On the Validation of Statistical Software

Ryan Lekivetz
Advanced Analytics Manager
JMP Statistical Discovery

Outline

- What is Software Testing?
- The Testing Challenge
- Test Selection as a Designed Experiment
- Covering Arrays

What is software testing?

- To show the software works as intended.
- To ensure there are no bugs in the software.
- Assures us the software does what it is supposed to do.

What is software testing?

- ~~To show the software works as intended.~~
- ~~To make sure there are no bugs in the software.~~
- ~~Assures us the software does what it is supposed to do.~~

These approach the problem as showing the software works, leading us down this path.

Consider

- “Can you check that this works?”

Vs.

- “Try and break it.”

What is software testing?

"Testing is the process of executing a program with the intent of finding errors."

G. Myers, The Art of Software Testing, Wiley, 1979

Where are the bugs?

“Bugs lurk in corners and congregate at boundaries.”

B. Beizer, Software Testing Techniques, Van Nostrand Reinhold, 1983

The testing challenge

1. Selection problem: How do you select test cases from the input space of the system so that the chance of finding faults, while staying within budget, is maximized?
2. Quality problem: Can you make informed assertions about “fitness for use” as testing unfolds?
3. Oracle problem: How do you determine expected behavior for each test case?

JMP STATISTICAL DISCOVERY

The oracle problem

How to determine if a test outcome is what was expected

- Discussion of statistical software usually focused on numerical accuracy
- We often want to go further than this
- How to determine the expected outcome?
- What does it mean if the results change?

JMP STATISTICAL DISCOVERY

The oracle problem

What is the expected result?

- Software crash
- Timing
- Visualization
- Set of statistics
- Numerics (how close is good enough?)
- And so on…

- May not be existing packages that the test engineer can use

The testing challenge

1. Selection problem: How do you select test cases from the input space of the system so that the chance of finding faults, while staying within budget, is maximized?
2. Quality problem: Can you make informed assertions about "fitness for use" as testing unfolds?
3. Oracle problem: How do you determine expected behavior for each test case?

JMP STATISTICAL DISCOVERY

Common techniques

Repeated running of unit tests as a regression test suite.

Unit Testing

Develop test cases for validating the smallest testable component (i.e., a unit) of a software package before focusing on the overall integrated system.

Regression Testing

Ensure the software still works as intended after a change.

Test selection via Design of Experiments (DOE)

- Given a set of inputs, how to test these effectively and efficiently
- Deterministic*
- Pass/Fail - based on oracle
- Fault detection, rather than model fitting
  - Looking for failure-inducing combinations
- What is a good design (test suite)?
  - “Bugs lurk in corners and congregate at boundaries.”

jmp STATISTICAL DISCOVERY

Fundamental principles of factorial effects

Wu & Hamada (2011)

Effect hierarchy - *i)* Lower order effects are more likely to be important than higher order effects. *ii)* Effects of the same order are equally likely to be important.

Effect sparsity - The number of relatively important effects will be small.

Effect heredity - An interaction is significant only if at least one of the parent factors involving the interaction is significant.

Fundamental principles of factorial effects

Effect hierarchy - *i)* Lower order effects are more likely to be important than higher order effects. *ii)* Effects of the same order are equally likely to be important.

Effect sparsity - The number of relatively important effects will be small.

Effect heredity - An interaction is significant only if at least one of the parent factors involving the interaction is significant.

- What if we think of failure-inducing combinations as important effects?

Fundamental principles of input combinations

Lekivetz & Morgan (2020)

Combination hierarchy: *i)* Combinations involving fewer inputs are more likely to be failure-inducing than those involving more inputs.

ii) Combinations of the same order are equally likely to be important.

Combination sparsity: The number of failure-inducing combinations will be small.

Combination heredity: A combination is *more likely* failure-inducing if at least one of the parent factors involving the interaction is known to be more likely involved in inducing failures.

Empirical Evidence

#factors involved in failure	Medical devices	Browser	Server	NASA GSFC
1	66	29	42	68
2	97	76	70	93
3	99	95	89	98
4	100	97	96	100
5		99	96	
6		100	100	

Cumulative percentage of faults in software systems triggered by interactions involving number of factors indicated in leftmost column (Kuhn et al., 2004).

STATISTICAL DISCOVERY

Combinatorial testing

"Bugs lurk in corners and congregate at boundaries."

A family of test case selection strategies used to test complex engineered systems. For a complex engineered system with m inputs, such as a software system, a strength t covering array will ensure that all possible combinations of the values for any set of $t \leq m$ inputs will appear in the derived test suite at least once.

- Solves the selection problem and the quality problem
  - A way to select cases and assert what has been tested (all combinations involving up to t inputs) – "pseudo-exhaustive"
  - Moves beyond one-factor-at-a-time (OFAT) - unit tests as the simplest units

Covering Arrays

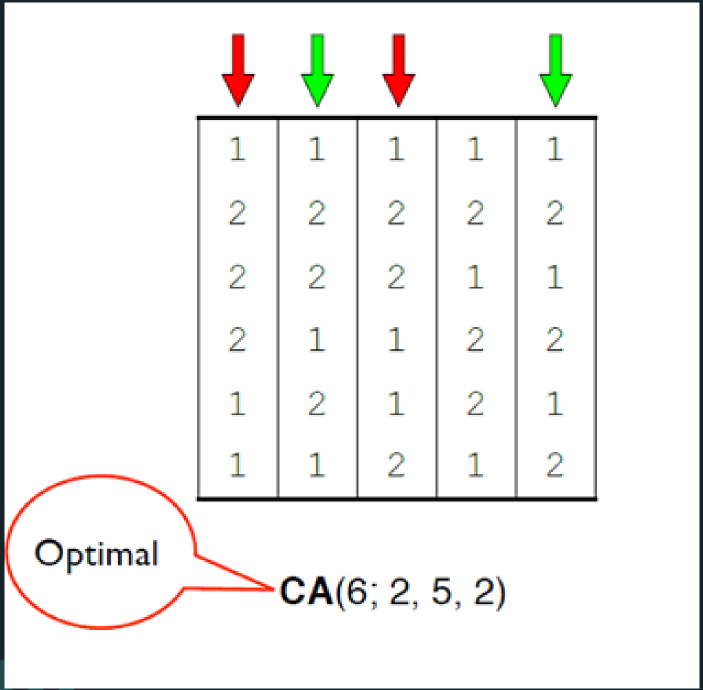
***Covering Arrays*:** For a set of factors, a t-covering array (or strength t) has the property that for any subset of t factors, every possible combination of levels occurs *at least once*.

Orthogonal Arrays: Every possible combination occurs the same number of times.

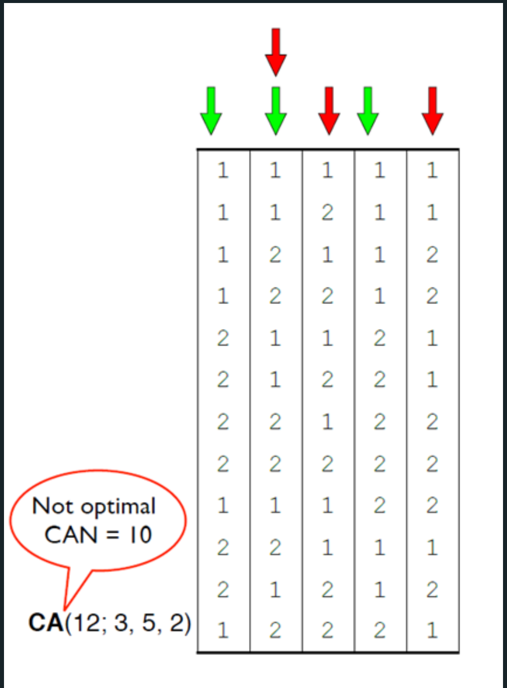
N	4	5	6	7	8	9	10	11	12	13	14
m	3	4	10	15	35	56	126	210	462	792	1716

CA(N; 2, 2^m), m ≤ 1716

Examples



1	1	1	1	1
2	2	2	2	2
2	2	2	1	1
2	1	1	2	2
1	2	1	2	1
1	1	2	1	2



1	1	1	1	1
1	1	2	1	1
1	2	1	1	2
1	2	2	1	2
2	1	1	2	1
2	1	2	2	1
2	2	1	2	2
2	2	2	2	2
1	1	1	2	2
2	2	1	1	1
2	1	2	1	2
1	2	2	2	1

Why Covering Arrays?

- Cost-efficient
- Selection problem – what to test (DOE)
- Quality problem - if all tests pass a strength *t* covering array, can ascertain there are no faults due to *t**-factor combinations *(t*<=t)*
- Disciplined approach to testing
- Another tool in the tool chest
- When do combination sparsity and hierarchy hold?

System under test (SUT)

i.e. the software

Covering Array	SUT
Columns	Inputs
Levels per column	Equivalence classes* per input
Strength (i.e. *t*)	Testing budget
Rows	Test cases

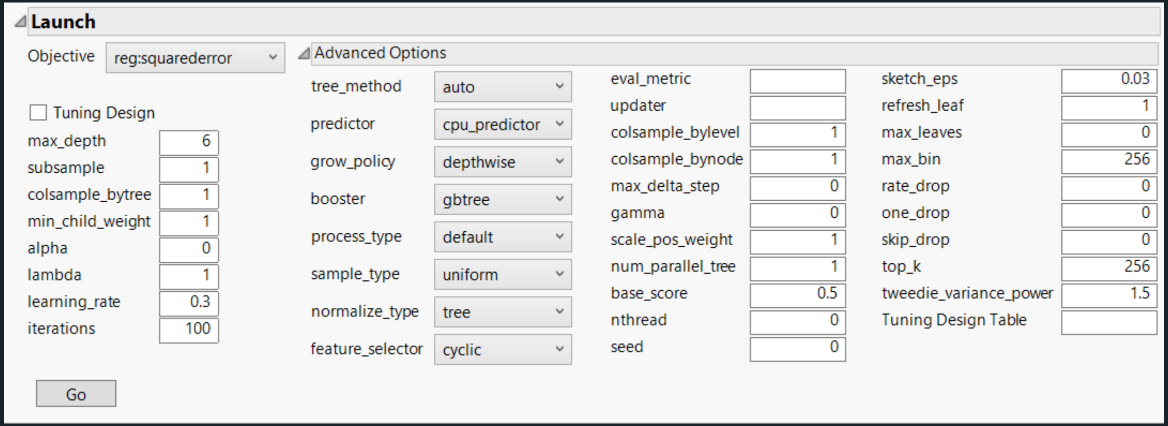
Other Considerations

- Fault localization – find the underlying cause when you have a failure.
  - Not as easy as it seems
- Constraints (disallowed combinations/forbidden edges)
- Locating / Detecting Arrays
- Random seeds (make sure to keep them)
- Random inputs/equivalence partitioning/boundary-value analysis
- Data set generation*
- Software (ACTS, JMP Pro, Hexawise, Testcover.com, etc…)

XGBoost

What is XGBoost?

“An optimized distributed gradient boosting library designed to be highly efficient, flexible and portable.”



T. Chen and C. Guestrin. 2016. XGBoost: A Scalable Tree Boosting System.

jmp STATISTICAL DISCOVERY

XGBoost Selected Inputs

	Hyperparameter	Type
1	Max_depth	*continuous*
2	Subsample	*continuous*
3	Colsample_bytree	*continuous*
4	Min_child_weight	*continuous*
5	Alpha	*continuous*
6	Lambda	*continuous*
7	Learning_rate	*continuous*
8	Iterations	*continuous*
9	Tree_method	*categorical (6)*
10	Predictor	*categorical (2)*
11	Grow_policy	*categorical (2)*
12	Booster	*categorical (3)*
13	Process_type	*categorical (2)*
14	Sample_type	*categorical (2)*
15	Feature_selector	*categorical (4)*
16	Colsample_bylevel	*continuous*
17	Colsample_bynode	*continuous*
18	Max_delta_step	*continuous*

	Hyperparameter	Type
19	Gamma	*continuous*
20	Scale_pos_weight	*continuous*
21	Num_parallel_tree	*continuous*
22	Base_score	*continuous*
23	Nthread	*continuous*
24	Seed	*continuous*
25	Sketch_eps	*continuous*
26	Refresh_leaf	*continuous*
27	Max_leaves	*continuous*
28	Max_bin	*continuous*
29	Rate_drop	*continuous*
30	One_drop	*continuous*
31	Skip_drop	*continuous*
32	Top_k	*continuous*
33	Tweedie_variance_power	*continuous*
34	Normalize_type	*categorical (2)*

JMP STATISTICAL DISCOVERY

XGBoost Selected Inputs

	Input	# of levels
1	Max_depth	3
2	Subsample	3
3	Colsample_bytree	3
4	Min_child_weight	3
5	Alpha	3
6	Lambda	3
7	Learning_rate	3
8	Iterations	3
9	Tree_method	6
10	Predictor	2
11	Grow_policy	2
12	Booster	3
13	Process_type	2
14	Sample_type	2
15	Feature_selector	4
16	Colsample_bylevel	3
17	Colsample_bynode	3
18	Max_delta_step	3

	Input	# of levels
19	Gamma	3
20	Scale_pos_weight	3
21	Num_parallel_tree	3
22	Base_score	3
23	Nthread	3
24	Seed	3
25	Sketch_eps	3
26	Refresh_leaf	2
27	Max_leaves	3
28	Max_bin	3
29	Rate_drop	3
30	One_drop	2
31	Skip_drop	3
32	Top_k	3
33	Tweedie_variance_power	3
34	Normalize_type	2

Results

- Input space: $6*4*3^{25}*2^{7}$ = 2,602,870,608,208,896 points
- Strength 2 CA can be constructed in 25 runs
  - 72% 3-coverage
  - 35% 4-coverage
- Strength 3 CA can be constructed in 150 runs
  - 90% 4-coverage

Thanks!

Ryan.Lekivetz@jmp.com

Lekivetz, R. and Morgan, J., 2021. On the Testing of Statistical Software. *Journal of Statistical Theory and Practice*, *15*(4), pp.1-18.

https://rdcu.be/cv7tv

jmp STATISTICAL DISCOVERY

jmp.com

References

1. B. Beizer, Software Testing Techniques, Van Nostrand Reinhold, 1983.

2. R. Bryce & C. Colbourn, “Prioritized interaction testing for pair-wise coverage with seeding and constraints,” Information & Software Technology, 48(10), pp. 960 – 970, 2006.

3. D. Cohen, S. Dalal, M. Fredman, & G. Patton, “The AETG System: An approach to testing based on Combinatorial Design,” IEEE TSE, 23(7), 1997, pp. 437-444.

4. M. Cohen, M. Dwyer & J. Shi, “Constructing interaction test suites for highly configurable systems in the presence of constraints: A greedy approach,” IEEE TSE, 34(5), 2008, pp. 633-650.

5. C. Colbourn & V. Syrotiuk, “Coverage, location, detection, and measurement,” IEEE 9th International Conference on Software Testing, Verification and Validation Workshops (ICSTW), 2016, pp. 19–25.

6. S. Dalal & C. Mallows, “Factor-covering designs for testing software,” Technometrics, 40(3), 1998, pp. 234-243.

7. G. Demiroz & C. Yilmaz, “Cost-aware combinatorial interaction testing,” Proc. of the International Conference on Advances in System Testing and Validation Lifecycles, 2012, pp. 9–16.

8. I. Dunietz, W. Ehrlich, B. Szablak, C. Mallows, & A. Iannino, “Applying design of experiments to software testing,” Proceedings of the 19th ICSE, New York, 1997, pp. 205-215.

9. L. Ghandehari, Y. Lei, D. Kung, R. Kacker, & R. Kuhn,“Fault localization based on failure inducing combinations,” IEEE 24th International Symposium on Software Reliability Engineering (ISSRE), 2013, pp. 168–177.

10. A. Hartman & L. Raskin, “Problems and algorithms for covering arrays,” Discrete Math, 284(1–3), 2004, pp. 149–156.

11. K. Johnson, & R. Entringer, “Largest induced subgraphs of the n-cube that contain no 4-cycles,” Journal of Combinatorial Theory, Series B, 46(3), 1989, pp. 346-355.

References

12. G. Katona, “Two applications (for search theory and truth functions) of Sperner type theorems,” Periodica Mathematica Hungarica, 3(1-2), 1973, pp. 19-26.

13. D. Kleitman & J. Spencer, “Families of k-independent sets,” Discrete Mathematics, 6(3), 1973, pp. 255-262.

14. R. Lekivetz, & J. Morgan, “On the testing of statistical software,” Journal of Statistical Theory and Practice (2021).

15. R. Lekivetz, & J. Morgan, “Fault localization: Analyzing covering arrays given prior information,” 2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C). IEEE, 2018.

16. R. Lekivetz & J. Morgan, “Combinatorial Testing: Using blocking to assign test cases for validating complex software systems,” Statistical Theory & Related Fields 5.2 (2021).

17. R. Lekivetz, & J. Morgan, “Covering Arrays: Using Prior Information for Construction, Evaluation and to Facilitate Fault Localization,” Journal of Statistical Theory and Practice 14.1 (2020): 7

18. C. King, J. Morgan, & R. Lekivetz. “Design Fractals: A Graphical Method for Evaluating Binary Covering Arrays,” 2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C). IEEE, 2019.

19. J. Morgan, R. Lekivetz, & T. Donnelly. “Covering arrays: Evaluating coverage and diversity in the presence of disallowed combinations,” 2017 IEEE 28th Annual Software Technology Conference (STC). IEEE, 2017.

20. J. Morgan, “Combinatorial Testing: An approach to systems and software testing based on covering arrays,” in Analytic Methods in Systems and Software Testing, eds., F. Ruggeri, R. Kennett, & F. Faltin, Wiley, pp. 131, 2018.

21. J. Morgan, "Combinatorial Testing,” Wiley StatsRef: Statistics Reference Online (2020): pp. 1-10.

22. G. Myers, The Art of Software Testing, Wiley, 1979.

STATISTICAL DISCOVERY