Applying Classification Techniques to Remotely-Collected Program Execution Data

Alessandro Orso
Georgia Institute of Technology

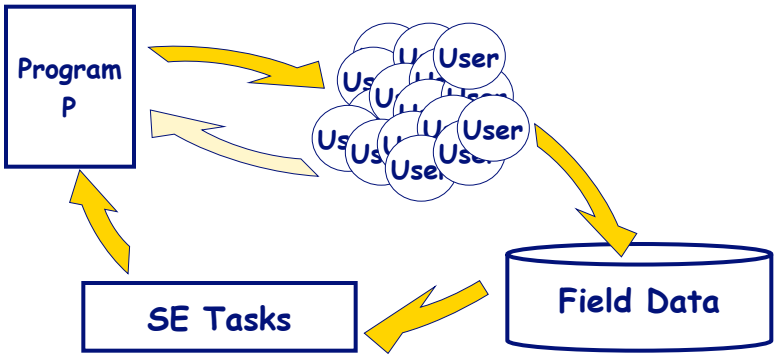
Murali Haran
Penn State University

Alan Karr, Ashish Sanil
National Institute of Statistical Sciences

Adam Porter
University of Maryland

This work was supported in part by NSF awards CCF-0205118 to NISS, CCR-0098158 and CCR-0205265 to University of Maryland, and CCR-0205422, CCR-0306372, and CCR-0209322 to Georgia Tech.

Testing & Analysis after Deployment



[Pavlopoulou99]	Test adequacy	Residual coverage data
[Hilbert00]	Usability testing	GUI interactions
[Dickinson01]	Failure classification	Caller/callee profiles
[Bowring02]	Coverage analysis	Partial coverage data
[Orso03]	Impact analysis	Dynamic slices
[Liblit05]	Fault localization	Various profiles (returns, …)
…	…	…

Tradeoffs of T&A after Deployment

- In-house
  - (+) Complete control (measurements, reruns, …)
  - (-) Small fraction of behaviors
- In the field
  - (+) All (exercised) behaviors
  - (-) Little control
    - Only partial measures, no reruns, …
    - In particular, no oracles
    - Currently, mostly crashes

Our Goal

Provide a technique for automatically identifying failures

- Mainly, in the field
- Useful in-house too
  - Automatically generated test cases

Overview

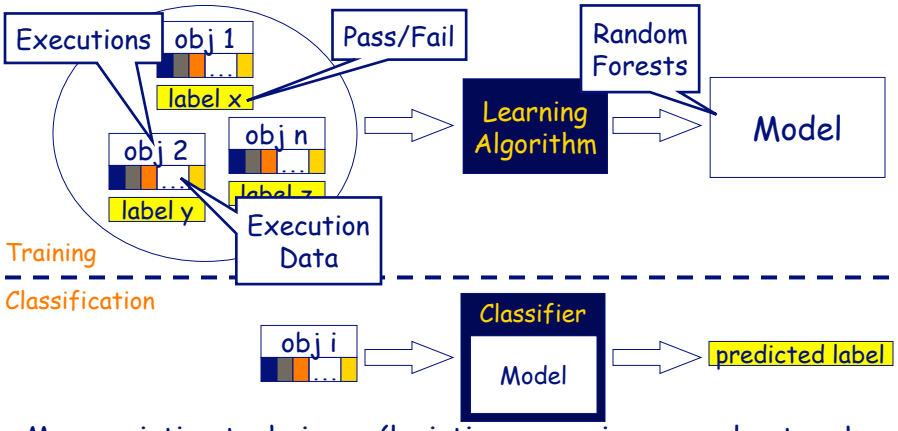
- Motivation and Goal
- General Approach
- Empirical Studies
- Conclusion and Future Work

Overview

- Motivation and Goal
- General Approach
- Empirical Studies
- Conclusion and Future Work

Background: Classification Techniques

Classification -> Supervised learning -> Machine learning



Many existing techniques (logistic regression, neural networks, tree-based classifiers, SVM, …)

Background: Random Forests Classifiers

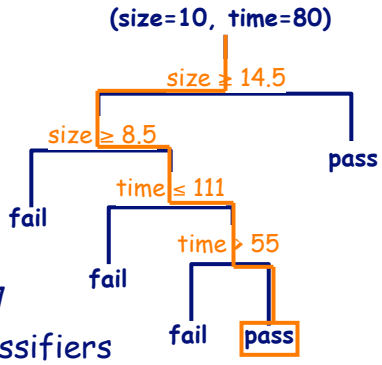
- *Tree-based classifiers*
  - Partition predictor space in hyper-rectangular regions
  - Regions are assigned a label

  (+) Easy to interpret

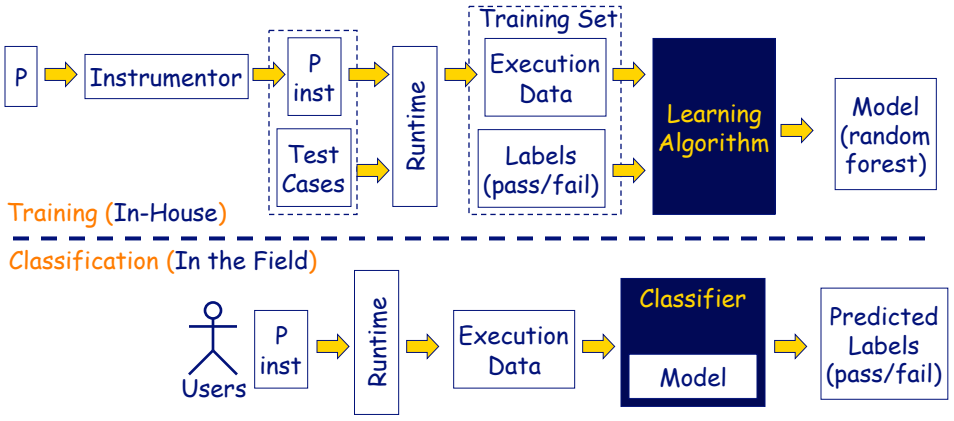
  (-) Unstable
- *Random forests [Breiman01]*
  - Integrate many (500) tree classifiers
  - Classification via a voting scheme

  (+) Easy to interpret

  (+) Stable



Our Approach



Some critical open issues

- What data should we collect?
- What tradeoffs exist between different types of data?
- How reliable/generalizable are the statistical analyses?

Specific Research Questions

RQ1: Can we reliably classify program outcomes using execution data?

RQ2: If so, what type of execution data should we collect?

RQ3: How can we reduce runtime data collection overhead while still producing accurate and reliable classifications?

⇒ Set of exploratory studies

Overview

- Motivation and Goal
- General Approach
- Empirical Studies
- Conclusion and Future Work

Experimental Setup (I)

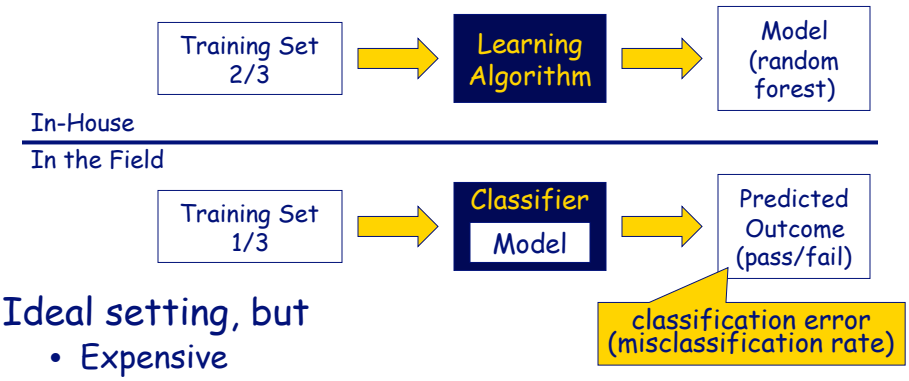
Subject program

- JABA bytecode analysis library
- 60 KLOC, 400 classes, 3000 methods
- 19 single-fault versions ("golden version" + 1 real fault)

Training set

- 707 test cases (7 drivers applied to 101 input programs)
- Collected various kinds of execution data (e.g., counts for throws, catch blocks, basic blocks, branches, methods, call edges, …)
- "Golden version" to label passing/failing runs

Experimental Setup (II)



Ideal setting, but

- Expensive
- Difficult to get enough data points
- Oracle problem

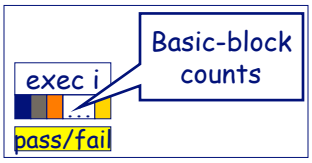
=> Simulate users' runs

RQ1 & RQ2: Can We Classify at All? How?

- RQ1: Can we reliably classify program outcomes using execution data?
- RQ2: Assuming we can classify program outcomes, what type of execution data should we collect?

- We first considered a specific kind of execution data: basic-block counts (~20K)
  (simple measure, intuitively related to faults)
- Results: classification error estimates always almost 0!
- But, time overheard ~15% and data volume not negligible

=> Other kinds of execution data



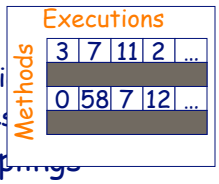
RQ1 & RQ2: Can We Classify at All? How?

- We considered other kinds of execution data:
  - Basic-block counts yielded almost perfect predictors => richer data not considered
  - Counts for: throws, catch-blocks, methods, and call-edges
- Results
  - Throw and catch-block counts are poor predictors
  - Method counts produced nearly perfect models
    - As accurate as block counts, but much cheaper to collect
    - 3000 methods vs. 20000 blocks (overhead < 5%)
  - Branch and call-edge counts equally accurate, but more costly than method counts

Preliminary conclusion (1): Possible to classify program runs; method counts provided high accuracy at low cost

RQ3: Can We Collect Less Information?

- Method-count models used between 2 and 7 method counts. Great for instrumentation, but…
- Two alternative hypotheses
  - Few methods are relevant -> must choose specifi
  - Many, redundant methods -> method selection le
- To investigate, performed 100 random samp
  - Took 10% random samples of method counts and rebuilt models
  - Models were excellent 90% of the times
  - Evidence that many method counts are good predictors



Preliminary conclusion (2): "failure signal" spread, rather than localized to single entities => estimates can be based on few measurements, collected with low overhead

Validity of the Analysis

Two main issues to consider

- Multiplicity
- Generality

Statistical Issues -- Multiplicity

When # of predictors far exceeds # of data points, the likelihood of finding spurious relationship increases

- i.e., random relationships confused for real ones

We took two steps to address the problem

- Consider method counts (least number of predictors)
- Conducted study in which we
  - Randomly permuted method counts
  - Took a 10% random sample of method counts and rebuilt models (100 times)

  => Never found good models based on this data

Methods \ Executions					
	3	7	11	2	…
	69	8	4	21	…
	0	58	7	12	…
	33	76	0	3	…

Preliminary conclusion (3): Results were unlikely to be due to random chance

Statistical Issues -- Generality

Classifiers for 1 specific bug are useful, but…

- We would like to have models that encode "correct behavior" for the application in general
- Looked for predictors that worked in general

⇒Found 11 excellent predictors for all versions

Programs typically contain more than 1 bug

- Applied our approach to 6 multi-bug versions
- Models had error rates less than 2% in most cases

Preliminary conclusion (4): Results promising w.r.t. generality (but need to investigate further)

Overview

- Motivation and Goal
- General Approach
- Empirical Studies
- Conclusion and Future Work

Summary

- Possible to classify program outcomes using execution data
- Method counts gave high accuracy at low cost
- Estimates can be computed based on very few data, collected with negligible overhead
- Our results are unlikely to depend on random chance and are promising in terms of generality
- **But**, these are still preliminary results, and we need to investigate further

Future Work

- Multiple faults
- Investigate relationship between predictors and failures
- Investigate relationship between predictors and faults
- Conduct further experiments with system(s) in actual use