A Learning-to-Rank Approach to Software Defect Prediction

Xiaoxing Yang, Ke Tang, *Senior Member, IEEE*, and Xin Yao, *Fellow, IEEE*

***Abstract*—Software defect prediction can help to allocate testing resources efficiently through ranking software modules according to their defects. Existing software defect prediction models that are optimized to predict explicitly the number of defects in a software module might fail to give an accurate order because it is very difficult to predict the exact number of defects in a software module due to noisy data. This paper introduces a learning-to-rank approach to construct software defect prediction models by directly optimizing the ranking performance. In this paper, we build on our previous work, and further study whether the idea of directly optimizing the model performance measure can benefit software defect prediction model construction. The work includes two aspects: one is a novel application of the learning-to-rank approach to real-world data sets for software defect prediction, and the other is a comprehensive evaluation and comparison of the learning-to-rank method against other algorithms that have been used for predicting the order of software modules according to the predicted number of defects. Our empirical studies demonstrate the effectiveness of directly optimizing the model performance measure for the learning-to-rank approach to construct defect prediction models for the ranking task.**

***Index Terms*—Software defect prediction, learning-to-rank, software metrics, count models, metric selection.**

Abbreviations & Acronyms

SDP	Software Defect Prediction
NBR	Negative Binomial Regression
SVM	Support Vector Machine
RF	Random Forest
LS	Least Squares
AAE	Average Absolute Errors
ARE	Average Relative Errors
CLC	Cumulative Lift Chart
FPA	Fault-Percentile-Average
LTR	Learning-to-Rank
BART	Bayesian Additive Regression Trees
RP	Recursive Partitioning
ZINBR	Zero-Inflated NBR
ZIPR	Zero-Inflated Poisson Regression
HNBR	Hurdle NBR
HPR	Hurdle Poisson Regression
InfoGain	Information Gain
LOC	Lines of Code
ROC	Receiver Operating Characteristic
AUC	Area under the ROC curve
CE	Cost-Effectiveness
EA	Evolutionary Algorithms
CoDE	Composite Differential Evolution
PCA	Principal Component Analysis
ranksum	Wilcoxon rank-sum test
s-	implies: statistical(ly)

Notations

$\mathbf{x}$	the vector of metrics of a software module, which can be written as $(x_1, x_2, \ldots\ldots, x_d)$
x_i	the value for the ith metric
d	number of metrics
$f(\mathbf{x})$	the predicted defect number of $\mathbf{x}$
α_i	the corresponding parameters for x_i to decide the prediction model
k	number of software modules
n_i	the actual defect number in module i
n	the total number of defects in all modules
$trapezoid_m$	the area of the trapezoid composed of two adjacent points and the axes in the CLC

Manuscript received February 06, 2013; revised October 25, 2013 and June 23, 2014; accepted July 03, 2014. Date of publication December 23, 2014; date of current version February 27, 2015. This work was supported in part by the 973 Program of China (Grant No. 2011CB707006), National Natural Science Foundation of China (Grants Nos. 61329302 and 61175065), the Program for New Century Excellent Talents in University (Grant No. NCET-12-0512), the Science and Technological Fund of Anhui Province for Outstanding Youth (Grant No. 1108085J16), EPSRC (Grant No. EP/J017515/1), and the European Union Seventh Framework Programme under grant agreements No. 247619 and No. 270428. The work of X. Yao was supported by a Royal Society Wolfson Research Merit Award. Associate Editor: C. Smidts. (*Corresponding author: K. Tang.*)

X. Yang and K. Tang are with the USTC-Birmingham Joint Research Institute in Intelligent Computation and Its Applications (UBRI), School of Computer Science and Technology, University of Science and Technology of China, Hefei, Anhui 230027, China (e-mail: apricot@mail.ustc.edu.cn; ketang@ustc.edu.cn).

X. Yao is with the USTC-Birmingham Joint Research Institute in Intelligent Computation and Its Applications (UBRI), School of Computer Science and Technology, University of Science and Technology of China, Hefei, Anhui 230027, China, and also with the Center of Excellence for Research in Computational Intelligence and Applications (CERCIA), the School of Computer Science, University of Birmingham, Edgbaston, Birmingham B15 2TT, U.K. (e-mail: x.yao@cs.bham.ac.uk).

Digital Object Identifier 10.1109/TR.2014.2370891

I. Introduction

SOFTWARE defect prediction (SDP) employs software metrics [1], [2] (also referred to as software features or software attributes, such as lines of code (LOC) [3], and change

TABLE I
A SIMPLE EXAMPLE OF DEFECT PREDICTION

Module Name	Lines of Code	Previous Defects	Lines Added	Defect Number
A	456	4	45	4
B	123	1	0	0
C	156	1	23	2
D	321	3	23	unknown
E	211	2	43	unknown
F	2354	15	432	unknown

information [4]) to predict their defect-proneness to support software testing activities [5], [6]. The most frequently investigated goals of SDP models include ordering new software modules based on their defect-proneness [7], and classifying whether or not new software modules have defects [8]. This paper considers the former case, which we refer to as SDP for the ranking task.

The process of SDP for the ranking task mainly includes two parts: data collection, and model construction. To illustrate the process more clearly, we describe it based on the simple example in Table I. Firstly, data are obtained from old software modules with known defect numbers (A, B, and C in Table I), including values of all software metrics (lines of code, previous defects, and lines added), and the numbers of defects (defect number). After that, we use modelling approaches such as machine learning methods to construct a model based on the obtained data. The learned model describes the relationship between software metrics and the number of defects. Therefore, after collecting all metric values of new software modules (D, E, and F in this example), we can use the learned model to predict the numbers of defects of these new modules. Thus we can obtain an order of D, E, and F based on the predicted defect numbers, and allocate testing resources according to the order (for instance, more testing resources for modules with more defects).

There are numerous approaches to constructing SDP models that sort new software modules based on their defect-proneness, including regression and classification methods, such as negative binomial regression (NBR) [3], support vector machine (SVM) [9], random forest (RF) [10], and Poisson regression [11]. These approaches obtain SDP models by maximum likelihood estimation or least squares (LS), focusing on the fitting of each sample. In other words, these approaches construct SDP models by minimizing the individual-based loss functions (classification or regression). There exist some performance measures such as average absolute errors (AAE) and average relative errors (ARE) for evaluating these prediction models [12].

However, the purpose of SDP for the ranking task is to predict which modules are likely to have most defects to allocate software quality enhancement efforts [7], [10], [11]. That is, the goal of SDP for the ranking task is to predict the relative defect number, although estimating the precise number of defects of the modules is better than estimating the ranks of modules, because the precise number of defects can give more information than the ranks. Nevertheless, to predict the precise number of defects of a module is hard or even impossible to do due to the lack of good quality data in practice. Actually, for those existing approaches that tried to predict explicitly the number of defects in a software module, they used these predicted numbers to rank the modules anyway, to direct the software quality assurance team in targeting the most faulty modules first [3], [10], [11]. Hence, Ohlsson [13] proposed the Alberg diagram [13] (also referred to as cumulative lift chart (CLC) [6]) as the performance measure, and Weyukers *et al.* [10] applied fault-percentile-average (FPA) and the percentage of defects in the first 20% modules to evaluate prediction models.

In other words, it is the order of software modules according to their defects instead of the specific defect numbers that is used for guiding the assignment of testing resources for SDP for the ranking task. In this case, it is more natural to learn the ranking of software modules directly, rather than to predict the precise number of defects in each module, and then use such numbers to rank the modules. Actually, for SDP for the ranking task, models with higher prediction accuracy (smaller AAE or ARE) might give a worse ranking. For example, assuming that D, E, and F in Table I have 2, 3, and 4 defects, model M predicts that D, E, and F have 2, 0, and 4 defects respectively, while model N predicts that D, E, and F have 0, 1, and 2 defects respectively. Although model M has a better prediction accuracy (according to AAE and ARE), the order given by model N is what we desire. Therefore, we propose to construct models by directly optimizing the ranking performance. To be specific, we propose a learning-to-rank (LTR) approach to solve SDP for the ranking task.

In this paper, our key contributions include the following.

1) A novel approach (the LTR approach) to SDP for the ranking task.
2) A comprehensive investigation of SDP for the ranking task including ten construction algorithms evaluated on eleven real-world data sets, some of which have multiple releases.
3) An investigation of the effectiveness of different software metrics for building SDP models for the ranking task using feature selection.

This paper is based on our previous work [14]. However, this paper differs from the previous work mainly in the following ways.

1) We apply the LTR method to the data sets with multiple releases, which was not included in our previous work.
2) We provide a comprehensive evaluation and comparison of the LTR approach against more algorithms for constructing SDP models for the ranking task. In our previous work [14], we compared the LTR approach with only three methods. In this paper, we firstly compare it with the same model obtained by LS, to verify whether or not directly optimizing the model performance measure can construct better SDP models. Subsequently, we compare the LTR approach with eight existing methods, including all models used in Weyuker *et al.*'s comparison study [10], and four count models used in Gao *et al.*'s investigation [11], which are NBR, Bayesian additive regression trees (BART), recursive partitioning (RP), zero-inflated NBR (ZINBR), zero-inflated Poisson regression (ZIPR), hurdle NBR (HNBR), hurdle Poisson regression (HPR), and RF.
3) We present an investigation of the relationship between two performance measures for evaluating SDP models for the ranking task: FPA, and CLC. Both FPA [10] and CLC

[13] were proposed to evaluate SDP models for the ranking task. Both are based on the whole ranking given by prediction models, and put more emphases on the former ranking. They seem to describe the same thing, but it is unclear how they relate to each other, and whether they capture the same characteristics of the prediction models, so it is interesting to investigate their relationship, which was not included in our previous work.

4) We study the effectiveness of different software metrics for constructing SDP models for the ranking task. Menzies *et al.* [15], and Wang *et al.* [16] pointed out that three metrics could work as well as all metrics. Filter-based metric selection methods have been applied for some data sets [17]. However, all such work was done for the classification task instead of the ranking task. In addition, filter-based metric selection was only applied to a limited number of data sets. Therefore, we investigate the effectiveness of different software metrics over eleven data sets by applying information gain (InfoGain). InfoGain is used because the empirical studies have demonstrated its effectiveness [15], [17] for SDP for the classification task.

The rest of this paper is organized as follows. Section II presents an overview of related work in SDP. In Section III, we describe our LTR approach. Section IV details the experimental methodologies. Experimental results are given in Section V. We describe the threats to validity in Section VI. Section VII draws the conclusions, and points out future work.

II. Related Work

SDP has been researched for decades because of its potential power in helping software developers to efficiently allocate testing resources. In this section, we review related work in two categories: software metrics, and model construction approaches.

A. Software Metrics

As pointed out by Fenton *et al.* [1], the quality of data is very important, and software metrics play an important role in describing the data. Early software metrics were based on size and structure, such as LOC and McCabe's Cyclomatic Complexity [18]. Subsequently, object orientation-based metrics [19] and change metrics [4] were introduced.

Graves *et al.* [20] computed correlation between different complexity metrics, and found that most of the complexity metrics were highly correlated to LOC, which implied the existence of redundant metrics. Zimmermann *et al.* [7] applied Spearman correlation to study the relationship between software metrics and defects, and found that a single metric was insufficient for predicting the number of defects.

Menzies *et al.* [15] applied InfoGain to select metrics based on their information, and found that using two or three out of 38 static code metrics could work as well as using all metrics, which reflected the usefulness of metric selection in SDP. However, they pointed out that the choice of learning methods is far more important than the choice of subsets of data used for learning. Wang *et al.* [16] utilized a threshold-based metric selection technique to remove irrelevant and redundant software metrics. Wang *et al.* [16] also found that an effective defect classifier could be built with only three metrics. Khoshgoftaar *et al.* [17] compared seven filter-based metric selection methods for SDP for the classification task, and showed the effectiveness of InfoGain and signal to noise ratio. However, specific results depended on specific classification models. Xu *et al.* [21] applied principal component analysis to obtain six components to reduce the dimension of the data, which could simplify the model.

B. Model Construction Approaches

According to Catal *et al.*'s review [22], machine learning algorithms were the most popular methods for constructing SDP models, including RF [23], and SVM [24]. With an increasing number of methods, some researchers conducted comparisons of model construction methods to define and motivate a baseline. For example, Lessmann *et al.* [25] conducted a large-scale benchmarking of 22 classification algorithms over 10 public-domain datasets from NASA MDP and PROMISE repositories. Lessmann *et al.* used AUC as the indicator, and found that RF was best, but no significant performance differences existed among 17 classifiers. However, Arisholm *et al.* [26], [27] pointed out that the regular confusion matrix criteria were not clearly related to SDP, and proposed a new indicator (cost-effectiveness (CE)) to more appropriately evaluate SDP models based on the assumption that testing cost was likely to be proportional to the size of software modules in some testing processes. Subsequently, Mende *et al.* [28] also proposed a similar indicator.

Compared with classification SDP, the research on SDP for the ranking task was limited [11]. Ohlsson and Alberg [13] used a linear regression method to construct SDP models. They used the models to predict the number of defects in software modules before coding started. Ohlsson *et al.* [13] applied the Alberg diagram (the same as CLC) and the percentages of defects in the top modules to evaluate the SDP models, and demonstrated the usefulness of the Alberg diagram.

Ostrand *et al.* [3] applied NBR and a very simple model based on only LOC to construct models to predict the expected number of defects in each module of the next release of a large commercial system, employing the percentages of defects in the top 20% modules to evaluate SDP models. NBR achieved better results than the simple model. However, from their graphs of the actual defects and models, there remained much space for improving the SDP models. Ostrand *et al.* also used the prediction models to give the order of modules according to the defect density.

Gao *et al.* [11] compared eight count models over a full-scale industrial software system. The comparative study showed that ZINBR and HNBR were more effective according to pairwise hypothesis testing techniques, information criteria-based comparative techniques, and Pearson's chi-square measure, but HPR with a threshold of 2 was better according to prediction accuracy AAE and ARE. The comparison results of model construction methods highly depended on the performance measures for evaluating models. Therefore, it is important to use the appropriate performance measures for evaluating SDP models.

Weyuker *et al.* [10] compared four approaches (NBR, RF, RP, and BART) for predicting the ranking of modules based on defects. Weyuker *et al.* proposed a new performance measure,

FPA. Similar to CLC, FPA was based on the whole ranking given by prediction models, and put more emphases on the former ranking. Experimental results indicated that NBR and RF models performed better than RP and BART models according to the percentages of defects in the top 20% modules and FPA. Because of a longer time for fitting RF and the non-deterministic results of RF models, Weyuker *et al.* concluded that linear and additive models were good and realistic for SDP.

All these existing methods constructed models by optimizing the indirect loss functions instead of directly optimizing the ranking performance measures, which can cause the potential problem that a good model according to the indirect loss functions could give poor results according to the ranking performance measures. In this paper, we describe a LTR approach to directly optimize the ranking performance.

From the above related work, we can see that most work with SDP for the ranking task ignored the module size. The possible reason might be that it is debatable whether knowing which files have the highest defect counts or knowing which files have the highest defect density is more helpful. Ostrand *et al.* [3] pointed out that "discussions with software testers convinced us that it is often more valuable for testers to know which files had the largest numbers of faults rather than the largest fault densities since that would allow them to better identify most of the faults quickly". Hence, we ignore the module size in this paper. When the ranking according to defect density is needed, we can change the predicted value of the number of defects into the defect density.

III. Learning-to-Rank Approach

Given a vector of metrics of a software module $\mathbf{x} = (x_1, x_2, \ldots\ldots, x_d)$, the goal of SDP for the ranking task is to predict its relative defect number, which is denoted as $f(\mathbf{x})$. Because Weyuker *et al.* [10] pointed out that linear and additive models were good and realistic for SDP, and because the generalized linear models (such as logistic models and Poisson models) give the same ranking as the simple linear model when their parameters are fixed, we study a simple linear model:

$$f(\mathbf{x}) = \sum_{i=1}^{d} \alpha_i x_i$$

where the α_i are the corresponding parameters obtained by training based on data from old software modules with known defect numbers. Once the α_i are fixed, the model is learned, and can be used for prediction.

In previous studies [10], [11], the parameters were often obtained by LS or maximum likelihood, which were fitted with explicit defect numbers. The potential problem of doing this is that a good regression model according to these functions might not give a good ranking evaluated by model performance measures, such as CLC [13], the percentages of defects contained in the top 20% modules [13], and FPA [10]. In our work, we directly optimize the ranking performance of SDP models to obtain the parameters. Because the performance measures of models are often non-differentiable, we employ meta-heuristic methods that can optimize non-differentiable objectives, such as evolutionary algorithms (EAs). In particular, we apply composite differential evolution (CoDE) [29], which has exhibited good performance compared with other methods. Details of using CoDE to optimize the parameters of linear models are shown as follows.

1) **Input** M training vectors $\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots, x_{id})$ (i $= 1, 2, \ldots, \mathrm{M}$), and the objective function (FPA).
2) **Initialize** the population size (number of solutions at each generation) to N. Randomly generate N solutions that compose P_0, and compute their objective function values. Set the generation number $t = 0$.
3) **While** $t < t_{max}$ (maximal generation number to decide termination), **do** the following.
   a) For each solution v_i in P_t, do crossover and mutation to generate three new solutions according to the following three generation strategies.
      i) rand-1-bin

$$u_{i,j,t} = \begin{cases} \alpha_{r1,j,t} + F \cdot (\alpha_{r2,j,t} - \alpha_{r3,j,t}) & \text{if rand} < C_r, \\ & \text{or j} = j_{rand}. \\ \alpha_{i,j,t} & \text{otherwise.} \end{cases}$$

      ii) rand-2-bin

$$u_{i,j,t} = \begin{cases} \alpha_{r1,j,t} + F_1 \cdot (\alpha_{r2,j,t} - \alpha_{r3,j,t}) & \\ \qquad +F \cdot (\alpha_{r4,j,t} - \alpha_{r5,j,t}) & \text{if rand} < C_r, \\ & \text{or j} = j_{rand}. \\ \alpha_{i,j,t} & \text{otherwise} \end{cases}$$

      iii) current-to-rand-1

$$\vec{u}_{i,t} = \vec{\alpha}_{i,t} + rand \cdot (\vec{\alpha}_{r1,t} - \vec{\alpha}_{i,t}) + F \cdot (\vec{\alpha}_{r2,t} - \vec{\alpha}_{r3,t})$$

      Each generation strategy creates a new solution $\vec{u}_{i,t}$. F and the crossover control parameter C_r are randomly chosen from three control parameter settings $[F = 1.0, C_r = 0.1]$, $[F = 1.0, C_r = 0.9]$, and $[F = 0.8, C_r = 0.2]$. r1–r5 are distinct integers randomly selected from the range $[1, N]$, *rand* and F_1 denote uniformly distributed random numbers between 0 and 1, and j_{rand} is a randomly chosen integer from 1 to d. The best vector among the three new solutions and v_i according to the objective function value is saved as a solution in the new population P_{t+1}.
   b) $\mathbf{t}++$.
4) **Return** the best solution according to the objective function in P_{t+1}, and **output** the model $f(\mathbf{x}) = \sum_{i=1}^{d} \alpha_i x_i$.

IV. Experimental Methodologies

In this section, we detail the methodologies used in our experimental studies, including research questions, data sets, compared model construction methods, evaluation measures, and the implementation.

A. Research Questions

1. About metrics: How many software metrics are most appropriate for building SDP models for the ranking task over two sets of data? Which metrics are most effective over these two sets of data?

For the classification task, related questions have been investigated by Wang *et al.* [16], and Menzies *et al.* [15]. We answer the above two questions for the ranking task. The total number of metrics is about 200 over all eleven data sets. The selected numbers of metrics are set to $2, 3(1+2), 5(2+3), 8(3+5), \ldots, 144(55+89)$, and all metrics. The LTR approach [14] is used as the model construction method, and 10-fold cross-validation is used to obtain the results.

InfoGain [15] is used as the metric selection method in this paper. Other feature selection methods such as principal component analysis (PCA) may also be used. We use InfoGain because the empirical studies have demonstrated its effectiveness for SDP for the classification task [15], [17]. Although PCA is a famous method, PCA converts the original metrics into a set of principal components that might be simultaneously related to several original metrics. The principal components are not the original metrics, and could not reflect the effectiveness of the original metrics directly. InfoGain can tell the effectiveness of the original metrics, and might tell which metrics should be collected in the first place.

InfoGain [15] is a metric selection method based on the rank of information gain. Given a metric's minimum and maximum values, a particular value n is replaced by $(n-min)/((max-min)/10)$. The number of bits required to encode an arbitrary class distribution C is H(C):

$$H(C) = -\sum_{c \in C} p(c) \log_2 p(c).$$

For classification problems, C includes c1 (defect-free) and c2 (defect-prone), and $p(c)$ is the corresponding frequency of defect-prone and defect-free modules. For ranking problems, there would be too many classes for C if we use all defect numbers because some modules have more than 100 defects. Therefore, we divide the defect numbers into m (square root of the maximum defect number) classes. If A is a set of metrics, the number of bits required to encode a class after observing a metric is

$$H(C \mid A) = -\sum_{a \in A} p(a) \sum_{c \in C} p(c \mid a) \log_2 p(c \mid a).$$

Then the highest ranked metric A_i is the one that most reduces the encoding required for the data after using that metric:

$$InfoGain(A_i) = H(C) - H(C \mid A_i).$$

In this paper, we adopt the iterative InfoGain subsetting, using the $i = 1, 2 \ldots, d$th top-ranked metrics.

2. What can the LTR approach bring to construct SDP models by directly optimizing the model performance measure?

The LTR approach obtains a linear model by optimizing the ranking performance directly. To verify whether or not directly optimizing the ranking performance measure can bring advantages for constructing SDP models for the ranking task, we compare it with the same linear model optimized by LS.

3. How does the performance of the LTR approach compare with existing methods over a large collection of data sets?

In this paper, we conduct a comprehensive study, including eight existing approaches on eleven data sets, to evaluate whether the LTR approach is comparable to existing nonlinear models.

4. How does the performance of the LTR approach compare with existing methods over data sets with more releases?

SDP models are often trained according to the old software, and then used to predict the defects of software modules in the new release [3], [7]. Results based on data sets with more releases are particularly interesting and useful in practice. Therefore, we compare the methods over data sets with more releases by using the former release as training data, and the latter one as testing data.

B. Data Sets

To facilitate the comparison between our work and others, and to make our work reproducible by others, we use existing benchmark data sets in our paper, including two sets of publicly available data.

One set is the benchmark presented by D'Ambros *et al.* [5], including data over a five year period from five open-source software systems: Eclipse JDT Core (eclipse), Eclipse PDE UI (pde), Equinox framework (equinox), Mylyn, and Apache Lucene (lucene). D'Ambros *et al.* [5] used a six month time interval for post-release defects for validation (i.e., not all defects in the history) to emulate a real-life scenario. The data were collected according to six sets of metrics: process metrics, previous defects, source code metrics, entropy of changes, churn of source code metrics, and entropy of source code metrics. To be specific, the data for each system included values of all the above metrics for each version of each class of the system, and the post-release defect number for each class. Some data were extracted from the change log, and the defects were extracted from the defect repository, linked to the transactions and the system classes referencing them. The data were designed to perform defect prediction at the class level. We combine all the metrics into one data set, and hence we can get five data sets with the corresponding defect numbers. The detailed process of data collection can be found in D'Ambros *et al.*'s work [5].

Considering that SDP models are usually used for predicting defects of a new release or a new software that is similar to the software providing training data, we also use the other data sets, the Eclipse data sets provided by Zimmermann *et al.* [7]. To avoid potential confusion, we denote the latter Eclipse as Eclipse_II. Eclipse_II involves three releases, so that we can use release 2.1 as the training data to construct a model, and then use the learned model to predict the defect numbers of modules in release 2.2, which is similar to real applications, and can better demonstrate the performance of the learned model. Eclipse_II was collected according to two kinds of metrics: complexity metrics, and metrics based on the structure of abstract syntax trees. One was collected at the file-level, and the other was collected at the package-level. The corresponding number of defects used here is the number of non-trivial defects reported in the first six months after release. The details can be found in Zimmermann *et al.*'s work [7].

The characteristics of these data sets are shown in Table II. The column of faulty modules records the number of modules

TABLE II
EXPERIMENTAL DATA SETS

Dataset Name	Module Number	Metric Number	Faulty Modules	Total Defects
Eclipse_II_File2.0(File2.0)	6729	198	975	1692
Eclipse_II_File2.1(File2.1)	7888	198	854	1182
Eclipse_II_File3.0(File3.0)	10593	198	1568	2679
Eclipse_II_Package2.0(Package2.0)	377	207	190	917
Eclipse_II_Package2.1(Package2.1)	434	207	194	662
Eclipse_II_Package3.0(Package3.0)	661	207	313	1534
eclipse	997	212	206	374
equinox	324	212	129	244
lucene	691	212	64	97
mylyn	1862	212	245	340
pde	1497	212	209	341

having defects, and the column of total defects records the total number of defects in all modules of the corresponding data set.

C. Existing Model Construction Methods

In this subsection, we briefly introduce eight existing model construction approaches used for comparison, including NBR, BART, RP, ZINBR, ZIPR, HNBR, HPR, and RF. Details can be found in Weyuker *et al.*'s comparison study [10], and Gao *et al.*'s investigation [11]. They are divided into two categories: variations of the Poisson regression model, and regression trees.

1) Variations of Poisson Regression Model: NBR has been popularly used for SDP [3], [10]. NBR, ZINBR, ZIPR, HNBR, and HPR are all variations of Poisson regression [11]. NBR models the logarithm of the expected number of defects as a linear combination of the metrics, so if the parameters are the same as the linear model, they can achieve the same ranking. Both zero-inflated models and hurdle models can explicitly model the excessive occurrence of zero defects. Zero-inflated models assume that modules having zero defects come from two different sources. The hurdle models directly partition the modules into a lower count group and a higher count group, assuming that each group follows some distributions.

2) Regression Trees: RP, BART, and RF are different kinds of regression trees [10]. RP is a basic regression tree, which iteratively constructs a binary decision tree to partition training samples to produce leaf nodes that contain homogeneous samples. BART constructs models as the sum of many trees with a normally distributed error. It consists of two parts: a sum-of-trees model, and the corresponding regularization prior on the parameters. RF is an ensemble classifier that consists of many trees, and outputs the average of individual trees [10].

D. Evaluation Measures

Performance measures decide which models are good. The percentage of defects in the former modules of the ranking is popularly used to evaluate SDP models for the ranking task [13], because it is practical and simple. However, considering that the relative performance can be sensitive to the arbitrary cutoff value, Weyuker *et al.* [10] proposed a general measure FPA. FPA takes into account both the practical use and the whole ranking performance of SDP models. CLC seems also suitable for evaluating SDP models for the ranking task because it is also based on the whole ranking and emphasizes the former ranking [13]. It is interesting to understand the relationship between FPA and CLC. In this subsection, we first introduce FPA, and then analyze the relationship between FPA and CLC.

1) FPA: In the beginning, the percentage of defects contained in the 20% of modules predicted to have the most faults was used to assess predictive accuracy [3]. In theory, random models predict only 20% defects in the first 20% of the modules. When testing resources are limited, software testers may allocate the testing resources to only the first 20% modules. Hence, if the first 20% modules have a larger percentage of defects, software testers may find more defects. However, the performance can be sensitive to the arbitrary cutoff value of 20%. Testing resources may be sufficient for testing the first 40% modules, or resources can test only the first 5% modules. Therefore, Weyuker *et al.* [10] proposed FPA, which could reflect the effectiveness of the different prediction models across all values of the cutoff.

Considering k modules listed in increasing order of predicted defect number as $f_1, f_2, \ldots, f_k$, and assuming that n_i is the actual defect number in the module i, $n = n_1 + n_2 + \cdots + n_k$ is the total number of defects, and the top m predicted modules should have $\sum_{i=k-m+1}^{k} n_i$ defects. The proportion of the actual defects in the top m predicted modules to the whole defects is

$$\frac{1}{n} \sum_{i=k-m+1}^{k} n_i.$$

Then the FPA [10] is defined as

$$\frac{1}{k} \sum_{m=1}^{k} \frac{1}{n} \sum_{i=k-m+1}^{k} n_i.$$

Therefore, FPA is actually the average of the proportions of actual defects in the top m $(m = 1, 2, \ldots, k)$ modules to the whole defects, which is a more comprehensive performance measure than the percentage of defects in the top 20% modules. A higher FPA means a better ranking, where the modules with most defects come first.

2) Relationship Between FPA and CLC: CLC [6] uses the percentages of modules as the x-axis, and the percentages of defects as the y-axis. When it is used to compare methods, we always use the area under CLC, which is simply denoted as CLC in this paper. Considering that k modules are listed in increasing order of predicted defect number as $f_1, f_2, \ldots, f_k$, and assuming that n_i is the actual defect number in the module $i, n = n_1 + n_2 + \cdots + n_k$ is the total number of defects. CLC should be computed as shown in (1) at the bottom of the next page.

According to the equation above, CLC and FPA are linearly related, and should be consistent to evaluate models for the ranking task. Therefore, one of them is sufficient to evaluate SDP models for ranking tasks. We adopt FPA as the performance measure in this paper because it is the up-to-date measure proposed to evaluate ranking SDP models, and its formula is easier to understand (the average of proportions of actual defects in the top predicted modules).

E. Implementation

To follow Weyuker *et al.*'s comparison study [10], all compared methods, including the simple linear model optimized by LS, are implemented in R. For RF, RP, and BART, we use the

TABLE III
MEAN FPA BASED ON DIFFERENT METRIC NUMBER

datasets	2	3	5	8	13	21	34	55	89	144	all
Files2.0	0.80	0.80	0.80	0.81	0.81	0.81	0.82	0.82	0.82	0.82	0.82
Files2.1	0.77	0.77	0.78	0.78	0.78	0.78	0.78	0.78	0.78	0.78	0.78
Files3.0	*0.78*	0.79	0.79	0.79	0.79	0.79	0.79	0.79	0.79	0.79	0.80
Package2.0	0.76	0.76	0.78	0.77	0.76	0.77	0.78	0.77	0.79	0.78	0.79
Package2.1	0.75	0.78	0.77	0.77	0.77	0.77	0.76	0.77	0.77	0.77	0.77
Package3.0	0.80	0.81	0.82	0.82	0.80	0.80	0.80	0.81	0.81	0.82	0.78
eclipse	0.82	0.82	0.82	0.83	0.82	0.83	0.83	0.83	0.82	0.82	0.82
equinox	0.80	0.81	0.80	0.81	0.80	0.81	0.80	0.80	0.80	0.81	0.80
lucene	0.80	0.84	0.83	0.85	0.85	0.84	0.83	0.82	0.81	0.81	0.80
Mylyn	0.74	0.74	0.75	0.75	0.75	0.75	*0.74*	0.78	0.80	0.79	0.79
pde	0.78	0.78	0.78	0.78	0.78	0.77	0.77	0.77	0.77	0.76	0.76

suggested parameters in Weyuker *et al.*'s comparison study. For other variations of Poisson regression, we use the default parameters. The LTR approach is implemented in Java. For CoDE in the LTR approach, we set the feasible solution space to $\Omega = \prod_{i=1}^{d}[-20, 20]$. The population size and maximal generation are set to 100, which are shown to be sufficient from our results (the algorithm converges to similar solutions). For eleven individual data sets, we apply 10-fold cross-validation to evaluate the methods, which splits the data sets into 10 folds, with 9 folds for training to build a model, and the remaining fold as a validation set to evaluate the model, with each fold used once as a validation set. To conduct cross-validation, we randomly produce the index before splitting, and hence all methods use the same training and testing sets each time. For Eclipse_II, we use the former release as the training set, and use the subsequent release as the validation set.

The Wilcoxon rank-sum test (which is called ranksum for short) at 0.05 significance is used as the statistical test, which is a non-parametric test of the null hypothesis that two populations are the same against an alternative hypothesis, especially that a particular population tends to have larger values than the other. Using ranksum is appropriate in our experiments, because the results meet the assumptions for the ranksum test.

According to the research questions, we conduct the following experiments.

1. Apply InfoGain to select different numbers of metrics to discover the appropriate number of metrics that can achieve competitive results with the whole set of metrics, and find out which metrics are effective for SDP for the ranking task.

TABLE IV
THREE METRICS SELECTED BY INFOGAIN ACCORDING TO DIFFERENT TASKS

datasets	selected metrics
Files2.0	NBD_max,SimpleName,SUM
Files2.1	SimpleName,NBD_max,SUM
Files3.0	NBD_max,Block,SimpleName
Package2.0	NBD_max,NOM_max,ImportDeclaration
Package2.1	NBD_max,Modifier,ImportDeclaration
Package3.0	NBD_max,NOCU,TLOC_max
eclipse	CvsEntropy,ent-wmc,ent-numberOfLinesOfCode
equinox	CvsLogEntropy,cbo,CvsEntropy
lucene	CvsLinEntropy,log-churn-wmc,CvsExpEntropy
mylyn	fanOut,exp-ent-numberOfLinesOfCode,exp-churn-numberOfLinesOfCode
pde	CvsLinEntropy,lin-ent-rfc,CvsExpEntropy

2. Compare our linear model with the linear model optimized by LS to understand the effects of directly optimizing the ranking performance measures.
3. Compare the LTR approach with eight existing methods over eleven data sets.
4. Compare LTR and other methods over data sets with more than one release.

V. EXPERIMENTAL RESULTS

In this section, we present the corresponding experimental results according to the four research questions.

A. Investigation of Metrics for SDP for the Ranking Task

The goal of this subsection is to answer the first research question. Table III shows the FPA results based on different numbers of metrics. The ranksum at 0.05 significance is used to test whether or not the models based on some of the metrics selected by InfoGain are *s*-significantly different from models based on all metrics. We use italics to mark the results if they are *s*-significantly different. According to Table III, models based on only two or three metrics work as well as models based on all metrics over all data sets. For some data sets such as Package3.0 and lucene, models based on only three metrics achieve larger FPA than models based on all metrics. We show the three most effective metrics selected by InfoGain for each data set in Table IV.

In Table IV, the metric NBR_max is selected by InfoGain over all six Eclipse_II data sets, which indicates the importance

$$
\begin{aligned}
CLC &= \sum_{m=1}^{k} trapezoid_m \\
&= (\frac{1}{k})(\frac{1}{2})((0+\frac{n_k}{n})+\cdots+(\frac{n_k+\cdots+n_2}{n}+\frac{n_k+\cdots+n_1}{n})) \\
&= (\frac{1}{2kn})((2k-1)n_k+(2k-3)n_{k-1}+\cdots+3n_2+n_1) \\
&= (\frac{1}{2kn})(2kn_k+2n_{k-1}(k-1)+\cdots+2n_1-n) \\
&= (\frac{1}{kn})(n_k k+n_{k-1}(k-1)+\cdots+n_1)-\frac{1}{2k} \\
&= (\frac{1}{k})(\frac{n_k}{n}+\frac{n_k+n_{k-1}}{n}+\cdots+\frac{n_k+\cdots+n_1}{n})-\frac{1}{2k} \\
&= FPA-(\frac{1}{2k}) \qquad (1)
\end{aligned}
$$

of max nested block depth for these data sets. Max nested block depth might reflect the difficulty of the corresponding modules in some degree, and thus relate to the defects. Other selected metrics for these data sets are mainly based on the structure of abstract syntax trees instead of complexity metrics. For the other five data sets, most selected metrics are change metrics, including the complexity code change, churn, and entropy. These observations are consistent with the previous studies [5], [20]. The similarities among different data sets could indicate which metrics are most useful for most projects, while the differences show that different data sets may favor different metrics, and metric selection may be needed in advance. Although different metrics might lead to different outcomes in different cases, the usefulness of the LTR approach is not affected significantly, because our approach is applicable in different cases, and does not depend on any particular metrics or data sets.

B. Comparison of Linear Models Optimized by Different Methods (LTR vs. LS)

In this subsection, we compare our LTR linear model with the linear model optimized by LS to investigate whether directly optimizing the model performance measure can construct better models than optimizing the indirect loss function. If LTR performs better than LS, it means that the idea of directly optimizing the model performance measure is useful for constructing SDP models for the ranking task.

To compare these two models, we apply 10 times 10-fold cross-validation over eleven data sets with all metrics, and the corresponding data sets with three metrics selected by InfoGain. We also compare them over Eclipse_II with multiple series, using the former release as the training data, and the latter release as the testing data. Because the training data and the testing data are fixed for Eclipse_II with multiple releases, we run them 10 times. All results are shown in Table V. The second and third columns are the two linear models (the LTR linear model, and the linear model by LS) over data sets with all metrics, and the fourth and fifth columns are the two linear models over data sets with three metrics. P2.0–2.1 means using Package2.0 as the training data and Package 2.1 as the testing data. Others are similar. The Wilcoxon ranksum at 0.05 significance is used to test whether or not the two models achieve *s*-significantly different results. If they are *s*-significantly different, the better results are in boldface. The p-value is in the corresponding brackets.

Over the data sets, the LTR linear model performs *s*-significantly better than the linear model by LS for ten out of eleven data sets with all metrics. When three metrics are used, LTR is never outperformed by LS on all eleven data sets. For data sets with multiple releases, LTR performs better than LS over most data sets. Hence, directly optimizing the ranking performance measure for LTR is useful for constructing better SDP models for the ranking task.

The above results imply that it is easier to capture the important factors related to defects by directly optimizing the ranking performance measure than by optimizing the prediction errors, especially when there are many coefficients (the number is equal to the number of metrics) to be optimized. These results also imply that constructing a linear model by LS might be inappropriate for solving this problem. There are some assumptions for using multiple linear regression models, one of which is that no correlation exists between the independent variables [30]. We use the condition numbers to analyze the correlation between metrics, which are shown in Table VI.

TABLE V
MEANS AND STANDARD DEVIATIONS OF FPA RESULTS OF LINEAR MODELS OPTIMIZED BY DIFFERENT METHODS OVER ELEVEN DATA SETS WITH ALL METRICS, AND THREE METRICS SELECTED BY INFOGAIN. RESULTS IN BOLDFACE ARE *S*-SIGNIFICANTLY BETTER

datasets	LTR all	LS all	LTR three	LS three
Files2.0	**0.825±.018**(.0)	0.805±.029	0.805±.021(.61)	0.804±.022
Files2.1	0.779±.022(.54)	0.777±.026	0.765±.022(.86)	0.765±.022
Files3.0	**0.795±.019**(.0)	0.780±.021	0.788±.016(.52)	0.787±.017
Package2.0	**0.785±.064**(.0)	0.670±.120	0.769±.063(.87)	0.766±.067
Package2.1	**0.772±.058**(.0)	0.691±.102	0.776±.062(.77)	0.774±.062
Package3.0	**0.814±.033**(.0)	0.734±.095	0.815±.031(.14)	0.807±.035
eclipse	**0.823±.036**(.0)	0.723±.105	0.822±.047(.90)	0.824±.044
equinox	**0.802±.039**(.0)	0.533±.131	0.805±.039(.87)	0.806±.038
lucene	**0.816±.090**(.0)	0.654±.194	0.842±.063(.95)	0.841±.065
mylyn	**0.795±.037**(.0)	0.711±.067	0.735±.055(.51)	0.740±.053
pde	**0.763±.054**(.0)	0.690±.122	**0.779±.051**(.0)	0.748±.060
P2.0-2.1	**0.777±.005**(.0)	0.757±.000	**0.762±.000**(.0)	0.756±.000
P2.1-3.0	**0.792±.017**(.0)	0.722±.000	**0.804±.000**(.0)	0.801±.000
F2.0-2.1	**0.769±.003**(.0)	0.744±.000	**0.764±.000**(.0)	0.7639±.000
F2.1-3.0	**0.758±.012**(.0)	0.727±.000	0.7869±.000(.0)	**0.7871±.000**

TABLE VI
CONDITION NUMBERS (CNS) FOR THE EXPERIMENTAL DATA SETS

datasets	CNs all	CNs three
Files2.0	Infinite	524489
Files2.1	Infinite	562976
Files3.0	Infinite	79368
Package2.0	Infinite	4804
Package2.1	Infinite	46874
Package3.0	Infinite	49767
eclipse	4.72E+21	1052
equinox	4.53E+28	640721
lucene	8.93E+18	329
mylyn	1.69E+19	1069946
pde	5.22E+18	372

The Infinite table entries are caused by the existence of metrics with only one value. According to He's definition [30], when the condition number is larger than 100, there exists severe multicollinearity. Therefore, all of these data sets have the problem of severe multicollinearity, which violates the assumption for multiple linear regression models. Under such a situation, the linear models by LS are not accurate, which implies the impropriety of linear models by LS. From Table V, LS constructs better models using three metrics than using all metrics over most data sets. The reason is that pruning useless metrics can reduce the condition number, and the model can be more accurate, which indicates the usefulness of metric selection.

To sum up, LTR can achieve better results than LS in most cases, especially when all metrics are used, which shows that it is easier for LTR to capture the important factors related to defects by directly optimizing the ranking performance measure than LS by optimizing the prediction errors.

C. Comparison of the Learning-to-Rank Approach With Eight Existing Methods

This subsection aims at finding whether the LTR linear model is comparable with existing nonlinear models. We

TABLE VII
COMPARISON OF THE LTR APPROACH WITH FOUR EXISTING METHODS OVER 11 DATA SETS WITH ALL METRICS. RESULTS IN BOLDFACE ARE *S*-SIGNIFICANTLY BETTER THAN LTR, AND RESULTS IN ITALIC ARE *S*-SIGNIFICANTLY WORSE THAN LTR. THE CORRESPONDING P-VALUES ARE SHOWN IN THE BRACKETS

datasets	LTR	RF	RP	BART	NBR	random	ideal
Files2.0	0.825±.018	**0.853±.014**(.00)	*0.788±.024*(.00)	*0.776±.043* (.00)	**0.830±.022**(.01)	*0.516±.104*	**0.952±.0.002**
Files2.1	0.779±.022	**0.794±.019**(.00)	*0.704±.028*(.00)	*0.742±.029*(.00)	**0.786±.024**(.00)	*0.513±.099*	**0.959±.001**
Files3.0	0.795±.019	**0.801±.015**(.04)	*0.737±.025*(.00)	*0.771±.025* (.00)	0.788±.025(.09)	*0.485±.086*	**0.951±.001**
Package2.0	0.785±.064	0.776±.070 (.30)	*0.709±.090* (.00)	*0.726±.085* (.00)	*0.656±.128*(.00)	*0.504±.124*	**0.890±.020**
Package2.1	0.772±.058	0.769±.054(.46)	*0.755±.058*(.02)	*0.750±.071*(.01)	*0.632±.123*(.00)	*0.520±.121*	**0.890±.017**
Package3.0	0.814±.033	0.811±.032 (.38)	*0.763±.046*(.00)	*0.780±.047*(.00)	*0.691±.108*(.00)	*0.496±.111*	**0.899±.009**
eclipse	0.823±.036	**0.864±.026** (.00)	*0.804±.045*(.00)	0.827±.040 (.24)	*0.678±.100*(.00)	*0.475±.132*	**0.938±.004**
equinox	0.802±.039	**0.816±.036** (.02)	*0.761±.055*(.00)	*0.779±.055*(.00)	*0.522±.139*(.00)	*0.491±.103*	**0.885±.015**
lucene	0.816±.090	**0.859±.066**(.00)	*0.719±.109*(.00)	*0.741±.138* (.00)	*0.604±.183*(.00)	*0.503±.195*	**0.972±.004**
mylyn	0.795±.037	**0.812±.038**(.00)	*0.740±.056*(.00)	*0.744±.070*(.00)	*0.684±.071*(.00)	*0.517±.130*	**0.952±.004**
pde	0.763±.054	**0.808±.044**(.00)	*0.699±.075*(.00)	*0.718±.085* (.00)	*0.621±.120*(.00)	*0.519±.157*	**0.954±.007**

TABLE VIII
MEANS, AND STANDARD DEVIATIONS (SD) OF 100 RESULTS OF NINE COMPARED METHODS OVER 11 DATA SETS WITH THREE METRICS. RESULTS IN BOLDFACE ARE *S*-SIGNIFICANTLY BETTER THAN LTR, AND RESULTS IN ITALIC ARE *S*-SIGNIFICANTLY WORSE THAN LTR ACCORDING TO MEANS. THE CORRESPONDING P-VALUES ARE SHOWN IN THE BRACKETS

means	LTR	RF	RP	BART	NBR	ZINBR	ZIPR	HNBR	HPR
Files2.0	0.805±.021	**0.814±.018**(.00)	*0.749±.035*(.00)	*0.782±.030*(.00)	*0.796±.023*(.01)	0.807±.021(.62)	NA	*0.798±.023*(.02)	NA
Files2.1	0.765±.022	*0.758±.021*(.01)	*0.722±.025*(.00)	*0.748±.025*(.00)	*0.749±.024*(.00)	0.764±.022(.74)	NA	*0.751±.023*(.00)	NA
Files3.0	0.788±.016	*0.783±.017*(.03)	*0.731±.025*(.00)	*0.756±.023*(.00)	*0.775±.021*(.00)	0.787±.017(.71)	*0.778±.021*(.00)	*0.777±.021*(.00)	*0.776±.021*(.00)
Package2.0	0.769±.063	0.756±.067(.11)	*0.676±.087*(.00)	*0.746±.067*(.00)	0.771±.061(.85)	0.773±.065(.49)	0.773±.063(.50)	0.772±.064(.55)	0.773±.063(.52)
Package2.1	0.776±.062	*0.758±.066*(.02)	*0.724±.073*(.00)	*0.722±.098*(.00)	0.774±.062(.83)	0.776±.062(.97)	0.776±.062(.99)	0.776±.062(.99)	0.775±.062(.97)
Package3.0	0.815±.031	*0.800±.033*(.00)	*0.745±.051*(.00)	*0.792±.037*(.00)	0.813±.031(.84)	0.814±.032(.97)	0.814±.032(.91)	0.814±.032(.95)	0.814±.032(.89)
eclipse	0.822±.047	0.826±.039(.65)	*0.799±.045*(.00)	*0.803±.045*(.00)	0.820±.054(.99)	0.825±.045(.63)	0.825±.046(.65)	0.822±.051(.87)	0.822±.051(.86)
equinox	0.805±.039	0.804±.040(.86)	*0.768±.051*(.00)	*0.781±.048*(.00)	0.804±.040(.94)	0.803±.040(.70)	0.803±.040(.65)	0.804±.040(.93)	0.804±.040(.93)
lucene	0.842±.063	0.836±.072(.66)	*0.748±.098*(.00)	*0.770±.113*(.00)	*0.828±.073*(.22)	0.839±.065(.79)	0.839±.065(.83)	0.842±.064(.93)	0.842±.064(.94)
mylyn	0.735±.055	0.737±.057(.70)	*0.684±.057*(.00)	*0.692±.072*(.00)	0.740±.053(.57)	0.739±.053(.64)	0.739±.053(.64)	0.741±.053(.56)	0.741±.053(.55)
pde	0.779±.051	0.774±.057(.54)	*0.697±.063*(.00)	*0.720±.083*(.00)	0.780±.050(.83)	0.779±.051(.94)	0.781±.050(.78)	0.780±.050(.88)	0.780±.050(.90)

compare the LTR approach with eight existing methods (RF, RP, BART, NBR, ZINBR, ZIPR, HNBR, HPR) over eleven data sets with all metrics, and three metrics selected by InfoGain. 10 times 10-fold cross-validation is used. All eight methods are implemented using R packages. As mentioned before, we use the suggested parameters in Weyuker *et al.*'s comparison study [10] for RF, RP, and BART; and use the default parameters for other variations of Poisson regression models because default parameters seemed to be used in Gao *et al.*'s work [11]. However, for some data sets, some of these methods cannot produce a result. To be specific, RF cannot produce results over Eclipse_II files2.1 and files3.0 data sets because it cannot allocate such a large memory size, so we use part (95% to 85%, depending on specific data sets) of the training data to train RF models over these two data sets. For NBR and other variations of Poisson regression models, they output errors such as "NA/NaN/Inf in foreign function call", which is triggered because of the absence of an inverse matrix, or the impossibility for a Gamma random variable that the deviance calculation returns something non-finite. Therefore, we try to tune parameters to produce a result when errors are produced. However, some methods cannot produce a result by tuning parameters over some data sets (e.g., ZINBR cannot produce a result over data sets with all metrics by tuning all possible parameters), so the corresponding results are recorded as NA. Results are shown in Tables VII and VIII. Only four existing methods are compared in Table VII because the other four methods (ZINBR, ZIPR, HNBR, HPR) output errors over all data sets with all metrics, which reflects that they have more constraints for the training data. To better evaluate these models, we also show the results of random models which predict modules randomly, and ideal models which give a perfect ranking (the accurate numbers of defects).

From Tables VII and VIII, we can see that RF and LTR obtain better results than other compared methods (except for the ideal models). When all metrics are used, LTR achieves comparable results over three data sets, but worse FPA results than RF over eight data sets. When using only three metrics, LTR obtains better results than RF over four data sets, worse results over only one data set, and comparable results over the other six data sets. LTR performs better than RF using fewer metrics, but worse than RF using all metrics. This might be caused by two potential reasons. One reason is that CoDE could not find the optimal solution for LTR when the number of metrics is large (which corresponds to the solution space). The other reason is that the linear model is more appropriate for three metrics, but not for all metrics. Nevertheless, LTR is an attractive choice for constructing SDP models because of its simplicity. It would be much easier to tell which metrics are more critical from the coefficients of the linear models than from RF models.

When using all metrics, LTR achieves better FPA results than NBR over eight data sets, worse results over only two data sets, and comparable results over one data set. When using three metrics, LTR obtains better results than NBR over four data sets, and no worse results over all data sets. Therefore, LTR is better than NBR as a whole. NBR is obtained by maximum likelihood, which also relates to the specific number of defects. With only a few samples, it is easier to optimize a model to predict the ranking of defects than to accurately predict the specific number of defects, especially when the solution space is

TABLE IX
TESTING RESULTS OVER THE SERIES OF ECLIPSE_II DATA SETS WITH ALL METRICS. RESULTS IN BOLDFACE ARE *S*-SIGNIFICANTLY BETTER THAN LTR, AND RESULTS IN ITALIC ARE *S*-SIGNIFICANTLY WORSE THAN LTR

datasets	LTR	RF	RP	BART	NBR	LS
P2.0-2.1	0.777±.005	0.779±.001(.47)	*0.706±.0*(.0)	*0.760±.010*(.0)	*0.761±.0*(.0)	*0.757±.0*(.0)
P2.1-3.0	0.792±.017	0.797±.001(.25)	*0.743±.0*(.0)	*0.784±.007*(.02)	*0.685±.0*(.0)	*0.722±.0*(.0)
F2.0-2.1	0.769±.003	0.770±.001(.91)	*0.725±.0*(.0)	*0.715±.002*(.0)	*0.765±.0*(.0)	*0.744±.0*(.0)
F2.1-3.0	0.758±.012	**0.769±.000**(.0)	*0.676±.0*(.0)	*0.712±.007*(.0)	*0.718±.0*(.0)	*0.727±.0*(.0)

TABLE X
TESTING RESULTS OVER THE SERIES OF REDUCED ECLIPSE_II DATA SETS WITH THREE METRICS. RESULTS IN BOLDFACE ARE *S*-SIGNIFICANTLY BETTER THAN LTR, AND RESULTS IN ITALIC ARE *S*-SIGNIFICANTLY WORSE THAN LTR

datasets	LTR	RF	RP	BART	NBR	LS
P2.0-2.1	0.7616±.000	**0.767±.001**(.0)	*0.708±.0*(.0)	*0.757±.002*(.0)	**0.7622±.0**(.0)	*0.756±.0*(.0)
P2.1-3.0	0.8038±.000	*0.793±.001*(.0)	*0.740±.0*(.0)	*0.772±.010*(.0)	**0.8048±.0**(.0)	*0.801±.0*(.0)
F2.0-2.1	0.764±.000	*0.755±.000*(.0)	*0.731±.0*(.0)	*0.751±.004*(.0)	*0.747±.0*(.0)	*0.764±.0*(.0)
F2.1-3.0	0.7869±.000	*0.756±.000*(.0)	*0.729±.0*(.0)	*0.765±.003*(.0)	*0.778±.0*(.0)	**0.7871±.0**(.0)

large (the number of metrics is large). The worse results of NBR than LTR over Files2.0, Files2.1, and Files3.0 using all metrics indicate that a good model according to individual-based loss functions might give a poor ranking. Therefore, LTR can obtain better results by optimizing the ranking performance directly in this situation.

LTR is no worse than ZINBR, ZIPR, HNBR, and HPR over all data sets with three metrics. In addition, ZINBR, ZIPR, HNBR, and HPR cannot produce results over data sets with all metrics, and both ZIPR and HPR cannot produce results over two data sets with three metrics. These methods are sensitive to the training data.

The worst performance of RP (a tree) among all methods indicates that a linear or generalized linear model seems to be more appropriate than a tree model for SDP for the ranking task. Nevertheless, BART (sum of trees) achieves larger mean FPA results than RP over most data sets, and RF (ensemble classifiers consisting of many trees) performs better than both BART and RP. This might reflect that ensemble algorithms might be useful to improve the performance of SDP models for the ranking task.

Compared with random models, these construction models are useful, and can give better module ranking according to the number of defects. Nevertheless, compared with ideal models, we still have much room to improve.

To sum up, the LTR linear model is comparable to existing nonlinear models. The LTR approach and RF perform better than other existing methods in most cases. The LTR approach is better for data sets with three metrics, and RF is better for data sets with all metrics. Compared with variations of Poisson regression models, directly optimizing the ranking performance measure is better than maximum likelihood, especially for small data sets with many metrics. Compared with RP and BART, the LTR approach obtains better results over all data sets.

D. Comparison of LTR and Existing Methods Over Data Sets With More Releases

SDP models are usually used to predict the defects of software modules in a new release [3], [7]. Results based on data sets with more releases are particularly interesting and useful in practice. This section aims at further comparing LTR with existing methods over data sets with more releases. To be specific, we apply them to Eclipse_II data sets, and their reduced data sets with three metrics selected by InfoGain. We use the former release as the training set, and the latter release as the testing set. The metrics of reduced data sets are selected based on only training data. Results are recorded in Tables IX and X. P2.0-2.1 in the tables means using package2.0 as the training set and package2.1 as the testing set, and others are similar. Because some algorithms cannot obtain models over some data sets, we compare LTR with only five methods in this experiment.

In general, results in Tables IX and X are consistent with results in Tables VII and VIII when comparing LTR and other existing methods. LTR and RF perform best among all these methods, RF is better for all metrics, and LTR is better for three metrics. The comparison of LTR and NBR further demonstrates that directly optimizing the ranking performance is better than optimizing other loss functions (LS and maximum likelihood) for most cases, especially when there are many metrics.

According to Tables IX and X, RP, NBR, and LS are stable methods which achieve fixed models when training data are fixed because their standard variance is 0. RF is also quite stable, whose standard variance is small (less than 0.001). LTR can obtain comparably stable models for three metrics, but not for all metrics. BART is not stable for either all metrics or three metrics. However, from Tables V, VII, and VIII, the standard deviations of all methods are relatively large, yet the standard deviation of LTR is even smaller than the stable methods in many cases. The high standard deviations of these models might suggest that their parameters are sensitive to data in some degree. The main reason might be over-fitting, especially for relatively fixed models such as RF, RP, NBR, and LS. For unstable methods such as LTR and BART, instability is another reason.

Because the percentage of defects in the top 20% modules is a practical measure in real situations [10], we show the defect numbers in the top 20% modules of one specific run (randomly chosen) over Eclipse_II with multiple releases in Fig. 1, to further compare these six methods. LinearR means linear models

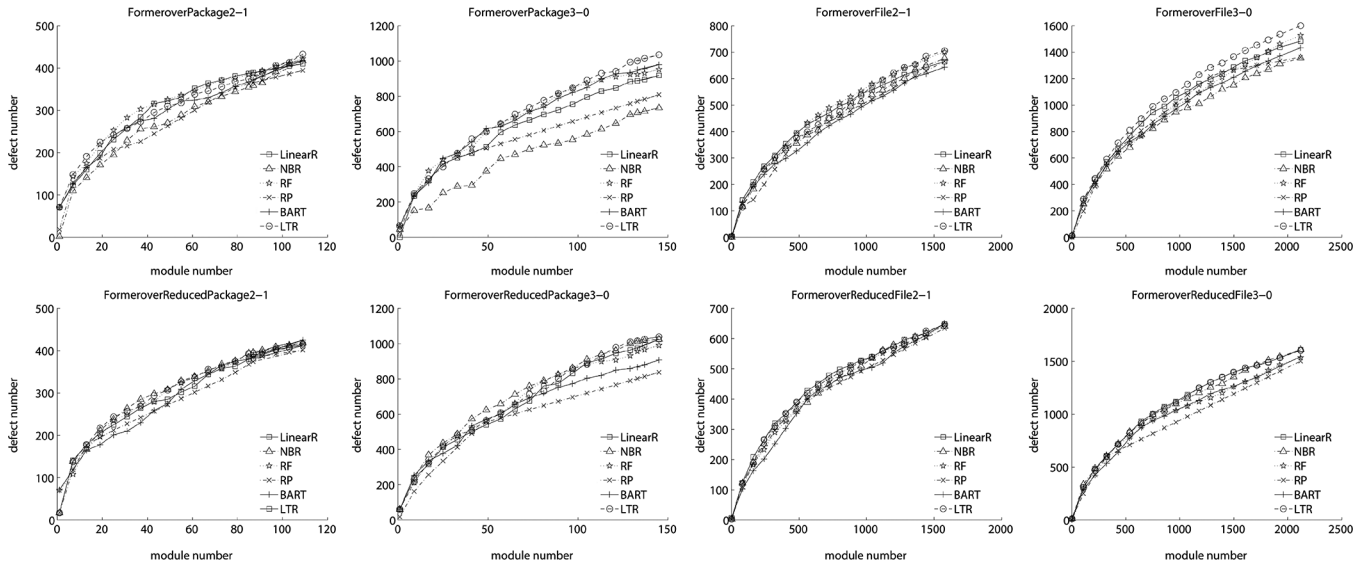

Fig. 1. Comparison of Six Methods over Eclipse_II, above using all metrics, below using three metrics.

by LS. As we can see, LTR performs best according to the defect number in the top 20% modules when all metrics are used. However, the differences between LTR, LS, NBR, and RF are not distinct for three metrics.

To sum up, LTR and RF perform best among all these methods. According to FPA, RF is better for all metrics, and LTR is better for three metrics. According to the percentage of defects in the top 20% modules, LTR performs best for all metrics, but the differences between LTR, LS, NBR, and RF are not distinct for three metrics.

VI. Threats to Validity

Eleven public datasets are used in our experiments, and eight existing methods for constructing SDP models are compared. The cross-validation method is used to obtain results. All of these make us confident that the obtained results are strongly related to the SDP domain, and the results are convincing. Despite our extensive experimental studies, some potential threats to validity should be considered.

A. Threats to Internal Validity

A threat to the validity of the results presented in this study is the choice of the performance measure for SDP for the ranking task. There are reasons for us to choose FPA; (1) has demonstrated that FPA is similar to the area under the Alberg diagram that was demonstrated to be reasonable for evaluating SDP models for the ranking task [13]; FPA is the up-to-date performance measure proposed for SDP for the ranking task, which overcomes the shortcoming of the percentage of the defects found in the top m modules [10].

However, when other goals of SDP models for the ranking task are considered (for example, only the percentage of defects found in the top m modules are considered), we should change the choice, which might lead to a different conclusion.

Another threat is the choice of parameters for the model construction methods. We simply set the parameters according to existing work [10] or default parameters. For different datasets or different numbers of metrics, the best parameters might be different, which might lead to different results.

B. Threats to External Validity

The main threat to external validity is the datasets. There are not many existing publicly available datasets for the ranking task. Our experiments are based on a large collection of publicly available datasets, including not only the commonly used Eclipse_II [7], but also the recently provided datasets that include many process metrics [5]. However, these two sets of data are only a very small part of all possible datasets, among which there are many datasets based on industrial software systems [11] that are not publicly available. Other datasets might include different metrics, and totally different modules. Therefore, the conclusions over these two sets of data might not hold for other datasets.

The idea of directly optimizing the ranking performance can be applied to any models, that is, linear or nonlinear models. However, the performance of the specific LTR model in this paper relies on CoDE. For other models, CoDE might be inappropriate.

Finally, we ignore the module size or testing cost in this paper, and only ranking models that predict an order of modules according to the number of defects are considered. Some applications might prefer ranking models according to the defect density, or require us to consider testing cost, whose predicted values should be different. Although the method might be applicable by simply changing the predicted value of the number of defects into the defect density, there is no guarantee that this method will work well on a different goal.

VII. Conclusions and Future Work

SDP for the ranking task helps to allocate testing resources more efficiently by predicting which modules are likely to have more defects [10]–[12]. SDP data are collected by different companies and by different people, which are noisy. As a result, predicting a precise number of defects for each software

module is hard or even impossible due to the lack of accurate historical data. Some researchers propose to use the performance measure based on ranking to evaluate SDP models, such as CLC [6], [13], and FPA [10]. However, existing SDP models are optimized to accurately predict a specific number of defects. Yet a good model according to individual-based loss functions might fail to give a good ranking. Therefore, we suggested constructing models by directly optimizing the ranking performance measure [14].

In this paper, we have applied the LTR approach to a wide range of real-world data sets, and given a comprehensive evaluation and comparison of LTR against other algorithms. We have also investigated the relationship between CLC and FPA, and the necessity of metric selection over two sets of data for SDP for the ranking task.

The main findings from our studies include the following.

1) According to the comparison of LTR with the linear model using LS, LTR models achieved *s*-significantly better results than LS models, especially over data sets with many metrics. These results showed that directly optimizing the model performance measure could lead to better prediction models than optimizing square errors in most cases, because directly optimizing the ranking performance measure could capture important factors related to defects more easily. Therefore, the idea of directly optimizing the ranking performance is useful.
2) According to the comparison of LTR with existing methods, LTR and RF were better than other methods. LTR was better on data sets with three metrics, and RF was better on the original data sets. According to the percentages of defects in the top 20% modules, LTR was better over data sets with multiple releases. Compared with RF, LTR models did not perform better in all cases. However, LTR has its own merits, including better performance in some cases, and clearer interpretation of the relationship between prediction values and metrics. As noted by Weyuker *et al.* [10], it is difficult for RF to interpret the relationship between prediction values and metrics. In contrast, linear models can give a clearer interpretation of the relationship that may indicate which metrics are more relevant to defects. Overall, LTR is a good choice for constructing SDP models for the ranking task with both high accuracy and clear interpretability.
3) The main reason for the unsuitability of linear regression models and generalized linear regression models such as NBR in solving SDP for the ranking task is that there exists severe multicollinearity in the experimental data.
4) According to (1), CLC and FPA should be consistent in evaluating a ranking. This means that using one of them is sufficient to evaluate SDP models for the ranking task.
5) Our empirical studies show that two or three metrics worked well for SDP for the ranking task.

As a whole, our approach has two major benefits. Firstly, LTR learns relative ranking only, and does not need to predict the number of defects for each software module precisely. Hence, LTR is more robust against noisy SDP data. Secondly, those existing approaches that try to predict explicitly the number of defects in a software module, actually used these predicted numbers to rank the modules, to allocate testing resources to the most faulty modules. In such cases, it is natural to learn the ranking directly. LTR provides such a way.

Directly optimizing the ranking performance measure of prediction models could give a better ranking than optimizing the individual-based loss functions, and LTR performed well in our experiments, especially over data sets with three metrics. However, when the number of metrics is large, the LTR linear model performed worse than RF. The potential reasons include the difficulty for CoDE to find the optimal solution in a large solution space, and the insufficiency of the linear model to characterize all information when there are many metrics. Therefore, in our future work, we study how to find the optimal solution for the LTR model in the large solution space, and investigate the LTR nonlinear model for SDP for the ranking task. In this paper, we have only applied InfoGain to select the metrics. In our future work, we will apply more metric selection methods to investigate more deeply the effectiveness of metrics for SDP for the ranking task.

REFERENCES

[1] N. E. Fenton and S. L. Pfleeger, *Software Metrics: A Rigorous and Practical Approach*. Boston, MA, USA: PWS Publishing, 1998.
[2] E. E. Mills, Software Metrics 1998, Tech. Rep., DTIC Document.
[3] T. J. Ostrand, E. J. Weyuker, and R. M. Bell, "Predicting the location and number of faults in large software systems," *IEEE Trans. Softw. Eng.*, vol. 31, no. 4, pp. 340–355, 2005.
[4] R. Moser, W. Pedrycz, and G. Succi, "A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction," in *Proc. ACM/IEEE 30th Int. Conf. Software Engineering*, 2008, pp. 181–190.
[5] M. D'Ambros, M. Lanza, and R. Robbes, "Evaluating defect prediction approaches: A benchmark and an extensive comparison," *Empiric. Softw. Eng.*, pp. 1–47, 2011.
[6] Y. Jiang, B. Cukic, and Y. Ma, "Techniques for evaluating fault prediction models," *Empiric. Softw. Eng.*, vol. 13, no. 5, pp. 561–595, 2008.
[7] T. Zimmermann, R. Premraj, and A. Zeller, "Predicting defects for eclipse," in *Proc. Int. Workshop Predictor Models in Software Engineering (PROMISE'07)*, 2007, pp. 9–15.
[8] S. Wang and X. Yao, "Using class imbalance learning for software defect prediction," *IEEE Trans. Rel.*, vol. 62, no. 2, pp. 434–443, Jun. 2013.
[9] S. Bibi, G. Tsoumakas, I. Stamelos, and I. Vlahavas, "Regression via classification applied on software defect estimation," *Expert Syst. Applicat.*, vol. 34, no. 3, pp. 2091–2101, 2008.
[10] E. J. Weyuker, T. J. Ostrand, and R. M. Bell, "Comparing the effectiveness of several modeling methods for fault prediction," *Empiric. Softw. Eng.*, vol. 15, no. 3, pp. 277–295, 2010.
[11] K. Gao and T. M. Khoshgoftaar, "A comprehensive empirical study of count models for software defect prediction," *IEEE Trans. Rel.*, vol. 56, no. 2, pp. 223–236, Jun. 2007.
[12] T. M. Khoshgoftaar and N. Seliya, "Fault prediction modeling for software quality estimation: Comparing commonly used techniques," *Empiric. Softw. Eng.*, vol. 8, no. 3, pp. 255–283, 2003.
[13] N. Ohlsson and H. Alberg, "Predicting fault-prone software modules in telephone switches," *IEEE Trans. Softw. Eng.*, vol. 22, no. 12, pp. 886–894, 1996.
[14] X. Yang, K. Tang, and X. Yao, "A learning-to-rank algorithm for constructing defect prediction models," in *Intelligent Data Engineering and Automated Learning-IDEAL 2012*, 2012, pp. 167–175.
[15] 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.
[16] 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.
[17] 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.

[18] T. J. McCabe, "A complexity measure," *IEEE Trans. Softw. Eng.*, no. 4, pp. 308–320, 1976.
[19] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Trans. Softw. Eng.*, vol. 20, no. 6, pp. 476–493, 1994.
[20] T. L. Graves, A. F. Karr, J. S. Marron, and H. Siy, "Predicting fault incidence using software change history," *IEEE Trans. Softw. Eng.*, vol. 26, no. 7, pp. 653–661, 2000.
[21] Z. Xu, T. M. Khoshgoftaar, and E. B. Allen, "Prediction of software faults using fuzzy nonlinear regression modeling," in *Proc. 5th IEEE Int. Symp. High Assurance Systems Engineering (HASE)*, 2000, pp. 281–290.
[22] C. Catal and B. Diri, "A systematic review of software fault prediction studies," *Expert Syst. Applicat.*, vol. 36, no. 4, pp. 7346–7354, 2009.
[23] L. Guo, Y. Ma, B. Cukic, and H. Singh, "Robust prediction of fault-proneness by random forests," in *Proc. 15th Int. Symp. Software Reliability Engineering (ISSRE)*, 2004, pp. 417–428.
[24] K. O. Elish and M. O. Elish, "Predicting defect-prone software modules using support vector machines," *J. Syst. Softw.*, vol. 81, no. 5, pp. 649–660, 2008.
[25] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking classification models for software defect prediction: A proposed framework and novel findings," *IEEE Trans. Softw. Eng.*, vol. 34, no. 4, pp. 485–496, 2008.
[26] E. Arisholm, L. C. Briand, and E. B. Johannessen, "A systematic and comprehensive investigation of methods to build and evaluate fault prediction models," *J. Syst. Softw.*, vol. 83, no. 1, pp. 2–17, 2010.
[27] E. Arisholm, L. C. Briand, and M. Fuglerud, "Data mining techniques for building fault-proneness models in telecom java software," in *Proc. 18th IEEE Int. Symp. Software Reliability (ISSRE)*, 2007, pp. 215–224.
[28] T. Mende and R. Koschke, "Revisiting the evaluation of defect prediction models," in *Proc. 5th Int. Conf. Predictor Models in Software Engineering*, 2009, pp. 1–10.
[29] Y. Wang, Z. Cai, and Q. Zhang, "Differential evolution with composite trial vector generation strategies and control parameters," *IEEE Trans. Evol. Computat.*, vol. 15, no. 1, pp. 55–66, 2011.
[30] X. He, *Practical Regression Analysis*. Beijing, China: Higher Education Press, 2008.

Xiaoxing Yang received the B.S. degree in computer science from School of Computer Science and Technology, University of Science and Technology of China (USTC), Hefei, China, in 2007. She is currently working toward her Ph.D. degree at the USTC-Birmingham Joint Research Institute in Intelligent Computation and Its Applications (UBRI), School of Computer Science and Technology, USTC.

Her research interests include software defect prediction, machine learning, evolutionary computation, and route planning.

Ke Tang (M'07–SM'13) received the B.Eng. degree from Huazhong University of Science and Technology, Wuhan, China, in 2002; and the Ph.D. degree from Nanyang Technological University, Singapore, in 2007.

From 2007–2011, he was a Lecturer and Associate Professor with the School of Computer Science and Technology, University of Science and Technology of China (USTC), Hefei, China. Since 2011, he has been a professor at the USTC-Birmingham Joint Research Institute in Intelligent Computation and Its Applications (UBRI), School of Computer Science and Technology, USTC. He has authored or co-authored more than 80 refereed publications. His major research interests include evolutionary computation, machine learning, data mining, large scale global optimization, dynamic and robust optimization, and real-world applications.

Dr. Tang is an associate editor of the IEEE COMPUTATIONAL INTELLIGENCE MAGAZINE, the *Computational Optimization and Applications Journal*, and the *Frontiers of Computer Science Journal*.

Xin Yao (M'91–SM'96–F'03) is a Chair (Professor) of Computer Science, and the Director of CERCIA (the Centre of Excellence for Research in Computational Intelligence and Applications), University of Birmingham, U.K. His major research interests include evolutionary computation and ensemble learning. He has more than 400 refereed publications in international journals and conferences.

He is the President (2014–2015) of IEEE Computational Intelligence Society (CIS). His work won the 2001 IEEE Donald G. Fink Prize Paper Award, 2010 IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION Outstanding Paper Award, 2010 BT Gordon Radley Award for Best Author of Innovation (Finalist), 2011 IEEE Transactions on Neural Networks Outstanding Paper Award, and many other best paper awards. He won the prestigious Royal Society Wolfson Research Merit Award in 2012, and the 2013 IEEE CIS Evolutionary Computation Pioneer Award. He was the Editor-in-Chief (2003–2008) of IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION.