An Empirical Study of Learning to Rank Techniques for Effort-Aware Defect Prediction

Xiao Yu[1,2], Kwabena Ebo Bennin[2], Jin Liu[1*], Jacky Wai Keung[2*], Xiaofei Yin[3], Zhou Xu[1]
[1]School of Computer Science, Wuhan University, Wuhan, China
[2]Department of Computer Science, City University of Hong Kong, Hong Kong, China
[3]School of Computer Science, Fudan University, Shanghai, China
*Corresponding authors: {jinliu@whu.edu.cn, jacky.keung@cityu.edu.hk}

***Abstract*—Effort-Aware Defect Prediction (EADP) ranks software modules based on the possibility of these modules being defective, their predicted number of defects, or defect density by using learning to rank algorithms. Prior empirical studies compared a few learning to rank algorithms considering small number of datasets, evaluating with inappropriate or one type of performance measure, and non-robust statistical test techniques. To address these concerns and investigate the impact of learning to rank algorithms on the performance of EADP models, we examine the practical effects of 23 learning to rank algorithms on 41 available defect datasets from the PROMISE repository using a module-based effort-aware performance measure (FPA) and a source lines of code (SLOC) based effort-aware performance measure ($Norm(P_{opt})$). In addition, we compare the performance of these algorithms when they are trained on a more relevant feature subset selected by the Information Gain feature selection method. In terms of FPA and $Norm(P_{opt})$, statistically significant differences are observed among these algorithms with BRR (Bayesian Ridge Regression) performing best in terms of FPA, and BRR and LTR (Learning-to-Rank) performing best in terms of $Norm(P_{opt})$. When these algorithms are trained on a more relevant feature subset selected by Information Gain, LTR and BRR still perform best with significant differences in terms of FPA and $Norm(P_{opt})$. Therefore, we recommend BRR and LTR for building the EADP model in order to find more defects by inspecting a certain number of modules or lines of codes.**

***Index Terms*—effort-aware defect prediction; learning to rank; empirical study; Scott-Knott ESD test.**

I. Introduction

Software defect prediction (SDP) has been an active research area in the field of software engineering attracting more and more attention from both industry and academia [1], [2]. SDP models predict whether a software module is defective based on some software features such as source lines of codes (SLOC) and McCabe's cyclomatic complexity. Accurate prediction results can help allocate limited testing resources by suggesting that software testers pay more attention on those predicted defective modules [3], [4], [5], [17], [18].

However, traditional SDP models [30], [32], [33] based on some binary classification algorithms are not sufficient for software testing in practice, since they do not distinguish between a module with many defects or high defect density (i.e., number of defects/lines of source codes) and a module with a small number of defects or low defect density [5], [6], [7], [34]. Clearly, both modules require a different amount of effort to inspect and fix, yet they are considered equal and allocated the same testing resources. Therefore, Mende et al. [8] proposed effort-aware defect prediction (EADP) models to rank software modules based on the possibility of these modules being defective, their predicted number of defects, or defect density.

Generally, EADP models are constructed by using learning to rank techniques [5]. These techniques can be grouped into three categories, i.e., the pointwise approach, the pairwise approach, and the listwise approach [11], [12], [29]. There exist a vast variety of learning to rank algorithms in literature. It is thus important to empirically and statistically compare the impact and effectiveness of different learning to rank algorithms for EADP. To the best of our knowledge, few prior studies [9], [10], [15], [16], [43] evaluated and compared the existing learning to rank algorithms for EADP.

Most of these studies however conducted their study with few learning to rank algorithms across a small number of datasets. Previous studies [15], [16], [43] conducted their study with as many as five EADP models and few datasets. For example, Jiang et al. [15] investigated the performance of only five classification-based pointwise algorithms for EADP on two NASA datasets. Nguyen et al. [43] investigated three regression based pointwise algorithm and two pairwise algorithms for EADP on five Eclipse CVS datasets.

In addition, the results of most studies were evaluated with an inappropriate or one type of performance measure and non-robust statistical tests. Evaluation of EADP models with AUC (Area Under the Curve), precision, recall and SRCC (Spearman Rank Correlation Coefficient) [43] is not suitable, because they do not take the effort into consideration [36], [37], [38]. Jiang et al. [15] and Yang et al. [10] employed one type of performance measure, i.e., three module-based effort-aware performance measures (Lift Chart (LC), Cumulative Lift Chart (CLC), and Fault-percentile-average (FPA)), while Mende et al. [16] and Bennin et al. [9] employed another type of performance measure, i.e., two SLOC-based effort-aware performance measures (P_{opt} and $Norm(P_{opt})$). Module-based performance measures evaluate how many defects can be found when we inspect a certain number of modules, while SLOC-based performance measures evaluate how many defects can be found when we inspect a certain number of lines of codes. It is thus crucial to investigate which learning to rank algorithm is best for different application scenarios, i.e., software testers may need to find more defects by inspecting a certain number of modules or lines of codes.

978-1-7281-0591-8/19/$31.00 © 2019 IEEE

Authorized licensed use limited to: Wuhan University. Downloaded on January 13,2025 at 08:00:44 UTC from IEEE Xplore. Restrictions apply.

Prior studies [9], [10] concluded that the effectiveness of different learning to rank algorithms are not significantly different from each other. These empirical findings may be susceptible to the statistical test techniques they employed in the experiments, such as the Friedman test with Nemenyi test in [9], [15], [16], because Ghotra et al. [19] pointed out that these techniques have some limitation for multiple comparison analysis. Lastly, previous results have shown that eliminating irrelevant features from the original dataset can improve the performance of predictive models [5], [31]. It is thus important to apply feature selection techniques to software defect datasets, since feature selection techniques are able to filter out irrelevant features by calculating the contributions of software features [25], [26], [35].

Considering the above issues, we address the following research questions using 41 releases of 11 available and commonly-used software projects from the PROMISE data repository [74].

(1) RQ1: Which is the best learning to rank algorithm for EADP?

For RQ1, we conduct an extensive comparative study on the impact of 6 classification-based pointwise learning to rank algorithms, 9 regression-based pointwise learning to rank algorithms, 4 pairwise learning to rank algorithms and 4 listwise learning to rank algorithms for EADP evaluated with the module-based effort-aware performance measure (FPA) and the SLOC-based effort-aware performance measure (*Norm*(P_{opt})). The experimental results show that Bayesian Ridge Regression (BRR) performs best in terms of FPA, and BRR and Learning to Rank (LTR) perform best in terms of *Norm*(P_{opt}). The experimental results are supported by the state-of-the-art multiple comparison technique, i.e., Scott-Knott ESD test [86].

(2) RQ2: Which is the best performing algorithm when trained on a relevant feature subset selected by Information Gain?

To filter out irrelevant features and significantly improve prediction performance, we adopt Information Gain as the feature selection method for this study, similar to the study in [5]. Experimental results show that eliminating irrelevant software features from the original dataset can improve the performance of EADP models. LTR and BRR perform best in terms of FPA and *Norm*(P_{opt}) when these algorithms are trained on a more relevant feature subset selected by Information Gain. The results are also supported by the Scott-Knott ESD test.

We recommend that software testers first employ Information Gain to eliminate irrelevant software features, and then use LTR and BRR to build the EADP model when they aim to inspect a certain number of modules or lines of code to find more defects.

The remainder of this paper is organized as follows. Section II briefly introduces the learning to rank algorithms. Section III and Section IV present the experiment setup and experiment results, respectively. Section V discusses the potential threats to validity. Section VI presents the related work. Finally, Section VII addresses the conclusion and points out the future work.

II. Background

A software module can be represented as $M_i=(\boldsymbol{x}_i,y_i)$, where $\boldsymbol{x}_i=(x_1, x_2,\ldots, x_m)$ is a m-dimensional software feature vector of the i-th module, and y_i is the number of defects in the i-th module. A software defect dataset can be represented as $S=\{M_i=(\boldsymbol{x}_i,y_i)\}_{i=1}^{n}$, where n is the number of modules in S. The goal of EADP is to learn from S to obtain a prediction model to rank new modules according to the possibility of these modules being defective, their predicted number of defects, or defect density, where $M_j > M_k$ means that the possibility of M_j being defective, the number of defects or defect density in M_j is larger than that in M_k.

In this section, we briefly introduce the 23 learning to rank algorithms due to the space limitation. These algorithms cover three families, including 15 pointwise algorithms, 4 pairwise algorithms, and 4 listwise algorithms. Figure 1 shows an overview of the three types of approaches, and Table I provides an overview of the 23 algorithms.

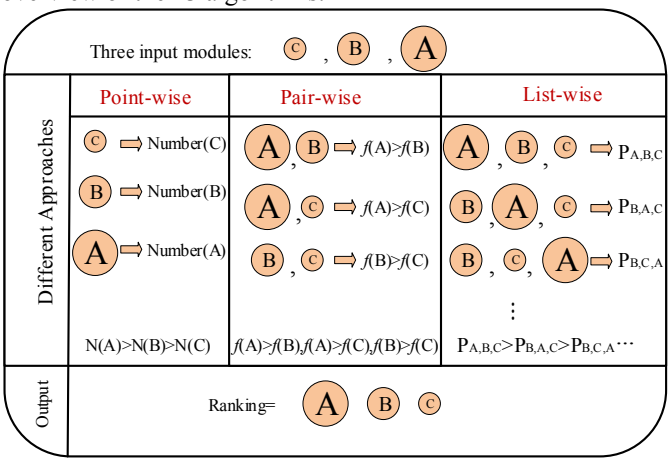

Figure 1. An overview of the three types of learning to rank approaches.

A. The Pointwise Approach

The pointwise approach tries to predict the possibility of a module being defective or the number of defects to rank the modules. As shown in Figure 1, there are three modules (i.e., A, B, and C) that need to be ranked. Assuming that the pointwise approach predicts that the number of defects in module A is Number(A), the number of defects in module B is Number(B), the number of defects in module C is Number(C), and Number(A)>Number(B)>Number(C), then the predicted ranking of A, B, and C is A > B > C. In the following, we first introduce the six classification-based learning to rank algorithms used in this study.

(1) Naïve Bayes (NB) [13]: It is a classification algorithm based on the Bayes' theorem with the "naive" assumption that every pair of software features are independent.

(2) Logistic regression (LogR) [54]: It is a classification algorithm to classify software modules into discrete outcomes. It maximizes the entropy of the labels conditioned on the software features with respect to the distribution.

(3) Classification and Regression Tree (CART) [55]: It partitions the training dataset into small segments using the Gini

Authorized licensed use limited to: Wuhan University. Downloaded on January 13,2025 at 08:00:44 UTC from IEEE Xplore. Restrictions apply.

TABLE I. PARAMETER VALUE OVERVIEW OF THE LEARNING TO RANK ALGORITHMS STUDIED IN OUR WORK

Family	Label	Algorithm	Parameter Description	Parameter Value
Classification-based Pointwise approach	1	Naïve Bayes(NB)	Laplace Correction	{0}
	2	Logistic Regression (LogR)	Tolerance for stopping criteria	{0.1,0.01,0.001,0.0001,0.00001}
	3	Classification and Regression Tree (CART)	The minimum number of samples required to split an internal node	{2 ,6,10,14,18}
			The minimum number of samples required to be at a leaf node	{1,3,5,7,9}
	4	Bagging	The number of base learners	{10,20,30,40,50}
	5	Random Forest (RF)	The number of trees in the forest	{10,20,30,40,50}
	6	K-nearest Neighbors (KNN)	Number of neighbors	{1,5,9,13,17}
Regression-based Pointwise approach	7	Decision Tree Regression (DTR)	The minimum number of samples required to split an internal node	{2 ,6,10,14,18}
			The minimum number of samples required to be at a leaf node	{1,3,5,7,9}
	8	Linear Regression (LR)	Whether the regressor will be normalized before	{true, false}
	9	Bayesian Ridge Regression (BRR)	Stop the algorithm if w has converged	{0.1,0.01,0.001,0.0001,0.00001}
	10	Neural Network Regression (NNR)	The size of hidden layers	{2,4,8,16,32,64}
			Size of minibatches for stochastic optimizers	{8,16,32,128,256}
	11	Support Vector Regression (SVR)	Penalty parameter C of the error term	{0.01,0.1,1,10,100}
	12	K-nearest Neighbors Regression (KNR)	The number of neighbors	{1,5,9,13,17}
	13	Gradient Boosting Regression (GBR)	The number of boosting stages to perform	{100,200,300,400,5000}
			The minimum number of samples required to split an internal node	{2 ,6,10,14,18}
			The minimum number of samples required to be at a leaf node	{1,3,5,7,9}
	14	Gaussian Process Regression (GPR)	The number of restarts of the optimizer for finding the kernel's parameters	{0,1,2,3,4}
	15	Stochastic Gradient Descent Regression (SDGR)	Constant that multiplies the regularization term	{0.1,0.01,0.001,0.0001,0.00001}
Pairwise approach	16	Ranking SVM	Penalty parameter C of the error term	{0.01,0.1,1,10,100}
	17	RankBoost	The number of rounds to train	{100, 200,300,400,500}
	18	RankNet	The number of epochs to train	{16,32,64,128,256}
	19	LambdaRank	The number of epochs to train	{16,32,64,128,256}
Listwise approach	20	ListNet	The number of epochs to train	{16,32,64,128,256}
	21	AdaRank	The number of rounds to train	{100, 200,300,400,500}
	22	Coordinate Ascent	The number of random restarts	{2,4,6,8,10}
	23	LTR	Feasible solution space	[-20,20]

index, and labels these small segments with one of the class labels (i.e., defective or non-defective).

(4) Bagging [56]: It is an ensemble classifier that fits a number of weak classifiers on the original dataset, and then combine them as a final strong classifier. In the experiment, we employ decision tree as the meta-classifier.

(5) Random Forest (RF) [14]: It is an ensemble classifier that fits a number of decision tree classifiers on various subsets of the original dataset, and use averaging to improve the predictive accuracy and control over-fitting.

(6) K-nearest Neighbors (KNN) [57]: It finds k training software modules closest to the new software module, and predicts the label of the new software module from these training modules.

Additionally, we introduce 9 regression-based learning to rank algorithms as follows.

(1) Decision Tree Regression (DTR) [58]. It builds a regression model in the form of a decision tree structure by learning from the training dataset.

(2) Linear Regression (LR) [59]. It trains a linear model:

$$y=\langle \boldsymbol{b}, \boldsymbol{x}\rangle + b_0 \quad (1)$$

where $\boldsymbol{b}=(b_1,b_2,\ldots,b_m)$ represents a m-dimensional vector of regression coefficients, $\boldsymbol{x}=(x_1, x_2,\ldots, x_m)$ is a m-dimensional software feature vector of the module, b_0 is the error term, and y is the number of defects or defect density in the module.

(3) Bayesian Ridge Regression (BRR) [60]. It is a probabilistic method that builds a regression model using Bayesian inference. It combines priori information about parameters (the coefficient of software features) with the observed training data to get the posterior distribution of the parameters.

(4) Neural Network Regression (NNR) [61]: It learns a nonlinear function approximator using backpropagation with no activation function in the output layer.

(5) Support Vector Regression (SVR) [62]: It produces a function $f(\boldsymbol{x})$ with at most ε-deviation from the target value y. Constructing an SVR model is formalized as solving:

$$\text{Minimize } \frac{1}{2}\|\boldsymbol{w}\|^2 \quad (2)$$

$$\text{subject to} \begin{cases} y_i - \langle \boldsymbol{w}, \boldsymbol{x_i}\rangle - b \le \varepsilon \\ \langle \boldsymbol{w}, \boldsymbol{x_i}\rangle + b - y_i \le \varepsilon \end{cases} \quad (3)$$

where $\boldsymbol{x}_i$ is the features of a module with target value y_i.

(6) K-nearest Neighbors Regression (KNR) [63]: It finds k training software modules closest to the new software module, and predicts the number of defects or the defect density of the new software module based the mean of the number of defects or the defect density of these nearest neighbors.

(7) Gradient Boosting Regression (GBR) [64]: It is a boosting regression method, which combines weak regression models to create a final strong regression model. In the experiment, we employ decision tree regression as the meta regression model.

Authorized licensed use limited to: Wuhan University. Downloaded on January 13,2025 at 08:00:44 UTC from IEEE Xplore. Restrictions apply.

(8) Gaussian Process Regression (GPR) [65]: It is a regression algorithm to undertake non-parametric regression with Gaussian processes.

(9) Stochastic Gradient Descent Regression (SDGR) [66]: It is a linear model fitted by minimizing a regularized empirical loss with SGD (Stochastic Gradient Descent).

B. The Pairwise Approach

The pairwise approach transforms EADP problem into a classification problem, i.e., learning a binary classifier f that can identity which module contains more defects or has higher defect density in a given module pair. As shown in Figure 1, assuming that the pairwise approach predicts that module A contains more defects than module B (i.e., $f(A)>f(B)$), module A contains more defects than module C (i.e., $f(A)>f(C)$), and module B contains more defects than module C (i.e., $f(B)>f(C)$), then the predicted ranking of A, B, and C becomes A>B>C. We introduce the four pairwise learning to rank algorithms used in this study as follows.

(1) Ranking SVM [67]: It first transforms the ranking problem into classification by computing $\boldsymbol{x}_1-\boldsymbol{x}_2$, where $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ are the feature vectors of a pair of modules (i.e., M_1 and M_2), and then uses SVM to classify $(\boldsymbol{x}_1-\boldsymbol{x}_2)$ into +1 or −1. If the class label is +1, M_1 contains more defects than M_2; otherwise, M_2 contains more defects than M_1.

(2) RankBoost [68]: It adopts AdaBoost to classify the modules pairs. The only difference between them is that the distribution is defined on modules pairs in RankBoost while that is defined on individual modules in AdaBoost. It aims to minimize the exponential loss on module pairs.

(3) RankNet [69]: The loss function of RankNet is also defined on module pairs, but the hypothesis is defined with the use of a scoring function, which is optimized by using the gradient descent method.

(4) LambdaRank [70]: LambdaRank optimizes an upgraded version of the loss function in RankNet with less computing complexity and better performance on measures using the gradient descent method.

C. The Listwise Approach

The listwise approach directly optimizes the performance measures to obtain a ranking model. As shown in Figure 1, assuming that the listwise approach predicts that the ranking list $P_{A,B,C}$ has the best performance measure among all possible ranking lists (i.e., $P_{A,B,C}$, $P_{A,C,B}$, $P_{B,A,C}$, $P_{B,C,A}$, $P_{C,A,B}$, and $P_{C,B,A}$), so the predicted ranking of A, B, and C will be A > B > C. We introduce the four pairwise learning to rank algorithms used in this study as follows.

(1) ListNet [71]: It uses a neural network approach with the gradient descent method to minimize a loss function, similar to RankNet. The loss function is defined using the probability distribution on all possible ranking lists.

(2) AdaRank [72]: It is another boosting method which combines weak rankers to create the final ranking model. AdaRank directly minimizes the performance measures by updating the distribution of software modules and computing the combination coefficient of the weak rankers.

(3) Coordinate Ascent [73]: It trains a ranking model by minimizing the mean average precision (MAP) values. It does a number of restarts to guarantee avoidance of the local minimum.

(4) Learning-to-Rank (LTR) [5]. It trains a simple linear model, i.e., $f(\boldsymbol{x})=\langle \boldsymbol{w}, \boldsymbol{x}\rangle$ by directly optimizing the FPA values using the composite differential evolution algorithm, and then ranks new modules based on the predicted relative number of defects or defect density.

III. Experimental Setup

A. Datasets

In this experiment, we employ 41 releases of 11 open source software projects, which can be obtained from the PROMISE data repository [74], [75], [76]. The details about the projects are shown in Table II, where *Module* represents the number of modules in the project, *#Defects* represents the total number of defects in the project, *%Defect* represents the percentage of defective modules in the project, and *Avg* is the average value of defects of all defective modules in the project. The 20 software features of the projects are listed in Table III.

TABLE II. Details of Experiment Dataset

Project	Release	Module	#Defects	%Defects	Avg
Ant	1.3,1.4,1.5,1.6,1.7	1692	637	20.7	1.82
Camel	1.0,1.2,1.4,1.6	2784	1371	20.2	2.44
Ivy	1.1,1.4,2.0	704	307	16.9	2.58
Jedit	3.2,4.0,4.1,4.2,4.3	1749	943	17.3	3.11
Log4j	1.0,1.1,1.2	449	645	57.9	2.48
Lucene	2.0,2.2,2.4	782	1314	56.0	3.0
Poi	1.5,2.0,2.5,3.0	1378	1377	51.3	1.95
Synapse	1.0,1.1,1.2	635	265	25.5	1.64
Velocity	1.4,1.5,1.6	639	731	57.4	1.99
Xalan	2.4,2.5,2.6,2.7	3320	2525	54.4	1.4
Xerces	init,1.2,1.3,1.4	1643	2071	39.8	3.17

TABLE III. Features of the Dataset

No.	Feature	Description
1	*wmc*	Weighted methods per class
2	*dit*	Depth of inheritance tree
3	*noc*	Number of children
4	*cbo*	Coupling between object classes
5	*rfc*	Response for a class
6	*lcom*	Lack of cohesion in methods
7	*ca*	Afferent couplings
8	*ce*	Efferent couplings
9	*npm*	Number of public methods
10	*lcom3*	Lack of cohesion in methods
11	*loc*	Lines of code
12	*dam*	Data access metric
13	*moa*	Measure of aggregation
14	*mfa*	Measure of functional abstraction
15	*cam*	Cohesion among methods of class
16	*ic*	Inheritance coupling
17	*cbm*	Coupling between methods
18	*amc*	Average method complexity
19	*max_cc*	Maximum McCabe's cyclomatic complexity
20	*avg_cc*	Average McCabe's cyclomatic complexity

Authorized licensed use limited to: Wuhan University. Downloaded on January 13,2025 at 08:00:44 UTC from IEEE Xplore. Restrictions apply.

B. Performance Measures

Menzies et al. [27], [28], [36], Kamei et al. [37], and D'Ambros et al. [38] suggested that software testers should consider the effort when testing the predicted defective software modules. Evaluation of EADP models with AUC, precision, recall, F-measure [43] is not suitable for evaluating the performance of EADP models, because they do not take the effort into consideration. Therefore, researchers have proposed some effort-aware performance measures to evaluate EADP models, such as cost effectiveness (*CE*) [39], [40], [41], P_{opt} [16], *Norm*(P_{opt}) [37], [20], cumulative lift chart (CLC) [15], and fault percentile average (FPA) [42].

CE, P_{opt} and *Norm*(P_{opt}) are SLOC based performance measures. The difference between *CE* and *Norm*(P_{opt}) is that *CE* compares the prediction model with the baseline model, while *Norm*(P_{opt}) compares the prediction model with the optimal model. That is, *CE* reports how much better than the baseline model a prediction model is, rather than tells us how closed to the optimal model a prediction model is. Therefore, we employ *Norm*(P_{opt}) as the performance measure in this paper.

$$(1)\ Norm(P_{opt}) = \frac{P_{opt}\text{-}\min(P_{opt})}{\max(P_{opt}) - \min(P_{opt})} \quad (4)$$

Here, P_{opt} is defined as 1-Δ_{opt}, where Δ_{opt} is the area between the optimal model (modules are ranked by decreasing actual defect densities) and the prediction model (modules are ranked by the decreasing predicted defect densities) in the SLOC-based cumulative lift chart (Figure 2). max(P_{opt}) is the P_{opt} value of the optimal model, while min(P_{opt}) is the P_{opt} value of the worst model (modules are ranked by increasing actual defect densities) in the SLOC-based cumulative lift chart (Figure 2). In this chart, the *x-axis* is the cumulative percentage of SLOC to inspect, and the *y-axis* is the cumulative percentage of defects found in the SLOC.

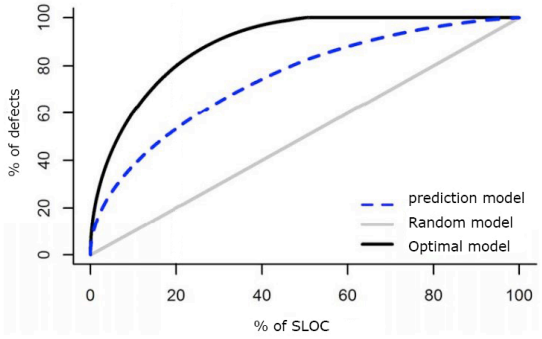

Figure 2. A SLOC-based cumulative lift chart.

CLC and FPA are module based performance measures. Yang et al. [5] have proved that FPA and CLC are linearly related. In the experiment, we also employ FPA to measure the performance, because Yang et al. [5] pointed out that FPA is the state of the art performance measure for evaluating EADP models.

FPA is the average of the proportions of actual defects in the top modules to the all defects in the defect dataset [5]. A higher FPA means a better ranking, where the modules with most defects are ranked first [5]. Assume that *n* modules in a software defect dataset are ranked by increasing order of the predicted number of defects, as M_1, M_2, M_3, …, M_n, and $Y=y_1+y_2+,\ldots,+y_n$ is the total number of defects in the software defect dataset. Therefore, M_n is predicted to contain most defects. The proportion of the actual defects in the top *m* predicted modules to the whole defects is:

$$\frac{1}{Y}\sum_{i=n-m+1}^{n} y_i. \quad (5)$$

Then, FPA is define as:

$$\frac{1}{n}\sum_{m=1}^{n}\frac{1}{Y}\sum_{i=n-m+1}^{n} y_i. \quad (6)$$

C. Experimental Procedure

We employ out-of-sample bootstrap validation technique recommended by Tantithamthavorn et al. [77], because it has been suggested to generate the best balance between the bias and variance of training and testing datasets. The out-of-sample bootstrap process is made up of the following three steps:

(1) *N* bootstrap modules are selected at random with replacement from an original defect dataset, where *N* is the number of software modules in the original defect dataset.

(2) An EADP model is trained using the bootstrap modules (i.e., training data). On average, 36.8% modules in the original dataset will not appear in the bootstrap modules.

(3) We calculate the *Norm*(P_{opt}) and FPA values for each learning to rank algorithm tested on the modules of the original defect dataset that do not appear in the bootstrap modules.

The experimental procedure is shown in Figure 3. We repeat the out-of-sample bootstrap 20 times. For the first research question, we compare the 23 learning to rank algorithms. The parameter configurations for each algorithm are presented in Table I. In this experiment, we use the grid search [78] to tune parameters, because it is commonly used in the field of software engineering [22], [23], [24], [79], [80], [81], [82].

For the second research question, we use Information Gain to investigate the effectiveness of different features on the experimental results. We employ Information Gain for the following reasons: (1) a number of empirical studies [83], [84], [85] have demonstrated the effectiveness of Information Gain for defect prediction; (2) empirical validation of feature selection for EADP is limited in literature. Only Yang et al. [5] applied Information Gain to EADP models. Therefore, we also employ Information Gain in this paper. In this paper, we adopt the iterative subset, by selecting the top 2,3,...,18,19 top features. For the implementations of the pointwise algorithms and Information Gain, we use the python machine learning library *sklearn* to avoid the potential faults as much as possible. For the implementation of the pairwise and listwise algorithms except LTR, we use the source code provided by Microsoft [91]. We carefully implemented LTR following the original paper [5], since the authors did not provide the source codes.

It is worth noting that we use the defect density (number of defects/SLOC) as the target variable and 19 software features (except SLOC) to build the EADP model when the model is evaluated with *Norm*(P_{opt}). We first discretize the number of defects into "defective" and "non-defective" classes, and further use these classes to train the classification-based pointwise learning to rank algorithms to build the EADP model.

Authorized licensed use limited to: Wuhan University. Downloaded on January 13,2025 at 08:00:44 UTC from IEEE Xplore. Restrictions apply.

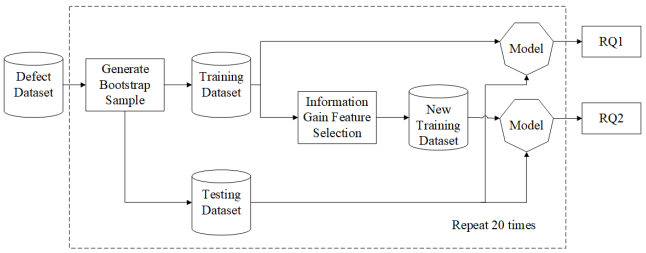

Figure 3. The experimental procedure.

D. Statistical Comparison Tests

The Scott-Knott test [87] is a multiple comparison technique that produces statistically distinct ranks at the significance level of 0.05 (α=0.05) using hierarchical clustering algorithm. This test ranks and clusters the learning to rank algorithms into significantly different groups, in which the learning to rank algorithms in distinct groups have significant differences while the learning to rank algorithms in the same group have no significant differences [31]. Therefore, the Scott-Knott test can group the learning to rank algorithms distinctly without any overlapping [88]. For more robust result analysis, we use the extended Scott-Knott with Cohen's *d* effect size awareness (Scott-Knott ESD) [86], which merges any pair of ranks that have a negligible Cohen's *d* effect size between them to post-processes the statistically distinct ranks produced by the traditional Scott-Knott test [86].

We use the double Scott-Knott test [89] to divide these learning to rank algorithms into different groups (α=0.05). The double Scott-Knott test contains two steps (shown in Figure 4): Initially, we provide the FPA and *Norm*(P_{opt}) values of the 20 bootstrap iterations of each learning to rank algorithms on each dataset to the Scott-Knott ESD test. This results in 41 different Scott-Knott ESD ranks (i.e, one from each dataset) for each learning to rank algorithm. Furthermore, we obtain the final rankings of these algorithms across all of the studied datasets with the 41 different Scott-Knott ESD ranks being the input to the Scott-Knott ESD test.

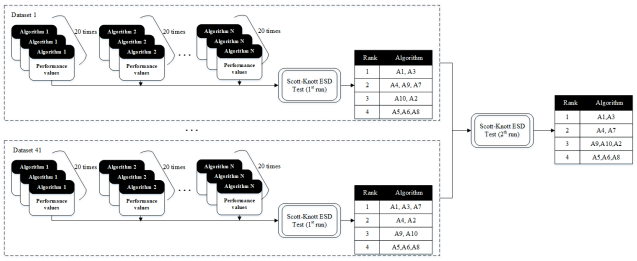

Figure 4. The procedure of the double Scott-Knott test.

IV. Experiment Results

In this section, the experimental results and answers to the two research questions in Section I are presented.

A. RQ1: Which is the best learning to rank algorithm for EADP?

To answer this question, we compare 23 learning to rank algorithms. The boxplots in Figure 5 show the distribution of FPA values of each algorithm with the Scott-Knott ESD test results across all studied datasets. Different colors of the boxplot indicate different Scott-Knott ESD test ranks. From top down, the order is red, pink, rose red, yellow, orange, chocolate, blue, sky blue, green, purple, gray, black. Table IV reports the algorithms that belong to the same group and the statistical properties of the algorithm rankings for each group in terms of FPA, including the median ranking, average ranking and standard deviation.

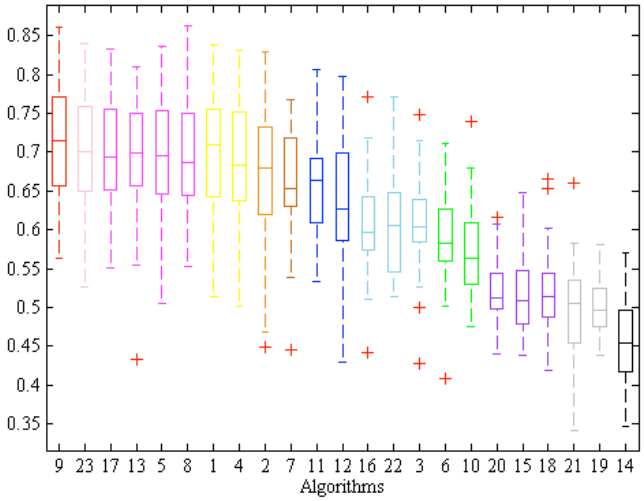

Figure 5. The boxplots of the FPA values.

TABLE IV. Statistical Results in Terms of FPA

Overall Ranking	Algorithms	Median Ranking	Average Ranking	Standard Deviation
1	BRR	2.07	2.07	0
2	LTR	2.56	2.56	0
3	RankBoost, GBR, RF, LR	2.92	2.94	0.080
4	NB, Bagging	3.29	3.45	0.159
5	LogR	4	4	0
6	DTR	4.97	4.97	0
7	SVR, KNR	5.46	5.65	0.033
8	Ranking SVM, Coordinate Ascent, CART	7.10	7.07	0.072
9	KNN, NNR	7.80	7.85	0.049
10	ListNet, SGDR, RankNet	9.66	9.62	0.172
11	AdaRank, LambdaRank	10.22	10.23	0.012
12	GPR	11.15	11.15	0

As shown in the Figure 5 and Table IV, we observe that the 23 learning to rank algorithms are clustered into twelve distinct groups without overlapping, which implies that there exist clear separations between these algorithms. As shown in Figure 5 and Table IV, BRR obtains the best ranking among all learning to rank algorithms in terms of FPA. LTR has a higher ranking than other learning to rank algorithms except BRR. A pairwise algorithm (RankBoost), two regression based pointwise algorithm (GBR and LR), a classification based pointwise algorithm (RF) belong to the third group. In addition, three classification based pointwise algorithms (NB, Bagging and LogR) perform well and belong to the fourth and fifth groups, respectively.

Authorized licensed use limited to: Wuhan University. Downloaded on January 13,2025 at 08:00:44 UTC from IEEE Xplore. Restrictions apply.

The boxplots in Figure 6 show the distribution of *Norm*(P_{opt}) values of each algorithms with the Scott-Knott ESD test results across all studied datasets. Table V reports the algorithms that belong to the same group and the statistical properties of the algorithm rankings for each group. As shown in Figure 6 and Table V, BRR and LTR attain the best ranking. GBR and LR (two regression based pointwise algorithms) belong to the second and third group, respectively.

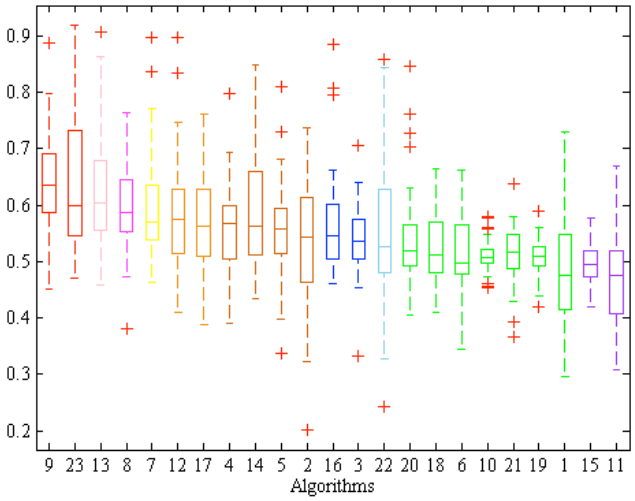

Figure 6. The boxplots of the *Norm*(P_{opt}) values.

TABLE V. STATISTICAL RESULTS IN TERMS OF NORM(POPT)

Overall Ranking	Algorithms	Median Ranking	Average Ranking	Standard Deviation
1	BRR, LTR	2.80	2.89	0.085
2	GBR	3.41	3.41	0
3	LR	3.85	3.85	0
4	DTR	4.51	4.51	0
5	KNR, RankBoost	5.06	5	0.06
6	Bagging, GPR, RF, LogR	5.5	5.439	0.107
7	Ranking SVM, CART	6.05	5.92	0.122
8	Coordinate Ascent	6.59	6.59	0
9	ListNet, RankNet, KNN, NNR, AdaRank, LambdaRank, NB	7.13	7.146	0.159
10	SGDR, SVR	7.90	7.83	0.073

In summary, BRR performs best among all learning to rank algorithms in terms of FPA, and BRR and LTR perform best among all learning to rank algorithms in terms of *Norm*(P_{opt}). The result is supported by the Scott-Knott ESD test.

B. Which is the best performing algorithm when trained on a relevant feature subset selected by Information Gain?

This question aims to explore whether the conclusion of RQ1 is consistent after removing irrelevant software features from the original datasets using Information Gain method. We adopt the iterative subset, by selecting the top 2,3,...,18,19 features. Since the average FPA and *Norm*(P_{opt}) values of all learning to rank algorithms on all datasets are highest when all algorithms are trained on top 10 features, we select the top 10 features following the setup in [90]. Due to the space limit, we do not list the detail FPA and *Norm*(P_{opt}) values of each algorithm trained on each feature subset. The average FPA value of all algorithms trained on top 10 features is 0.629, which is higher than that (0.617) of all algorithms trained on original datasets. The average *Norm*(P_{opt}) value of all algorithms trained on top 10 features is 0.552, which is higher than that (0.549) of all algorithms trained on original datasets.

The boxplots in Figure 7 show the distribution of FPA values of each algorithm trained on the top 10 features across all studied datasets with the Scott-Knott ESD test results. Table VI reports the algorithms that belong to the same group and the statistical properties of the algorithm rankings for each group in terms of FPA. As shown in the Figure 7 and Table VI, BRR and LTR belong to the first group. LR and RankBoost belong to the second and third group, respectively. Three classification based pointwise algorithms (NB, LogR and RF) perform well and belong to the fourth group.

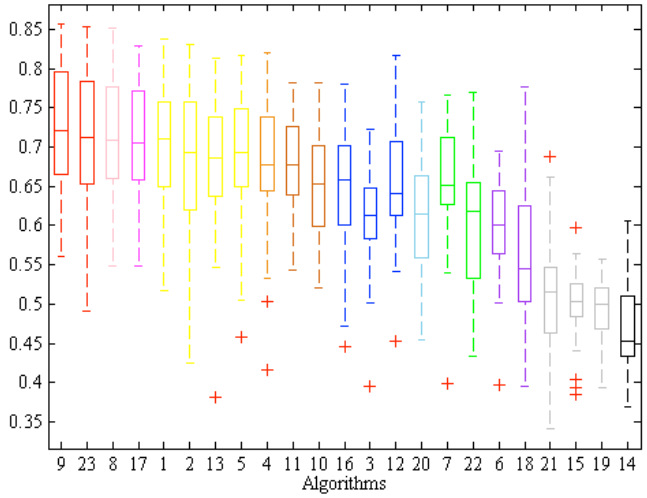

Figure 7. The boxplots of the FPA values when the algorithms are trained on the top 10 features.

TABLE VI. STATISTICAL RESULTS IN TERMS OF FPA WHEN THE ALGORITHMS ARE TRAINED ON TOP 10 FEATURES

Overall Ranking	Algorithms	Median Ranking	Average Ranking	Standard Deviation
1	BRR, LTR	1.98	1.95	0.024
2	LR	2.51	2.51	0
3	RankBoost	2.80	2.80	0
4	NB, LogR, GBR, RF	3.61	3.61	0.062
5	Bagging	4.22	4.22	0
6	SVR, NNR	4.80	4.88	0.073
7	Ranking SVM, DTR, KNR	5.36	5.37	0.090
8	ListNet	6.61	6.61	0
9	CART, Coordinate Ascent	7.17	7.23	0.061
10	KNN, RankNet	7.68	7.90	0.220
11	AdaRank, SGDR, LambdaRank	9.537	9.577	0.172
12	GPR	10.366	10.366	0

The boxplots in Figure 8 show the distribution of *Norm*(P_{opt}) values of each algorithm trained on the top 10 features across all studied datasets with the Scott-Knott ESD test results. Table VII reports the algorithms that belong to the

Authorized licensed use limited to: Wuhan University. Downloaded on January 13,2025 at 08:00:44 UTC from IEEE Xplore. Restrictions apply.

same group and the statistical properties of the algorithm rankings for each group in terms of *Norm*(P_{opt}). As shown in the Figure 8 and Table VII, LTR and BRR belong to the first group, and GBR and LR belong to the second group. RankBoost, RF, DTR and LogR belong to the third group.

In summary, LTR and BRR perform best among all learning to rank algorithms trained on the top 10 features in terms of FPA and *Norm*(P_{opt}). The result is supported by the Scott-Knott ESD test.

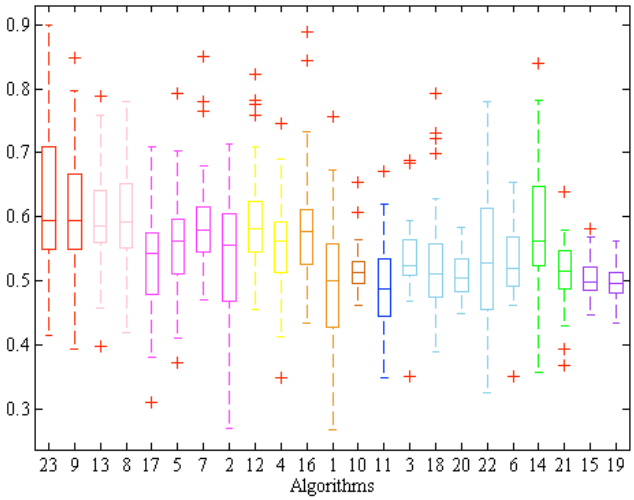

Figure 8. The boxplots of the *Norm*(P_{opt}) values when these algorithms are trained on the top 10 features.

TABLE VII. STATISTICAL RESULTS IN TERMS OF NORM(POPT) WHEN THE ALGORITHMS ARE TRAINED ON TOP 10 FEATURES

Overall Ranking	Algorithms	Median Ranking	Average Ranking	Standard Deviation
1	LTR, BRR	2.34	2.89	0.146
2	GBR, LR	3.22	3.22	0
3	RankBoost, RF, DTR, LogR	4.29	4.24	0.128
4	KNR, Bagging	5.56	4.57	0.012
5	Ranking SVM, NB	4.80	4.91	0.012
6	NNR	5.39	5.39	0
7	SVR	5.59	5.59	0
8	CART, RankNet, ListNet, Coordinate Ascent, KNN	6.63	6.59	0.148
9	GPR, AdaRank	7.32	7.49	0.171
10	SGDR, LambdaRank	7.92	8.02	0.097

C. Discussion

The performance of the three types of learning to rank algorithms might be explained via the difference of the training utility function.

(1) As mentioned in Section III, the listwise approach directly optimizes the performance measure to obtain a ranking function. LTR performs well in terms of FPA (LTR belongs to the second group and first group, when it is trained on all feature sets and top-10 feature subsets, respectively), because it directly optimizes the FPA value. LTR also performs well for *Norm*(P_{opt}), because *Norm*(P_{opt}) and FPA are both effort-aware performance measures. The only difference between *Norm*(P_{opt}) and FPA is that *Norm*(P_{opt}) is SLOC-based, and FPA is module-based. However, other listwise algorithms have poor performance. The reason might be that the goal of these algorithms is to optimize some information retrieval performance measures, such as mean average precision (MAP), which do not take the effort into consideration [36], [37], [38].

(2) Some regression-based pointwise algorithms perform well in terms of FPA, such as BRR, GBR and LR. These algorithms outperform other regression algorithms for several reasons. BRR and LR are multiple linear regression models, whereas the other regression algorithms are not linear models. Furthermore, there is a strong linear relationship between software features and the number of defects [5]. Therefore, the two algorithms have strong capability to identify and build the relationship between the software features and the number of defects. In addition, there exist strong correlations among the software features, i.e., multicollinearity [10]. BRR can reduce multicollinearity when constructing EADP models [10]. This is one reason why BRR performs best. GBR is an ensemble learning algorithm, which grow an ensemble of regression trees and allow them to vote on the decision to improve the performance.

(3) Some classification-based pointwise algorithms perform well in terms of FPA, such as RF, NB, Bagging, LogR (belong to the third or fourth group). In contrast to current practices in defect prediction studies, the classification-based pointwise algorithms lead to better performance when the number of defects in the defect datasets is not used to build prediction models. The finding is in agreement with a recent study [21], which found that building defect prediction classifiers using the number of defects does not always lead to better performance. One possible reason for other types of learning to rank algorithms using the number of defects having poorer performance than these classification-based pointwise algorithms is that the number of defects in each module in these datasets is highly imbalanced. That is, the modules with many defects occupy only a small part of this project, whereas the defect-free modules occupy a great part of this project, followed by the modules with one defect. These imbalanced datasets can be better handled by these classification-based algorithms, as they employ the class labels instead of the information of the number of defects. In addition, Bagging and RF are ensemble learning algorithms, which grow an ensemble of classification trees and allow them to vote on the decision to handle the data imbalance problem.

(4) In most cases, the pairwise algorithms have poor performance. The reasons might be as follows. The goal of pairwise algorithms is to minimize the number of incorrect rankings. Here, an incorrect ranking means that a module with less defects or lower defect density is ranked ahead of a module with more defects or higher defect density in a given module pair. When a model ranks the modules with more defects or higher defect density correctly, the model will obtain higher FPA value or *Norm*(P_{opt}) value. Therefore, EADP models should rank the modules with more defects or higher defect density correctly, and a higher cost should be assigned to the incorrect ranking of a module with more defects or higher defect density than a module with less defects or lower defect density. Subsequently, pairwise algorithms allocate the same costs for incorrect ranked modules with more defects or higher

Authorized licensed use limited to: Wuhan University. Downloaded on January 13,2025 at 08:00:44 UTC from IEEE Xplore. Restrictions apply.

TABLE VIII. RELATED WORKS ABOUT EMPIRICAL STUDIES FOR EADP

Study	Datasets used	Learning to Rank Techniques	Performance Measures	Statistical Tests	Main Findings
Jiang et al. [15]	Corpus: NASA Number of datasets: 8	NB, LogR, KNN, C4.5, Bagging	CLC	Friedman test, Nemenyi test	KNN outperforms others on the PC1 dataset, and C4.5 outperforms others on the KC2 dataset.
Mende et al. [16]	Corpus: NASA Number of datasets: 13	NB, LogR, CART, Bagging, RF	*CE*, P_{opt}	Nemenyi's post-hoc test	Bagging outperforms others.
Nguyen et al. [43]	Corpus: Eclipse CVS Number of datasets: 5	KNR, LR, MARS (multivariate adaptive regression splines), Ranking SVM, RankBoost	SRCC	None	RankBoost has more stable prediction performance.
Bennin et al. [9]	Corpus: Open Source Software Number of datasets: 25	LR, LAR (least angel regression), RVM (relevance vector machine), KNR, K*, NNR, SVR, DTR, GBR, RF	$Norm(P_{opt})$	Nemenyi test	K* outperforms best when using cross-validation setup, M5 performs best when using cross-release setup. There is not statistically significant difference among all compared algorithms.
Yang et al. [10]	Corpus: PROMISE Number of datasets: 41	LAR (lasso regression), RR, NBR (negative binomial regression), PCR (principal component regression), RF, LTR	CLC, FPA	Wilcoxon rank-sum test	RR can achieve better results than LR and NBR, slightly (not significantly) better results than LAR, PCR and LTR, and slightly worse results than RF when using cross-release setup.
Our study	Corpus: PROMISE Number of datasets: 41	NB, LogR, CART, Bagging, RF, KNN, DTR, LR, BRR, NNR, SVR, KNR, GBR, GPR, SDGR, Ranking SVM, RankBoost, RankNet, LambdaMart, ListNet, AdaRank, Coordinate Ascent, LTR	FPA, $Norm(P_{opt})$	Scott-Knott ESD test	In terms of FPA and $Norm(P_{opt})$, statistically significant differences are observed among these algorithms with BRR (Bayesian Ridge Regression) performing best in terms of FPA, and BRR and LTR performing best in terms of $Norm(P_{opt})$. When these algorithms are trained on a more relevant feature subset selected by Information Gain, LTR and BRR still perform best with significant differences in terms of FPA and $Norm(P_{opt})$.

defect density and incorrect ranked modules with one defect or lower defect density. That is, the training utility function of these pairwise algorithms also do not take the effort into consideration.

V. THREATS TO VALIDITY

In this section, we discuss several validity threats that may have impacted the results of our empirical studies.

External validity. Threats to external validity occur when the results of our experiments cannot be generalized. Although these datasets have been widely used in many software defect prediction studies [75], [76], we cannot generalize the results for all datasets especially commercial datasets. Additionally, we acknowledge the existence of several prediction models. Our study employed 23 prediction models which is sufficient for an empirical study. Adoption of other prediction models not used in this study is left for a future study.

Internal validity. Threats to internal validity refer to the bias of the choice of learning to rank algorithms and feature selection method. The reasons we employ the learning to rank algorithms are as follows: (1) Our selection of the classification-based pointwise algorithms closely resembles the choice by Jiang et al. [15] and Mende et al. [16]. (2) Our selection of the regression-based pointwise algorithms closely resembles the choice by Chen et al. [52] and Rathore et al. [53]. (3) Our selection of the pairwise and listwise algorithms closely resembles the choice by Nguyen et al. [43] and Shi et al. [11]. Shi et al. [11] investigated the same pairwise and listwise algorithms for bug localization, which is also a research hotspot in the field of software engineering [92]. We use Information Gain as the feature selection method following the work in [5]. In addition, with regards to the experimental implementation, the results could be influenced by the parameters used and experiment setup.

Construct validity. In our experiments, we use FPA and $Norm(P_{opt})$ as the evaluation measures, because the former is module-based effort-aware performance measure, and the latter is SLOC-based effort-aware performance measure. The two performance measures can investigate which learning to rank algorithm is best for different application scenarios, i.e., software testers may need to find more defects by inspecting a certain number of modules or lines of codes.

Conclusion validity. Threats to conclusion validity focus on the statistical analysis method. In this work, we use Scott-Knott ESD test to statistically analyze the learning to rank algorithms, because Tantithamthavorn et al. [87] have suggested that the Scott-Knott ESD test is superior to other post-hoc tests.

VI. RELATED WORK

Table VIII compares our study with the prior empirical studies for EADP. As shown in Table VIII, these prior studies have some limitations: (1) Comparison of few learning to rank algorithms considering small number of datasets. For example, Nguyen et al. [43] investigated five algorithms on five open

Authorized licensed use limited to: Wuhan University. Downloaded on January 13,2025 at 08:00:44 UTC from IEEE Xplore. Restrictions apply.

source software projects. (2) Inappropriate or only one type of performance measure. For example, Nguyen et al. [43] employed Spearman rank correlation coefficient as the performance measures. Jiang et al. [15] and Yang et al. [10] employed three module-based effort-aware performance measures, i.e., LC, CLC, and FPA. Mende et al. [16] and Bennin et al. [9] employed two SLOC-based effort-aware performance measures, i.e., P_{opt} and $Norm(P_{opt})$. (3) Inappropriate statistical test techniques. Prior studies employ inappropriate statistical test techniques, such as the Friedman test and Nemenyi test in [9], [15], [16]. Ghotra et al. [64] pointed out that these statistical test techniques have limitations for multiple comparison analysis. The results of prior studies are inconsistent, since the studies were conducted under different conditions, e.g., datasets of different domains, using different learning to rank algorithms, using different experiment setups.

It is worthy to mention that in recent years, various regression algorithms have been applied to predict the number of defects, including Poisson regression (PR) [44], [45], [46], [47], genetic programming (GP) [48], [49], [50], decision tree regression (DTR) [51], etc. In addition, Chen et al. [52] and Rathore et al. [53] performed an empirical study of some regression algorithms for predicting the number of defects, and found that DTR, LR, and BRR achieved better root mean square error (RMSE) and average absolute error (AAE) values. However, Yang et al. [5] pointed out that these approaches with higher predictive accuracy (smaller RMSE or AAE value) may result in a worse ranking of modules. Therefore, we revisit the impact of these regression-based learning to rank algorithms for EADP by using $Norm(P_{opt})$ and FPA as the performance measures.

VII. Conclusion and Future Work

Effort-Aware defect prediction (EADP) models can help to allocate testing resources more efficiently in the absence of testing resources. A number of learning to rank algorithms have been used for building EADP models. However, it is still a challenge on deciding on the best performing learning to rank algorithms for EADP. Accordingly, in this paper, we conduct a large-scale empirical study to investigate the impact of 23 learning to rank algorithms for EADP. In order to obtain a comprehensive evaluation, we use both a module-based effort-aware performance measure (FPA) and a SLOC-based effort-aware performance measure ($Norm(P_{opt})$) to compare the prediction performance of the 23 algorithms. The experimental results show that BRR performs best in terms of FPA, and BRR and LTR perform best in terms of $Norm(P_{opt})$ among the 23 algorithms when they are trained on original feature subset, and LTR and BRR still perform best when they are trained on the top 10 features. In the future, we also plan to employ more project datasets to validate the generalization of our findings.

Acknowledgment

This work is supported in part by the National Key R&D Program of China (No.2018YFC1604000), the grands of the National Natural Science Foundation of China (61572374, U163620068, U1135005, 61572371, 61772525), the Open Fund of Key Laboratory of Network Assessment Technology from CAS, Guangxi Key Laboratory of Trusted Software (No.kx201607), the Academic Team Building Plan for Young Scholars from Wuhan University (WHU2016012), the General Research Fund of the Research Grants Council of Hong Kong (No. 11208017), the research funds of City University of Hong Kong (No. 9678149 and 7005028), and the Research Support Fund by Intel.

References

[1] Yu X, Wu M, Jian Y, et al. Cross-company defect prediction via semi-supervised clustering-based data filtering and MSTrA-based transfer learning. Soft Computing, 2018, 22(10): 3461-3472.

[2] Bennin K E, Keung J, Monden A, et al. The significant effects of data sampling approaches on software defect prioritization and classification. Proceedings of the 11th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. IEEE Press, 2017: 364-373.

[3] M. Shepperd, D. Bowes, and T. Hall. Researcher Bias: The Use of Machine Learning in Software Defect Prediction. IEEE Transactions on Software Engineering,40(6):603-616, 2014.

[4] Yu X, Liu J, Peng W, et al. Improving Cross-Company Defect Prediction with Data Filtering. International Journal of Software Engineering and Knowledge Engineering, 2017, 27(09n10): 1427-1438.

[5] X. Yang, K. Tang, and X. Yao. A Learning-to-Rank Approach to Software Defect Prediction. IEEE Transactions on Reliability,64(1): 234-246, 2015.

[6] C. Catal. Software fault prediction: A literature review and current trends. Expert systems with applications, 38(4): 4626-4636, 2011.

[7] R. Malhotra. A systematic review of machine learning techniques for software fault prediction, Applied Soft Computing, 27: 504-518, 2015.

[8] T. Mende, R. Koschke. Effort-Aware Defect Prediction Models. European Conference on Software Maintenance and Reengineering, IEEE Computer Society Press, 2010; 109–118.

[9] Bennin K E, Toda K, Kamei Y, et al. Empirical evaluation of cross-release effort-aware defect prediction models. Software Quality, Reliability and Security, 2016 IEEE International Conference on. IEEE, 2016: 214-221.

[10] Yang X, Wen W. Ridge and Lasso Regression Models for Cross-Version Defect Prediction. IEEE Transactions on Reliability, 2018, 67(3): 885-896.

[11] Shi Z, Keung J, Bennin K E, et al. Comparing learning to rank techniques in hybrid bug localization. Applied Soft Computing, 2018, 62: 636-648.

[12] Liu T Y. Learning to rank for information retrieval, International ACM SIGIR Conference on Research and Development in Information Retrieval. ACM, 2010:904-904.

[13] Rish I. An empirical study of the naive Bayes classifier. Journal of Universal Computer Science, 2001, 1(2):127.

[14] Liaw A, Wiener M, Liaw A. Classification and Regression by Random Forests. R News, 2002, 23(23).

[15] Jiang Y, Cukic B, Ma Y. Techniques for evaluating fault prediction models, Empirical Software Engineering, 2008, 13(5):561-595.

[16] Mende T, Koschke R. Revisiting the evaluation of defect prediction models, International Conference on Predictor MODELS in Software Engineering. ACM, 2009:1-10.

[17] Xu Z, Liu J, Luo X, et al. Cross-version defect prediction via hybrid active learning with kernel principal component analysis. 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER). IEEE, 2018: 209-220.

Authorized licensed use limited to: Wuhan University. Downloaded on January 13,2025 at 08:00:44 UTC from IEEE Xplore. Restrictions apply.

[18] Bennin K E, Keung J, Phannachitta P, et al. Mahakil: Diversity based oversampling approach to alleviate the class imbalance issue in software defect prediction. IEEE Transactions on Software Engineering, 2018, 44(6): 534-550.

[19] B. Ghotra, S. McIntosh, and A. E. Hassan. Revisiting the impact of classification techniques on the performance of defect prediction models. In Proceedings of the 37th International Conference on Software Engineering (ICSE). IEEE, 789-800, 2015.

[20] Bennin, K.E., Keung, J., Monden, A., Kamei, Y. and Ubayashi, N., 2016, June. Investigating the effects of balanced training and testing datasets on effort-aware fault prediction models. In Computer Software and Applications Conference (COMPSAC), 2016 IEEE 40th Annual (Vol. 1, pp. 154-163). IEEE.

[21] Rajbahadur G K, Wang S, Kamei Y, et al. The impact of using regression models to build defect classifiers. Proceedings of the 14th International Conference on Mining Software Repositories. 2017: 135-145.

[22] Tantithamthavorn C, McIntosh S, Hassan A E, et al. The impact of automated parameter optimization on defect prediction models. IEEE Transactions on Software Engineering, 2018.

[23] Tantithamthavorn C, Hassan A E. An experience report on defect modelling in practice: Pitfalls and challenges. Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice. ACM, 2018: 286-295.

[24] Tantithamthavorn C, McIntosh S, Hassan A E, et al. Automated parameter optimization of classification techniques for defect prediction models. Software Engineering (ICSE), 2016 IEEE/ACM 38th International Conference on. IEEE, 2016: 321-332.

[25] Jiarpakdee J, Tantithamthavorn C, Ihara A, et al. A study of redundant metrics in defect prediction datasets. Software Reliability Engineering Workshops, 2016 IEEE International Symposium on. IEEE, 2016: 51-52.

[26] Jiarpakdee J, Tantithamthavorn C, Hassan A E. The impact of correlated metrics on defect models. arXiv preprint arXiv:1801.10271, 2018.

[27] T. Mende, R. Koschke, M. Leszak. Evaluating Defect Prediction Models for a Large Evolving Software System. European Conference on Software Maintenance and Reengineering, 2009; 247–250.

[28] T. Mende, R. Koschke, J. Peleska. On the Utility of a Defect Prediction Model during HW/SW Integration Testing: A Retrospective Case Study. European Conference on Software Maintenance and Reengineering, IEEE Computer Society Press, 2011; 259–268.

[29] Yu X, Li Q, Liu J. Scalable and parallel sequential pattern mining using spark. World Wide Web, 2018: 1-30.

[30] Kamei Y, Fukushima T, McIntosh S, et al. Studying just-in-time defect prediction using cross-project models. Empirical Software Engineering, 2016, 21(5): 2072-2106.

[31] Xu Z, Liu J, Yang Z, et al. The impact of feature selection on defect prediction performance: An empirical comparison. Software Reliability Engineering (ISSRE), 2016 IEEE 27th International Symposium on. IEEE, 2016: 309-320.

[32] Yu X, Liu J, Yang Z, et al. Learning from Imbalanced Data for Predicting the Number of Software Defects. Software Reliability Engineering (ISSRE), 2017 IEEE 28th International Symposium on. IEEE, 2017: 78-89.

[33] Xu Z, Liu J, Luo X, et al. Software defect prediction based on kernel PCA and weighted extreme learning machine. Information and Software Technology, 2018.

[34] Panichella A, Alexandru C V, Panichella S, et al. A search-based training algorithm for cost-aware defect prediction. Proceedings of the Genetic and Evolutionary Computation Conference 2016: 1077-1084.

[35] Bettenburg N, Nagappan M, Hassan A E. Think locally, act globally: Improving defect and effort prediction models. Proceedings of the 9th IEEE Working Conference on Mining Software Repositories. IEEE Press, 2012: 60-69.

[36] T. Menzies, Z. Milton, B. Turhan, B. Cukic, and Y. J. A. Bener. Defect prediction from static code features: current results, limitations, new approaches. Automated Software Engineering, 17:375–407, 2010.

[37] Y. Kamei, S. Matsumoto, A. Monden, K. Matsumoto, B. Adams, and A. Hassan. Revisiting common bug prediction findings using effort-aware models. In Software Maintenance (ICSM), 2010 IEEE International Conference on, pages 1–10, 2010.

[38] M. D'Ambros, M. Lanza, and R. Robbes. Evaluating defect prediction approaches: A benchmark and an extensive comparison. Empirical Software Engineering, 17(4-5):531–577, 2012.

[39] Arisholm E, Briand L C, Johannessen E B. A systematic and comprehensive investigation of methods to build and evaluate fault prediction models. Journal of Systems and Software, 2010, 83(1): 2-17.

[40] Rahman F, Posnett D, Devanbu P. Recalling the imprecision of cross-project defect prediction, Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering. ACM, 2012: 61.

[41] Rahman F, Devanbu P. How, and why, process metrics are better, Software Engineering (ICSE), 2013 35th International Conference on. IEEE, 2013: 432-441.

[42] E.J.Weyuker,T.J.Ostrand,andR.M.Bell, Comparing the effectiveness of several modeling methods for fault prediction, Empiric. Softw. Eng., vol. 15, no. 3, pp. 277–295, 2010.

[43] Nguyen T T, An T Q, Hai V T, et al. Similarity-based and rank-based defect prediction. International Conference on Advanced Technologies for Communications. IEEE, 2015:321-325.

[44] T. J. Ostrand, E. J. Weyuker and R. M. Bell. Predicting the location and number of faults in large software systems. IEEE Transactions on Software Engineering, 31(4): 340-355, 2005.

[45] A. Janes, M. Scotto, and W. Pedrycz. Identification of defect-prone classes in telecommunication software systems using design metrics, Information sciences, 176(24): 3711-3734, 2006.

[46] T. M. Khoshgoftaar and K. Gao. Count models for software quality estimation. IEEE Transactions on Reliability, 56(2): 212-222, 2007.

[47] K. Gao and T. M. Khoshgoftaar, A comprehensive empirical study of count models for software fault prediction. IEEE Transactions on Reliability, 56(2): 223-236, 2007.

[48] W. Afzal, R. Torkar, and R.Feldt. Prediction of fault count data using genetic programming. Multitopic Conference, 2008. INMIC 2008. IEEE International. IEEE, 2008.

[49] Rathore S S and Kuamr S. Comparative analysis of neural network and genetic programming for number of software faults prediction. National Conference on Recent Advances in Electronics & Computer Engineering (RAECE), 328-332, 2015.

[50] S. S. Rathore and S. Kumar. Predicting number of faults in software system using genetic programming. Procedia Computer Science, 62: 303-311, 2015.

[51] S. S. Rathore and S.Kumar. A Decision Tree Regression based Approach for the Number of Software Faults Prediction. ACM SIGSOFT Software Engineering Notes, 41(1): 1-6, 2016.

[52] M. Chen and Y. Ma. An empirical study on predicting defect numbers. In Proceedings of the 28th International Conference on Software Engineering and Knowledge Engineering, 397-402, 2015.

[53] S. S. Rathore and S. Kumar. An empirical study of some software fault prediction techniques for the number of faults prediction. Soft Computing, 1-18, 2016.

[54] Hosmer Jr D W, Lemeshow S, Sturdivant R X. Applied logistic regression. John Wiley & Sons, 2013.

[55] Loh W Y. Classification and regression trees. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, 2011, 1(1): 14-23.

[56] Breiman L. Bagging predictors. Machine learning, 1996, 24(2): 123-140.

Authorized licensed use limited to: Wuhan University. Downloaded on January 13,2025 at 08:00:44 UTC from IEEE Xplore. Restrictions apply.

[57] Peterson L E. K-nearest neighbor. Scholarpedia, 2009, 4(2): 1883.

[58] Xu M, Watanachaturaporn P, Varshney P K, et al. Decision tree regression for soft classification of remote sensing data. Remote Sensing of Environment, 2005, 97(3): 322-336.

[59] Asai H T S U K. Linear regression analysis with fuzzy model. IEEE Transaction Systems Man and Cybermatics, 1982, 12(6): 903-07.

[60] Hoerl A E, Kennard R W. Ridge regression: Biased estimation for nonorthogonal problems. Technometrics, 1970, 12(1): 55-67.

[61] Gardner M W, Dorling S R. Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences. Atmospheric environment, 1998, 32(14-15): 2627-2636.

[62] Basak D, Pal S, Patranabis D C. Support vector regression. Neural Information Processing-Letters and Reviews, 2007, 11(10): 203-224.

[63] Altman. An introduction to kernel and nearest-neighbor nonparametric regression. The American Statistician, 1992, 46(3): 175-185.

[64] Carpenter B. Lazy sparse stochastic gradient descent for regularized multinomial logistic regression. Alias-i, Inc., Tech. Rep, 2008: 1-20.

[65] Quiñonero-Candela J, Rasmussen C E. A unifying view of sparse approximate Gaussian process regression. Journal of Machine Learning Research, 2005, 6(Dec): 1939-1959.

[66] Carpenter B. Lazy sparse stochastic gradient descent for regularized multinomial logistic regression. Alias-i, Inc., Tech. Rep, 2008: 1-20.

[67] R. Herbrich, T. Graepel, and K. Obermayer. Large Margin Rank Boundaries for Ordinal Regression. Advances in Large Margin Classifiers, pages 115-132, 2000.

[68] Freund Y, Iyer R D, Schapire R E, et al. An Efficient Boosting Algorithm for Combining Preferences. Fifteenth International Conference on Machine Learning. Morgan Kaufmann Publishers Inc. 1998:170-178.

[69] Burges C, Shaked T, Renshaw E, et al. Learning to rank using gradient descent. Proceedings of the 22nd international conference on Machine learning. ACM, 2005: 89-96.

[70] Wu Q, Burges C J C, Svore K M, et al. Adapting boosting for information retrieval measures[J]. Information Retrieval, 2010, 13(3): 254-270.

[71] Cao Z, Qin T, Liu T Y, et al. Learning to rank: from pairwise approach to listwise approach. Proceedings of the 24th international conference on Machine learning. ACM, 2007: 129-136.

[72] Xu J, Li H. Adarank: a boosting algorithm for information retrieval. Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval. ACM, 2007: 391-398.

[73] Metzler D, Croft W B. Linear feature-based models for information retrieval. Information Retrieval, 2007, 10(3): 257-274.

[74] G. Boetticher, T. Menzies and T. Ostrand, The PROMISE Repository of Empirical Software Engineering Data, <http://promisedata.org/repository>, 2007.

[75] Z. He, F. Peters, T. Menzies, and Y. Yang. Learning from open-source projects: An empirical study on defect prediction. In 2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement, Baltimore, Maryland, USA, 45–54, 2013.

[76] S. Wang, T. Liu, and L. Tan. Automatically Learning Semantic Features for Defect Prediction. In Proceedings of International Conference on Software Engineering, 2016.

[77] C. Tantithamthavorn, S. Mcintosh, A. E. Hassan, et al, An Empirical Comparison of Model Validation Techniques for Defect Prediction Models, IEEE Transactions on Software Engineering, 2017, 43(1):1-18.

[78] LaValle S M, Branicky M S, Lindemann S R. On the relationship between classical grid search and probabilistic roadmaps. The International Journal of Robotics Research, 2004, 23(7-8): 673-692.

[79] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, K. Matsumoto, Automated parameter optimization of classification techniques for defect prediction models, in: Proceedings of the 38th International Conference on Software Engineering, ACM, 2016, pp. 321–332.

[80] S. Lessmann, B. Baesens, C. Mues, S. Pietsch, Benchmarking classification models for software defect prediction: A proposed framework and novel findings, IEEE Trans. Softw Eng. 34 (4) (2008) 485–496.

[81] Osman H, Ghafari M, Nierstrasz O. Hyperparameter optimization to improve bug prediction accuracy, Machine Learning Techniques for Software Quality Evaluation (MaLTeSQuE), IEEE Workshop on. IEEE, 2017: 33-38.

[82] Shan C, Zhu H, Hu C, et al. Software defect prediction model based on improved LLE-SVM, Computer Science and Network Technology, 2015 4th International Conference on. IEEE, 2015, 1: 530-535.

[83] T. Menzies, J. Greenwald, and A. Frank, Data mining static code attributes to learn defect predictors, IEEE Trans. Softw. Eng., vol. 33, no. 1, pp. 2–13, 2007.

[84] H. Wang, T. M. Khoshgoftaar, and N. Seliya, How many software metrics should be selected for defect prediction, in Proc. 24th Int. Florida Artificial Intelligence Research Society Conf., 2011, pp. 69–74.

[85] T. M. Khoshgoftaar, K. Gao, and A. Napolitano, An empirical study of feature ranking techniques for software quality prediction, Int. J. Softw. Eng. Knowl. Eng., vol. 22, no. 2, pp. 161–183, 2012.

[86] C. Tantithamthavorn. ScottKnottESD: The Scott-Knott Effect Size Difference (ESD) Test. https://cran.r- project.org/web/packages/ScottKnottESD/index.html, 2016.

[87] A. J. Scott and M. Knott. A cluster analysis method for grouping means in the analysis of variance. Biometrics, 507-512, 1974.

[88] L. C. Borges and D. F. Ferreira. Power and type I errors rate of Scott- Knott, Tukey and Newman-Keuls tests under normal and no-normal distributions of the residues. Revista de Matemática e Estatística, 21(1): 67-83, 2003.

[89] B. Ghotra, S. McIntosh, and A. E. Hassan. Revisiting the impact of classification techniques on the performance of defect prediction models. In37th International Conference on Software Engineering, 789-800, 2015.

[90] Khoshgoftaar T M, Golawala M, Van Hulse J. An empirical study of learning from imbalanced data using random forest. Tools with Artificial Intelligence, 2007. ICTAI 2007. 19th IEEE international conference on. IEEE, 2007, 2: 310-317.

[91] https://www.microsoft.com/

[92] Yu X, Liu J, Yang Z, et al. The Bayesian Network based program dependence graph and its application to fault localization. Journal of Systems and Software, 2017, 134: 44-53.

Authorized licensed use limited to: Wuhan University. Downloaded on January 13,2025 at 08:00:44 UTC from IEEE Xplore. Restrictions apply.