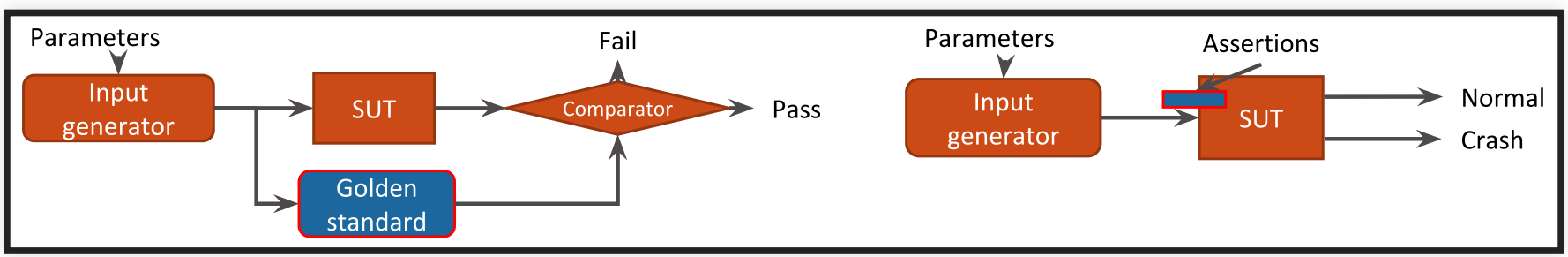
THE ORACLE PROBLEM

How do we know the expected output of a test?

```
assertEquals(??, factorPrime(15485863));
```

TEST CASE GENERATION & THE ORACLE PROBLEM

- Manually construct input-output pairs (does not scale, cannot automate)
- Comparison against gold standard (e.g., alternative implementation, executable specification)
- Checking of global properties only -- crashes, buffer overflows, code injections
- Manually written assertions -- partial specifications checked at runtime



MANUALLY CONSTRUCTING OUTPUTS

```
@Test
void testNextDate() {
  assert nextDate(2010, 8, 20) == (2010, 8, 21);
  assert nextDate(2024, 7, 15) == (2024, 7, 16);
  assert nextDate(2011, 10, 27) == (2011, 10, 28);
  assert nextDate(2024, 5, 4) == (2024, 5, 5);
  assert nextDate(2013, 8, 27) == (2013, 8, 28);
  assert nextDate(2010, 2, 30) throws InvalidInputException;
}
```

```
@Test
void testCancerPrediction() {
  assert cancerModel.predict(loadImage("random1.jpg")) == true;
  assert cancerModel.predict(loadImage("random2.jpg")) == true;
  assert cancerModel.predict(loadImage("random3.jpg")) == false;
}
```

(tedious, labor intensive; possibly crowd sourced)

COMPARE AGAINST REFERENCE IMPLEMENTATION

assuming we have a correct implementation

```
@Test
void testNextDate() {
  assert nextDate(2010, 8, 20) == referenceLib.nextDate(2010, 8,
  assert nextDate(2024, 7, 15) == referenceLib.nextDate(2024, 7,
  assert nextDate(2011, 10, 27) == referenceLib.nextDate(2011, 1
  assert nextDate(2024, 5, 4) == referenceLib.nextDate(2024, 5,
  assert nextDate(2013, 8, 27) == referenceLib.nextDate(2013, 8,
  assert nextDate(2010, 2, 30) == referenceLib.nextDate(2010, 2,
}
```

```
@Test
void testCancerPrediction() {
  assert cancerModel.predict(loadImage("random1.jpg")) == ???;
}
```

(usually no reference implementation for ML problems)

CHECKING GLOBAL SPECIFICATIONS

Ensure, no computation crashes

```
@Test
void testNextDate() {
  nextDate(2010, 8, 20)
  nextDate(2024, 7, 15)
  nextDate(2011, 10, 27)
  nextDate(2024, 5, 4)
  nextDate(2013, 8, 27)
  nextDate(2010, 2, 30)
}
```

```
@Test
void testCancerPrediction() {
  cancerModel.predict(generateRandomImage())
  cancerModel.predict(generateRandomImage())
  cancerModel.predict(generateRandomImage())
}
```

(we usually do fear crashing bugs in ML models)

INVARIANTS AS PARTIAL SPECIFICATION

```
class Stack {
  int size = 0;
  int MAX_SIZE = 100;
  String[] data = new String[MAX_SIZE];
  // class invariant checked before and after every method
  private void check() {
    assert(size>=0 && size<=MAX_SIZE);
  }
  public void push(String v) {
    check();
    if (size<MAX_SIZE)
      data[+size] = v;
    check();
  }
  public void pop(String v) { check(); ... }
}
```

AUTOMATED TESTING / TEST CASE GENERATION / FUZZING

- Many techniques to generate test cases
- Dumb fuzzing: generate random inputs
- Smart fuzzing (e.g., symbolic execution, coverage guided fuzzing): generate inputs to maximally cover the implementation
- Program analysis to understand the shape of inputs, learning from existing tests
- Minimizing redundant tests
- Abstracting/simulating/mocking the environment
- Typically looking for crashing bugs or assertion violations

TEST GENERATION EXAMPLE (SYMBOLIC EXECUTION)

Code:

```
void foo(a, b, c) {
    int x=0, y=0, z=0;
    if (a) x=-2;
    if (b<5) {
        if (!a && c) y=1;
        z=2;
    }
    assert(x+y+z!=3)
}
```

Paths:

- $a \wedge (b < 5)$: x=-2, y=0, z=2
- $a \wedge \neg(b < 5)$: x=-2, y=0, z=0
- $\neg a \wedge (\neg a \wedge c)$: x=0, z=1, z=2
- $\neg a \wedge (b < 5) \wedge \neg(\neg a \wedge c)$: x=0, z=0, z=2
- $\neg a \wedge (b < 5) \wedge \neg(\neg a \wedge c)$: x=0, z=0, z=2
- $\neg a \wedge \neg(b < 5)$: x=0, z=0, z=0

Speaker notes

example source: http://web.cs.iastate.edu/~weile/cs641/9.SymbolicExecution.pdf

GENERATING INPUTS FOR ML PROBLEMS

- Completely random data generation (uniform sampling from each feature's domain)
- Using knowledge about feature distributions (sample from each feature's distribution)
- Knowledge about dependencies among features and whole population distribution (e.g., model with probabilistic programming language)
- Mutate from existing inputs (e.g., small random modifications to select features)

MACHINE LEARNED MODELS = UNTESTABLE SOFTWARE?

```
@Test
void testCancerPrediction() {
  cancerModel.predict(generateRandomImage())
}
```

- Manually construct input-output pairs (does not scale, cannot automate)
  - **too expensive at scale**
- Comparison against gold standard (e.g., alternative implementation, executable specification)
  - **no specification, usually no other "correct" model**
  - comparing different techniques useful? (see ensemble learning)
  - semi-supervised learning as approximation?
- Checking of global properties only -- crashes, buffer overflows, code injections - **??**
- Manually written assertions -- partial specifications checked at runtime - **??**

INVARIANTS IN MACHINE LEARNED MODELS (METAMORPHIC TESTING)

Exploit relationships between inputs

- If two inputs differ only in **X** -> output should be the same
- If inputs differ in **Y** output should be flipped
- If inputs differ only in feature **F**, prediction for input with higher F should be higher
- ...

INVARIANTS IN MACHINE LEARNED MODELS?

EXAMPLES OF INVARIANTS

- Credit rating should not depend on gender:
  - $\forall x.\ f(x[\text{gender} \leftarrow \text{male}]) = f(x[\text{gender} \leftarrow \text{female}])$
- Synonyms should not change the sentiment of text:
  - $\forall x.\ f(x) = f(\text{replace}(x, \text{"is not"}, \text{"isn't"}))$
- Negation should swap meaning:
  - $\forall x \in \text{"X is Y"}.\ f(x) = 1 - f(\text{replace}(x, \text{" is "}, \text{" is not "}))$
- Robustness around training data:
  - $\forall x \in \text{training data}.\ \forall y \in \text{mutate}(x, \delta).\ f(x) = f(y)$
- Low credit scores should never get a loan (sufficient conditions for classification, "anchors"):
  - $\forall x.\ x.\ \text{score} < 649 \Rightarrow \neg f(x)$

Identifying invariants requires domain knowledge of the problem!

METAMORPHIC TESTING

Formal description of relationships among inputs and outputs (*Metamorphic Relations*)

In general, for a model f and inputs x define two functions to transform inputs and outputs g_I and g_O such that:

$$\forall x.\ f(g_I(x)) = g_O(f(x))$$

e.g. $g_I(x) = \text{replace}(x, \text{" is ", " is not "})$ and $g_O(x) = \neg x$

MORE EXAMPLES

Some capability tests can be expressed as invariants and automatically encoded as transformations to existing test data

- Negation should flip sentiment analysis result
- Typos should not affect sentiment analysis result
- Changes to locations or names should not affect sentiment analysis results

Robust.	***INV:*** Add randomly generated URLs and handles to tweets	9.6	13.4	24.8	11.4	7.4	@JetBlue that selfie was extreme. @pi9QDK INV @united stuck because staff took a break? Not happy 1K.... https://t.co/PWK1jb INV
	INV: Swap one character with its neighbor (typo)	5.6	10.2	10.4	5.2	3.8	@JetBlue → @JeBtlue I cri INV @SouthwestAir no thanks → thakns INV
NER	***INV:*** Switching locations should not change predictions	7.0	20.8	14.8	7.6	6.4	@JetBlue I want you guys to be the first to fly to # Cuba → Canada... INV @VirginAmerica I miss the #nerdbird in San Jose → Denver INV
	INV: Switching person names should not change predictions	2.4	15.1	9.1	6.6	2.4	...Airport agents were horrendous. Sharon → Erin was your saviour INV @united 8602947, Jon → Sean at http://t.co/58tuTgli0D, thanks. INV

From: Ribeiro, Marco Tulio, Tongshuang Wu, Carlos Guestrin, and Sameer Singh. "Beyond Accuracy: Behavioral Testing of NLP Models with CheckList." In Proceedings ACL, p. 4902–4912. (2020).

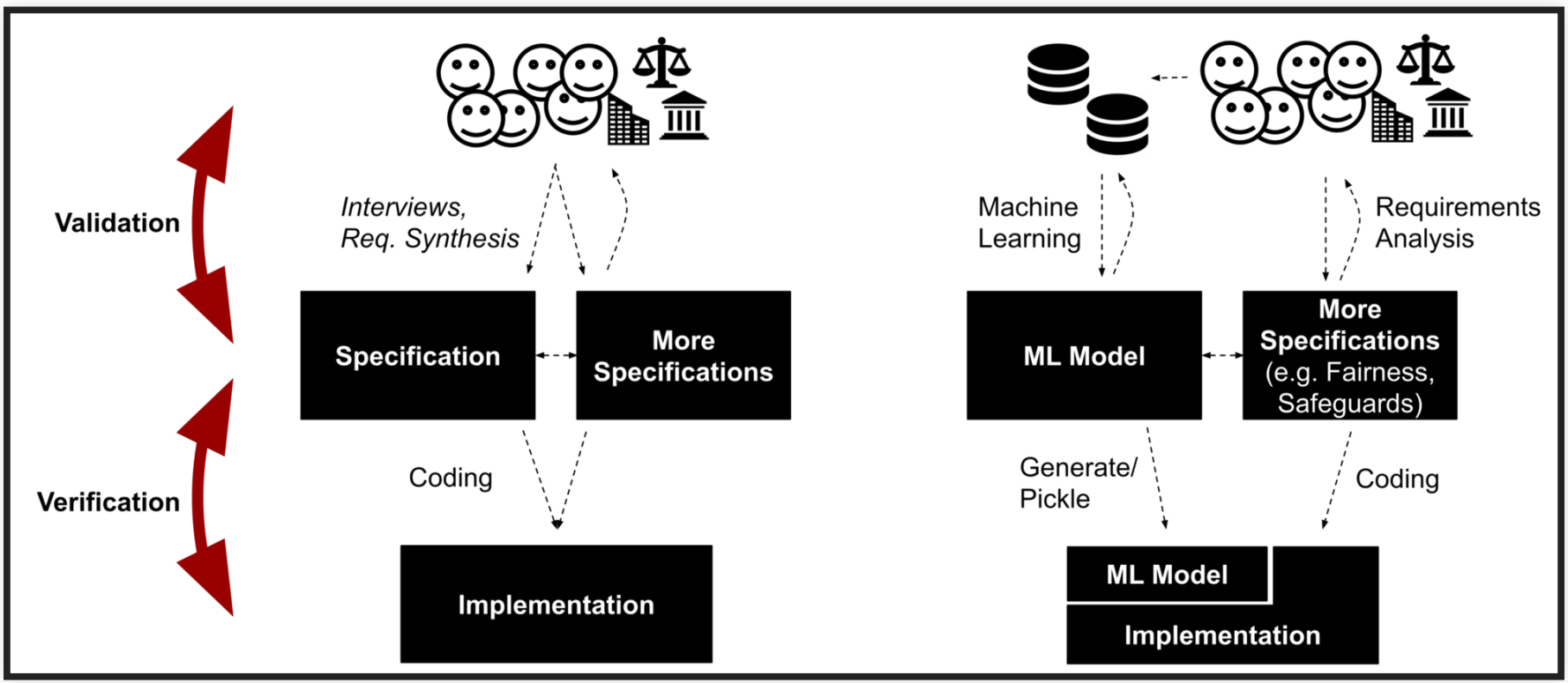
ON TESTING WITH INVARIANTS/ASSERTIONS

- Defining good metamorphic relations requires knowledge of the problem domain
- Good metamorphic relations focus on parts of the system
- Invariants usually cover only one aspect of correctness -- maybe capabilities
- Invariants and near-invariants can be mined automatically from sample data (see *specification mining* and *anchors*)

Further reading:

- Segura, Sergio, Gordon Fraser, Ana B. Sanchez, and Antonio Ruiz-Cortés. "A survey on metamorphic testing." IEEE Transactions on software engineering 42, no. 9 (2016): 805-824.
- Ribeiro, Marco Tulio, Sameer Singh, and Carlos Guestrin. "Anchors: High-precision model-agnostic explanations." In Thirty-Second AAAI Conference on Artificial Intelligence. 2018.

INVARIANT CHECKING ALIGNS WITH REQUIREMENTS VALIDATION



APPROACHES FOR CHECKING IN VARIANTS

- Generating test data (random, distributions) usually easy
- Transformations of existing test data
- Adversarial learning: For many techniques gradient-based techniques to search for invariant violations -- that's roughly analogous to symbolic execution in SE
- Early work on formally verifying invariants for certain models (e.g., small deep neural networks)

Further readings: Singh, Gagandeep, Timon Gehr, Markus Püschel, and Martin Vechev. "An abstract domain for certifying neural networks." Proceedings of the ACM on Programming Languages 3, no. POPL (2019): 1-30.

USING INVARIANT VIOLATIONS

- Are invariants strict?
  - Single violation in random inputs usually not meaningful
  - In capability testing, average accuracy in realistic data needed
  - Maybe strict requirements for fairness or robustness?
- Do invariant violations matter if the input data is not representative?

ONE MORE THING: SIMULATION-BASED TESTING

- In some cases it is easy to go from outputs to inputs:

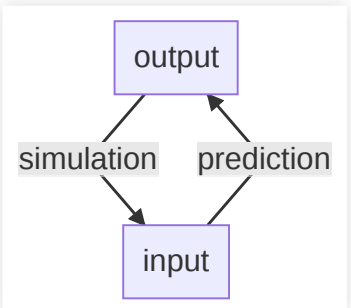
```
assertEquals(??, factorPrime(15485862));
```

```
randomNumbers = [2, 3, 7, 7, 52673]
assertEquals(randomNumbers,
    factorPrime(multiply(randomNumbers)));
```

Similar idea in machine-learning problems?

ONE MORE THING: SIMULATION-BASED TESTING

- Derive input-output pairs from simulation, esp. in vision systems
- Example: Vision for self-driving cars:
    - Render scene -> add noise -> recognize -> compare recognized result with simulator state
- Quality depends on quality of the simulator and how well it can produce inputs from outputs:
    - examples: render picture/video, synthesize speech, ...
    - Less suitable where input-output relationship unknown, e.g., cancer detection, housing price prediction, shopping recommendations



Further readings: Zhang, Mengshi, Yuqun Zhang, Lingming Zhang, Cong Liu, and Sarfraz Khurshid. "DeepRoad: GAN-based metamorphic testing and input validation framework for autonomous driving systems." In Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, pp. 132-142. 2018.

SUMMARY: INVARIANTS AND GENERATION

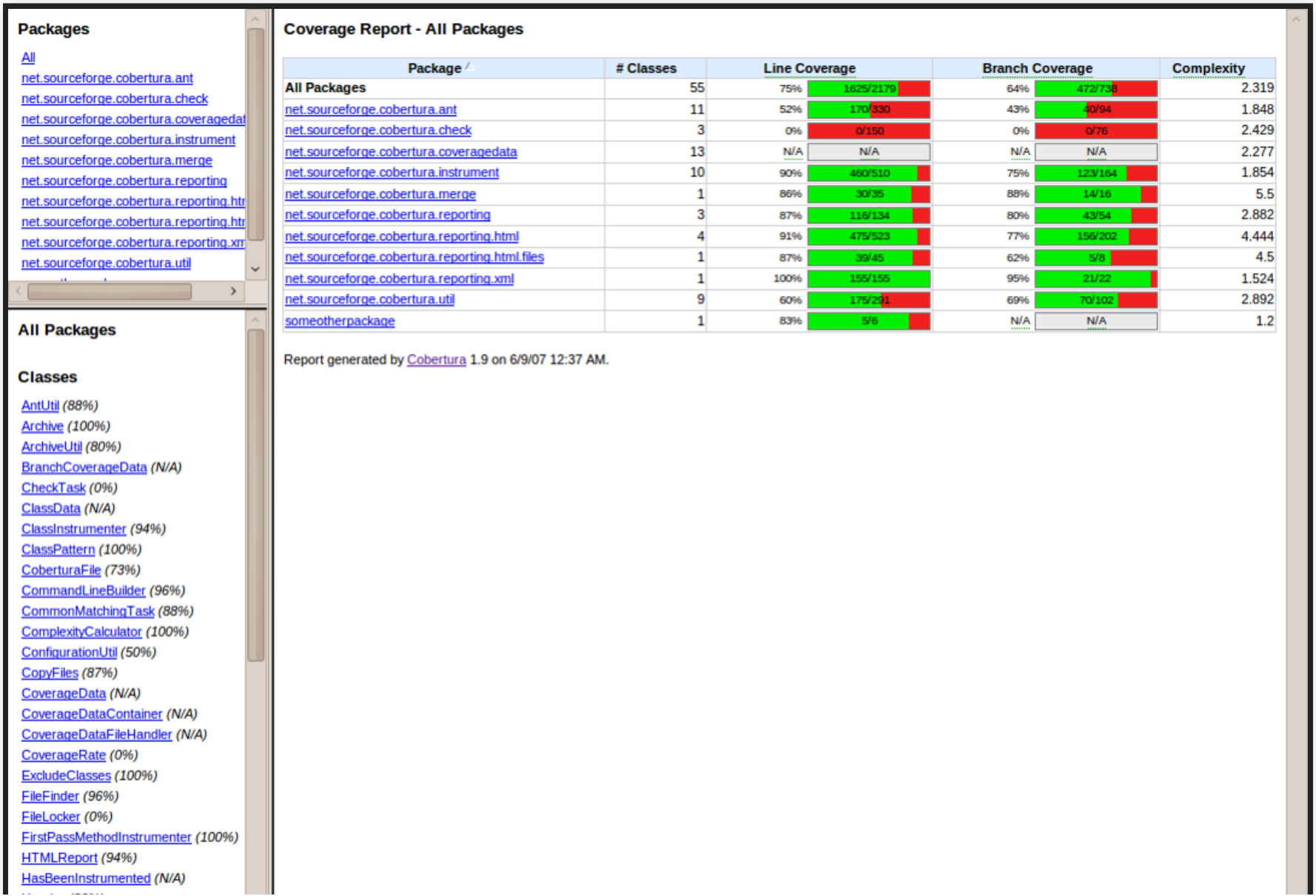
- Generating sample inputs is easy, but knowing corresponding outputs is not (oracle problem)
- Crashing bugs are not a concern
- Invariants + generated data can check capabilities or properties (metamorphic testing)
    - Inputs can be generated realistically or to find violations (adversarial learning)
- If inputs can be computed from outputs, tests can be automated (simulation-based testing)

ON TERMINOLOGY

- *Metamorphic testing* is a software engineering term that's not common in ML literature, it generalizes many concepts regularly reinvented
- Much of the security, safety and robustness literature in ML focuses on invariants

OTHER TESTING CONCEPTS

TEST COVERAGE



Packages

All
net.sourceforge.cobertura.ant
net.sourceforge.cobertura.check
net.sourceforge.cobertura.coveragedat
net.sourceforge.cobertura.instrument
net.sourceforge.cobertura.merge
net.sourceforge.cobertura.reporting
net.sourceforge.cobertura.reporting.htr
net.sourceforge.cobertura.reporting.htr
net.sourceforge.cobertura.reporting.xm
net.sourceforge.cobertura.util

All Packages

Classes

AntUtil *(88%)*
Archive *(100%)*
ArchiveUtil *(80%)*
BranchCoverageData *(N/A)*
CheckTask *(0%)*
ClassData *(N/A)*
ClassInstrumenter *(94%)*
ClassPattern *(100%)*
CoberturaFile *(73%)*
CommandLineBuilder *(96%)*
CommonMatchingTask *(88%)*
ComplexityCalculator *(100%)*
ConfigurationUtil *(50%)*
CopyFiles *(87%)*
CoverageData *(N/A)*
CoverageDataContainer *(N/A)*
CoverageDataFileHandler *(N/A)*
CoverageRate *(0%)*
ExcludeClasses *(100%)*
FileFinder *(96%)*
FileLocker *(0%)*
FirstPassMethodInstrumenter *(100%)*
HTMLReport *(94%)*
HasBeenInstrumented *(N/A)*

Coverage Report - All Packages

Package	# Classes	Line Coverage		Branch Coverage		Complexity
All Packages	55	75%	1625/2179	64%	472/738	2.319
net.sourceforge.cobertura.ant	11	52%	170/330	43%	40/94	1.848
net.sourceforge.cobertura.check	3	0%	0/150	0%	0/76	2.429
net.sourceforge.cobertura.coveragedata	13	N/A	N/A	N/A	N/A	2.277
net.sourceforge.cobertura.instrument	10	90%	460/510	75%	123/164	1.854
net.sourceforge.cobertura.merge	1	86%	30/35	88%	14/16	5.5
net.sourceforge.cobertura.reporting	3	87%	116/134	80%	43/54	2.882
net.sourceforge.cobertura.reporting.html	4	91%	475/523	77%	156/202	4.444
net.sourceforge.cobertura.reporting.html.files	1	87%	39/45	62%	5/8	4.5
net.sourceforge.cobertura.reporting.xml	1	100%	155/155	95%	21/22	1.524
net.sourceforge.cobertura.util	9	60%	175/291	69%	70/102	2.892
someotherpackage	1	83%	5/6	N/A	N/A	1.2

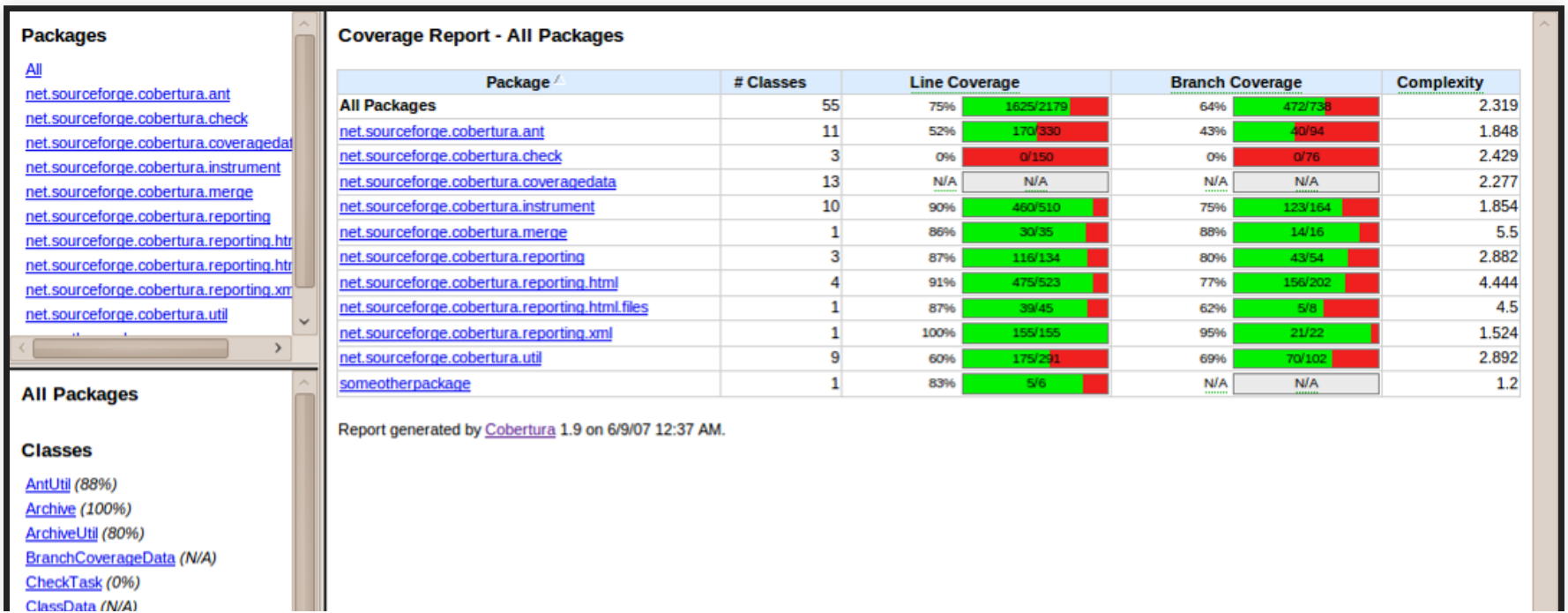
Report generated by Cobertura 1.9 on 6/9/07 12:37 AM.


Header (80%)
IOUtil (62%)
Ignore (100%)
IgnoreBranches (0%)

EXAMPLE: WHITE-BOX TESTING

```
int divide(int A, int B) {
  if (A==0)
    return 0;
  if (B==0)
    return -1;
  return A / B;
}
```

minimum set of test cases to cover all lines? all decisions? all path?



Packages

All
net.sourceforge.cobertura.ant
net.sourceforge.cobertura.check
net.sourceforge.cobertura.coveragedat
net.sourceforge.cobertura.instrument
net.sourceforge.cobertura.merge
net.sourceforge.cobertura.reporting
net.sourceforge.cobertura.reporting.ht
net.sourceforge.cobertura.reporting.ht
net.sourceforge.cobertura.reporting.xm
net.sourceforge.cobertura.util

All Packages

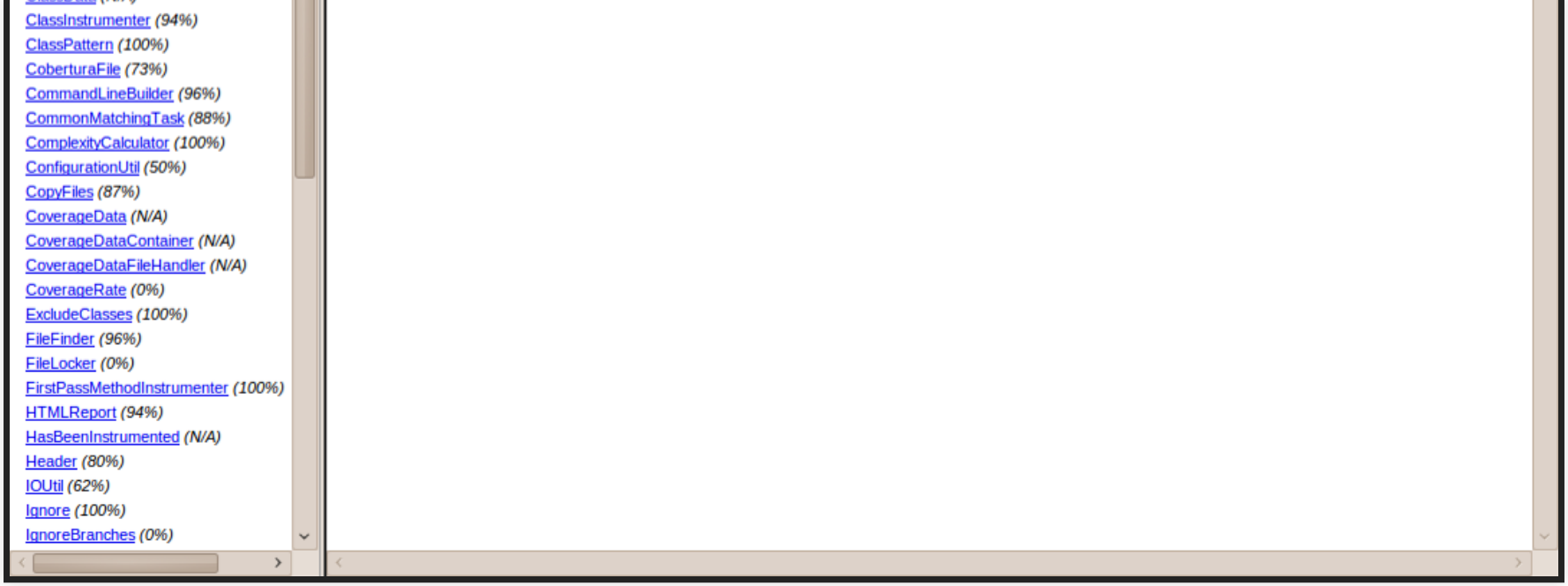
Classes

AntUtil *(88%)*
Archive *(100%)*
ArchiveUtil *(80%)*
BranchCoverageData *(N/A)*
CheckTask *(0%)*
ClassData *(N/A)*

Coverage Report - All Packages

Package	# Classes	Line Coverage		Branch Coverage		Complexity
All Packages	55	75%	1625/2179	64%	472/738	2.319
net.sourceforge.cobertura.ant	11	52%	170/330	43%	40/94	1.848
net.sourceforge.cobertura.check	3	0%	0/150	0%	0/76	2.429
net.sourceforge.cobertura.coveragedata	13	N/A	N/A	N/A	N/A	2.277
net.sourceforge.cobertura.instrument	10	90%	460/510	75%	123/164	1.854
net.sourceforge.cobertura.merge	1	86%	30/35	88%	14/16	5.5
net.sourceforge.cobertura.reporting	3	87%	116/134	80%	43/54	2.882
net.sourceforge.cobertura.reporting.html	4	91%	475/523	77%	156/202	4.444
net.sourceforge.cobertura.reporting.html.files	1	87%	39/45	62%	5/8	4.5
net.sourceforge.cobertura.reporting.xml	1	100%	155/155	95%	21/22	1.524
net.sourceforge.cobertura.util	9	60%	175/291	69%	70/102	2.892
someotherpackage	1	83%	5/6	N/A	N/A	1.2

Report generated by Cobertura 1.9 on 6/9/07 12:37 AM.


ClassInstrumenter (94%)
ClassPattern (100%)
CoberturaFile (73%)
CommandLineBuilder (96%)
CommonMatchingTask (88%)
ComplexityCalculator (100%)
ConfigurationUtil (50%)
CopyFiles (87%)
CoverageData (N/A)
CoverageDataContainer (N/A)
CoverageDataFileHandler (N/A)
CoverageRate (0%)
ExcludeClasses (100%)
FileFinder (96%)
FileLocker (0%)
FirstPassMethodInstrumenter (100%)
HTMLReport (94%)
HasBeenInstrumented (N/A)
Header (80%)
IOUtil (62%)
Ignore (100%)
IgnoreBranches (0%)

DEFINING WHITE-BOX TESTING

- Test case creation is driven by the implementation, not the specification
- Typically aiming to increase coverage of lines, decisions, etc
- Automated test generation often driven by maximizing coverage (for finding crashing bugs)

WHITEBOX ANALYSIS IN ML

- Several coverage metrics have been proposed
  - All path of a decision tree?
  - All neurons activated at least once in a DNN? (several papers "neuron coverage")
  - Linear regression models??
- Often create artificial inputs, not realistic for distribution
- Unclear whether those are useful
- Adversarial learning techniques usually more efficient at finding invariant violations

REGRESSION TESTING

- Whenever bug detected and fixed, add a test case
- Make sure the bug is not reintroduced later
- Execute test suite after changes to detect regressions
    - Ideally automatically with continuous integration tools

- Maps well to curating test sets for important populations in ML

MUTATION ANALYSIS

- Start with program and passing test suite
- Automatically insert small modifications ("mutants") in the source code
  - a+b -> a-b
  - a<b -> a<=b
  - ...
- Can program detect modifications ("kill the mutant")?
- Better test suites detect more modifications ("mutation score")

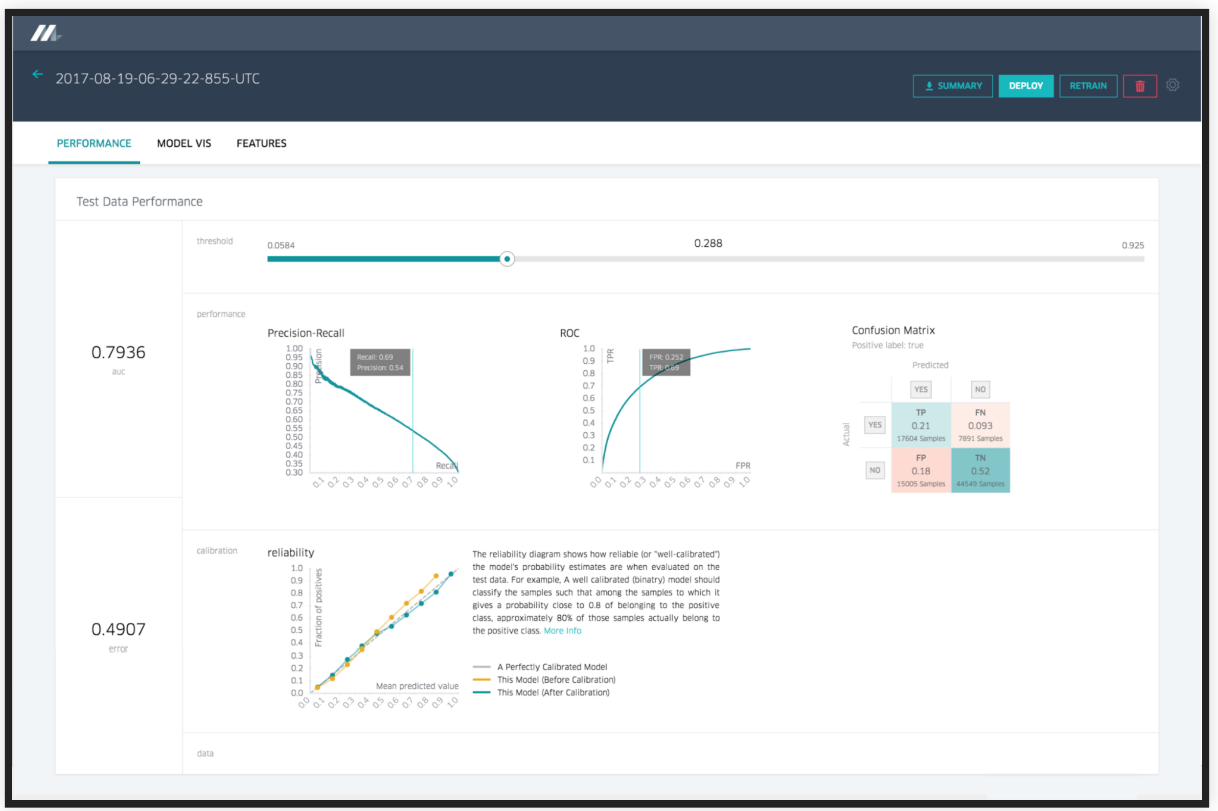
```
int divide(int A, int B) {
  if (A==0)     // A!=0, A<0, B==0
    return 0;   // 1, -1
  if (B==0)     // B!=0, B==1
    return -1;  // 0, -2
  return A / B; // A*B, A+B
}
assert(1, divide(1,1));
assert(0, divide(0,1));
assert(-1, divide(1,0));
```

MUTATION ANALYSIS

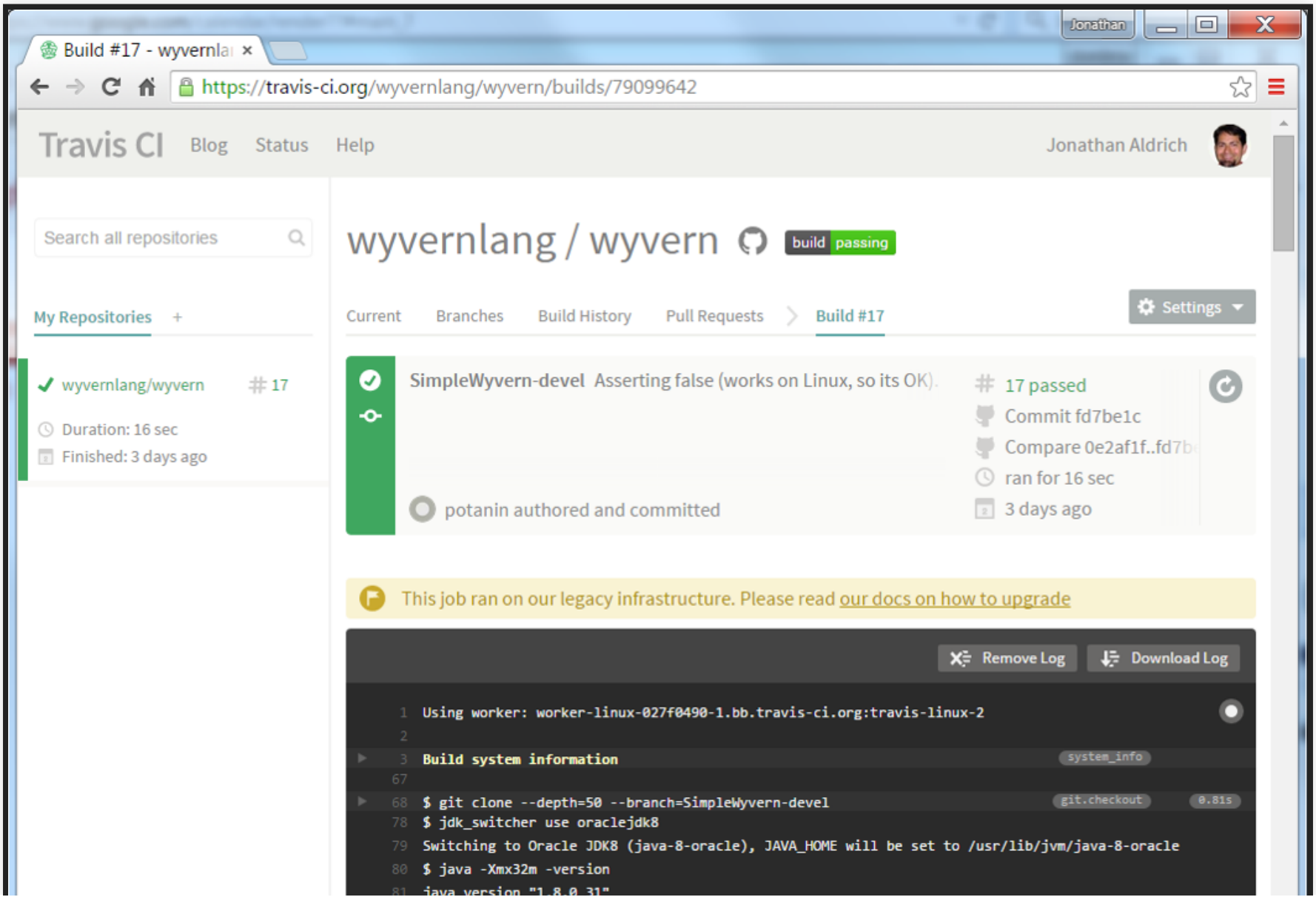
- Some papers exist, but strategy unclear
- Mutating model parameters? Mutating hyperparameters? Mutating inputs?
- What's considered as killing a mutant, if we don't have specifications?

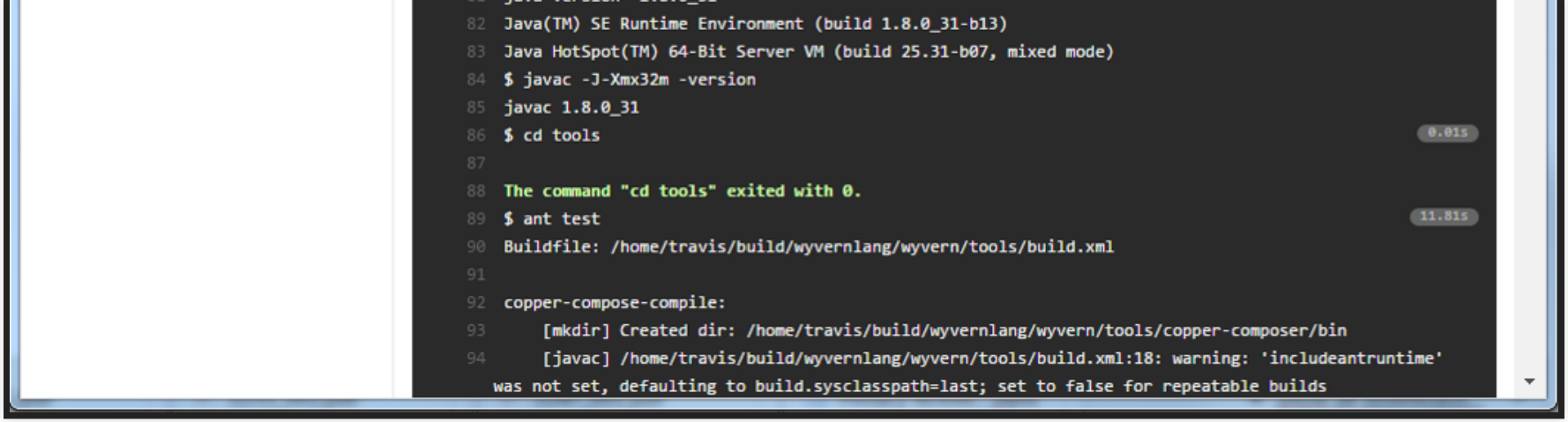
- Still unclear application...

CONTINUOUS INTEGRATION FOR MODEL QUALITY



CONTINUOUS INTEGRATION




82 Java(TM) SE Runtime Environment (build 1.8.0_31-b13)
83 Java HotSpot(TM) 64-Bit Server VM (build 25.31-b07, mixed mode)
84 $ javac -J-Xmx32m -version
85 javac 1.8.0_31
86 $ cd tools
0.01s
87
88 The command "cd tools" exited with 0.
89 $ ant test
11.81s
90 Buildfile: /home/travis/build/wyvernlang/wyvern/tools/build.xml
91
92 copper-compose-compile:
93 [mkdir] Created dir: /home/travis/build/wyvernlang/wyvern/tools/copper-composer/bin
94 [javac] /home/travis/build/wyvernlang/wyvern/tools/build.xml:18: warning: 'includeantruntime'
was not set, defaulting to build.sysclasspath=last; set to false for repeatable builds

CONTINUOUS INTEGRATION FOR MODEL QUALITY?

CONTINUOUS INTEGRATION FOR MODEL QUALITY

- Testing script
  - Existing model: Implementation to automatically evaluate model on labeled training set; multiple separate evaluation sets possible, e.g., for critical subcommunities or regressions
  - Training model: Automatically train and evaluate model, possibly using cross-validation; many ML libraries provide built-in support
  - Report accuracy, recall, etc. in console output or log files
  - May deploy learning and evaluation tasks to cloud services
  - Optionally: Fail test below quality bound (e.g., accuracy <.9; accuracy < accuracy of last model)
- Version control test data, model and test scripts, ideally also learning data and learning code (feature extraction, modeling, ...)
- Continuous integration tool can trigger test script and parse output, plot for comparisons (e.g., similar to performance tests)
- Optionally: Continuous deployment to production server

DASHBOARDS FOR MODEL EVALUATION RESULTS

2017-08-19-06-29-22-855-UTC

SUMMARY | DEPLOY | RETRAIN

PERFORMANCE | MODEL VIS | FEATURES

Test Data Performance

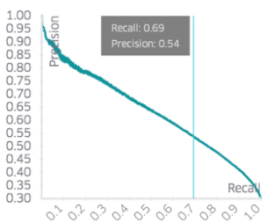
threshold

0.0584 — 0.288 — 0.925

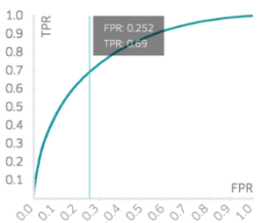
performance

0.7936
auc

Precision-Recall



ROC



Confusion Matrix

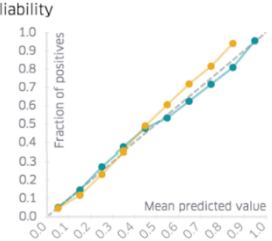
Positive label: true

Actual \ Predicted	YES	NO
YES	TP 0.21 17604 Samples	FN 0.093 7891 Samples
NO	FP 0.18 15005 Samples	TN 0.52 44549 Samples

0.4907
error

calibration

reliability

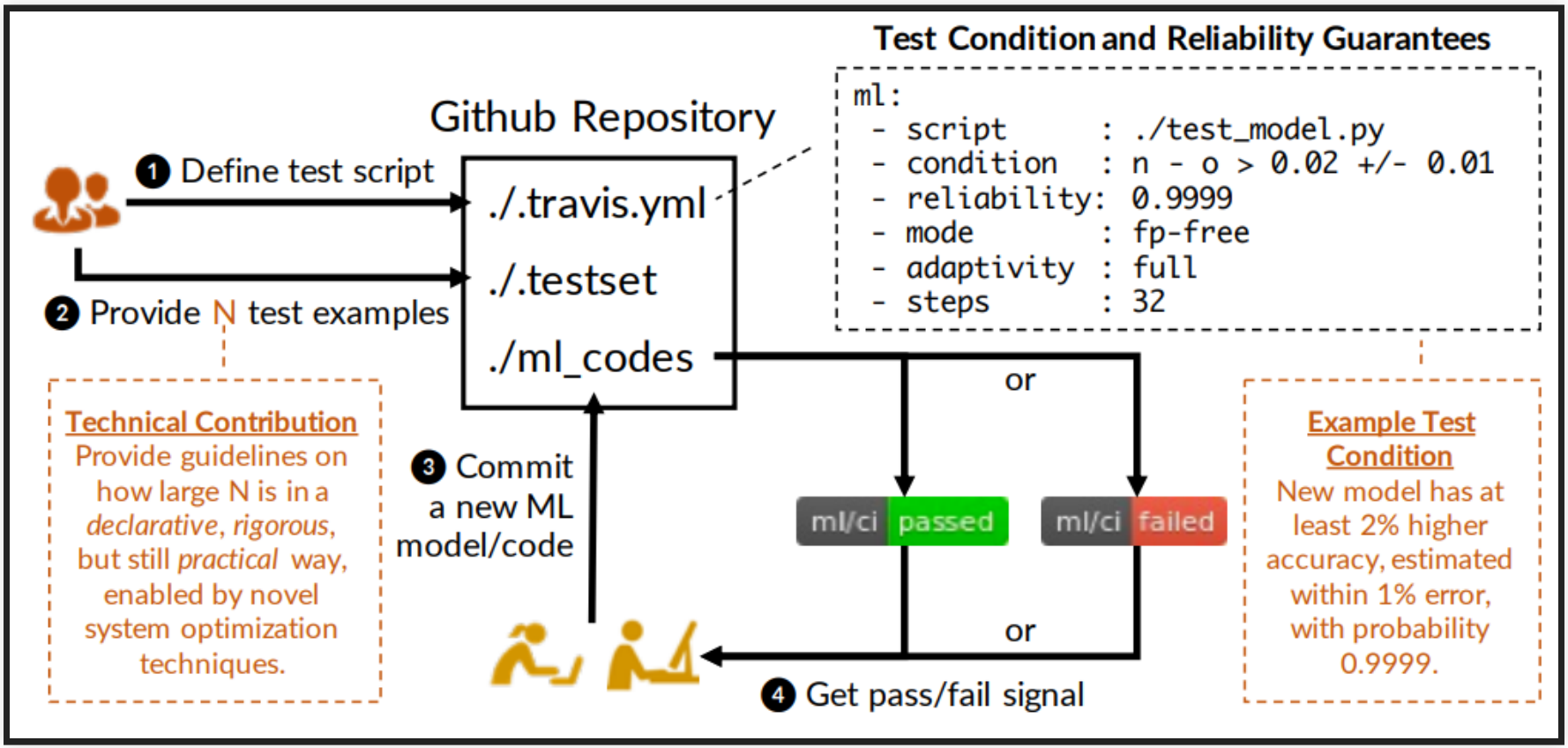


The reliability diagram shows how reliable (or "well-calibrated") the model's probability estimates are when evaluated on the test data. For example, A well calibrated (binatry) model should classify the samples such that among the samples to which it gives a probability close to 0.8 of belonging to the positive class, approximately 80% of those samples actually belong to the positive class. More Info

- A Perfectly Calibrated Model
- This Model (Before Calibration)
- This Model (After Calibration)

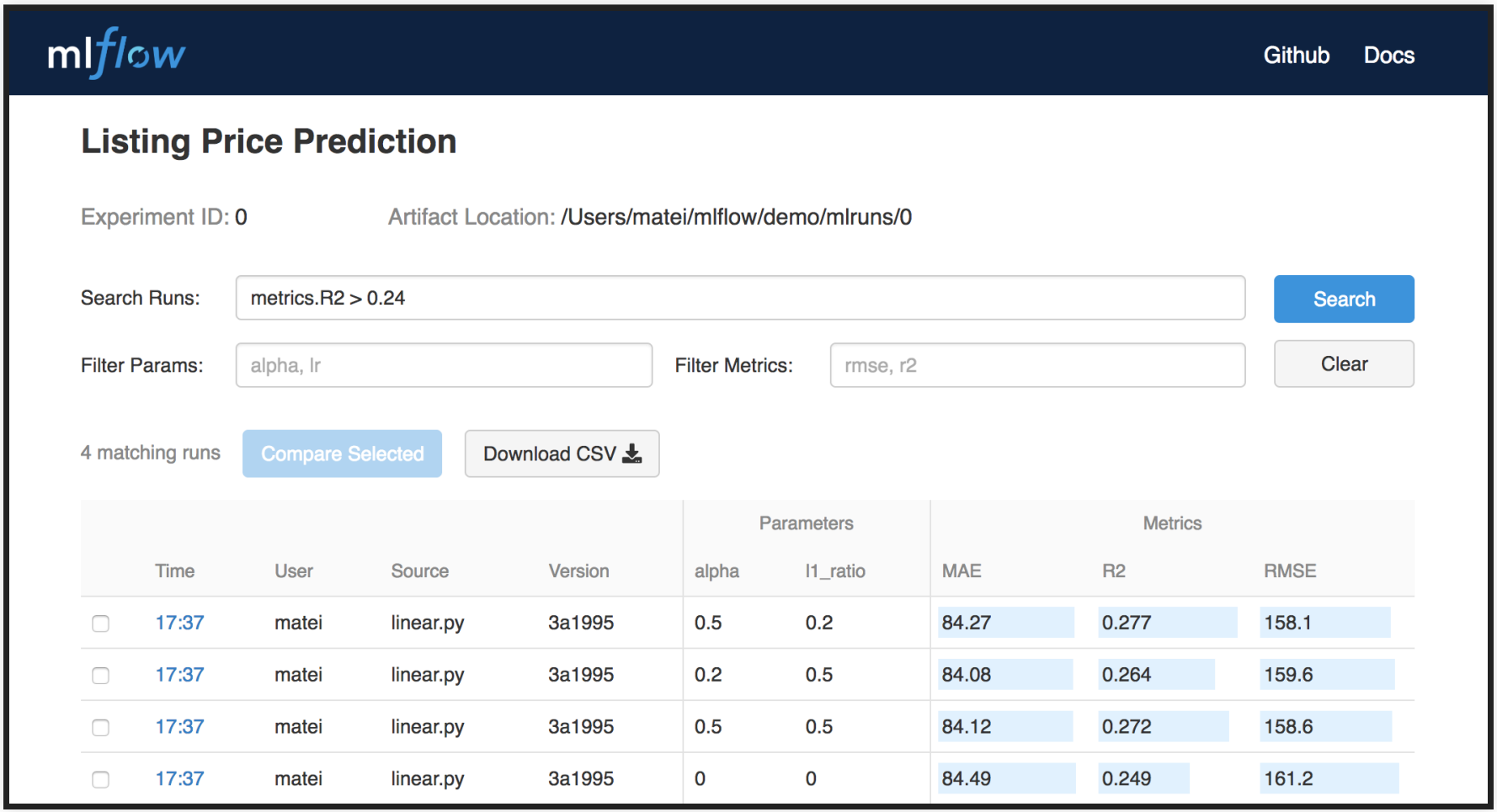
data

SPECIALIZED CI SYSTEMS



Renggli et. al, Continuous Integration of Machine Learning Models with ease.ml/ci: Towards a Rigorous Yet Practical Treatment, SysML 2019

DASHBOARDS FOR COMPARING MODELS



Matei Zaharia. Introducing MLflow: an Open Source Machine Learning Platform, 2018

SUMMARY

- Model prediction accuracy only one part of system quality
- Select suitable measure for prediction accuracy, depending on problem
- Use baselines for interpreting prediction accuracy; ensure independence of test and validation data
- Software bugs vs model fit in the absence of specifications
- Curating test data
    - Analyzing specifications, capabilities
    - Not all inputs are equal: Identify important inputs (inspiration from blackbox testing)
    - Slice data for evaluation
- Automated random testing
    - Feasible with invariants (e.g. metamorphic relations)
    - Sometimes possible with simulation
- Automate the test execution with continuous integration

FURTHER READINGS

- Ribeiro, Marco Tulio, Sameer Singh, and Carlos Guestrin. "Semantically equivalent adversarial rules for debugging NLP models." In Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp. 856-865. 2018.
- Barash, Guy, Eitan Farchi, Ilan Jayaraman, Orna Raz, Rachel Tzoref-Brill, and Marcel Zalmanovici. "Bridging the gap between ML solutions and their business requirements using feature interactions." In Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 1048-1058. 2019.
- Kaestner, Christian. "Machine Learning is Requirements Engineering — On the Role of Bugs, Verification, and Validation in Machine Learning." Medium Blog Post. 2020.
- Ashmore, Rob, Radu Calinescu, and Colin Paterson. "Assuring the machine learning lifecycle: Desiderata, methods, and challenges." arXiv preprint arXiv:1905.04223. 2019.
- D'Amour, Alexander, Katherine Heller, Dan Moldovan, Ben Adlam, Babak Alipanahi, Alex Beutel, Christina Chen et al. "Underspecification presents challenges for credibility in modern machine learning." *arXiv preprint arXiv:2011.03395* (2020).
- Segura, Sergio, Gordon Fraser, Ana B. Sanchez, and Antonio Ruiz-Cortés. "A survey on metamorphic testing." IEEE Transactions on software engineering 42, no. 9 (2016): 805-824.