Regression Models for Performance Ranking of Configurable Systems: A Comparative Study

Yuntianyi Chen, Yongfeng Gu, Lulu He, and Jifeng Xuan*

School of Computer Science, Wuhan University
Wuhan, China
{yuntianyichen, yongfenggu, luluhe, jxuan}@whu.edu.cn

Abstract. Finding the best configurations for a highly configurable system is challenging. Existing studies learned regression models to predict the performance of potential configurations. Such learning suffers from the low accuracy and the high effort of examining the actual performance for data labeling. A recent approach uses an iterative strategy to sample a small number of configurations from the training pool to reduce the number of sampled ones. In this paper, we conducted a comparative study on the rank-based approach of configurable systems with four regression methods. These methods are compared on 21 evaluation scenarios of 16 real-world configurable systems. We designed three research questions to check the impacts of different methods on the rank-based approach. We find out that the decision tree method of Classification And Regression Tree (CART) and the ensemble learning method of Gradient Boosted Regression Trees (GBRT) can achieve better ranks among four regression methods under evaluation; the sampling strategy in the rank-based approach is useful to save the cost of sampling configurations; the measurement, i.e., rank difference correlates with the relative error in several evaluation scenarios.

Keywords: Regression Methods · Performance Prediction · Sampling · Software Configurations.

1 Introduction

A highly configurable system integrates many configuration options (i.e., a configurable feature of a system) to provide choices for system administrators and end users. For instance, a web server, such as Apache Http Server, supports the configuration of server performance to adapt to the resource limit or particular hardware platforms. Given a configurable system, many configurations may result in performance issues, e.g., the resource consumption of a compiler or the response time of a web server [14]. Due to the large search space of examining all configurations, it is expensive to find out the optimal configuration for a complicated system [8]. A highly configurable system contains numerous

* Corresponding author: Jifeng Xuan

configuration options; in a large configurable system, even examining one configuration may be time-consuming, e.g., deploying and rebooting a web server. This makes manually examining all configurations impossible.

Existing studies have introduced machine learning methods to predict the performance of a configuration [6,14]. The predicted performance of a configuration serves as a reference to the user. This prediction can be used to assist the choice of configurations. A general way of performance prediction is to use a regression method to approximate an exact performance value. Such a regression method evaluates the predicted performance with the relative error of performance [14,15,9,8,16,10]. The Classification And Regression Tree (CART) is viewed as one of the state-of-the-art methods for the regression on performance prediction [15,10]. An ideal result is that the regression method can predict performance for all configurations. However, the user of performance prediction may not care about the performance of all the configurations. Instead, choosing a configuration with the best performance is practical. Nair et al. [8] revisited the goal of performance prediction: getting the (near) optimal configuration for users. Therefore, the original regression problem is treated as a ranking problem: ranking configurations based on their predicted performance and choosing a "good" configuration rather than predicting performance for all configurations. Such a method is called a *rank-based approach*. A rank-based approach evaluates the result with the *rank difference*, which is defined as the actual rank for a configuration that is predicted as the best.

The study by Nair et al. [8] also revealed an important problem of performance prediction, i.e., the number of sampled configurations for machine learning. A user cannot collect many configurations with known performance since examining the actual performance of configurations is costly. Nair et al. [8] used an iterative sampling method to reduce the number of sampled configurations in the rank-based approach. This sampling method is considered to shorten the cost of sampling configurations in real-world applications.

In this paper, we conducted a comparative study on the rank-based approach of configurable systems with four regression methods, including Classification And Regression Tree (CART), Support Vector Regression (SVR), Gaussian Process Regression (GPR), and Gradient Boosted Regression Trees (GBRT). Methods in the study are compared on 21 evaluation scenarios of 16 real-world configurable systems. We designed three Research Questions (RQs) to investigate the impacts of different methods on the rank-based approach. In the study, we evaluated the result of each regression method via learning a model from known configurations and measured the result with the rank difference and the Mean Magnitude of Relative Error (MMRE). Each experiment is repeated for 50 times to reduce the disturbance of randomness. We find that among four regression methods under evaluation, the decision tree method CART and the ensemble learning method GBRT can achieve better ranks while SVR and GPR can save more sampled configurations; the sampling strategy in the rank-based approach is useful to the reduction of sampled configurations; the measurement, i.e., rank difference correlates with the relative error in several evaluation scenarios.

This paper makes the following major contributions:

- We compared four different regression methods for the problem of performance ranking of configurable systems on 21 evaluation scenarios of 16 real-world systems.
- We empirically studied the saved cost of sampling configurations and the correlation between evaluation measurements.

The rest of this paper is organized as follows. Section 2 presents the background of this study. Section 3 shows the study setup, including three research questions and the data preparation. Section 4 describes the results of our comparative study. Section 5 lists the threats to the validity. Section 6 presents the related work and Section 7 concludes.

2 Background and Motivation

Accurately predicting the performance of configurations is challenging due to the time cost of sampling and the accuracy of building performance models.

2.1 Problem Formalization

In this paper, we followed Nair et al. [8] to adopt the rank-based approach. In this approach, a regression model is learned from a training set of configurations and then is used to predict the performance for each configuration in the test set. The rank-based approach sorts all configurations in the test set in terms of the predicted performance. We call the sorting results a *predicted sequence*, denoted as ps; let ps_k denote the top-k configurations in ps.

Let X be the search space of all configurations in a dataset. Denoting the number of configuration options as n and one configuration as x, we can define a configurable option in a configuration as x_i and $x = \{x_1, x_2, ..., x_n\}$. The domain of x_i is defined as $D(x_i)$. Thus $X \subseteq D(x_1) \times D(x_2) \times ... \times D(x_n)$. Let $perf(x)$ be the performance of a configuration x. Then we denote the actual performance of x as $perf_a(x)$ and the predicted performance that is calculated by a regression method as $perf_p(x)$.

Previous studies of performance prediction generally employs the Mean Magnitude of Relative Error (MMRE) to evaluate the accuracy of performance prediction [14,15]. MMRE is defined as follow,

$$\text{MMRE} = \frac{1}{n} \sum_{x \in X} \frac{|perf_a(x) - perf_p(x)|}{perf_a(x)}$$

Nair et al. [8] introduced the rank difference to evaluate the actual rank of configurations that are predictively ranked as the top. The rank difference can be defined as the minimum of actual rank values of the top-k configurations, i.e., $MAR(k)$. The rank difference $MAR(k)$ can be used to pick a configuration out from the predicted sequence. We use $MAR(k)$ to evaluate the ability of ranking configurations to the top of the predicted sequence.

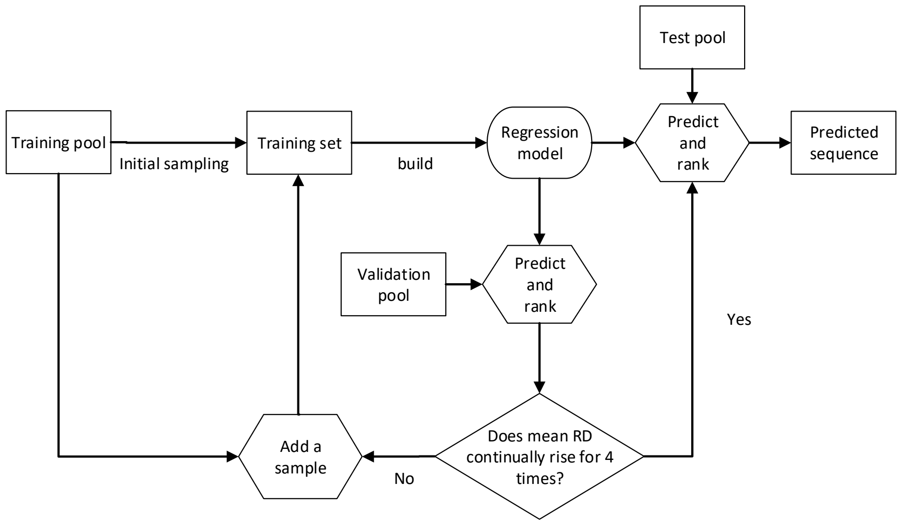

Fig. 1. Overview of the rank-based approach.

2.2 Rank-based Approach

The state-of-the-art approach to solve the ranking problem of software configurations is called the rank-based approach by Nair et al. [8]. They considered ranking as the ultimate goal of performance prediction and claimed that users concerned ranks rather than exact performance. They adopted a rank-based sampling method to reduce the number of training samples for building regression models. To evaluate the ranking deviation of a configuration, they introduced a measurement *rank difference*, which means the rank distance between the predicted rank and the actual rank of a configuration. We denote a predicted rank as $rank_p(x)$ and an actual rank as $rank_a(x)$.

$$RD(x) = |rank_a(x) - rank_p(x)|$$

Fig. 1 shows the overview of the rank-based approach by Nair et al. [8]. The rank-based approach employs an iterative sampling strategy to collect a training set. At the beginning of sampling, they randomly select a certain number of samples from the training pool as the training set. The training set is used to build a regression model and to predict the performance on the validation pool. Then, the mean of rank differences RD is calculated for all configurations to decide whether the iterative sampling process terminates. Once the mean RD continually rises for a pre-defined number of times, the iteration ends. Otherwise, a new sampled configuration is added into the training set from the training pool. Finally, a regression model is learned from the training set and is used to predict and rank the test pool. The result we get after ranking is called the predicted sequence of the test pool.

2.3 Motivation

Highly configurable systems are common in our daily lives. A software usually contains a large number of configuration options. For instance, Project lrzip is a compression software for large files. As for performance related configuration options, we can identify 19 features, which lead to 432 configurations [13]. A user could tune these configuration options to change the performance, namely the compression time in Project lrzip. Rich choices of configuration can facilitate the functions of systems, but may hide faults and performance issues [14,18,16,17]. Due to the large number of configurations, manual examination on all configurations is arduous. Thus, it is feasible to predict the performance for configurations with unknown performance via machine learning algorithms[6].

Learning a regression model requires sampled configurations, which are selected by users and examined by deploying the system with the configurations. However, sampling is time-consuming. Existing studies have proposed several sampling strategies to reduce the number of sampled configurations [14,12,20,9]. In this paper, we expect to evaluate the cost of learning a regression model, i.e., the number of configurations that contain known performance.

There are many regression methods in machine learning. The CART serves as the state-of-the-art method in performance prediction [6,12,20,10,9,8,16]. In this paper, we expect to find out which regression method is the best on performance ranking. Can the CART reduce the number of configurations with known performance in learning regression models? In this paper, we comprehensively compare the result of performance ranking with four regression methods.

3 Study Setup

We describe the data preparation, four regression methods, and three RQs in this section.

3.1 Dataset Preparation and Evaluation Setup

In this paper, we empirically evaluate the performance ranking on 16 subject systems with 21 evaluation scenarios. A *evaluation scenario* is a subject system deployed on a particular running environment.

One subject system may contain one or more evaluation scenarios. For instance, *wc-6d-c1-obj1* and *wc-6d-c1-obj2* are two evaluation scenarios that use the same subject system and hardware environments, but different performance indicators: *wc-6d-c1-obj1* measures performance with throughput while *wc-6d-c1-obj2* measures performance with latency [7]. Table 1 presents the details of 21 evaluation scenarios in our study.

The performance of configurable systems can be measured with different indicators. For example, the performance is measured by throughput in *wc-6d-c1-obj1*. That is, the larger the value is, the better the performance is. In *BerkeleyC*, the performance is defined as the response time. For the sake of unification, we

Table 1. Configurations of 21 evaluation scenarios in 16 subject systems.

Scenario	Options	Configurations	Performance	Ref.
AJStats	19	30,256	analysis time	[13]
Apache	9	192	max response rate	[13]
BerkeleyC	18	2,560	response time	[13]
BerkeleyJ	26	180	response time	[13]
clasp	19	700	solving time	[13]
Dune	11	2,304	solving time	[8]
HSMGP_num	14	3,456	average time	[8]
lrzip	19	432	compression time	[8]
noc	4	259	runtime	[21]
snw	3	206	throughput	[21]
spear	14	16,384	solving time	[13]
SQL	39	4,553	response time	[13]
wc+rs-3d-c4-obj1	3	196	throughput	[7]
wc+rs-3d-c4-obj2	3	196	latency	[7]
wc-6d-c1-obj1	6	2,880	throughput	[7]
wc-6d-c1-obj2	6	2,880	latency	[7]
wc-c1-3d-c1-obj1	3	1,343	throughput	[7]
wc-c1-3d-c1-obj2	3	1,343	latency	[7]
WGet	16	188	main memory	[8]
x264	16	2,047	encoding time	[13]
XZ	16	1,078	execution time	[16]

pre-processed the raw data. For instance, we transferred the throughput value *tp* in *wc-6d-c1-obj1* into $100/tp$.

In our study, the whole dataset is randomly divided into three parts, including the training pool, the test pool, and the validation pool, which respectively account for 40%, 40%, and 20% of the total number of configurations. Each regression method is evaluated on the same division of the dataset and with the same random seed. The experiment is repeatedly run for 50 times to count the average.

We evaluate the results with two measurements, $MAR(k)$ and $MMRE(k)$, which denote the minimum of actual ranks for the top-k ranking and the mean magnitude of relative errors of the performance prediction for the top-k ranking, respectively.

We also measure the correlation between $MAR(k)$ and $MMRE(k)$ with the *Pearson correlation coefficient*. We repeated running each experiment for fifty times and collected each 50 pairs of $MAR(k)$ and $MMRE(k)$. Let Y and X be the vectors of $MAR(k)$ and $MMRE(k)$ values. Then the Pearson correlation coefficient between $MAR(k)$ and $MMRE(k)$ is calculated as follows,

$$\rho(X,Y) = \frac{\mathrm{cov}(X,Y)}{\sigma_X \sigma_Y}$$

where $\rho(X, Y)$ and $\mathrm{cov}(X, Y)$ are the correlation coefficient and the covariance between X and Y, σ_X and σ_Y are the standard deviation of X and Y, respectively.

Our experiment is implemented with Python 3.7.3. We use the Python package *scikit-learn* to build regression models and to predict the performance of configurations. Our running environment is a PC with Intel Core i7 Quad CPU 2.60GHz and 8GB RAM memory.

3.2 Regression Methods

In this paper, we selected four regression methods: three basic regression methods, Classification and regression tree (CART), Support Vector Regression (SVR), and Gaussian Process Regression (GPR), and one ensemble method, Gradient Boosted Regression Trees (GBRT). In our evaluation, we do not assume that there is a specific function between the performance and configurations of a scenario.

CART is a classic decision tree algorithm, which builds a classification or regression model within a tree structure [1]. The decision tree iteratively partitions into branches to generate a decision rule. Leaf nodes of the tree contain the prediction result. *SVR* is the regression model of the support vector machine algorithm [2]. SVR uses a hyperplane to discriminate samples and sets a tolerance for the loss function. In our study, we chose the Radial Basis Function (RBF) as the kernel algorithm of SVR because we assume configuration data are non-linear. *GPR* is a non-parametric regression model that uses probability distribution to predict results [11]. We also use the RBF as the kernel. *GBRT* is an ensemble learning algorithm of multiple decision trees, i.e., CART [3]. GBRT uses multiple weak learners to fit a strong learner. GBRT iteratively finds decision trees to turn down the value of loss functions.

3.3 Research Questions

Our experiments are designed to compare the ranking ability of four different regression methods. We aim at investigate the performance ranking via answering three RQs:

– RQ1. Which regression method can achieve better ranking for the performance ranking?
– RQ2. How many configurations are sampled in the rank-based performance ranking?
– RQ3. Does the rank difference in performance ranking correlate with the relative error in performance prediction?

In RQ1, we check which regression method can perform well in the performance ranking of configurable systems. The CART method is considered as a state-of-the-art method [8]. Then what is the effectiveness of other typical regression methods? We plan to evaluate the four regression methods on the performance ranking via two measurements, the minimum of actual ranks and the relative error.

In RQ2, we aim at evaluating the number of sampled configurations for learning a rank-based model. Nair et al. [8] employed an iterative strategy to sample configurations for the model learning. We focused on the number of sampled ones by different regression methods.

In RQ3, we use the Pearson correlation coefficient to measure the relevance between two measurements. We collect results from fifty repeated experiments and calculate the correlation between the minimum of actual ranks $MAR(k)$ and the mean magnitude of relative error $MMRE(k)$. The correction is evaluated by checking $k = 1$, $k = 5$, and $k = 10$, respectively, i.e., the measurements of top-1, top-5, and top-10.

4 Comparative Study

We conducted a comparative study on using four regression methods in the performance ranking. These regression methods are evaluated with experiments on 21 evaluation scenarios of 16 subject systems.

4.1 RQ1. Which regression method can achieve better ranking for the performance ranking?

We followed the rank-based approach by Nair et al. [8] to use an iterative strategy of sampling. This strategy samples the training set from a given training pool and then a regression model is learned from the training set. All configurations in a validation pool are used to validate the prediction by the learned regression model. The sampling process repeats until a pre-defined threshold reaches. In the training pool, configurations that are not sampled can be viewed as configurations without known performance. Then the sampling strategy can save the effort of examining the actual performance of configurations. If the sampling strategy terminates, the learned regression model is used to predicted performance for all configurations in the test set. According to the predicted performance, configurations in the test set are sorted and measured with two defined measurements $MAR(k)$ and $MMRE(k)$ in Section 3.1.

Table 2 shows the minimum of actual ranks, i.e., $MAR(k)$, on 21 evaluation scenarios of four regression methods, CART, SVR, GBRT, and GPR. We compared the minimum of actual ranks for top-1, top-5, and top-10 configurations in the predicted sequence.[1] As shown in Table 2, CART can obtain the best rank in 7 out of 21 evaluation scenarios for top-10 configurations; GBRT can obtain the best rank in 11 scenarios; SVR and GPR obtain the best rank in one and two scenarios, respectively. CART and its enhanced method GBRT could achieve better rank than the other methods. The difference among four

[1] Ranks in the experiment are zero-based; that is, the MAR value of the best configuration is zero.

Table 2. Minimum of actual ranks on 21 evaluation scenarios of 4 regression methods

Scenario	Top-1				Top-5				Top-10			
	CART	SVR	GBRT	GPR	CART	SVR	GBRT	GPR	CART	SVR	GBRT	GPR
AJStats	4381.56	4159.68	**3920.20**	9611.40	**949.62**	1985.46	1219.68	7630.66	739.48	1254.20	**689.62**	6514.96
Apache	5.52	6.54	**2.66**	4.78	1.02	1.30	**0.32**	0.32	0.36	0.50	**0.14**	0.18
BerkeleyC	122.70	161.82	**83.14**	110.52	**35.34**	73.08	47.18	39.10	24.82	49.38	29.98	**18.54**
BerkeleyJ	11.30	15.86	**10.92**	17.34	3.56	4.60	**3.36**	7.48	**1.52**	2.68	1.54	2.98
clasp	17.08	38.84	**6.20**	54.34	4.86	16.80	**1.64**	21.64	1.76	8.22	**0.46**	14.98
Dune	161.74	**87.06**	110.10	276.38	55.52	**27.42**	31.22	119.92	37.88	**11.80**	18.58	83.52
HSMGP_num	88.42	29.46	**14.60**	71.92	23.92	10.48	**2.28**	16.62	12.10	7.58	**1.26**	13.08
lrzip	**10.80**	35.34	13.68	27.80	**3.16**	19.72	3.30	8.80	**1.42**	11.76	1.62	6.60
noc	6.50	3.60	**1.66**	27.18	0.80	1.46	**0.04**	2.24	0.18	0.94	**0.02**	1.86
snw	3.66	4.32	**1.30**	20.04	0.52	0.40	**0.12**	3.58	0.32	0.18	**0.00**	2.02
spear	**1072.44**	1360.02	1237.38	1335.10	**246.9**	776.60	646.78	549.14	**204.34**	651.74	444.7	378.26
SQL	749.38	**589.60**	590.74	682.52	288.48	237.08	**212.36**	271.84	123.34	155.80	**106.22**	128.72
wc+rs-3d-c4-obj1	13.94	26.94	**5.74**	21.92	2.66	6.24	**1.86**	5.94	0.84	4.08	**0.16**	3.56
wc+rs-3d-c4-obj2	**4.56**	16.08	18.74	49.52	**1.00**	6.40	4.78	23.36	**0.42**	3.84	1.92	10.06
wc-6d-c1-obj1	268.60	395.60	**261.06**	499.56	**56.20**	150.82	103.18	339.24	**26.28**	94.72	45.70	201.44
wc-6d-c1-obj2	**80.58**	225.50	209.58	395.46	**34.42**	106.80	116.88	296.38	**18.78**	86.10	83.56	247.50
wc-c1-3d-c1-obj1	56.36	102.68	**31.74**	494.58	16.26	**4.58**	5.12	440.70	7.88	2.20	**1.70**	400.86
wc-c1-3d-c1-obj2	82.46	94.84	**64.62**	481.22	16.20	18.58	**16.02**	450.64	11.38	8.62	**8.18**	380.94
WGet	**25.48**	30.92	26.38	47.50	7.34	8.64	**6.96**	23.04	**3.70**	4.06	3.78	11.36
x264	92.42	72.06	**17.60**	300.90	25.84	16.00	**3.28**	129.32	14.22	9.72	**1.82**	86.94
XZ	86.84	93.60	134.84	**85.04**	**17.14**	27.10	37.88	26.26	11.52	14.56	18.46	**10.98**

regression methods is not slight. For instance, in top-10 of Scenario AJStats, the rank of GBRT is 689.62 while the rank of GPR is 6514.96; in top-10 of Scenario XZ, the rank of GBRT is 18.46 while the rank of GPR is 10.98. In top-10, four regression methods can nearly rank the actually optimal configuration to the top for several evaluation scenarios, e.g., Apache, BerkeleyJ, noc, snw, and wc+rs-3d-c4-obj1.

We also present the results of MMRE in Table 3. In top-10, CART reaches the best MMRE in 7/21 evaluation scenarios; SVR can obtain the best MMRE values in 3/21 scenarios; GBRT achieves the best MMRE in 11/21 scenarios; GPR cannot obtain the best MMRE in these scenarios.

We consider the results across Table 2 and Table 3. For Scenario XZ, CART performs the best for all MMRE values in Table 3 while GPR performs the best for top-1 and top-10 for MAR values in Table 2. This observation reveals that the MAR value may not correlate with the MMRE value. We study such correlation in Section 4.3.

4.2 RQ2. How many configurations are sampled in the rank-based performance ranking?

The iterative strategy of sampling in the rank-based approach [8] can save the effort of examining actual performance of configurations. In RQ2, we evaluate the sampling via the number of sampled configurations, denoted as $\#samples$. In the rank-based approach, $\#samples$ counts all configurations that have known actual performance, including both the training set and the validation pool.

Table 3. Mean magnitude of relative errors on 21 evaluation scenarios of 4 regression methods

Project	Top-1				Top-5				Top-10			
	CART	SVR	GBRT	GPR	CART	SVR	GBRT	GPR	CART	SVR	GBRT	GPR
AJStats	0.02	**0.02**	0.03	0.85	0.02	**0.02**	0.03	0.84	0.02	**0.02**	0.03	0.84
Apache	0.10	0.76	**0.06**	0.14	0.09	0.69	**0.05**	0.11	0.09	0.63	**0.05**	0.11
BerkeleyC	**0.15**	1.08	0.98	4.86	**0.15**	1.29	0.91	3.75	**0.15**	1.36	0.83	3.15
BerkeleyJ	0.06	0.47	**0.06**	0.65	0.05	0.46	**0.05**	0.59	0.05	0.45	**0.04**	0.53
clasp	0.05	0.41	**0.03**	0.29	0.06	0.40	**0.04**	0.28	0.06	0.40	**0.04**	0.28
Dune	0.18	0.40	**0.14**	1.00	0.18	0.40	**0.13**	0.89	0.18	0.38	**0.12**	0.83
HSMGP_num	0.19	1.67	**0.16**	1.59	0.21	1.43	**0.14**	1.35	0.22	1.31	**0.14**	1.20
lrzip	**0.09**	2.04	0.69	2.52	**0.10**	1.97	0.57	1.92	**0.10**	1.94	0.47	1.47
noc	0.03	0.11	**0.02**	0.20	0.03	0.09	**0.02**	0.11	0.04	0.08	**0.02**	0.09
snw	0.08	0.38	**0.06**	3.04	0.07	0.31	**0.06**	1.90	0.07	0.26	**0.05**	1.26
spear	**0.46**	27.68	8.69	9.15	**0.51**	28.61	8.16	6.02	**0.50**	28.36	8.37	4.84
SQL	0.08	**0.04**	0.08	0.99	0.08	**0.04**	0.08	0.99	0.08	**0.04**	0.08	0.99
wc+rs-3d-c4-obj1	0.28	0.44	**0.24**	3345.07	**0.26**	0.49	0.27	1937.15	0.24	0.49	**0.24**	1245.09
wc+rs-3d-c4-obj2	**0.12**	165.99	108.94	1997.79	**0.11**	93.29	107.11	1469.57	**0.15**	76.28	67.58	1638.20
wc-6d-c1-obj1	**0.35**	0.83	1.27	450.99	**0.35**	0.74	1.19	407.35	**0.34**	0.77	1.10	380.17
wc-6d-c1-obj2	**0.17**	14.49	27.16	504.66	**0.20**	11.07	21.70	348.91	**0.20**	10.22	20.78	334.28
wc-c1-3d-c1-obj1	0.08	**0.07**	0.07	4.53	0.07	0.09	**0.06**	4.12	0.07	0.10	**0.06**	3.84
wc-c1-3d-c1-obj2	0.10	0.16	**0.08**	8.59	0.09	0.21	**0.09**	8.13	0.10	0.22	**0.08**	7.75
WGet	**0.04**	0.08	0.14	0.86	0.07	**0.05**	0.11	0.83	0.07	**0.05**	0.09	0.81
x264	0.07	0.16	**0.04**	0.74	0.07	0.17	**0.04**	0.73	0.07	0.16	**0.04**	0.72
XZ	**0.59**	2.78	1.33	0.85	**0.64**	3.12	1.25	0.86	**0.67**	3.00	1.08	0.83

Table 4 shows the number of sampled configurations by four regression methods. For the sake of space, we only show the MAR results of top-10 and remove the MMRE results.

As presented in Table 4, iterative sampling can save nearly up to half of samples compared with the *No sampling* method, especially for projects with a large number of configurations like Projects AJStats and spear. As for four regression methods, SVR and GPR can respectively obtain the least samples in 12 and 7 scenarios. The difference between CART and GBRT is slight: both have relatively more samples than the other two methods and can achieve the least samples in only one scenario. For instance, in Scenario wc-6d-c1-obj1, CART has 617.66 samples on average and its MAR is 26.28. SVR and GPR have 606.34 and 602.28 samples on average but their MARs are 94.72 and 201.44, which are much higher than the MAR of the CART.

4.3 RQ3. Does the rank difference in performance ranking correlate with the relative error in performance prediction?

As mentioned in Section 4.3, the rank-based approach can be measured with MAR and MMRE. We investigate whether there exists correlation between the values of MAR and MMRE.

Table 4. Number of sampled configurations for top-10 results on 21 evaluation scenarios by four regression methods. *No sampling* denotes the number of all configurations without sampling.

Project	Iteratively sampling								No sampling
	CART		SVR		GBRT		GPR		
	MAR	#Samples	MAR	#Samples	MAR	#Samples	MAR	#Samples	#Samples
AJStats	739.48	6088.48	1254.20	6080.46	**689.62**	6089.28	6514.96	**6073.56**	12102
Apache	0.36	73.18	0.50	**64.28**	**0.14**	69.32	0.18	70.40	76
BerkeleyC	24.82	557.36	49.38	**546.32**	29.98	561.08	**18.54**	562.20	1024
BerkeleyJ	**1.52**	68.46	2.68	62.84	1.54	64.72	2.98	**59.34**	72
clasp	1.76	190.26	8.22	**171.56**	**0.46**	182.60	14.98	181.60	280
Dune	37.88	500.66	**11.80**	495.08	18.58	497.22	83.52	**490.98**	921
HSMGP_num	12.10	**733.48**	7.58	734.86	**1.26**	737.52	13.08	752.16	1382
lrzip	**1.42**	124.98	11.76	**114.86**	1.62	129.30	6.60	121.96	172
noc	0.18	91.68	0.94	**80.36**	**0.02**	89.56	1.86	82.42	103
snw	0.32	76.46	0.18	**65.38**	**0.00**	74.94	2.02	68.74	82
spear	**204.34**	3310.18	651.74	3311.72	444.70	**3308.46**	378.26	3327.34	6553
SQL	123.34	943.42	155.80	**934.00**	**106.22**	949.52	128.72	935.24	1821
wc+rs-3d-c4-obj1	0.84	73.56	4.08	**62.32**	**0.16**	75.64	3.56	62.62	78
wc+rs-3d-c4-obj2	**0.42**	71.24	3.84	**62.56**	1.92	68.90	10.06	62.66	78
wc-6d-c1-obj1	**26.28**	617.66	94.72	606.34	45.70	616.84	201.44	**602.28**	1152
wc-6d-c1-obj2	**18.78**	630.96	86.10	**601.92**	83.56	607.18	247.50	601.94	1152
wc-c1-3d-c1-obj1	7.88	308.08	2.20	**293.34**	**1.70**	313.80	400.86	297.22	537
wc-c1-3d-c1-obj2	11.38	305.36	8.62	296.36	**8.18**	307.78	380.94	**295.26**	537
WGet	**3.70**	74.42	4.06	62.32	3.78	67.06	11.36	**60.36**	75
x264	14.22	453.94	9.72	**427.26**	**1.82**	468.04	86.94	432.38	818
XZ	11.52	248.98	14.56	250.82	18.46	249.54	**10.98**	**243.82**	431

Each experiment repeated for 50 individual runs. Then we collected the values of MAR and MMRE and calculated the Pearson correlation coefficient. The absolute value of the coefficient shows the strength of the correlation. A positive value means that one measurement increases with the other; a negative value means that one measurement increases when the other decreases. Table 5 shows the Pearson correlation coefficient between MAR and MMRE for the top rankings. We labeled the significant values (p-value<0.05) with $*$ and highlighted the values whose absolute values of correlation coefficients are higher than 0.6, which means strong correlation between MAR and MMRE.

As presented in Table 5, the top-1 configuration has stronger correlation than the top-5 and top-10 configurations. The more configurations we take from predicted sequences, the less number of scenarios of strong correlation we would have. For instance, CART has five scenarios of strong correlation in top-1 but only has two and one in top-5 and top-10. In top-1, CART, SVR, GBRT, and GPR have 5, 4, 6, and 5 scenarios of strong correlation out of 21 evaluation scenarios, respectively. It means MAR and MMRE may not have strong correlation for regression methods. Besides, SVR has a negative correlation between MAR and MMRE for most of significant scenarios, which is different from the other three regression methods.

Table 5. Pearson correlation coefficient on 21 scenarios of four regression methods. The mark * denotes the coefficient has p-value < 0.05.

Project	Top-1				Top-5				Top-10			
	CART	SVR	GBRT	GPR	CART	SVR	GBRT	GPR	CART	SVR	GBRT	GPR
AJStats	0.1929	-0.3878*	**0.8760***	0.3400*	0.2882*	-0.3500*	**0.6025***	0.4193*	0.1209	-0.1834	0.3132*	0.5578*
Apache	0.2363	-0.5089*	0.3028*	**0.7257***	0.2577	-0.2406	0.1390	0.4475*	0.1507	-0.1594	0.1255	0.5294*
BerkeleyC	**0.6092***	-0.2703	0.0192	-0.2037	0.2650	-0.2635	0.0769	-0.0721	-0.0761	-0.2365	0.0526	-0.0594
BerkeleyJ	0.4720*	-0.5279*	**0.7545***	0.3710*	0.3415*	-0.5902*	0.3960*	0.3766*	0.1297	-0.5623*	0.1247	0.3543*
clasp	**0.7202***	**-0.7174***	0.0626	**0.8735***	**0.6506***	-0.5339*	0.1743	**0.7088***	0.5672*	-0.4681*	0.2322	**0.6332***
Dune	0.3255*	**-0.9613***	**0.7844***	0.2381	0.4888*	**-0.6208***	0.2781	0.3103*	**0.6569***	-0.3618*	0.1407	0.4032*
HSMGP_num	-0.0163	**-0.8033***	0.2286	-0.1414	0.4586*	-0.5325*	0.1049	-0.0743	0.4032*	-0.5314*	-0.1091	-0.0395
lrzip	**0.6610***	-0.4326*	0.3091*	-0.0281	0.2278	-0.2810*	0.2769	0.1778	0.2602	-0.2354	0.1424	0.1828
noc	**0.6121***	-0.2536	0.0248	**0.8451***	0.5143*	-0.1117	N/A	**0.8481***	0.2964*	-0.1327	N/A	**0.8905***
snw	0.2423	-0.5291*	-0.2146	-0.0768	0.1591	-0.3073*	0.0480	0.2994*	0.0847	-0.0889	N/A	0.1206
spear	0.2691	-0.2080	-0.0716	-0.5949*	-0.1824	-0.1299	-0.0437	-0.4684*	-0.2601	-0.1335	-0.1131	-0.4651*
SQL	0.1724	0.2529	**0.8438***	0.1353	0.1438	-0.1746	**0.6293***	-0.0827	0.1952	-0.1272	0.5459*	0.2039
wc+rs-3d-c4-obj1	0.5080*	-0.2223	0.3822*	-0.209	0.5200*	**-0.6263***	0.4198*	-0.0922	0.5343*	-0.5536*	0.0040	-0.1581
wc+rs-3d-c4-obj2	**0.6604***	-0.4572*	-0.2554	**-0.6365***	**0.6475***	-0.3564*	-0.1413	-0.5552*	0.4188*	-0.3014*	-0.0684	-0.2407
wc-6d-c1-obj1	0.2732	-0.4223*	0.1438	-0.1919	0.2342	-0.3863*	0.2297	-0.1120	-0.0327	-0.2669	0.0694	-0.0013
wc-6d-c1-obj2	0.4586*	-0.3422*	-0.2649	-0.5077*	0.244	-0.2266	-0.184	-0.4362*	0.4112*	-0.2421	-0.1593	-0.4416*
wc-c1-3d-c1-obj1	0.3199*	0.3387*	0.2233	-0.0751	0.0664	-0.3066*	0.0856	0.2208	0.3260*	-0.3595*	-0.0887	0.3529*
wc-c1-3d-c1-obj2	0.1695	-0.3289*	**0.812***	-0.5458*	0.2465	-0.3552*	0.5888*	-0.4455*	-0.0588	-0.3955*	**0.7149***	-0.2672
WGet	0.3138*	**0.6341***	**0.6819***	0.3722*	0.3934*	-0.2516	0.3587*	0.2987*	0.4472*	-0.0445	0.3907*	0.1540
x264	0.2806*	-0.5289*	0.4741*	**0.6381***	0.5089*	-0.3099*	0.4471*	0.5058*	0.3811*	-0.3079*	0.3384*	0.4681*
XZ	-0.0780	-0.3862*	-0.2468	-0.1039	0.3236*	-0.5986*	-0.1742	0.0054	0.2098	**-0.6915***	-0.1401	-0.1965

Summary. From three RQs, we can understand the reason that many existing studies used CART as their regression method in performance prediction of software configurations. However, our study shows that all four regression methods can behave well in several evaluation scenarios. The rank-based approach for the performance ranking can be further improved to reduce the ranking difference and the number of sampled configurations.

5 Threats to Validity

We discuss threats to the validity to our comparative study in the following three dimensions.

Threats to construct validity. Our experiments used four regression methods to check the impact on performance ranking. It is feasible to conduct a large experiment via involving other existing regression methods. The dataset in our study is selected from existing work [13,8,21,7,16]. All projects in the dataset are real-world configurable systems. However, selecting the projects may lead to the bias of favoring specific application domains. This may be a threat to the project selection in the experiment.

Threats to internal validity. In the experiment, we repeated individual runs for 50 times to reduce the randomness of the dataset division, i.e., dividing a dataset into three subsets, the training pool, the validation pool, and the test pool. The 50 times of repetitive experiments is used to avoid the influence of randomness. However, the randomness cannot be totally removed.

Threats to external validity. We compared four regression methods in the study. We do not claim that the comparative results can be generalized to other projects or to other regression methods. Considering the large number of existing configurable systems, projects and regression methods can be viewed as a sample of performance ranking on real-world systems.

6 Related Work

In this paper, we studied the performance ranking with four regression methods. Previous studies on the performance of configurable systems generally focused on sampling methods and learning methods. We present the related work as follows.

Guo et al. [6] used the CART method to predict the performance of configurations. Their empirical results show that the method can reach a prediction accuracy of 94% on average based on small random samples. They claimed that prediction accuracy would robustly increase with the increasing of the sample size. Valov et al. [15] compared the predicting accuracy of four typical regression methods. They selected regressions methods, including CART, SVM, random forests, and bagging. Their results show that bagging does better in performance prediction when all configurations are evaluated with the relative errors. Siegmund et al. [13] proposed an approach that learns a performance-influence model for a configurable system. Their approach can flexibly combine both binary and numeric configuration options. Sarkar et al. [12] compared two sampling strategies, progressive sampling and projective sampling, with the prediction accuracy and the measurement effort and found that the projective sampling is better than the progressive sampling. Gu et al. [4] proposed a multi-objective optimization method for configuration sampling. This method considers both the number of configurations in the training set and the rank difference in the test set.

Zhang et al. [20] proposed a Fourier learning method to learn performance functions for configurable systems. They showed that this method can generate performance predictions with guaranteed accuracy at the confidence level. Nair et al. [9] proposed a fast spectral learner and three new sampling techniques. They conducted experiments on six real-world configurable software systems. Xuan et al. [19] designed a method genetic configuration sampling, which used a genetic algorithm to reduce the number of sampled configurations to reveal the internal system faults, which result in system crashes. Gu et al. [5] learned a predictive model to identify whether the root cause of crashing faults resides in the lines of stack traces. This model is built on the features of stack traces and faulty source code.

The rank-based approach by Nair et al. [8] reconsidered the problem of performance prediction and refined the problem as a ranking problem. Their work first used the rank difference instead of predicting accuracy as the measurement in the evaluation. They also proposed an iterative sampling strategy and conducted experiments on nine real-world systems to compare with two state-of-the-art residual-based approaches. The results show that the rank-based approach uses fewer sampled configurations than residual-based approaches and the rank-based approach is effective to find out the optimal configurations for most systems under evaluation.

7 Conclusions

It is challenging to find the optimum among numerous configurations for highly configurable systems. The rank-based approach aims to rank the optimal configurations to the top and to assist the decision by system administrators and users. To find out which regression method can rank the best configuration to the top, we compare four different regression methods in the rank-based approach. We conducted experiments on 21 evaluation scenarios of 16 real-world systems. Empirical results show that the decision tree method CART and the ensemble learning method GBRT can achieve better ranks while SVR can save more sampled configurations in the rank-based approach. Meanwhile, the results indicate that the minimum of actual ranks may not strongly correlate with the relative error.

In future work, we plan to design a new sampling strategy to maintain the predicting accuracy and reduce the number of measurements for training regression models. We also want to check other sampling strategies for the rank-based approach in the future.

Acknowledgments

The work is supported by the National Key R&D Program of China under Grant No. 2018YFB1003901, the National Natural Science Foundation of China under Grant Nos. 61872273 and 61502345, the Open Research Fund Program of CETC Key Laboratory of Aerospace Information Applications under Grant No. SXX18629T022, and the Advance Research Projects of Civil Aerospace Technology, Intelligent Distribution Technology of Domestic Satellite Information, under Grant No. B0301.

References

1. Breiman, L.: Classification and regression trees. Routledge (2017)
2. Drucker, H., Burges, C.J., Kaufman, L., Smola, A.J., Vapnik, V.: Support vector regression machines. In: Advances in neural information processing systems. pp. 155–161 (1997)
3. Friedman, J.H.: Stochastic gradient boosting. Computational statistics & data analysis **38**(4), 367–378 (2002)
4. Gu, Y., Chen, Y., Jia, X., Xuan, J.: Multi-objective configuration sampling for performance ranking in configurable systems. In: Proceedings of the the 26th Asia-Pacific Software Engineering Conference (APSEC 2019), Putrajaya, Malaysia, December 2-5, 2019 (2019)

5. Gu, Y., Xuan, J., Zhang, H., Zhang, L., Fan, Q., Xie, X., Qian, T.: Does the fault reside in a stack trace? assisting crash localization by predicting crashing fault residence. Journal of Systems and Software **148**, 88–104 (2019). https://doi.org/10.1016/j.jss.2018.11.004, https://doi.org/10.1016/j.jss.2018.11.004
6. Guo, J., Czarnecki, K., Apel, S., Siegmund, N., Wasowski, A.: Variability-aware performance prediction: A statistical learning approach. In: 2013 28th IEEE/ACM International Conference on Automated Software Engineering, ASE 2013, Silicon Valley, CA, USA, November 11-15, 2013. pp. 301–311 (2013). https://doi.org/10.1109/ASE.2013.6693089, https://doi.org/10.1109/ASE.2013.6693089
7. Jamshidi, P., Casale, G.: An uncertainty-aware approach to optimal configuration of stream processing systems. In: 24th IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, MASCOTS 2016, London, United Kingdom, September 19-21, 2016. pp. 39–48 (2016). https://doi.org/10.1109/MASCOTS.2016.17, https://doi.org/10.1109/MASCOTS.2016.17
8. Nair, V., Menzies, T., Siegmund, N., Apel, S.: Using bad learners to find good configurations. In: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2017, Paderborn, Germany, September 4-8, 2017. pp. 257–267 (2017). https://doi.org/10.1145/3106237.3106238, http://doi.acm.org/10.1145/3106237.3106238
9. Nair, V., Menzies, T., Siegmund, N., Apel, S.: Faster discovery of faster system configurations with spectral learning. Autom. Softw. Eng. **25**(2), 247–277 (2018). https://doi.org/10.1007/s10515-017-0225-2
10. Nair, V., Yu, Z., Menzies, T., Siegmund, N., Apel, S.: Finding faster configurations using FLASH. CoRR **abs/1801.02175** (2018), http://arxiv.org/abs/1801.02175
11. Rasmussen, C.E.: Gaussian processes in machine learning. In: Summer School on Machine Learning. pp. 63–71. Springer (2003)
12. Sarkar, A., Guo, J., Siegmund, N., Apel, S., Czarnecki, K.: Cost-efficient sampling for performance prediction of configurable systems (T). In: 30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015, Lincoln, NE, USA, November 9-13. pp. 342–352 (2015). https://doi.org/10.1109/ASE.2015.45, https://doi.org/10.1109/ASE.2015.45
13. Siegmund, N., Grebhahn, A., Apel, S., Kästner, C.: Performance-influence models for highly configurable systems. In: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015, Bergamo, Italy, August 30 - September 4, 2015. pp. 284–294 (2015). https://doi.org/10.1145/2786805.2786845, http://doi.acm.org/10.1145/2786805.2786845
14. Siegmund, N., Kolesnikov, S.S., Kästner, C., Apel, S., Batory, D.S., Rosenmüller, M., Saake, G.: Predicting performance via automated feature-interaction detection. In: 34th International Conference on Software Engineering, ICSE 2012, June 2-9, 2012, Zurich, Switzerland. pp. 167–177 (2012). https://doi.org/10.1109/ICSE.2012.6227196, https://doi.org/10.1109/ICSE.2012.6227196
15. Valov, P., Guo, J., Czarnecki, K.: Empirical comparison of regression methods for variability-aware performance prediction. In: Proceedings of the 19th International Conference on Software Product Line, SPLC 2015, Nashville, TN, USA, July 20-

24, 2015. pp. 186–190 (2015). https://doi.org/10.1145/2791060.2791069, http://doi.acm.org/10.1145/2791060.2791069
16. Valov, P., Petkovich, J., Guo, J., Fischmeister, S., Czarnecki, K.: Transferring performance prediction models across different hardware platforms. In: Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering, ICPE 2017, L'Aquila, Italy, April 22-26, 2017. pp. 39–50 (2017). https://doi.org/10.1145/3030207.3030216, http://doi.acm.org/10.1145/3030207.3030216
17. Xu, Y., Jia, X., Xuan, J.: Writing tests for this higher-order function first: Automatically identifying future callings to assist testers. In: Proceedings of the 11th Asia-Pacific Symposium on Internetware (Internetware 2019), Fukuoka, Japan, October 28-29, 2019 (2019)
18. Xuan, J., Cornu, B., Martinez, M., Baudry, B., Seinturier, L., Monperrus, M.: B-refactoring: Automatic test code refactoring to improve dynamic analysis. Information & Software Technology **76**, 65–80 (2016). https://doi.org/10.1016/j.infsof.2016.04.016, https://doi.org/10.1016/j.infsof.2016.04.016
19. Xuan, J., Gu, Y., Ren, Z., Jia, X., Fan, Q.: Genetic configuration sampling: Learning a sampling strategy for fault detection of configurable systems. In: In Proceedings of the 5th International Workshop on Genetic Improvement (GI@GECCO 2018), Kyoto, Japan, July 15-19, 2018 (2018). https://doi.org/10.1145/3205651.3208267, https://doi.org/10.1145/3205651.3208267
20. Zhang, Y., Guo, J., Blais, E., Czarnecki, K.: Performance prediction of configurable software systems by fourier learning (T). In: 30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015, Lincoln, NE, USA, November 9-13, 2015. pp. 365–373 (2015). https://doi.org/10.1109/ASE.2015.15, https://doi.org/10.1109/ASE.2015.15
21. Zuluaga, M., Krause, A., Püschel, M.: e-pal: An active learning approach to the multi-objective optimization problem. J. Mach. Learn. Res. **17**, 104:1–104:32 (2016), http://jmlr.org/papers/v17/15-047.html