Empirical Comparison of Regression Methods for Variability-Aware Performance Prediction

Pavel Valov
University of Waterloo
200 University Avenue West
Waterloo, ON, Canada
pvalov@gsd.uwaterloo.ca

Jianmei Guo
University of Waterloo
200 University Avenue West
Waterloo, ON, Canada
gjm@gsd.uwaterloo.ca

Krzysztof Czarnecki
University of Waterloo
200 University Avenue West
Waterloo, ON, Canada
kczarnec@gsd.uwaterloo.ca

ABSTRACT

Product line engineering derives product variants by selecting features. Understanding the correlation between feature selection and performance is important for stakeholders to acquire a desirable product variant. We infer such a correlation using four regression methods based on small samples of measured configurations, without additional effort to detect feature interactions. We conduct experiments on six real-world case studies to evaluate the prediction accuracy of the regression methods. A key finding in our empirical study is that one regression method, called Bagging, is identified as the best to make accurate and robust predictions for the studied systems.

1. INTRODUCTION

In product line engineering, product functionalities are abstracted as *features*, and product variants are derived by feature selection. To acquire a desirable product among a set of alternative variants, stakeholders are interested in the performance of the variants. Performance of a product is often subject to a wide variety of influencing factors, with feature selection being one of them.

Understanding the correlation between performance and feature selection is non-trivial. A straightforward approach to reveal the correlation is to measure the performance of all product variants, but it is usually infeasible, due to the exponential number of variants and the potentially high measurement effort (e.g., executing a complex benchmark) [5]. Moreover, quantifying the performance influence of each individual feature is insufficient in most cases, because feature interactions may cause unpredictable performance anomalies [13]. Siegmund et al. [13] overcame this hurdle by detecting performance-relevant feature interactions using specific sampling heuristics that meet different feature-coverage criteria. However, in practice, the product variants that we can measure or that we already have at our disposal may not meet any feature-coverage criterion. Guo et al. [5] addressed this issue by using regression analysis based on small random samples of measured variants. However, they used only one regression technique, called Classification And Regression Trees (CART), and they did not perform systematic comparison with other regression methods.

In this paper, we aim at an empirical comparison of different regression methods for variability-aware performance prediction. We compare CART to three other regression techniques: Bagging [2], Random Forest [3], and Support Vector Machines (SVMs) [17]. We evaluate the prediction accuracy of each regression method based on small random samples of measured program variants. Moreover, we assess the robustness of each regression method when using the best, the average, and the worst parameter settings of the method. To cover the parameter space of each method as evenly as possible, we generate the parameter settings using a state-of-the-art parameter sweep technique, called Sobol sampling [11]. We conduct experiments on a dataset covering six real-world systems, which has been a benchmark used in [13, 5] for variability-aware performance prediction. In summary, we make the following contributions:

- **Methods.** We extend the previous work [5] and use four regression methods, including CART, Bagging, Random Forest, and SVM, for variability-aware performance prediction based on small random samples of measured product variants.
- **Evaluation.** We evaluate the prediction accuracy of each regression method through experiments on six real-world software programs. Moreover, we compare the robustness of the regression methods under the best, the average, and the worst settings of parameters.
- **Findings.** Our empirical results show that Bagging outperforms all other techniques in terms of prediction accuracy and robustness, as it is the most likely to produce the highest prediction accuracy when using the best, the average, and the worst parameter settings, for the studied systems.

Source code and data to reproduce our experiments are available online at https://bitbucket.org/valovp/splc2015.

2. MOTIVATING EXAMPLE

To motivate our work, we use an example from previous work [5], a configurable tool x264 for encoding video streams into the H.264/MPEG-4 AVC format. We consider 16 encoder features of x264, such as encoding with multiple reference frames and parallel encoding on multiple CPUs. The encoding time is used to indicate the performance of x264 in different configurations. A *configuration* represents a program variant with a certain selection of features. This

SPLC 2015, July 20 - 24, 2015, Nashville, TN, USA
© 2015 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ISBN 978-1-4503-3613-0/15/07…$15.00
DOI: http://dx.doi.org/10.1145/2791060.2791069

Table 1: A sample of 16 randomly selected configurations of x264 and corresponding performance measurements (seconds)

Conf.	Features																Perf. (s)
$\mathbf{c}_i$	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	x_{16}	y_i
$\mathbf{c}_1$	1	1	0	0	1	1	0	0	1	0	0	1	1	1	0	0	292
$\mathbf{c}_2$	1	0	0	0	1	1	0	1	1	1	0	0	1	0	1	0	571
$\mathbf{c}_3$	1	1	1	0	0	1	0	0	1	0	1	0	1	0	0	1	681
$\mathbf{c}_4$	1	1	0	1	1	0	0	0	1	0	1	0	1	1	0	0	263
$\mathbf{c}_5$	1	1	1	0	1	1	0	0	1	0	1	0	1	0	1	0	536
$\mathbf{c}_6$	1	0	0	1	0	1	1	1	1	0	1	0	1	1	0	0	305
$\mathbf{c}_7$	1	1	1	1	0	1	1	0	1	0	1	0	1	0	1	0	408
$\mathbf{c}_8$	1	0	0	1	1	0	1	1	1	0	1	0	1	1	0	0	278
$\mathbf{c}_9$	1	0	0	1	1	0	0	1	1	0	0	1	1	0	1	0	519
$\mathbf{c}_{10}$	1	1	0	0	1	1	0	1	1	0	1	0	1	0	0	1	781
$\mathbf{c}_{11}$	1	0	0	1	1	1	0	1	1	1	0	0	1	0	0	1	822
$\mathbf{c}_{12}$	1	1	0	0	1	0	0	0	1	0	1	0	1	0	0	1	713
$\mathbf{c}_{13}$	1	0	1	0	0	0	1	0	1	0	0	1	1	0	1	0	381
$\mathbf{c}_{14}$	1	0	1	1	1	1	0	1	1	1	0	0	1	0	1	0	564
$\mathbf{c}_{15}$	1	1	1	1	1	0	0	0	1	0	0	1	1	0	1	0	489
$\mathbf{c}_{16}$	1	1	1	1	0	1	1	1	1	1	0	0	1	1	0	0	275

example with only 16 features gives rise to 1,152 configurations. In practice, often only a limited set of configurations can be measured, either by simulation or by monitoring in the field. How can we determine the performance of other configurations based on a small random sample of measured configurations?

To formulate the above problem, we represent a feature as a binary decision variable x. Assume that there are N features in total, then a configuration is an N-tuple $\mathbf{c}$, assigning 0 or 1 to each variable. For example, each configuration of x264 is represented by 16-tuple, e.g., $\mathbf{c}_1 = (x_1 = 1, x_2 = 1, ..., x_{16} = 0)$. All valid configurations of a program are denoted by $\mathbf{C}$.

Each configuration $\mathbf{c}$ has an actual measured performance value y. All performance values of all configurations $\mathbf{C}$ form set Y. Suppose that we acquire a random sample of configurations $\mathbf{C}_S \subset \mathbf{C}$ and their actual measured performance values $Y_S \subset Y$, together forming sample S. The problem of variability-aware performance prediction is to predict the performance of other configurations in $\mathbf{C} \backslash \mathbf{C}_S$ based on the measured sample S. In other words, we predict a quantitative response y based on a set of categorical predictors X, which is a typical regression problem [7]. Due to feature interactions [13], the above issue is reduced to a non-linear regression problem, where the response depends non-linearly on one or more predictors [5].

3. REGRESSION METHODS

We compare four regression methods: CART, Bagging, Random Forest and Support Vector Machines (SVMs). *CART* is a classic regression analysis method based on a tree-structure model. It starts with a sample S of measured configurations, and recursively partitions it into smaller sections. Then, all sections are combined into a global prediction model in the form of a binary decision tree, which we call a *regression tree*. *Bagging* [2] enhances CART by generating multiple regression trees and averaging their predictions in order to produce the final result. *Random forest* [3] is another enhanced version of CART. Similar to Bagging, it averages over multiple regression trees. The key difference is that Random Forest considers a randomly selected subset of all features when generating regression trees, while CART and Bagging take all features X into account. *SVM* [17] creates a hyperplane and uses it for approximate regression. SVM is considered to perform well on prediction problems that are non-linear and high-dimensional [19], which fits our variability-aware performance prediction problem. We adopt ε-SV regression [17], which is the simplest SVM for regression.

We implement regression methods using R 3.0.1 [12]. R is an open-source language and environment for statistical computing and graphics. We use packages RPART [16], RANDOMFOREST [10], and KERNLAB [9] for implementing CART, Bagging, Random Forest and SVM. When implementing each regression method, we consider a number of key parameters for tuning. To choose them, we analyse all parameters accessible in respective R packages and select those that are commonly modified by stakeholders, while fixing other parameters to their default values. We consider the following key parameters for CART: *minsplit* controls the minimum amount of observations that must exist in a tree node in order for it to be partitioned; *minbucket* specifies the minimum amount of observations that must be present in any leaf of a regression tree; *maxdepth* controls the maximum depth of a regression tree; *cp* is a complexity parameter that controls optimal size of a regression tree. For Bagging and Random Forest, we consider the following key parameters: *ntree* specifies the number of trees to be grown; *nodesize* is analogous to the minbucket parameter of CART; *minsplit* parameter is not directly accessible in RANDOMFOREST package and is calculated automatically; *mtry* specifies the number of feature-selection variables that are considered when creating a regression tree and indicates the key difference between Bagging and Random Forest. For Bagging, *mtry* is fixed to the number of all feature-selection variables, i.e., $|X|$. For Random Forest, it varies in the integer value range $[|X|/2, |X| - 1]$. Parameters that we consider for SVM are as follows: *epsilon*, i.e., ε, specifies the width of error-insensitive boundary of a SVM hyperplane for regression; *sigma*, i.e., σ, specifies inverse kernel width for Radial Basis Kernel; C represents the cost of constraint violation.

4. PARAMETER TUNING

Most of machine-learning algorithms have parameters that guide their execution [1] and have a strong influence on their prediction accuracy. However, they are often chosen manually. We use *random search* with quasi-random numbers for parameter tuning of regression methods. Random search generates a set of parameter combinations that are randomly selected according to sequences of *pseudo-random* or *quasi-random* numbers [1]. Pseudo-random numbers are generated in terms of a specified formula that meets certain properties of random numbers [15]. Quasi-random numbers are also generated using a specified formula, but they fill the parameter space more uniformly [11], therefore often recommended and used in systematic parameter tuning. Among many techniques that generate quasi-random numbers, we choose Sobol sampling [11], because it explicitly minimizes the density differences across samples and covers the parameter space more evenly than other quasi-random techniques.

We implement Sobol sampling using RANDTOOLBOX [4] package. For generation of Sobol sequences RANDTOOLBOX provides function SOBOL with the following parameters: n specifies length of a sequence, which is set to 10^P, where P is a number of parameters of a tuned regression method; *dim* controls dimension of a Sobol sequence and is set to P. For example, suppose that we want to generate a Sobol sequence for a method that has $P = 3$ parameters. Therefore to create a Sobol sequence we run function SOBOL with: $\{n = 10^P = 10^3, dim = P = 3\}$. Function SOBOL returns

Table 2: Key parameters of regression methods

Method	Tuning Parameters			
	Name	Type	Minimum	Maximum
CART	minsplit	integer	2	10
	minbucket	integer	1	10
	maxdepth	integer	2	10
	cp	real	0	0.01
Bagging	ntree	integer	2	20
	nodesize	integer	1	10
	mtry	integer	$\|X\|$	$\|X\|$
Random Forest	ntree	integer	2	20
	nodesize	integer	1	10
	mtry	integer	$\|X\|/2$	$\|X\| - 1$
SVM	epsilon	real	0.01	1
	sigma	real	0.01	1
	C	real	1	100

Table 3: Overview of investigated systems; Lang. - Language; LOC - Lines of code; C - number of all valid configurations; N - number of all features

System	Lang.	LOC	C	N
Apache	C	230,277	192	9
LLVM	C++	47,549	1,024	11
x264	C	45,743	1,152	16
Berkeley DB	C	219,811	2,560	18
Berkeley DB	Java	42,596	400	26
SQLite	C	312,625	3,932,160	39

a matrix of real numbers from range $[0, 1]$, representing a 3-dimensional Sobol sequence, which can be converted to actual parameter values.

5. EVALUATION

We conducted a series of experiments to evaluate the four regression methods for variability-aware performance prediction. We aim at comparing the relative prediction error of these methods and their probability of achieving the highest possible prediction accuracy.

We chose a publicly available dataset deployed with the SPL Conqueror tool [13], presented in Table 3. It contains six real-world configurable systems with different sizes, implementation languages, and configuration mechanisms. The dataset includes all configurations of each system and their performance measurements (the exception is SQLite, for which it contains 4,553 configurations for prediction modeling and 100 additional random configurations for prediction evaluation [13]).

5.1 Experimental Setup

Our experimental setup can be divided into three main stages: data preparation, regression methods preparation, and regression analysis.

During data preparation, for each configurable software system (e.g., Apache) we generate five sampling sizes in the form of $T \times N$, where T is the training coefficient with values from 1 to 5, and N is the number of features. For each combination of a system and a sampling size, which we call a *test case*, we generate ten random samples from the original system data.

During method preparation, for each regression method (e.g. CART) we identify a set of tuning parameters and their value ranges. We use Sobol sampling to generate P-dimensional sequence of 10^P quasi-random numbers, where P is the number of tuning parameters for the regression method. We convert the generated sequence of Sobol numbers to actual parameter values, thus creating a sequence of actual parameter settings.

During regression analysis, we perform described steps for each regression method and each test case. First of all, we initialize each regression method with one of the parameter settings, generated during method preparation stage. Then we run each method on each of the ten random samples of the test case, generated during data preparation stage, and calculate the relative error of each run using the following formula:

$$Relative\ Error = \left| \frac{actual - predicted}{actual} \right|$$

The relative error is the relative difference between the actual measured performance value and the predicted performance value. Finally, we average the relative errors of all runs of the regression method and produce the mean relative error.

5.2 Results and Discussion

We present prediction results of the regression methods. For each regression method, we give the prediction results for the best, the average, and the worst parameter settings, i.e., settings that provided the smallest, the closest to the mean, and the largest relative error, respectively. Table 4 summarizes the results as means and standard deviations of relative prediction errors of regression methods for all systems, sampling sizes and parameter settings.

Subject Systems Comparison. According to the experimental results, different subject systems may make or break variability-aware performance prediction using regression analysis. For the purpose of comparison, we define a mean relative error smaller than 10% as an *acceptable* threshold. We can see that for systems LLVM and SQLite all regression methods can produce acceptable results for all sampling sizes and parameter settings. For systems BerkeleyJ and x264, regression methods CART, Bagging and Random Forest were able to achieve acceptable accuracy for all parameter settings and most of the sampling sizes. However, for Apache and BerkeleyC, CART, Bagging and Random Forest were able to show acceptable results using the best parameter settings, but no results are acceptable when using the average or worst parameter settings.

Comparison of Regression Methods. We observe a stable decreasing of the means and standard deviations of relative errors with the increasing sample size for all regression methods. That is, acquiring more measurements always helps improving the prediction accuracy of regression methods.

For each combination of a parameter setting (e.g., average) and a test case, we identify **highest accuracy** as the lowest mean of relative error out of four means of relative errors provided by each regression method. In Table 4, the number in bold indicates the highest accuracy for each combination of parameter setting and test case.

We define a **winner** method for each parameter setting and each test case as a method that achieves the highest accuracy. For example, for the test case of Apache system and $5 \times N$ sampling size, four regression methods with average parameter settings provide the mean relative errors: 10.3, 7.37, 9.4, 21.12. In this case, the highest accuracy is 7.37, and thus the winner is Bagging.

For each parameter setting (e.g., average), we calculate the

Table 4: Relative error means and standard deviations of different regression methods for different subject systems, sample sizes (i.e., $\lvert S \rvert$), and parameter settings including the best, the average, and worst cases

System	$\lvert S \rvert$	Parameter Settings											
		Best				Average				Worst			
		CART	Bagging	Random Forest	SVM	CART	Bagging	Random Forest	SVM	CART	Bagging	Random Forest	SVM
Apache	N	18.82 ± 5.59	**17.32** ± 6.80	19.80 ± 6.39	28.70 ± 7.61	**20.88** ± 6.92	21.22 ± 7.95	23.35 ± 8.38	29.95 ± 7.91	**22.42** ± 8.57	23.72 ± 9.40	30.33 ± 11.64	30.66 ± 8.11
	2 × N	**11.32** ± 4.20	11.64 ± 4.50	14.84 ± 5.22	22.46 ± 7.63	18.64 ± 9.98	**15.77** ± 6.84	19.99 ± 7.30	24.35 ± 8.26	28.74 ± 13.76	**21.25** ± 11.06	29.65 ± 10.85	25.59 ± 8.65
	3 × N	9.30 ± 2.70	**7.03** ± 2.47	7.88 ± 2.26	21.10 ± 7.13	16.45 ± 6.51	**10.19** ± 4.38	13.21 ± 4.21	24.31 ± 7.91	22.83 ± 9.38	**17.16** ± 10.35	22.74 ± 8.31	26.39 ± 8.43
	4 × N	7.91 ± 2.06	7.60 ± 2.14	**6.71** ± 2.54	18.34 ± 7.30	12.95 ± 6.95	**9.26** ± 2.69	10.72 ± 3.88	21.61 ± 8.36	19.26 ± 12.02	**13.25** ± 4.38	23.47 ± 8.63	23.98 ± 8.99
	5 × N	7.15 ± 2.17	6.50 ± 1.99	**6.38** ± 2.08	17.57 ± 7.35	10.30 ± 4.58	**7.37** ± 2.22	9.40 ± 3.11	21.12 ± 8.49	14.39 ± 8.15	**8.82** ± 2.82	15.66 ± 6.59	24.14 ± 8.96
BerkeleyC	N	**135.12** ± 111.59	213.23 ± 231.64	179.67 ± 138.09	212.37 ± 136.01	268.34 ± 226.03	296.92 ± 288.18	**265.61** ± 220.85	472.22 ± 336.38	575.80 ± 420.78	**380.08** ± 377.95	381.78 ± 345.62	650.45 ± 476.13
	2 × N	**30.66** ± 55.00	74.33 ± 143.65	40.86 ± 35.92	125.82 ± 69.71	**117.30** ± 130.59	127.44 ± 222.23	140.71 ± 148.32	387.36 ± 253.53	**169.56** ± 172.12	178.56 ± 282.76	233.29 ± 234.62	605.68 ± 403.95
	3 × N	**10.05** ± 10.10	19.58 ± 16.95	22.34 ± 25.66	97.60 ± 44.57	109.50 ± 163.05	**43.17** ± 63.43	95.81 ± 136.94	345.84 ± 231.77	143.96 ± 240.89	**83.76** ± 132.26	235.99 ± 330.63	581.10 ± 381.70
	4 × N	**6.41** ± 6.96	15.75 ± 24.64	11.29 ± 9.29	92.90 ± 43.73	67.77 ± 65.07	**44.57** ± 83.94	47.10 ± 50.57	299.46 ± 195.00	110.70 ± 107.30	**96.80** ± 190.37	140.48 ± 148.78	526.02 ± 333.45
	5 × N	**2.78** ± 2.88	8.70 ± 11.86	7.35 ± 5.32	89.40 ± 46.43	47.40 ± 35.47	**21.48** ± 37.00	35.17 ± 41.68	271.11 ± 174.76	94.67 ± 74.11	**43.96** ± 83.57	107.78 ± 132.23	499.85 ± 312.06
BerkeleyJ	N	6.69 ± 4.96	6.74 ± 4.12	**6.08** ± 3.71	30.82 ± 22.90	23.80 ± 10.96	**17.97** ± 13.31	19.59 ± 18.94	32.40 ± 23.12	36.96 ± 15.86	**29.23** ± 22.46	44.19 ± 50.03	33.09 ± 23.18
	2 × N	2.01 ± 0.96	**1.97** ± 0.94	**1.97** ± 1.21	28.90 ± 21.56	6.98 ± 7.10	**4.99** ± 4.91	6.63 ± 6.01	31.93 ± 22.43	**20.10** ± 26.81	15.02 ± 20.25	24.42 ± 31.46	33.43 ± 22.67
	3 × N	1.76 ± 0.87	**1.51** ± 0.75	1.67 ± 0.95	26.13 ± 19.22	2.69 ± 1.13	**1.90** ± 0.91	2.81 ± 1.51	30.49 ± 20.76	**2.98** ± 1.66	3.12 ± 1.69	8.28 ± 7.96	32.59 ± 21.16
	4 × N	1.50 ± 0.71	**1.50** ± 0.93	**1.50** ± 0.90	24.63 ± 16.48	2.64 ± 0.93	**1.71** ± 1.08	2.20 ± 1.32	30.00 ± 17.66	3.00 ± 1.24	**2.20** ± 1.32	10.11 ± 12.31	32.77 ± 17.98
	5 × N	1.54 ± 0.72	1.39 ± 0.78	**1.25** ± 0.67	21.65 ± 12.54	2.39 ± 0.88	**1.47** ± 0.89	1.78 ± 0.98	28.22 ± 14.04	2.85 ± 1.03	**1.65** ± 1.07	4.26 ± 3.49	31.62 ± 14.51
LLVM	N	5.08 ± 1.62	4.83 ± 1.95	4.74 ± 1.78	**4.04** ± 1.87	5.72 ± 1.97	5.19 ± 2.22	**5.11** ± 2.19	5.37 ± 2.52	5.96 ± 2.34	**5.93** ± 2.36	5.98 ± 2.43	6.76 ± 2.89
	2 × N	4.18 ± 1.88	3.89 ± 1.56	3.80 ± 1.40	**2.64** ± 1.79	5.13 ± 2.15	**4.52** ± 1.80	4.71 ± 1.89	4.30 ± 2.22	**5.72** ± 2.34	5.73 ± 2.01	5.76 ± 2.40	6.23 ± 2.87
	3 × N	3.13 ± 1.05	3.18 ± 1.22	2.91 ± 1.13	**2.47** ± 1.61	4.48 ± 1.50	**3.70** ± 1.52	3.81 ± 1.64	4.06 ± 2.04	5.45 ± 1.91	**4.63** ± 1.88	5.21 ± 2.17	6.32 ± 2.87
	4 × N	2.72 ± 1.05	2.53 ± 0.94	2.77 ± 1.01	**2.14** ± 1.58	3.91 ± 1.51	**3.09** ± 1.23	3.47 ± 1.42	3.43 ± 1.98	4.87 ± 1.83	**3.90** ± 1.58	5.10 ± 2.19	6.10 ± 2.75
	5 × N	2.37 ± 1.03	2.14 ± 0.78	2.31 ± 0.94	**2.03** ± 1.48	3.48 ± 1.26	**2.70** ± 0.91	3.04 ± 1.41	3.28 ± 1.85	4.23 ± 1.57	**3.57** ± 1.30	4.62 ± 2.29	6.22 ± 2.71
Sqlite	N	4.53 ± 1.77	4.56 ± 1.96	4.46 ± 1.78	**4.17** ± 1.86	4.73 ± 2.01	4.70 ± 2.11	4.65 ± 2.04	**4.39** ± 2.32	5.50 ± 2.26	**5.17** ± 2.21	5.42 ± 2.21	5.42 ± 2.59
	2 × N	4.52 ± 1.64	4.26 ± 1.63	**4.17** ± 1.61	4.30 ± 1.66	4.63 ± 1.86	4.37 ± 1.76	**4.32** ± 1.81	4.45 ± 2.28	5.39 ± 2.10	4.84 ± 1.87	4.85 ± 1.95	**4.68** ± 2.50
	3 × N	4.28 ± 1.59	**4.04** ± 1.73	4.11 ± 1.57	4.11 ± 1.59	4.38 ± 1.74	**4.20** ± 1.82	4.26 ± 1.75	4.33 ± 2.23	4.98 ± 2.07	4.69 ± 1.89	4.84 ± 1.89	**4.49** ± 2.47
	4 × N	3.92 ± 1.60	3.87 ± 1.59	**3.80** ± 1.56	4.01 ± 1.52	4.06 ± 1.76	3.99 ± 1.67	**3.93** ± 1.67	4.35 ± 2.23	4.62 ± 2.03	**4.48** ± 1.75	4.55 ± 1.76	4.50 ± 2.45
	5 × N	3.83 ± 1.67	3.75 ± 1.53	**3.65** ± 1.57	3.90 ± 1.54	4.02 ± 1.77	3.85 ± 1.61	**3.79** ± 1.67	4.36 ± 2.32	4.42 ± 2.06	**4.33** ± 1.69	4.38 ± 1.75	4.51 ± 2.48
x264	N	10.30 ± 4.35	**8.52** ± 3.44	8.72 ± 3.09	22.18 ± 10.39	17.69 ± 6.61	**10.75** ± 4.43	11.89 ± 4.52	30.66 ± 14.40	31.22 ± 15.16	**15.21** ± 5.72	21.95 ± 9.95	35.07 ± 16.13
	2 × N	6.55 ± 2.35	6.18 ± 3.19	**5.79** ± 2.46	15.88 ± 9.93	10.56 ± 3.83	**7.75** ± 3.97	8.28 ± 3.54	29.73 ± 21.64	18.62 ± 8.44	**9.61** ± 4.76	20.31 ± 9.82	36.74 ± 27.47
	3 × N	4.96 ± 2.31	4.72 ± 2.51	**4.70** ± 2.20	9.05 ± 4.70	8.27 ± 3.39	**6.22** ± 3.11	7.01 ± 3.04	24.25 ± 14.19	9.89 ± 4.40	**8.03** ± 3.91	15.80 ± 6.83	34.97 ± 21.41
	4 × N	**3.36** ± 1.74	3.46 ± 1.74	4.01 ± 1.92	6.79 ± 3.63	7.26 ± 2.76	**4.90** ± 2.34	5.80 ± 2.63	21.68 ± 13.53	9.01 ± 4.15	**6.82** ± 3.09	10.43 ± 3.78	34.56 ± 23.82
	5 × N	**2.46** ± 1.38	3.16 ± 1.72	2.98 ± 1.46	5.22 ± 2.98	6.72 ± 2.53	**4.36** ± 2.21	4.92 ± 2.12	19.91 ± 12.24	8.40 ± 3.42	**6.18** ± 3.01	9.18 ± 3.26	34.38 ± 22.30

Table 5: Winner probabilities of regression methods for different parameter settings and closeness ranges

	Parameter Settings											
	Best				Average				Worst			
Closeness Range	CART	Bagging	Random Forest	SVM	CART	Bagging	Random Forest	SVM	CART	Bagging	Random Forest	SVM
0%	30.00%	23.33%	**36.67%**	20.00%	6.67%	**70.00%**	16.67%	6.67%	13.33%	**80.00%**	0.00%	6.67%
5%	40.00%	**46.67%**	43.33%	26.67%	16.67%	**86.67%**	23.33%	13.33%	23.33%	**93.33%**	23.33%	16.67%
10%	**56.67%**	**56.67%**	46.67%	33.33%	26.67%	**96.67%**	40.00%	20.00%	26.67%	**100.00%**	26.67%	20.00%
15%	**66.67%**	**66.67%**	60.00%	33.33%	30.00%	**100.00%**	60.00%	26.67%	33.33%	**100.00%**	30.00%	26.67%

winner probability that a certain regression method (e.g., Bagging) is a winner among all test cases. For example, when using average parameter settings, Bagging became winner for 21 out of 30 test cases, and thus the winner probability of Bagging is $21/30 \times 100\% = 70\%$. This way, we use the winner probability as a metric to compare the robustness of different regression algorithms.

Furthermore, from Table 4, we can see that sometimes for the same test case the difference of mean relative error between a winner method and others is close. In order to assess this difference between the highest accuracy and other predicted accuracy, we define metric called **closeness** that expresses this difference in percentage:

$$closeness = \left| \frac{highest - predicted}{highest} \right|$$

Using this definition of closeness, we can calculate probability that a given regression method achieves highest accuracy within a specified closeness range, i.e., the method becomes a winner within this closeness range. For example, as discussed earlier, when using average parameter settings, Bagging became a winner in 70% of the test cases. However, if we allow 5% closeness range, Bagging becomes winner in 86.67% of the test cases.

Table 5 compares the winner probabilities of different regression methods for different parameter settings and closeness ranges. The first row of Table 5 presents the winner probabilities if we do not allow any closeness range, i.e., probabilities of achieving the highest accuracy by regression methods. We can see that for the best parameter settings, Random forest outperformed all other regression methods by showing 36.67% winner probability. However, for the average and the worst parameter settings, Bagging has shown a high winner probability of 70% and 80% respectively, outperforming all other regression methods.

The second row of Table 5 presents the winner probabilities if we allow 5% closeness range. We can see that this time Bagging outperformed all other regression methods for all parameter settings by showing 46.67%, 86.67% and 93.33% winner probabilities for the best, the average and the worst parameter settings, respectively.

The third and the fourth rows of Table 5 present winner probabilities for 10% and 15% closeness ranges respectively. Here, Bagging showed the highest winner probability along with CART for the best parameter settings, while completely outperforming all other regression methods for the average and the worst parameter settings.

6. THREATS TO VALIDITY

To increase internal validity of our work, we performed automated sampling. Training samples for each system and sampling size were selected randomly 10 times from the whole population of configurations, while taking the rest of the population as testing samples.

Given the number of parameters and their value ranges, the parameter space of each regression method might be huge. To this end, we randomly generate 10^P parameter settings, for each regression method and for each subject system, where P is the number of tuning parameters for each method. Moreover, to cover the possibly huge parameter space as evenly as possible, we generate parameter settings following Sobol sampling, a state-of-the-art parameter sweep technique.

We use the same metric (i.e., relative error) used in [13, 5] to evaluate prediction accuracy of regression methods. However, we are aware that any existing metric may provide specific, but incomplete quantification of accuracy [6]. A potential future work to mitigate this issue is to combine multiple evaluations metrics for an ensemble evaluation.

To enhance external validity, we performed our experiments using six real-world software systems of different sizes, different implementation languages, and different application domains. However, we are aware that the results of our experiments are not automatically transferable to all other software systems.

7. RELATED WORK

Westermann et al. [18] proposed an approach that builds prediction models of expected accuracy using the least possible number of measurements. Their approach uses three measurement-point-selection algorithms for exploring the parameter space and two validation strategies for assessing the performance of prediction model. However, their approach assumes that all features involved in the prediction models are performance-relevant, while our work considers all features of a software system.

Siegmund et al. [14] proposed SPLConqueror that automatically detects performance-relevant feature interactions using specific sampling heuristics that meet different feature-coverage criteria. On the contrary, the regression methods use random samples as basis and avoid the effort of detecting feature interactions.

Hutter et al. [8] investigated predicting performance of algorithms designed for solving hard combinatorial problems. They used machine learning approaches for generating predictive models based on Random Forest and Gaussian processes. In contrast, we focused on the performance prediction of configurable software systems, and as well compared different regression methods. Moreover, we use Sobol sampling and the winner probability for parameter tuning and robustness evaluation.

This paper extends our previous work [5] using CART for variability-aware performance prediction. We add three popular regression methods and compare the prediction accuracy of all regression methods.

8. CONCLUSION

We regard configurable software as a black box and infer the correlation between performance and feature selection using non-linear regression analysis. We used four regression methods for variability-aware performance prediction, and performed systematic parameter tuning of those methods using Sobol sampling. We conducted experiments on six real-world configurable software systems and evaluated the relative prediction error of the regression methods when predicting performance of those systems. We use the winner probability of achieving the highest accuracy by a performance prediction model as a metric for robustness comparison of different regression methods. Empirical results show that Bagging turns out to be the ideal method for the studied systems, since it achieves the highest accuracy with higher probability than any other regression method for different parameter settings.

In future work, we plan to further evaluate more regression methods and parameter tuning techniques for variability-aware performance prediction. We also want to investigate different experimental design policies for performance prediction.

9. REFERENCES

[1] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305, Feb. 2012.

[2] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[3] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[4] C. Dutang and P. Savicky. *randtoolbox: Generating and Testing Random Numbers*, 2013.

[5] J. Guo, K. Czarnecki, S. Apel, N. Siegmund, and A. Wasowski. Variability-aware performance prediction: A statistical learning approach. In *Proc. ASE*. IEEE, 2013.

[6] J. Guo, E. Zulkoski, R. Olaechea, D. Rayside, K. Czarnecki, S. Apel, and J. M. Atlee. Scaling exact multi-objective combinatorial optimization by parallelization. In *Proc. ASE*. ACM, 2014.

[7] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, 2nd Edition*. Springer, 2009.

[8] F. Hutter, L. Xu, H. H. Hoos, and K. Leyton-Brown. Algorithm runtime prediction. methods and evaluation. *Artificial Intelligence*, 206:79–111, 2014.

[9] A. Karatzoglou, A. Smola, K. Hornik, and A. Zeileis. kernlab – an S4 package for kernel methods in R. *Journal of Statistical Software*, 11(9):1–20, 2004.

[10] A. Liaw and M. Wiener. Classification and regression by randomforest. *R News*, 2(3):18–22, 2002.

[11] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C (2nd Ed.): The Art of Scientific Computing*. Cambridge Univ. Press, 1992.

[12] R Core Team. *R: A Language and Environment for Statistical Computing*, 2013.

[13] N. Siegmund, S. S. Kolesnikov, C. Kästner, S. Apel, D. Batory, M. Rosenmüller, and G. Saake. Predicting performance via automated feature-interaction detection. In *Proc. ICSE*. IEEE Press, 2012.

[14] N. Siegmund, M. Rosenmüller, M. Kuhlemann, C. Kästner, S. Apel, and G. Saake. SPL Conqueror: Toward optimization of non-functional properties in software product lines. *Software Quality Journal*, 20(3-4):487–517, 2012.

[15] I. Sobol and Y. Levitan. A pseudo-random number generator for personal computers. *Computers and Mathematics with Applications*, 37(4–5):33–40, 1999.

[16] T. Therneau, B. Atkinson, and B. Ripley. *rpart: Recursive Partitioning and Regression Trees*, 2014.

[17] V. N. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag New York, Inc., 1995.

[18] D. Westermann, J. Happe, R. Krebs, and R. Farahbod. Automated inference of goal-oriented performance prediction functions. In *Proc. ASE*. ACM, 2012.

[19] G. J. Williams. *Data Mining with Rattle and R: The art of excavating data for knowledge discovery*. Springer, 2011.