MODEL QUALITY 2

SLICING, CAPABILITIES, INVARIANTS, AND OTHER TESTING STRATEGIES

Christian Kaestner

Required reading:

- 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).

ADMINISTRATIVA

- Finalized waitlist
- Team and VM updates

- Back to in-person
    - Subscribe to `#lecture` slack channel
    - Experimental: using slack for breakout groups and polls
    - Ask immediate question in person, background questions in Slack

TEAMWORK REMARK: DIVIDING THE WORK

- Coordinate at meetings
- Read assignment before meeting
- Discuss big picture and how to divide work (inner teams?)
- Consider task dependencies

- **Write down explicit deliverables**
  - *Who* does *what* by *when*
  - Be explicit about expected results, should be verifiable
  - Track completion, check off when done
  - GitHub issues, Trello board, Google docs, ... -- **single source of truth, with history tracking**
- Complete deliverable list **during meeting**: everybody writes their own deliverables, others read all deliverables to check understanding
  - if not completed during meeting or team member not at meeting, email assignment after meeting to everybody; no objection within 24h counts as agreement with task assignment

LEARNING GOALS

- Curate validation datasets for assessing model quality, covering subpopulations and capabilities as needed
- Explain the oracle problem and how it challenges testing of software and models
- Use invariants to check partial model properties with automated testing
- Select and deploy automated infrastructure to evaluate and monitor model quality

MODEL QUALITY

FIRST PART: MEASURING PREDICTION ACCURACY

the data scientist's perspective (last lecture)

SECOND PART: WHAT IS CORRECTNESS ANYWAY?

the role and lack of specifications, validation vs verification (last lecture)

THIRD PART: LEARNING FROM SOFTWARE TESTING

unit testing, test case curation, invariants, test case generation (this lecture)

LATER: TESTING IN PRODUCTION

monitoring, A/B testing, canary releases (next week)

CURATING VALIDATION DATA & INPUT SLICING

(Learning from Software Testing)

BREAKOUT DISCUSSION

Write a few tests for the following program:

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

A test may look like:

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

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

Post answer to `#lecture` in Slack using template:

Selection strategy: ...
Test quantity: ...
AndrewIDs: ...

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
- Specification-based testing ("black box"): Derive test cases from specifications
    - Boundary value analysis
    - Equivalence classes
    - Combinatorial testing
    - Random testing
- Structural testing ("white box"): 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 *specification-based testing*.

IMPORTANT INPUT GROUPS FOR CANCER PROGNOSIS?

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

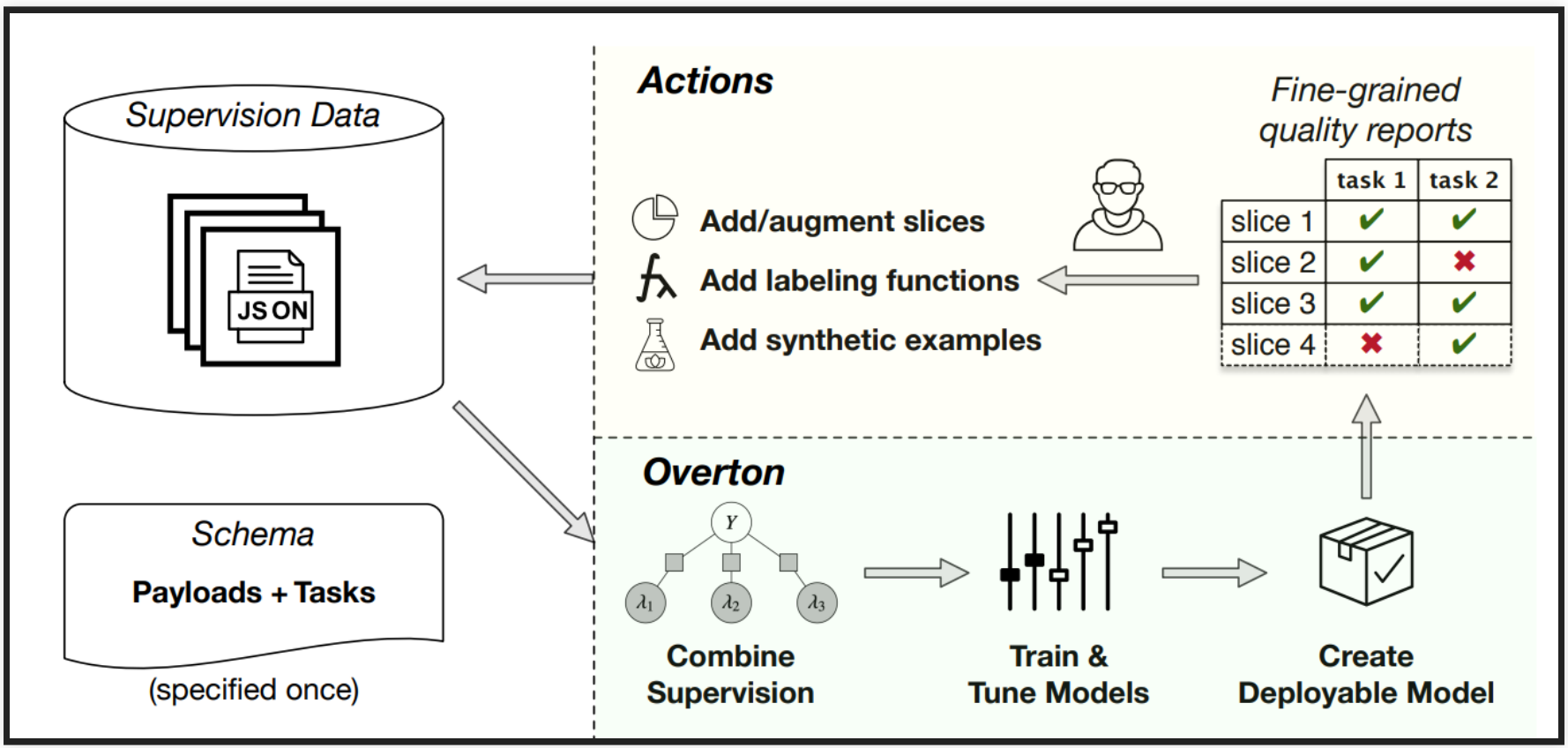
Input divided by genre, rating, and length. Accuracy differs, but also amount of test data used ("support") differs, highlighting low confidence areas.

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 prognosis?

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 MODEL CAPABILITIES

("stress testing")

Further reading: Christian Kaestner. Rediscovering Unit Testing: Testing Capabilities of ML Models. Toward Data Science, 2021.

TESTING CAPABILITIES

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

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).

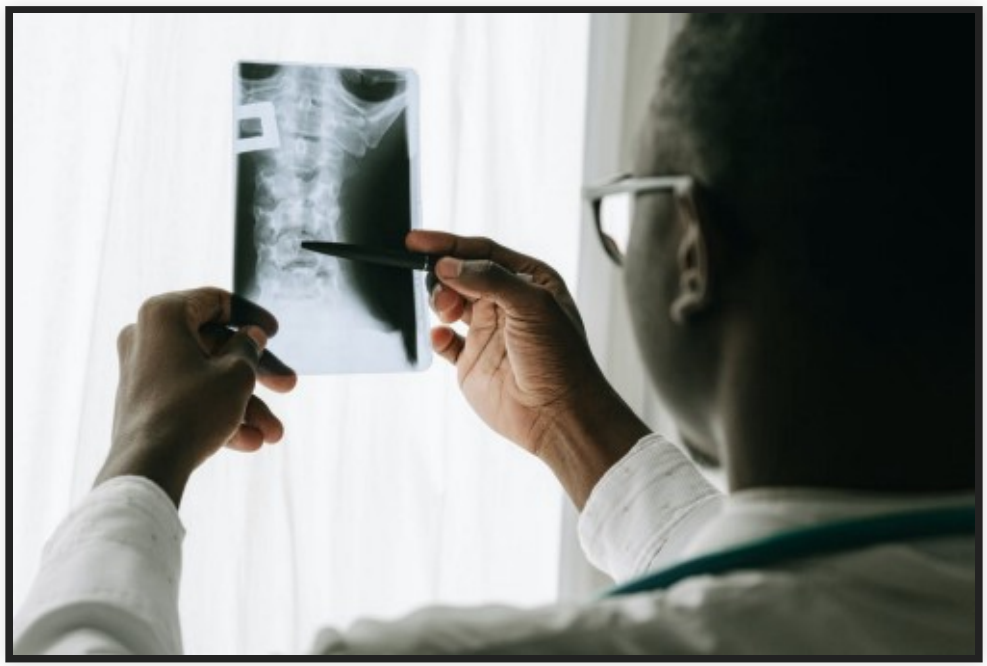
TESTING CAPABILITIES

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

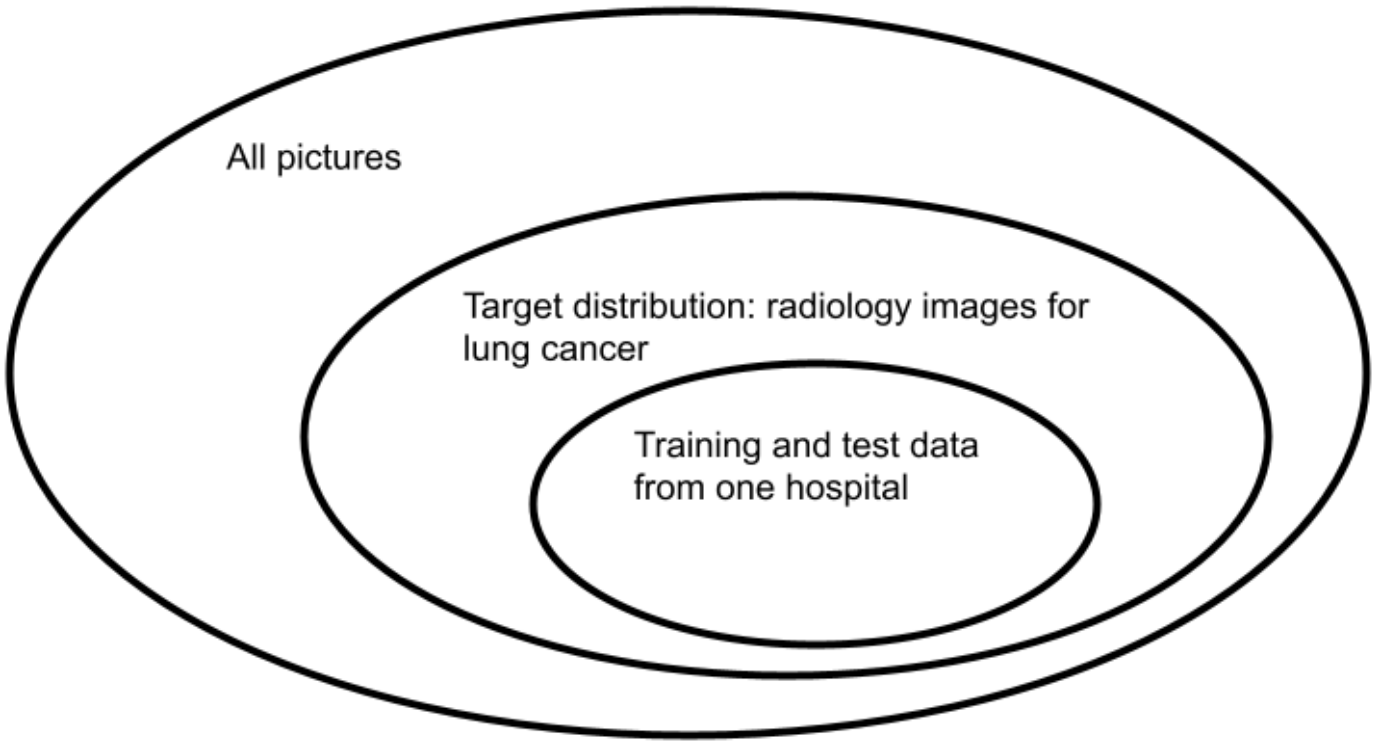
From: Ribeiro, Marco Tulio, Tongshuang Wu, Carlos Guestrin, and Sameer Singh. "Beyond Accuracy: Behavioral Testing of NLP Models with CheckList." In Proceedings ACL, p. 4902–4912. (2020).

EXAMPLES OF CAPABILITIES

What could be capabilities of the cancer classifier?



RECALL: 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"

RECALL: SHORTCUT LEARNING

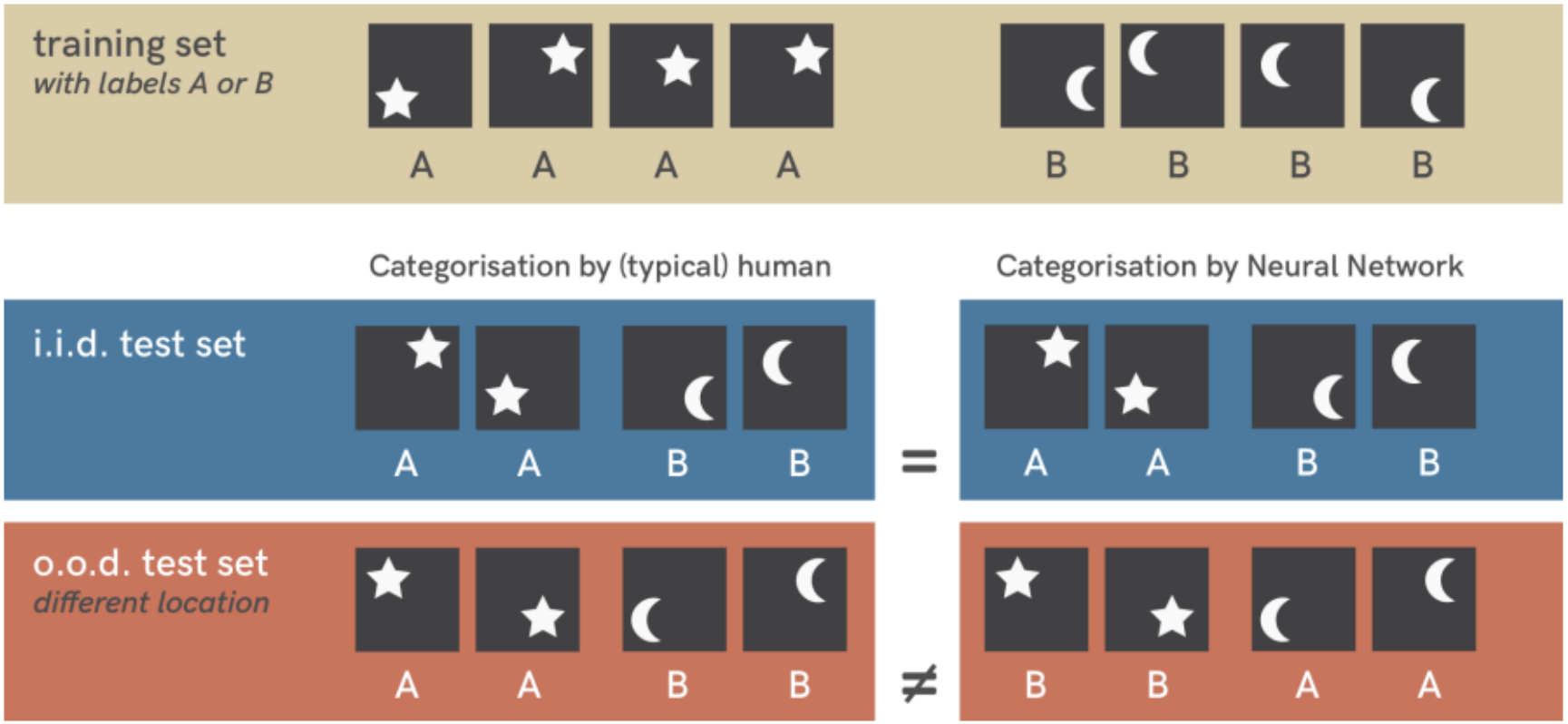
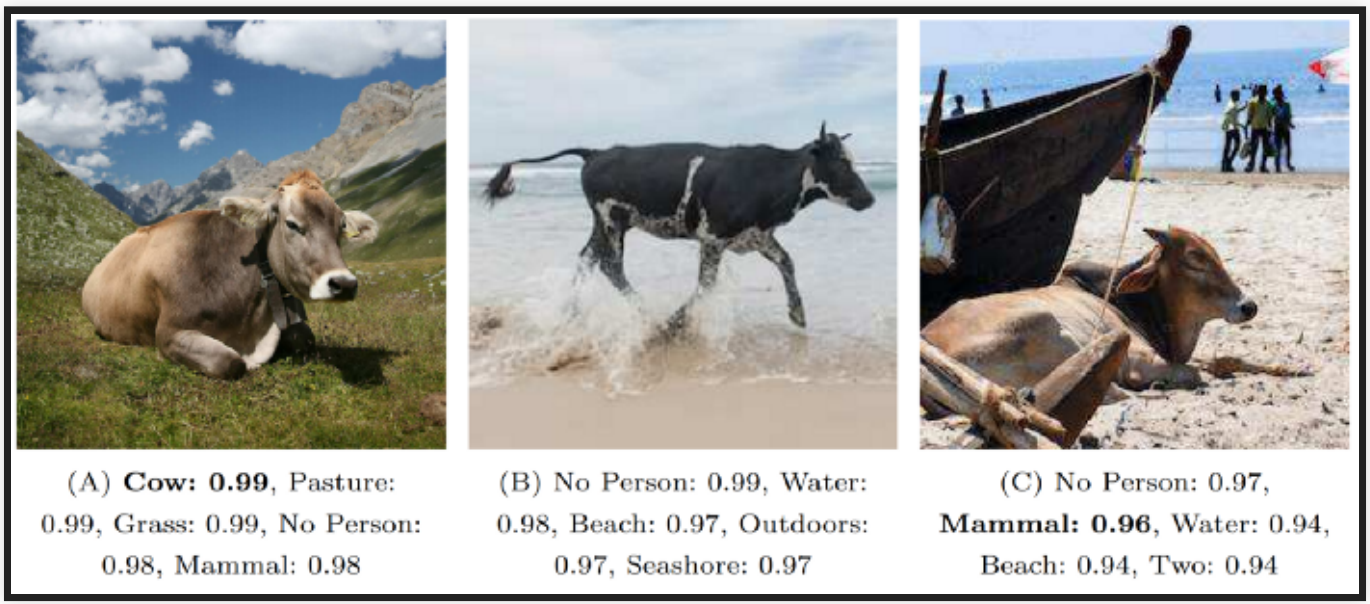


Figure from: Geirhos, Robert, Jörn-Henrik Jacobsen, Claudio Michaelis, Richard Zemel, Wieland Brendel, Matthias Bethge, and Felix A. Wichmann. "Shortcut learning in deep neural networks." Nature Machine Intelligence 2, no. 11 (2020): 665-673.

MORE SHORTCUT LEARNING :)



(A) **Cow: 0.99**, Pasture: 0.99, Grass: 0.99, No Person: 0.98, Mammal: 0.98

(B) No Person: 0.99, Water: 0.98, Beach: 0.97, Outdoors: 0.97, Seashore: 0.97

(C) No Person: 0.97, **Mammal: 0.96**, Water: 0.94, Beach: 0.94, Two: 0.94

Figure from Beery, Sara, Grant Van Horn, and Pietro Perona. “Recognition in terra incognita.” In Proceedings of the European Conference on Computer Vision (ECCV), pp. 456–473. 2018.

GENERALIZATION BEYOND TRAINING DISTRIBUTION?

- Typically training and validation data from same distribution (i.i.d. assumption!)
- 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
    - Capabilities are inherently domain specific
    - Curate capability-specific test data for a 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).

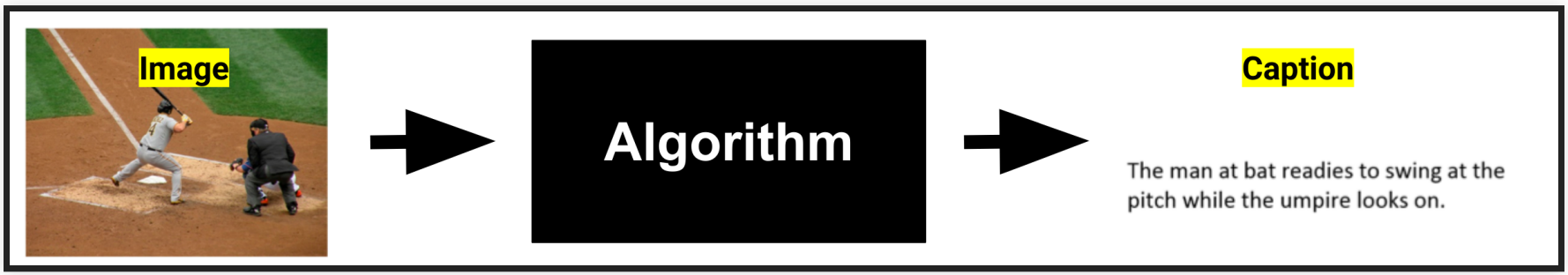
STRATEGIES FOR IDENTIFYING CAPABILITIES

- Analyze common mistakes (e.g., classify past mistakes in cancer prognosis)
- Use existing knowledge about the problem (e.g., linguistics theories)
- Observe humans (e.g., how do radiologists look for cancer)
- Derive from requirements (e.g., fairness)
- Causal discovery from observational data?

Further reading: Christian Kaestner. Rediscovering Unit Testing: Testing Capabilities of ML Models. Toward Data Science, 2021.

EXAMPLES OF CAPABILITIES

What could be capabilities of image captioning system?



GENERATING TEST DATA FOR CAPABILITIES

Idea 1: Domain-specific generators

Testing *negation* in sentiment analysis with template:

`I {NEGATION} {POS_VERB} the {THING}.`

Testing texture vs shape priority with artificial generated images:

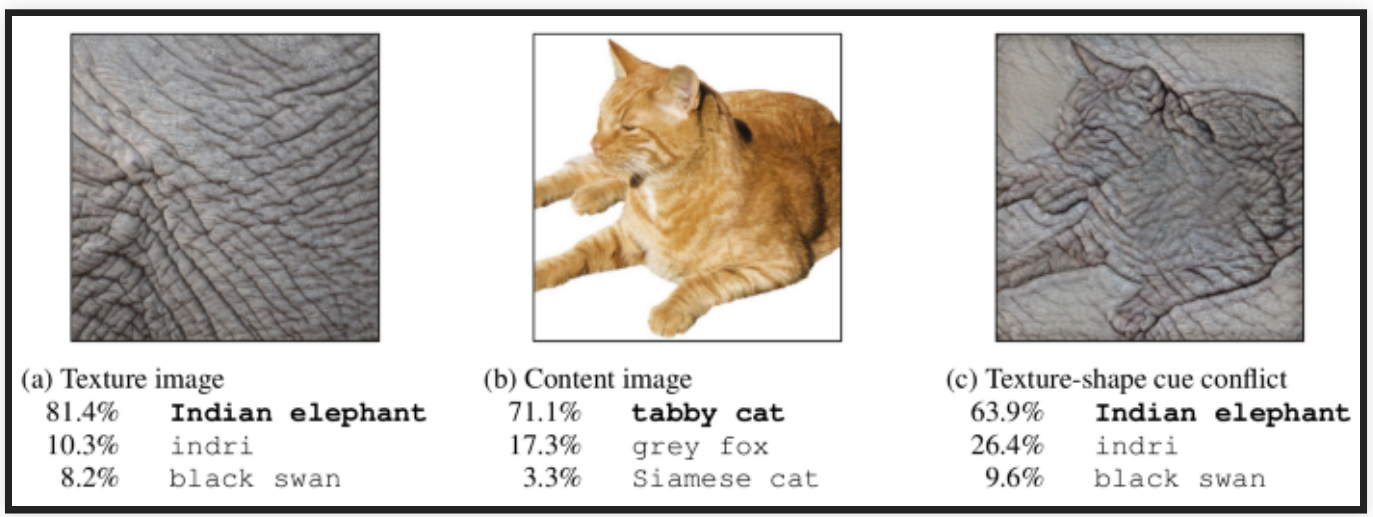


Figure from Geirhos, Robert, Patricia Rubisch, Claudio Michaelis, Matthias Bethge, Felix A. Wichmann, and Wieland Brendel. “ImageNet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness.” In Proc. International Conference on Learning Representations (ICLR), (2019).

GENERATING TEST DATA FOR CAPABILITIES

Idea 2: Mutating existing inputs

Testing *synonyms* in sentiment analysis by replacing words with synonyms, keeping label

Testing *robust against noise and distraction* add `and false is not true` or random URLs to text

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

Figure 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).

GENERATING TEST DATA FOR CAPABILITIES

Idea 3: Crowd-sourcing test creation

Testing *sarcasm* in sentiment analysis: Ask humans to minimally change text to flip sentiment with sarcasm

Testing *background* in object detection: Ask humans to take pictures of specific objects with unusual backgrounds

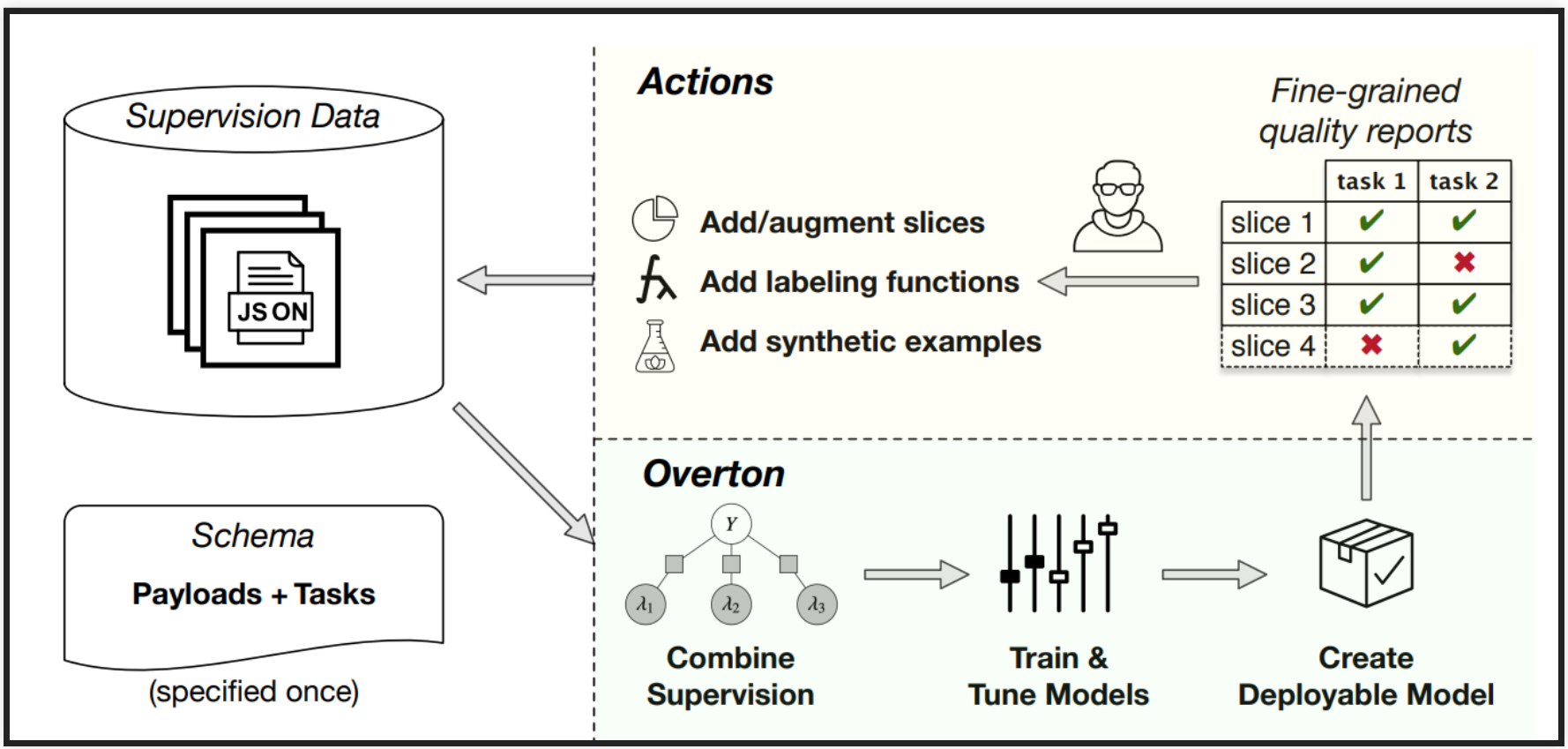
Recasting *fact* as *hoped for*	The world of Atlantis, hidden beneath the earth's core, is fantastic The world of Atlantis, hidden beneath the earth's core is **supposed** to be fantastic
Suggesting sarcasm	thoroughly captivating **thriller-drama, taking a deep and realistic** view thoroughly mind numbing **"thriller-drama", taking a "deep" and "realistic" (who are they kidding?)** view
Inserting modifiers	The presentation of simply Atlantis' landscape and setting The presentation of Atlantis' **predictable** landscape and setting

Figure from: Kaushik, Divyansh, Eduard Hovy, and Zachary C. Lipton. "Learning the difference that makes a difference with counterfactually-augmented data." In Proc. International Conference on Learning Representations (ICLR), (2020).

GENERATING TEST DATA FOR CAPABILITIES

Idea 4: Slicing test data

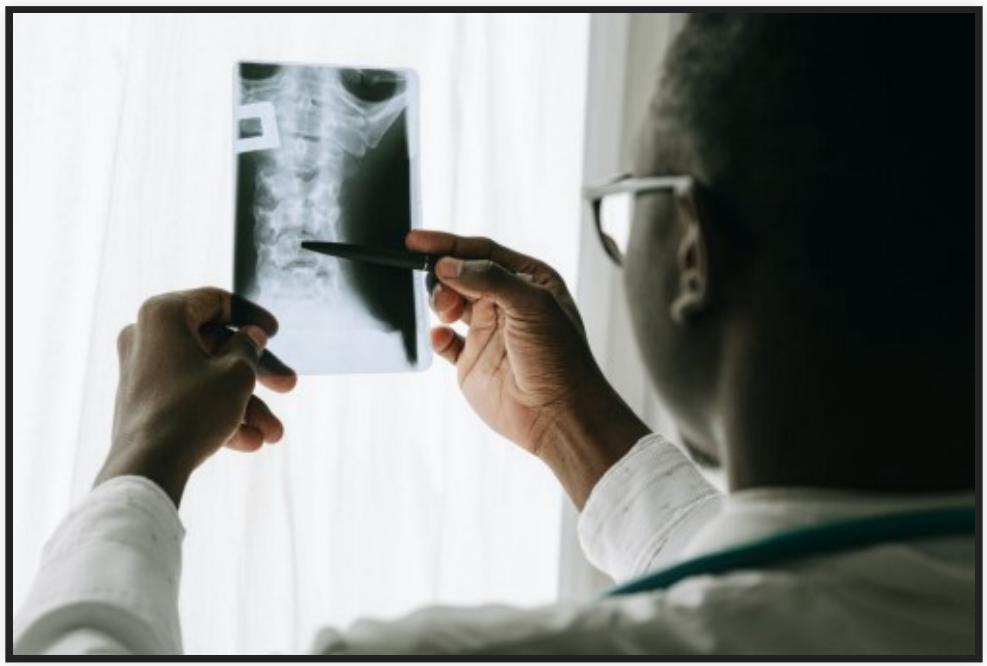
Testing *negation* in sentiment analysis by finding sentences containing 'not'



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).

EXAMPLES OF CAPABILITIES

How to generate test data for capabilities of the cancer classifier?



TESTING VS TRAINING CAPABILITIES

- Dual insight for testing and training
- Strategies for curating test data can also help select training data
- Generate capability-specific training data to guide training (data augmentation)

Further reading on using domain knowledge during training: Von Rueden, Laura, Sebastian Mayer, Jochen Garcke, Christian Bauckhage, and Jannis Schuecker. "Informed machine learning–towards a taxonomy of explicit integration of knowledge into machine learning." Learning 18 (2019): 19-20.

PRELIMINARY SUMMARY: SPECIFICATION-BASED 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

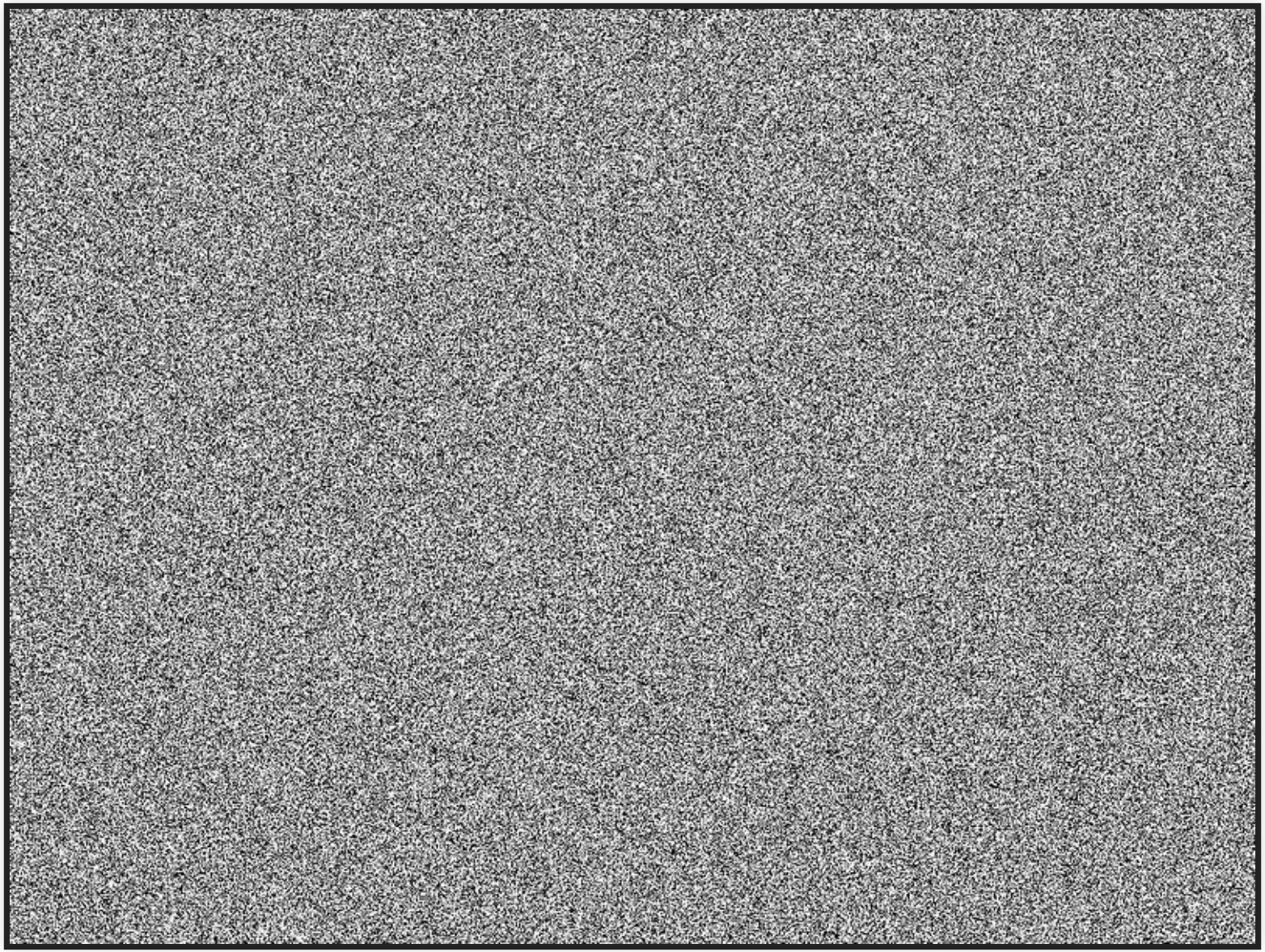
(if it wasn't for that darn oracle problem)

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?

CANCER IN RANDOM IMAGE?



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?

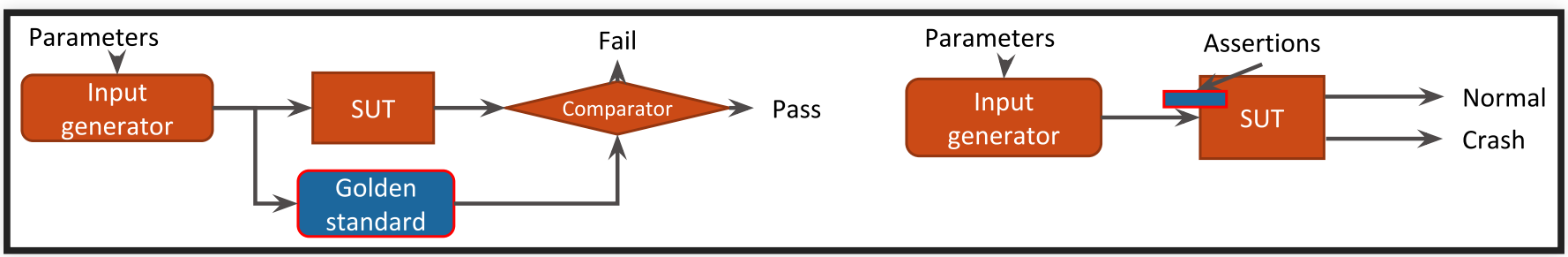
THE ORACLE PROBLEM

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)
- Generate "fake data" with Generative Adversarial Networks

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?

SOME CAPABILITIES ARE INVARIANTS

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).

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:
  - $\forall x. f(x) = f(\texttt{replace}(x, \text{"is not", "isn't"}))$
- Negation should swap meaning:
  - $\forall x \in \text{"X is Y"}. f(x) = 1-f(\texttt{replace}(x, \text{" is ", " is not "}))$
- 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. $g_I(x)= \texttt{replace}(x, \text{" is ", " is not "})$ and $g_O(x)=\neg x$

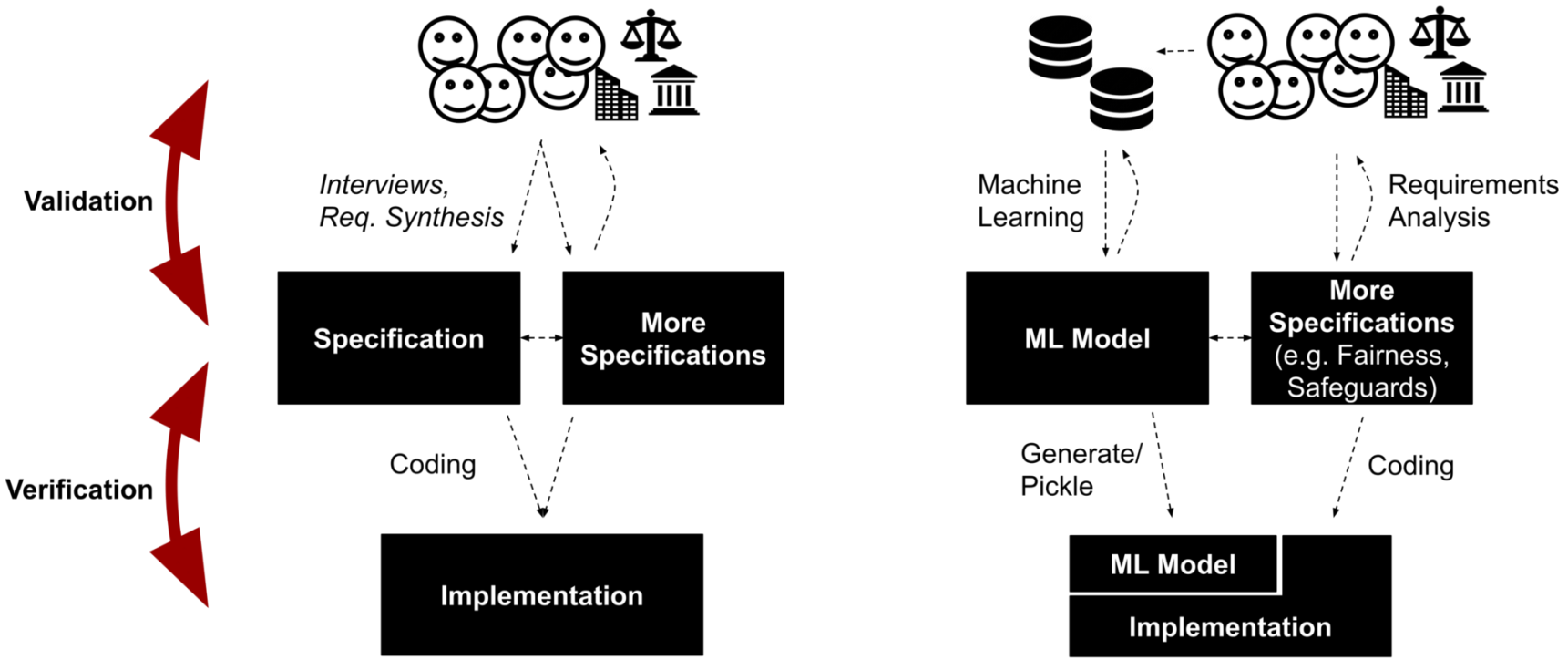
ON TESTING WITH INVARIANTS/ASSERTIONS

- 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:

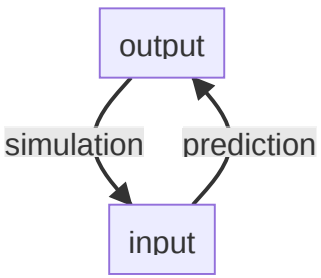
```
assertEquals(??, factorPrime(15485862));
```

```
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 prognosis, 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.

PRELIMINARY SUMMARY: INVARIANTS AND GENERATION

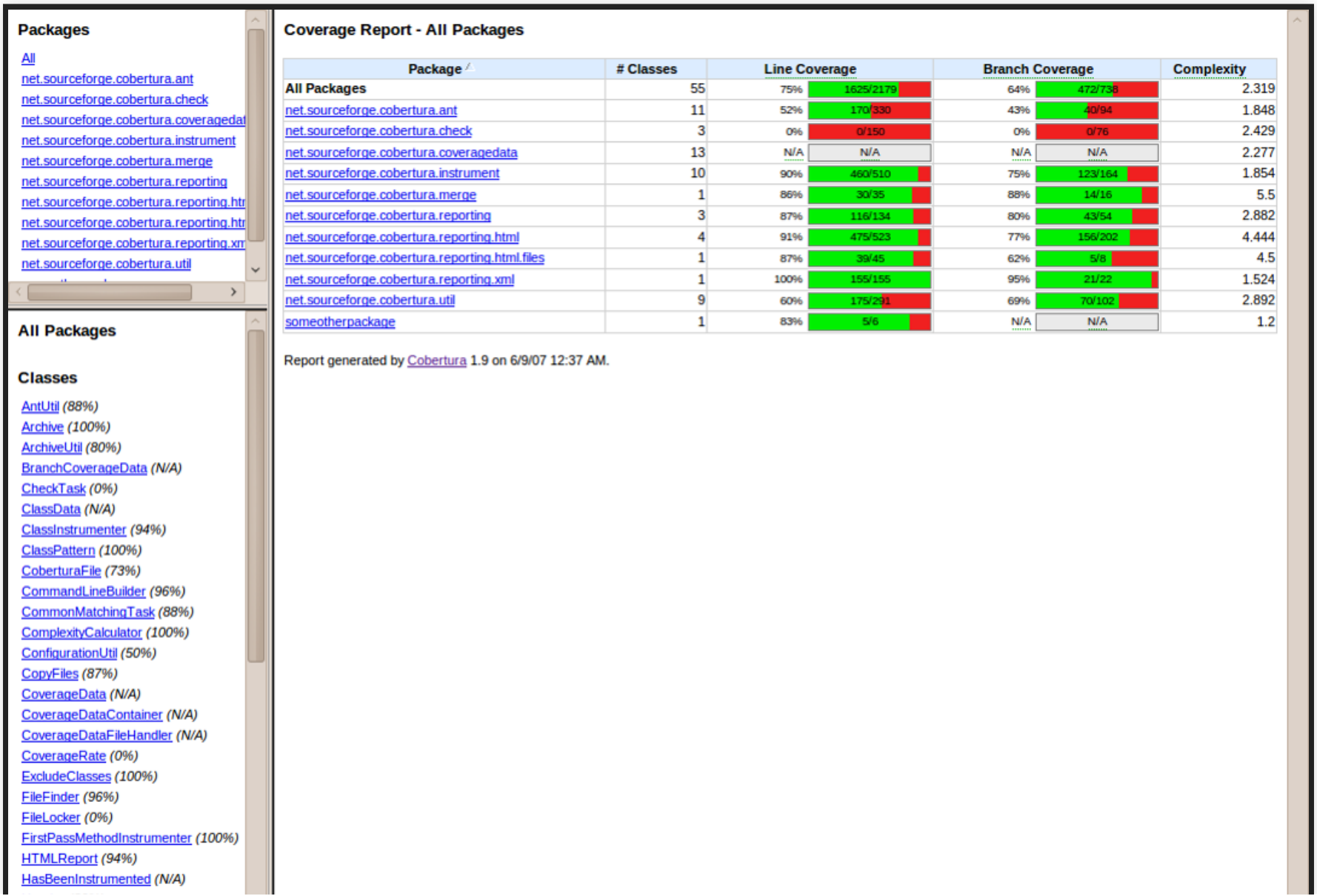
- Generating sample inputs is easy, but knowing corresponding outputs is not (oracle problem)
- Crashing bugs are not a concern
- Invariants + generated data can check capabilities or properties (metamorphic testing)
    - Inputs can be generated realistically or to find violations (adversarial learning)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

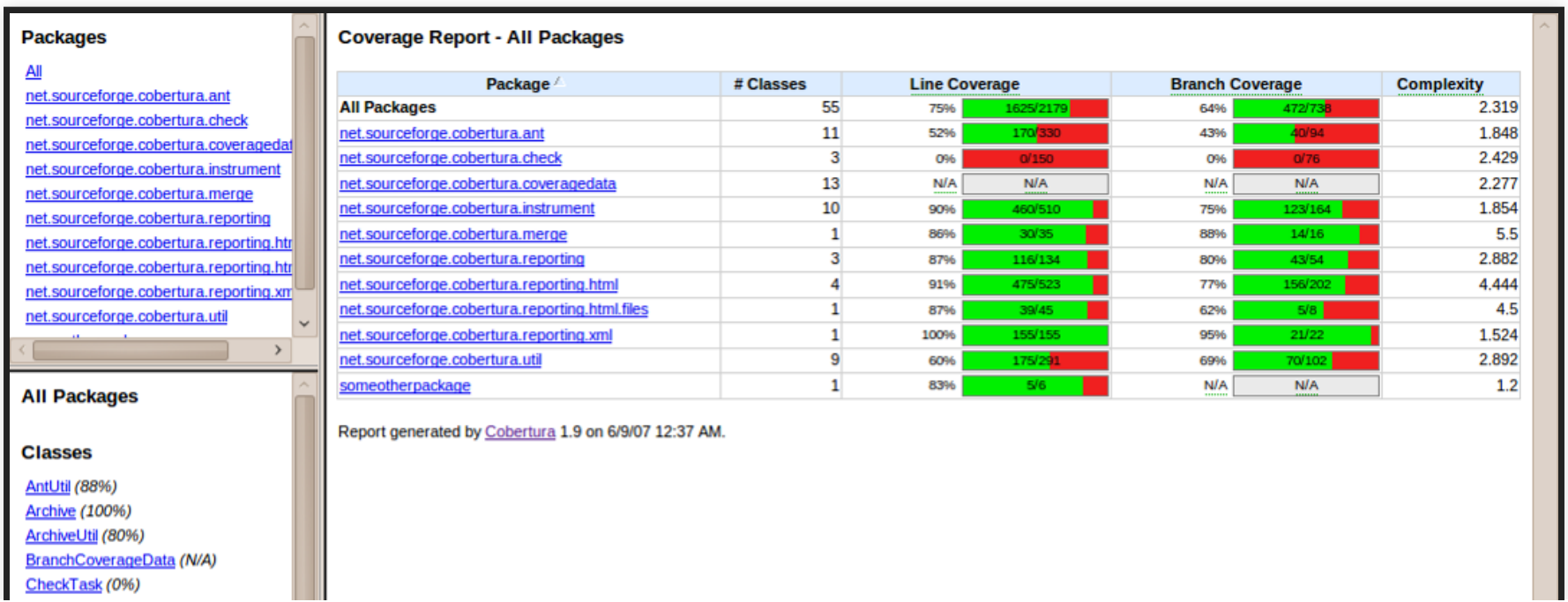
Report generated by Cobertura 1.9 on 6/9/07 12:37 AM.


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

EXAMPLE: STRUCTURAL 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.htr
net.sourceforge.cobertura.reporting.htr
net.sourceforge.cobertura.reporting.xm
net.sourceforge.cobertura.util

All Packages

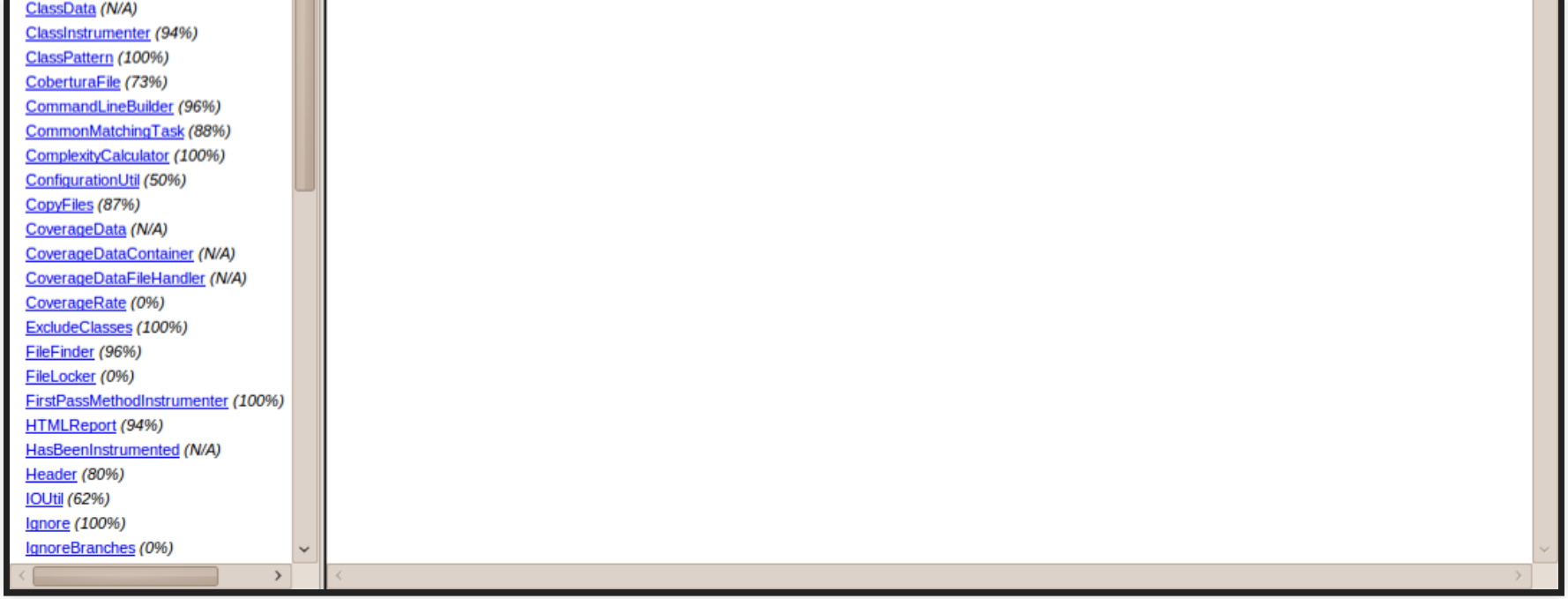
Classes

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

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.


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

DEFINING STRUCTURAL TESTING ("WHITE BOX")

- 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")

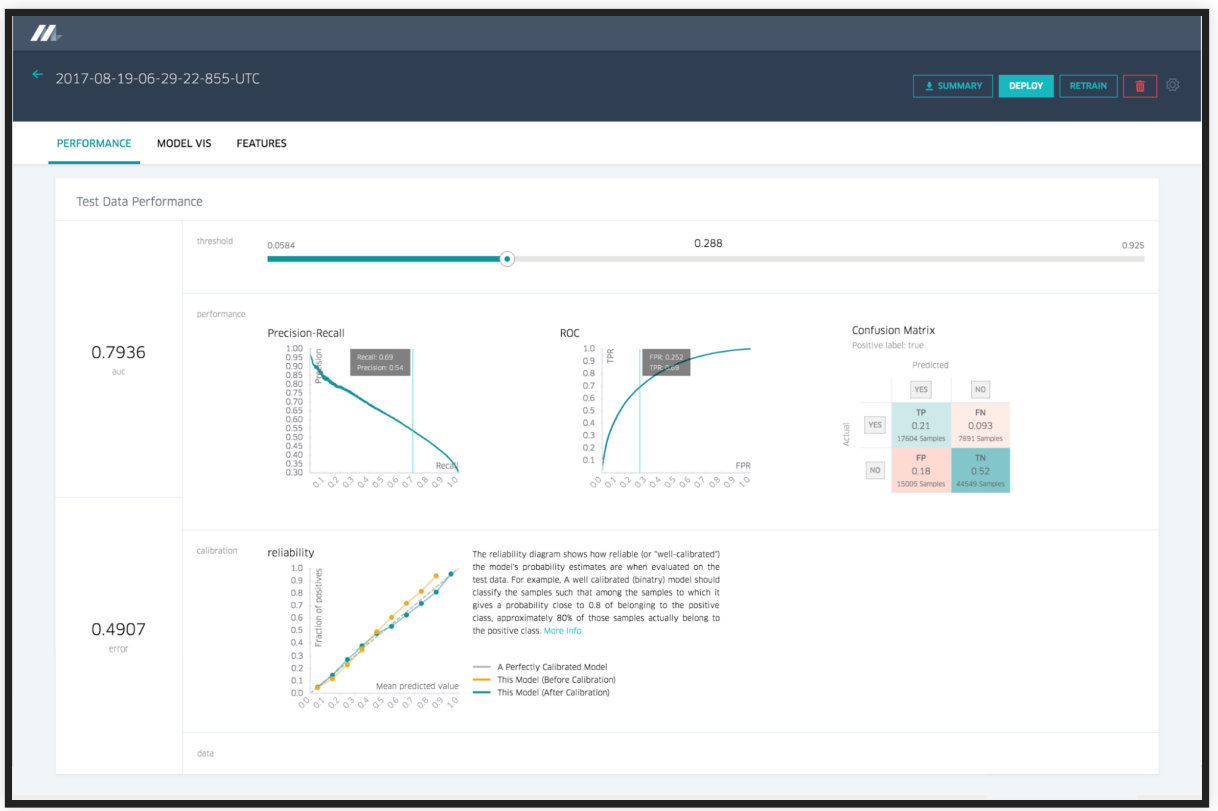
```
int divide(int A, int B) {
  if (A==0)     // A!=0, A<0, B==0
    return 0;   // 1, -1
  if (B==0)     // B!=0, B==1
    return -1;  // 0, -2
  return A / B; // A*B, A+B
}
assert(1, divide(1,1));
assert(0, divide(0,1));
assert(-1, divide(1,0));
```

MUTATION ANALYSIS

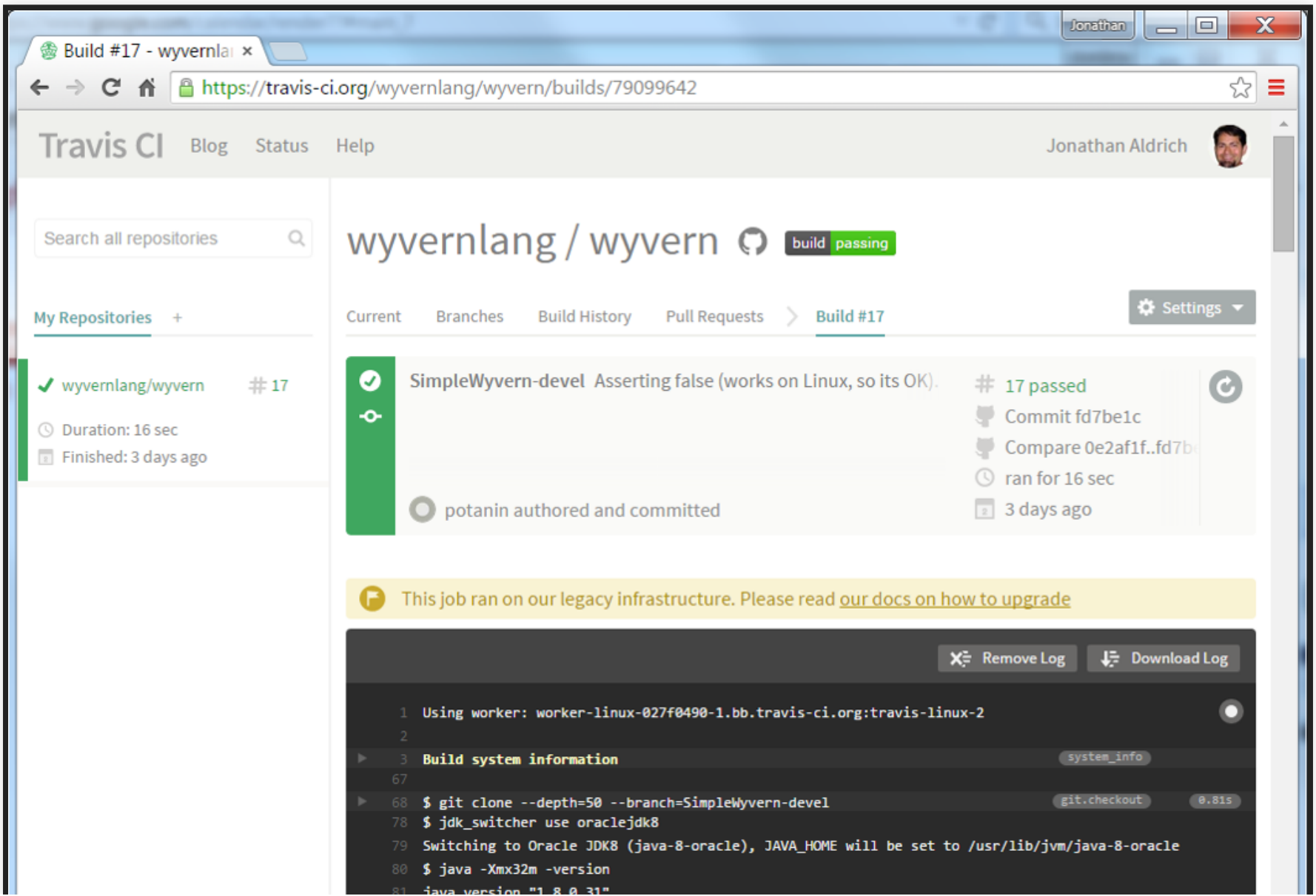
- Some papers exist, but strategy unclear
- Mutating model parameters? Mutating hyperparameters? Mutating inputs?
- What's considered as killing a mutant, if we don't have specifications?

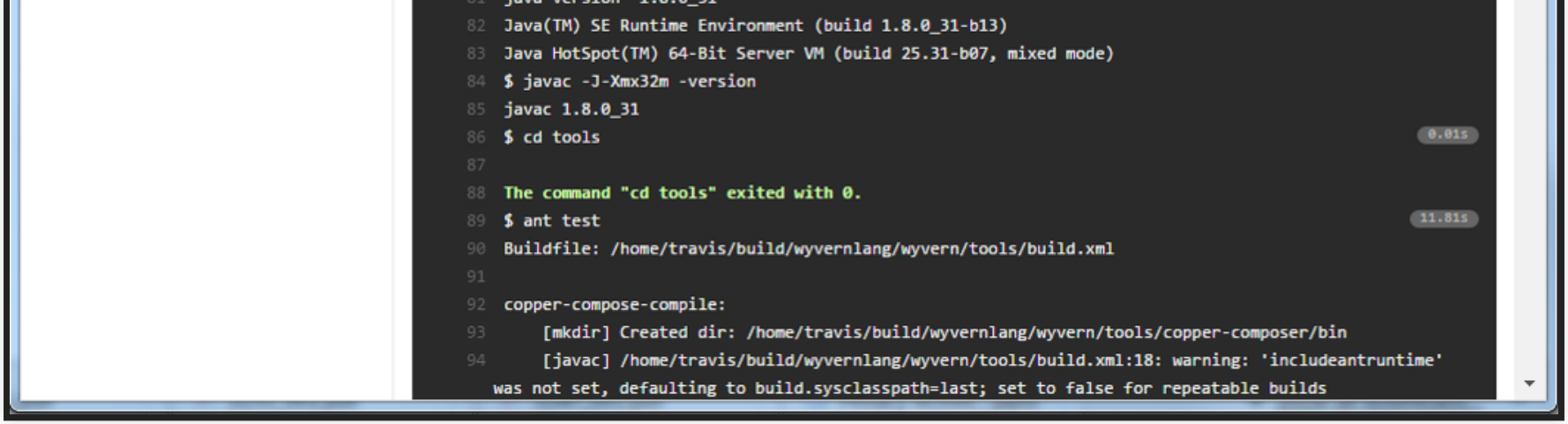
- Still unclear application...

CONTINUOUS INTEGRATION FOR MODEL QUALITY



CONTINUOUS INTEGRATION




Java(TM) SE Runtime Environment (build 1.8.0_31-b13)
Java HotSpot(TM) 64-Bit Server VM (build 25.31-b07, mixed mode)
$ javac -J-Xmx32m -version
javac 1.8.0_31
$ cd tools
0.01s
The command "cd tools" exited with 0.
$ ant test
11.81s
Buildfile: /home/travis/build/wyvernlang/wyvern/tools/build.xml
copper-compose-compile:
[mkdir] Created dir: /home/travis/build/wyvernlang/wyvern/tools/copper-composer/bin
[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

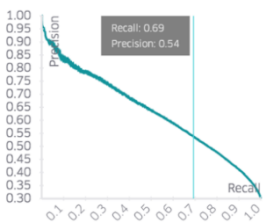
threshold

0.0584 — 0.288 — 0.925

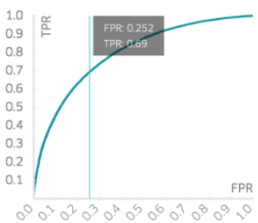
performance

0.7936
auc

Precision-Recall



ROC



Confusion Matrix

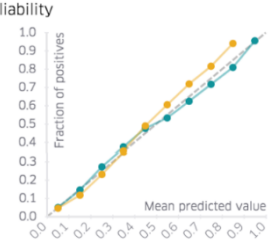
Positive label: true

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

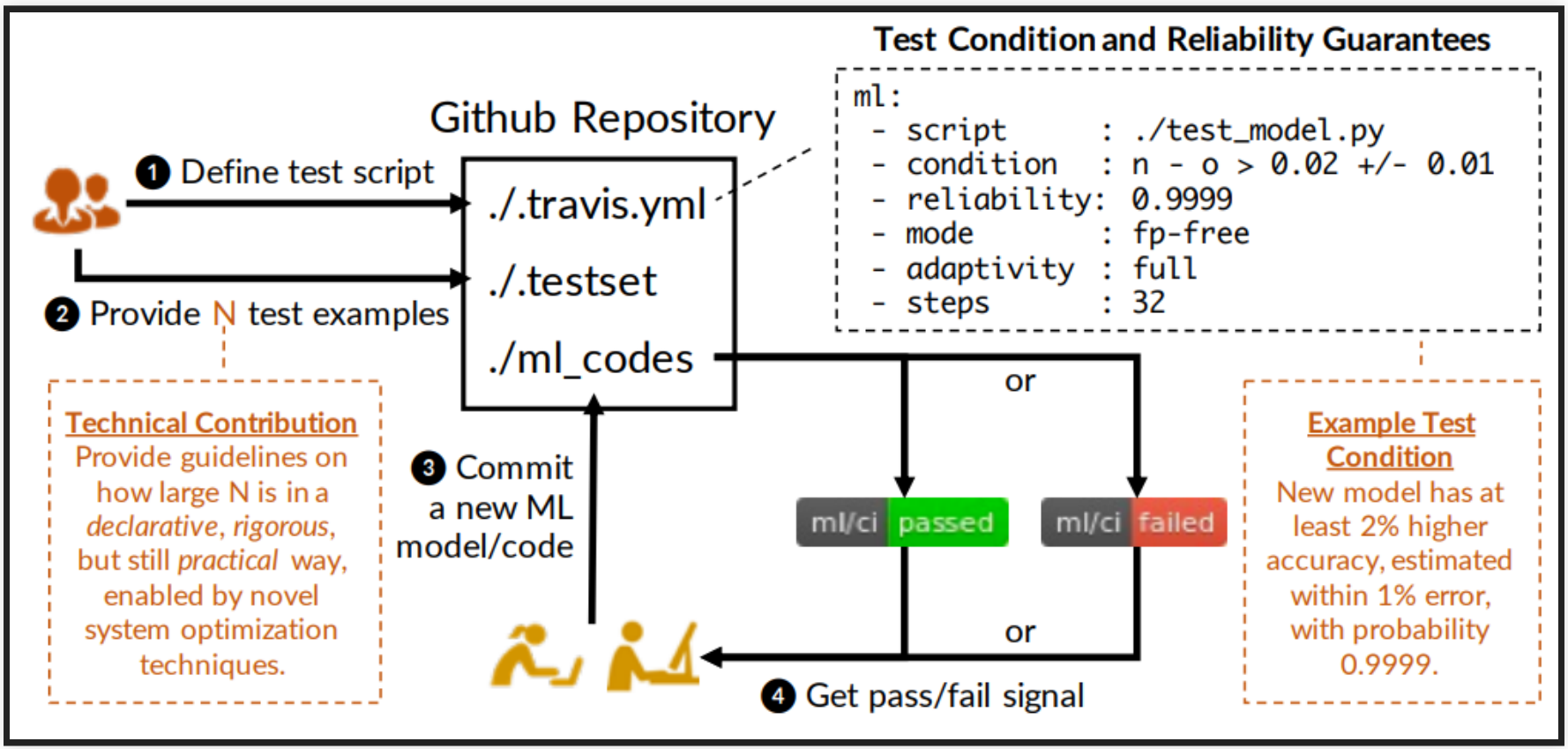


The reliability diagram shows how reliable (or "well-calibrated") the model's probability estimates are when evaluated on the test data. For example, A well calibrated (binatry) model should classify the samples such that among the samples to which it gives a probability close to 0.8 of belonging to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

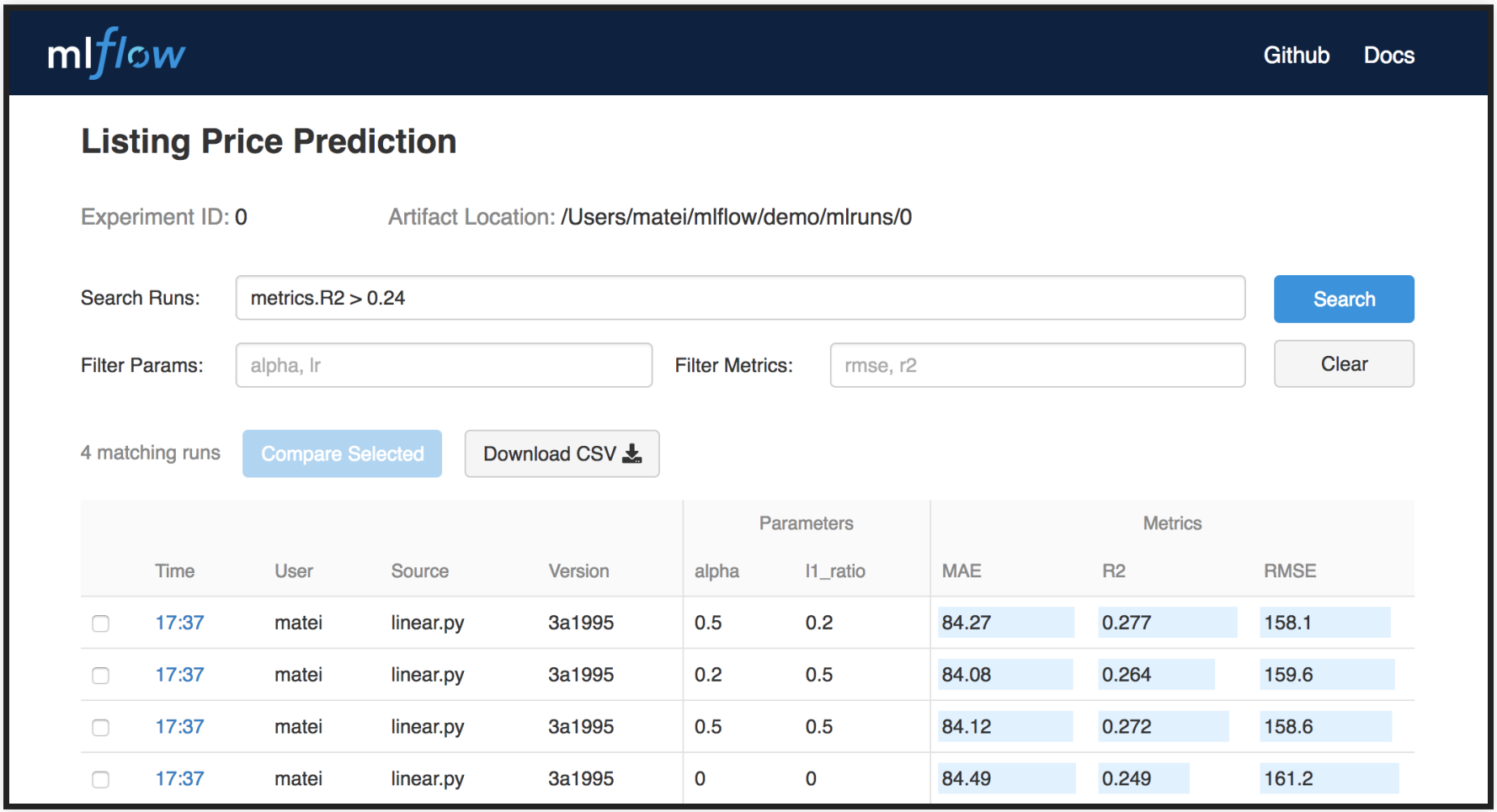
data

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

- Curating test data
  - Analyzing specifications, capabilities
  - Not all inputs are equal: Identify important inputs (inspiration from specification-based testing)
  - Slice data for evaluation
  - Identifying capabilities and generating relevant tests
- 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.
- Ashmore, Rob, Radu Calinescu, and Colin Paterson. "Assuring the machine learning lifecycle: Desiderata, methods, and challenges." arXiv preprint arXiv:1905.04223. 2019.
- Christian Kaestner. Rediscovering Unit Testing: Testing Capabilities of ML Models. Toward Data Science, 2021.
- 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.