Applications of Machine Learning in Software Testing

Lionel C. Briand
Simula Research Laboratory
and
University of Oslo
Acknowledgments

- Yvan Labiche
- Xutao Liu
- Zaheer Bawar
- Kambiz Frounchi
Motivations

• There are many examples of ML applications in the testing literature, but not always where it could be the most useful or practical

• Limited usage of ML in commercial testing tools and practice

• Application of ML in testing has not reached its full potential

• Examples: Applications of machine learning for supporting test specifications, test oracles, and debugging

• General conclusions from these experiences
Black-box Test Specifications

- Context: Black-box, specification testing
- Black-box, specification testing is the most common practice for large components, subsystems, and systems. But it is error-prone.
- Learning objective: relationships between inputs & execution conditions and outputs
- Usage: detect anomalies in black-box test specifications, iterative improvement
- User’s role: define/refine categories and choices (Category-partition)
- Just learning from traces is unlikely to be practical in many situations: Exploit test specifications
Iterative Improvement Process

1. Generate Abstract Test Suite
2. C4.5 Decision Tree
3. Analysis of Decision Tree (DT)
4. Update Test Suite
5. Update Category-Partition

Activities:
- Automated activity
- Partially automated activity
- Manual activity (with heuristic support)
Abstract Test Cases

• Using Category and choices to derive abstract test cases
  – Categories (e.g., triangle side $s_1 = s_2$), choices (e.g., true/false)
  – CP definitions must be sufficiently precise
  – $(1,2,2) \Rightarrow (s_1 \neq s_2, s_2 = s_3, s_1 \neq s_3)$
  – Output equivalence class: Isosceles, etc.
  – Abstract test cases make important properties of test cases explicit
  – Facilitate learning
Examples with Triangle Program

1. \( (a \ vs. \ b) = a!\neq b \)
2. \( | \ (c \ vs. \ a+b) = c\leq a+b \)
3. \( | \quad | \ (a \ vs. \ b+c) = a\leq b+c \)
4. \( | \quad | \quad | \ (b \ vs. \ a+c) = b\leq a+c \)
5. \( | \quad | \quad | \ | \ (b \ vs. \ c) = b=c \)
6. \( | \quad | \quad | \ | \ | \ (a) = a>0: \text{Isosceles} \ (22.0) \)

Example 1:
(a vs. b) = a=b
| (b vs. c) = b!=c
| | (a vs. b+c) = a<=b+c
| | | (c vs. a+b) = c<=a+b: Isosceles (24.0/2.0)

Example 2:
(a vs. b) = a=b
| (c) = c>0
| | (b vs. c) = b!=c: Isosceles (24.0/2.0)

Examples of Detected Problems: Misclassifications
Example: ill-defined Choices

• Ill-defined choices make render a category a poor predictor of output equivalence classes

• Example: Category (c vs. a+b)
  
  \[ c < a + b \quad (\text{should be } <=) \]
  
  \[ c >= a + b \quad (\text{should be } >) \]

• Misclassifications where c = a+b
Linking Problems to Potential Causes

<table>
<thead>
<tr>
<th>Problems</th>
<th>Causes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Missclassifications</td>
<td>Missing Category</td>
</tr>
<tr>
<td>Too Many Test Cases for a Rule</td>
<td>Ill-defined Choices</td>
</tr>
<tr>
<td>Unused Categories</td>
<td>Missing Test Cases</td>
</tr>
<tr>
<td>Missing Combinations of Choices</td>
<td>Redundant Test Cases</td>
</tr>
<tr>
<td></td>
<td>Useless Categories</td>
</tr>
<tr>
<td></td>
<td>Impossible Combinations of Choices</td>
</tr>
</tbody>
</table>

Missed classifications
Too Many Test Cases for a Rule
Unused Categories
Missing Combinations of Choices

Missing Category
Ill-defined Choices
Missing Test Cases
Redundant Test Cases
Useless Categories
Impossible Combinations of Choices
Case Study: Summary of Results

- Experiments with students defining and refining test case specifications using category partition
- Taxonomies of decision tree problems and causes complete
- Student achieved a good CP specification in two or three iterations
- Reasonable increase in test cases led to a significant number of additional faults.
- Our heuristic to remove redundant test cases leads to significant reduction in test suite size (~50%), but a small reduction in the number of faults detected may also be observed.
Test Oracles

- Context: Iterative development and testing, no precise test oracles
- Learning objectives: Model expert knowledge in terms of output correctness and similarity
- Usage: avoid expensive (automate) re-testing of previously successful test cases (segmentations)
- User’s role: Expert must help devise a training set to feed the ML algorithm.
- Example is image segmentation algorithms for heart ventricles
Heart Ventricle Segmentation
Iterative Development of Segmentation Algorithms
Study

• Many (imperfect) similarity measures between segmentations in the literature

• Oracle: Are two segmentations of the same image similar enough to be confidently considered equivalent or consistent?
  – Vi Correct & Vi+1 consistent => Vi+1 correct
  – Vi Correct & Vi+1 inconsistent => Vi+1 incorrect
  – Vi Incorrect & Vi+1 consistent => Vi+1 incorrect

• Machine learning uses training set of instances where that question was answered by experts + similarity measures
Classification Tree Predicting Consistency of Segmentations

All Rows
- Count: 215
- $G^2$: 297.49025
- LogWorth: 41.362539

Similarity measures

DSC_MF < 0.84987577
- Count: 79
- $G^2$: 42.464569
- LogWorth: 4.5957099

DSC_MF ≥ 0.84987577
- Count: 136
- $G^2$: 140.9536
- LogWorth: 14.275805

ANVD > 0.2903414
- Count: 63
- $G^2$: 0

ANVD < 0.2903414
- Count: 16
- $G^2$: 21.170024

AVD > 9.169
- Count: 47
- $G^2$: 64.622912

AVD < 9.169
- Count: 89
- $G^2$: 26.237865

Consistency
Results

• Three similarity measures selected
• Cross-validation ROC area: 94%
• For roughly 75% of comparisons, the decision tree can be trusted with a high level of confidence
• For 25% of comparisons, the expert will probably have to perform manual checks
• More similarity measures to consider
• Similar results with other rule generation algorithms (PART, Ripper)
Fault Localization (Debugging)

- Context: Black-box, specification testing
- Learning objective: relationships between inputs & execution conditions and failure occurrences
- Usage: Learn about failure conditions, refine statement ranking techniques in the presence of multiple faults
- User’s role: define categories and choices (Category-partition)
- Techniques ranking statements are unlikely to be of sufficient help for debugging
- Still need to address the case of multiple faults (failures caused by different faults)
- Failure conditions must be characterized in an easily understood form
Generating Rules - Test case classification

• Using C4.5 to analyze abstract test cases
  – A failing rule generated by the C4.5 models a possible condition of failure
  – Failing test cases associated with a same C4.5 rule (similar conditions) are likely to fail due to the same faults

Rule: \( s_1 = s_2 \) and \( s_3 = s_1 \)

1. \( s_1 = s_2 \)
2. \( s_2 = s_3 \)
3. \( s_3 = s_1 \)
4. Fail
5. Pass
6. Pass
7. Pass
Accuracy of Fail Rules (Space)

<table>
<thead>
<tr>
<th>Predicted</th>
<th></th>
<th></th>
<th>Fail Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fail</td>
<td>Pass</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Actual</td>
<td>Fail</td>
<td>Pass</td>
<td></td>
</tr>
<tr>
<td>6045</td>
<td>335</td>
<td>550</td>
<td>6655</td>
</tr>
</tbody>
</table>

- Fail test cases:
  - 92% precision, 95% recall
- Similar for Pass test cases

1. defines a triangular grid of antennas (condition 1),
2. defines a uniform amplitude and phase of the antennas (conditions 2 and 3),
3. defines the triangular grid with angle coordinates or Cartesian coordinates, and a value is missing when providing the coordinates (conditions 4 and 5);

Table 1 Example Fail rules with Fail probabilities and test case coverage

<table>
<thead>
<tr>
<th>Rules</th>
<th>length</th>
<th>TC</th>
<th>Fail Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>F109</td>
<td>1</td>
<td>30</td>
<td>100.00%</td>
</tr>
<tr>
<td>F025</td>
<td>5</td>
<td>667</td>
<td>99.55%</td>
</tr>
<tr>
<td>F104</td>
<td>2</td>
<td>240</td>
<td>98.75%</td>
</tr>
<tr>
<td>F106</td>
<td>1</td>
<td>304</td>
<td>98.36%</td>
</tr>
<tr>
<td>F105</td>
<td>2</td>
<td>158</td>
<td>97.47%</td>
</tr>
<tr>
<td>F107</td>
<td>1</td>
<td>114</td>
<td>97.37%</td>
</tr>
<tr>
<td>F009</td>
<td>12</td>
<td>147</td>
<td>97.28%</td>
</tr>
<tr>
<td>F010</td>
<td>12</td>
<td>132</td>
<td>96.97%</td>
</tr>
<tr>
<td>F003</td>
<td>12</td>
<td>37</td>
<td>94.59%</td>
</tr>
<tr>
<td>F004</td>
<td>11</td>
<td>200</td>
<td>94.50%</td>
</tr>
<tr>
<td>F012</td>
<td>11</td>
<td>97</td>
<td>93.81%</td>
</tr>
<tr>
<td>F005</td>
<td>11</td>
<td>548</td>
<td>93.80%</td>
</tr>
<tr>
<td>F070</td>
<td>10</td>
<td>271</td>
<td>93.73%</td>
</tr>
<tr>
<td>F069</td>
<td>9</td>
<td>116</td>
<td>92.24%</td>
</tr>
<tr>
<td>F087</td>
<td>9</td>
<td>199</td>
<td>91.96%</td>
</tr>
<tr>
<td>F016</td>
<td>7</td>
<td>12</td>
<td>91.67%</td>
</tr>
<tr>
<td>F086</td>
<td>8</td>
<td>700</td>
<td>91.57%</td>
</tr>
</tbody>
</table>
Statement ranking strategy

- Select high accuracy rules based on a sufficiently large number of (abstract) test cases.

- Consider test cases in each rule separately.

- In each test case set matching a failing rule, the more test cases executing a statement, the more suspicious it is, and the smaller its weight: \( \text{Weight}(R_i, s) \in [-1 \ 0] \)

- For passing rules, the more test cases executing a statement, the safer it is: \( \text{Weight}(R_i, s) \in [0 \ 1] \)

\[
\text{Weight}(s) = \sum_{R_i \in R} \text{Weight}(R_i, s)
\]
Statement Ranking: Space

- Scenario: for each iteration, fix all the faults in reachable statements
Case studies: summary

• RUBAR more effective than Tarantula at ranking faulty statements thanks to the C4.5 classification rules

• The generated C4.5 classification rules based on CP choices characterizing failure conditions accurately predict failures

• Experiments with human debuggers are needed to assess the cost-effectiveness of the approach
Lessons Learned

- In all considered applications, it is difficult to imagine how the problem could have been solved without human input, e.g., categories and choices.

- Machine learning has shown to help decision making -- but it does not help fully automate solutions to the test specification, oracle, and fault localization problems.

- Search for full automation is often counter-productive: It leads to impractical solutions.

- Important question: What is best handled/decided by the expert and what is best automated (through ML algorithms)?

- Solutions that best combine human expertise and automated support.
References


Questions
RUBAR iterative debugging process

System under test → Execution/Coverage Analysis

(2) Program slice by TC

Test suite → Test case transformation

(1) Abstract test suite

Category Partition definition

Rule generation

C4.5 rules

RUBAR algorithm

Statement ranking

(4) Fault removing

(5) Fault removing strategy

Test result