A Multi-Objective Learning Method for Building Sparse Defect Prediction Models

1st Xin Li
School of Data and Computer Science
Sun Yat-Sen University
Guangzhou, China
lixin49@mail2.sysu.edu.cn

2nd Xiaoxing Yang
College of Big Data and Internat
Shenzhen Technology University
Shenzhen, China
yangxiaoxing@sztu.edu.cn

3rd Jianmin Su
School of Intelligent Manufacturing
Wuyi University
Jiangmen, China
jackysura@163.com

4th Wushao Wen
School of Data and Computer Science
Sun Yat-Sen University
Guangzhou, China
wenwsh@mail.sysu.edu.cn

***Abstract*—Software defect prediction constructs a model from the previous version of a software project to predict defects in the current version, which can help software testers to focus on software modules with more defects in the current version. Most existing methods construct defect prediction models through minimizing the defect prediction error measures. Some researchers proposed model construction approaches that directly optimized the ranking performance in order to achieve an accurate order. However, the model complexity is also considered. Therefore, defect prediction can be seen as a multi-objective optimization problem and should be solved by multi-objective approaches. And hence, in this paper, we employ an existing multi-objective evolutionary algorithm and propose a new multi-objective learning method based on it, to construct defect prediction models by simultaneously optimizing more than one goal. Experimental results over 30 sets of cross-version data show the effectiveness of the proposed multi-objective approaches.**

***Index Terms*—software defect prediction, multi-objective learning method, ranking task, cross-version defect prediction**

I. Introduction

Software defect prediction employs software metrics (also referred to as features or attributes) of software modules (such as classes, files, packages, etc) to construct defect prediction models, in order to predict defect information of new software modules [15]. It can support software testing activities, by helping software testers to focus on software modules with defects, in order to timely detect and repair defects before releasing [13]. Cross-version defect prediction constructs a model from the previous version of a software project to predict defects in the current version, so it can help software testers to focus on software modules with more defects in the current version [11].

There are two most frequently investigated goals of software defect prediction: to predict an order of software modules based on the predicted defect numbers (which we call as ranking task) [10], [15], and to predict whether a software module has defects or not (which we call as classification task) [8]. In this paper, the former goal is studied. The detailed process is as follows [15]: first, data is obtained from software modules according to metrics such as lines of code, and response for a class [4]; secondly, a model is constructed based on the data from modules with known defect numbers; and finally, the model is used to predict defects of software modules with unknown defect numbers, so that an order of these modules based on the predicted defect numbers is obtained.

Most existing methods, such as linear regression [9], negative binomial regression [10], recursive partitioning, Bayesian additive regression trees, and random forest [12], construct defect prediction models by maximum likelihood estimation or least squares, focusing on the fitting of each sample. Considering that prediction models constructed by these methods might fail to give an accurate order (because of the difficulty to predict the exact number of defects in a software module due to noisy data), Yang et al. [14] proposed an effective learning-to-rank method, which constructed defect prediction models by directly optimizing the ranking performance.

However, in some cases, the model complexity is also considered. That is, when performance is the same, a simple model is better than a complex model. Furthermore, in some cases, both a good order of software modules and a high prediction accuracy are desired. In other words, an accurate order is needed to help software testers to allocate testing resources (for instance, more testing resources for modules with more defects), and the predicted defect numbers should be reasonable or meaningful (for example, if true defect number is 3, a predicted defect number around 3 might be better than 30000). Therefore, defect prediction for the ranking task can be seen as a multi-objective optimization problem. And hence, in this paper, we investigate the application of multi-objective learning methods to construct defect prediction models by simultaneously optimizing both ranking performance and model complexity.

This work is supported by National Natural Science Foundation of China (Grants No. 61602534) and Fundamental Research Funds for the Central Universities (No.17lgpy122). (Corresponding author: Xiaoxing Yang)

The rest of this paper is organized as follows. Section 2 presents related work. In Section 3, an overview of the investigated multi-objective optimization methods are described, including the original one and the proposed one. In Section 4, we detail the experimental methodologies. Experimental results are reported in Section 5. Section 6 presents the threats to validity, and Section 7 draws the conclusions.

II. Related Work

Software defect prediction for the ranking task can help software testers to focus on software modules with more defects, which attracts many researchers. For example, Gao et al. [3] compared eight count models, and they concluded that zero-inflated negative binomial regression, hurdle negative binomial regression and hurdle Poisson regression with threshold 2 were more effective. Weyuker et al. [12] compared four approaches (negative binomial regression, random forest, recursive partitioning and Bayesian additive regression trees) to predict the number of defects in software modules, and they concluded that negative binomial regression and random forest models performed better than the other two models. Yang and Wen [15] investigated two penalized regression methods for constructing prediction models, and they concluded that both penalized regression performed better than linear regression and negative binomial regression for cross-version defect prediction because they could handle the multi-collinearity problems in the datasets.

The above-mentioned methods construct models by minimizing prediction error measures such as average, relative, or mean square errors, which might not provide a direct insight into the ranking performance [6]. Therefore, some researchers employed evolutionary algorithm to directly optimize the ranking performance. For example, Khoshgoftaar et al. [6] used a GP-based approach to optimize tree size and performances for the 95%, 90%, 80%, and 70% cutoff percentile values to obtain prediction models, and they concluded better performance of the proposed models. Yang et al. [14] proposed a learning-to-rank approach to directly optimize the ranking performance measure (fault-percentile-average), and the experimental results showed that the learning-to-rank approach and random forest were better than other methods to construct defect prediction models for the ranking task.

Realizing that single-objective approaches might be insufficient for solving software defect prediction problems, some researchers proposed multi-objective learning algorithms to construct models [7]. For example, in 2004, Khoshgoftaar et al. [6] used a GP-based approach to optimize five objectives, in order to obtain a better ranking. To be noted, the five objectives were in order, i.e., the first objective had priority over other objectives, and then came the second objective, etc. In 2007, Khoshgoftaar et al. [5] proposed a multi-objective optimization method to construct models to fit different applications, which simultaneously optimized a balance of classification accuracy, available resources and model simplicity. In order to deal with the imbalanced problem in software defect prediction, Carvalho et al. [1] applied multi-objective particle swarm optimization to obtain a set of rules with sensitivity and specificity as the objectives, and these rules composed a classifier. Shukla et al. [11] compared a multi-objective logistic regression method with four traditional machine learning algorithms for cross-version defect prediction for the classification task, and they concluded better performance of the multi-objective logistic regression. Li et al. [7] compared a multi-objective approach that optimized two objectives and a single-objective approach that directly optimized a trade-off of the two objectives for classification software defect prediction, and they pointed out that when the trade-off of objectives was known, it might be better to choose single-objective approaches to directly optimize the trade-off, instead of using multi-objective approaches.

III. Multi-Objective Optimization

According to related work, existing studies about software defect prediction for the ranking task constructed software defect prediction models by optimizing prediction error measures, or the ranking performance, ignoring model complexity. In some cases, a simple model is desired, and both a good order to help testers to allocate testing resources, and a good predicted accuracy as reference are desired. Therefore, defect prediction can be seen as a multi-objective optimization problem, and hence we employ a multi-objective evolutionary algorithm to solve it.

In this section, we firstly introduce the involved objectives (fitness functions). Subsequently, the employed multi-objective evolutionary approach and the revised method are described.

A. Fitness Functions

As mentioned above, all three objectives (a good order, a high prediction accuracy, and a simple model) might be desired in some cases. Hence, three fitness functions are described in this subsection.

In this study, we adopt fault-percentile-average (FPA) [12] as the ranking performance measure. Considering m modules $f_1, f_2, \ldots, f_m$, listed in increasing order of predicted defect number, s_i as the actual defect number in the module f_i, and s=s_1+s_2+...+s_m as the total number of defects in all modules, the proportion of actual defects in the top t predicted modules (i.e. top t modules predicted to have most defects) to the whole defects is $\frac{1}{s}\sum_{i=m-t+1}^{m} s_i$. Then FPA is defined as follows [12]:

$$\frac{1}{m}\sum_{t=1}^{m}\frac{1}{s}\sum_{i=m-t+1}^{m} s_i$$

FPA is an average of the proportions of actual defects in the top i (i: *1* to m) predicted modules. Larger FPA means better ranking performance, and can help allocate testing resources better on the whole.

Mean square error (MSE) is adopted as prediction accuracy in this paper. Considering m modules $f_1, f_2, \ldots, f_m$, s_i as the actual defect number in the module f_i, and p_i as the predicted

defect number in the module f_i, MSE is simply computed as follows:

$$\frac{1}{m}\sum_{i=1}^{m}(p_i - s_i)^2$$

Smaller MSE means better prediction accuracy.

In this paper, we adopt a linear model as a base predictor for multi-objective approaches. Given n training metric vectors of software modules $\mathbf{x_i}$= ($x_{i,1}$, $x_{i,2}$, $\cdots\cdots$, $x_{i,d}$) (i: 1 to n, $\mathbf{x_i}$ is the vector of the i^{th} software module, $x_{i,j}$ is the j^{th} metric value in vector $\mathbf{x_i}$, and d is number of metrics), which compose a metric matrix $\mathbf{X}$, and corresponding defect number y_i, the predicted defect number of testing module $f(\mathbf{z_i})$ is computed as follows:

$$f(\mathbf{z_i}) = \sum_{j=1}^{d}\alpha_j z_{i,j} + \alpha_0 \quad (1)$$

where α_js (including α_0) are the corresponding parameters obtained by training. Once α_js are fixed, the model is learned. If more α_js equal to zero, it costs less computation, and the model is simpler. Therefore, we use the number of nonzero α_js to measure the model complexity, which is denoted as NNZ. Smaller NNZ means a simpler model.

B. Multi-Objective Approaches

Multi-objective approaches are used to construct a set of Pareto-optimal models. In this study, we adopt a linear model as a base predictor, and Non-dominated Sorting Genetic Algorithm II (NSGA-II) [2] as the multi-objective learning algorithm. As mentioned in the above subsection, the predicted defect number of testing module $f(\mathbf{z_i})$ is computed as $f(\mathbf{z_i}) = \sum_{j=1}^{d}\alpha_j z_{i,j} + \alpha_0$. The goal of NSGA-II is to find out a set of α_js, which optimizes two or three objectives (FPA, MSE, or/and NNZ). The detailed process of NSGA-II is as follows (more details in [2]).

- Initialize: Set population size (number of solutions) to N, and randomly generate N solutions that compose population P_0. Sort the solution in P_0 using fast non-dominated sorting, and compute the non-dominated rank value of each solution.
  Set the generation number $t = 0$.
- While $t < t_{max}$ (maximal generation number, used to decide the termination condition), do
  - Use binary tournament selection to select individuals from P_t for crossover and mutation to generate the offspring population Q_t.
  - Combine solutions in P_t and Q_t to get $R_t = P_t U Q_t$.
  - Sort R_t based on non-domination rank value and crowding distance, and select N elitist individuals to compose the new parent population P_{t+1}.
- EndWhile and Return Pareto-optimal solutions in P_{t+1}.

Because it is difficult for NSGA-II to achieve zero α_js using crossover and mutation operation, we revise the process of NSGA-II by setting parameters to zero when they are smaller than a certain ratio of the maximum feasible value. The detailed process of the revised NSGA-II is as follows.

- Initialize: Set population size (number of solutions) to N, and randomly generate N solutions that compose population P_0. Sort the solution in P_0 using fast non-dominated sorting, and compute the non-dominated rank value of each solution.
  Set the generation number $t = 0$.
- While $t < t_{max}$ (maximal generation number, used to decide the termination condition), do
  - Use binary tournament selection to select individuals from P_t for crossover and mutation to generate the offspring population Q_t.
  - Combine solutions in P_t and Q_t to get $R_t = P_t U Q_t$.
  - Sort R_t based on non-domination rank value and crowding distance, and select N elitist individuals to compose the new parent population P_{t+1}.
- EndWhile and Return Pareto-optimal solutions in P_{t+1}.

C. Abbreviations and Acronyms

Define abbreviations and acronyms the first time they are used in the text, even after they have been defined in the abstract. Abbreviations such as IEEE, SI, MKS, CGS, ac, dc, and rms do not have to be defined. Do not use abbreviations in the title or heads unless they are unavoidable.

D. Units

- Use either SI (MKS) or CGS as primary units. (SI units are encouraged.) English units may be used as secondary units (in parentheses). An exception would be the use of English units as identifiers in trade, such as "3.5-inch disk drive".
- Avoid combining SI and CGS units, such as current in amperes and magnetic field in oersteds. This often leads to confusion because equations do not balance dimensionally. If you must use mixed units, clearly state the units for each quantity that you use in an equation.
- Do not mix complete spellings and abbreviations of units: "Wb/m^2" or "webers per square meter", not "webers/m^2". Spell out units when they appear in text: ". . . a few henries", not ". . . a few H".
- Use a zero before decimal points: "0.25", not ".25". Use "cm^3", not "cc".)

E. Equations

Number equations consecutively. To make your equations more compact, you may use the solidus (/), the exp function, or appropriate exponents. Italicize Roman symbols for quantities and variables, but not Greek symbols. Use a long dash rather than a hyphen for a minus sign. Punctuate equations with commas or periods when they are part of a sentence, as in:

$$a + b = \gamma \quad (2)$$

Be sure that the symbols in your equation have been defined before or immediately following the equation. Use "(2)", not "Eq. (2)" or "equation (2)", except at the beginning of a sentence: "Equation (2) is . . ."

F. LaTeX-Specific Advice

Please use "soft" (e.g., `\eqref{Eq}`) cross references instead of "hard" references (e.g., `(1)`). That will make it possible to combine sections, add equations, or change the order of figures or citations without having to go through the file line by line.

Please don't use the `{eqnarray}` equation environment. Use `{align}` or `{IEEEeqnarray}` instead. The `{eqnarray}` environment leaves unsightly spaces around relation symbols.

Please note that the `{subequations}` environment in LaTeX will increment the main equation counter even when there are no equation numbers displayed. If you forget that, you might write an article in which the equation numbers skip from (17) to (20), causing the copy editors to wonder if you've discovered a new method of counting.

BibTeX does not work by magic. It doesn't get the bibliographic data from thin air but from .bib files. If you use BibTeX to produce a bibliography you must send the .bib files.

LaTeX can't read your mind. If you assign the same label to a subsubsection and a table, you might find that Table I has been cross referenced as Table IV-B3.

LaTeX does not have precognitive abilities. If you put a `\label` command before the command that updates the counter it's supposed to be using, the label will pick up the last counter to be cross referenced instead. In particular, a `\label` command should not go before the caption of a figure or a table.

Do not use `\nonumber` inside the `{array}` environment. It will not stop equation numbers inside `{array}` (there won't be any anyway) and it might stop a wanted equation number in the surrounding equation.

G. Some Common Mistakes

- The word "data" is plural, not singular.
- The subscript for the permeability of vacuum μ_0, and other common scientific constants, is zero with subscript formatting, not a lowercase letter "o".
- In American English, commas, semicolons, periods, question and exclamation marks are located within quotation marks only when a complete thought or name is cited, such as a title or full quotation. When quotation marks are used, instead of a bold or italic typeface, to highlight a word or phrase, punctuation should appear outside of the quotation marks. A parenthetical phrase or statement at the end of a sentence is punctuated outside of the closing parenthesis (like this). (A parenthetical sentence is punctuated within the parentheses.)
- A graph within a graph is an "inset", not an "insert". The word alternatively is preferred to the word "alternately" (unless you really mean something that alternates).
- Do not use the word "essentially" to mean "approximately" or "effectively".
- In your paper title, if the words "that uses" can accurately replace the word "using", capitalize the "u"; if not, keep using lower-cased.
- Be aware of the different meanings of the homophones "affect" and "effect", "complement" and "compliment", "discreet" and "discrete", "principal" and "principle".
- Do not confuse "imply" and "infer".
- The prefix "non" is not a word; it should be joined to the word it modifies, usually without a hyphen.
- There is no period after the "et" in the Latin abbreviation "et al.".
- The abbreviation "i.e." means "that is", and the abbreviation "e.g." means "for example".

An excellent style manual for science writers is [7].

H. Authors and Affiliations

The class file is designed for, but not limited to, six authors. A minimum of one author is required for all conference articles. Author names should be listed starting from left to right and then moving down to the next line. This is the author sequence that will be used in future citations and by indexing services. Names should not be listed in columns nor group by affiliation. Please keep your affiliations as succinct as possible (for example, do not differentiate among departments of the same organization).

I. Identify the Headings

Headings, or heads, are organizational devices that guide the reader through your paper. There are two types: component heads and text heads.

Component heads identify the different components of your paper and are not topically subordinate to each other. Examples include Acknowledgments and References and, for these, the correct style to use is "Heading 5". Use "figure caption" for your Figure captions, and "table head" for your table title. Run-in heads, such as "Abstract", will require you to apply a style (in this case, italic) in addition to the style provided by the drop down menu to differentiate the head from the text.

Text heads organize the topics on a relational, hierarchical basis. For example, the paper title is the primary text head because all subsequent material relates and elaborates on this one topic. If there are two or more sub-topics, the next level head (uppercase Roman numerals) should be used and, conversely, if there are not at least two sub-topics, then no subheads should be introduced.

J. Figures and Tables

a) Positioning Figures and Tables: Place figures and tables at the top and bottom of columns. Avoid placing them in the middle of columns. Large figures and tables may span across both columns. Figure captions should be below the figures; table heads should appear above the tables. Insert figures and tables after they are cited in the text. Use the abbreviation "Fig. 1", even at the beginning of a sentence.

Figure Labels: Use 8 point Times New Roman for Figure labels. Use words rather than symbols or abbreviations when writing Figure axis labels to avoid confusing the reader. As an

TABLE I
TABLE TYPE STYLES

Table Head	Table Column Head		
	Table column subhead	*Subhead*	*Subhead*
copy	More table copy[a]		

[a]Sample of a Table footnote.

Fig. 1. Example of a figure caption.

example, write the quantity "Magnetization", or "Magnetization, M", not just "M". If including units in the label, present them within parentheses. Do not label axes only with units. In the example, write "Magnetization (A/m)" or "Magnetization {A[m(1)]}", not just "A/m". Do not label axes with a ratio of quantities and units. For example, write "Temperature (K)", not "Temperature/K".

ACKNOWLEDGMENT

The preferred spelling of the word "acknowledgment" in America is without an "e" after the "g". Avoid the stilted expression "one of us (R. B. G.) thanks ...". Instead, try "R. B. G. thanks...". Put sponsor acknowledgments in the unnumbered footnote on the first page.

REFERENCES

Please number citations consecutively within brackets [1]. The sentence punctuation follows the bracket [2]. Refer simply to the reference number, as in [3]—do not use "Ref. [3]" or "reference [3]" except at the beginning of a sentence: "Reference [3] was the first ..."

Number footnotes separately in superscripts. Place the actual footnote at the bottom of the column in which it was cited. Do not put footnotes in the abstract or reference list. Use letters for table footnotes.

Unless there are six authors or more give all authors' names; do not use "et al.". Papers that have not been published, even if they have been submitted for publication, should be cited as "unpublished" [4]. Papers that have been accepted for publication should be cited as "in press" [5]. Capitalize only the first word in a paper title, except for proper nouns and element symbols.

For papers published in translation journals, please give the English citation first, followed by the original foreign-language citation [6].

REFERENCES

[1] G. Eason, B. Noble, and I. N. Sneddon, "On certain integrals of Lipschitz-Hankel type involving products of Bessel functions," Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955.

[2] J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

[3] I. S. Jacobs and C. P. Bean, "Fine particles, thin films and exchange anisotropy," in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.

[4] K. Elissa, "Title of paper if known," unpublished.

[5] R. Nicole, "Title of paper with only first word capitalized," J. Name Stand. Abbrev., in press.

[6] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, "Electron spectroscopy studies on magneto-optical media and plastic substrate interface," IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].

[7] M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.

IEEE conference templates contain guidance text for composing and formatting conference papers. Please ensure that all template text is removed from your conference paper prior to submission to the conference. Failure to remove the template text from your paper may result in your paper not being published.

REFERENCES

[1] A. B. Carvalho and S. R. Vergilio. A symbolic fault-prediction model based on multiobjective particle swarm optimization. *Systems and Software*, pages 868–882, 2010.

[2] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast elitist multi-objective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6:182–197, 2000.

[3] K. Gao and T. Khoshgoftaar. A comprehensive empirical study of count models for software defect prediction. *IEEE Transactions on Reliability*, 56(2):223–236, 2007.

[4] M. Jureczko and L. Madeyski. Towards identifying software project clusters with regard to defect prediction. In *Proceedings of the 6th International Conference on Predictive Models in Software Engineering*, PROMISE '10, pages 9:1–9:10, 2010.

[5] T. M. Khoshgoftaar and Y. Liu. A multi-objective software quality classification model using genetic programming. *IEEE Transactions on Reliability*, 56(2):237–245, 2007.

[6] T. M. Khoshgoftaar, Y. Liu, and N. Seliya. A multiobjective module-order model for software quality enhancement. *IEEE Transactions on Evolutionary Computation*, 8(6):593–608, 2004.

[7] Y. Li, J. Su, and X. Yang. Multi-objective vs. single-objective approaches for software defect prediction. In *International Conference on Management Engineering, Software Engineering and Service Sciences, ICMSS 2018*, pages 122–127, 2018.

[8] T. Menzies, J. Greenwald, and A. Frank. Data mining static code attributes to learn defect predictors. *IEEE Transactions on Software Engineering*, 33(1):2–13, 2007.

[9] N. Ohlsson and H. Alberg. Predicting fault-prone software modules in telephone switches. *IEEE Transactions on Software Engineering*, 22(12):886–894, 1996.

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

[11] S. Shukla, T. Radhakrishnan, and K. Muthukumaran. Multi-objective cross-version defect prediction. *Soft Computing*, pages 1–22, 2016.

[12] E. Weyuker, T. Ostrand, and R. Bell. Comparing the effectiveness of several modeling methods for fault prediction. *Empirical Software Engineering*, 15(3):277–295, 2010.

[13] Z. Xu, J. Liu, X. Luo, and T. Zhang. Cross-version defect prediction via hybrid active learning with kernel principal component analysis. In *25th IEEE International Conference on Software Analysis, Evolution, and Reengineering*, pages 209–220, 2018.

[14] 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.

[15] X. Yang and W. Wen. Ridge and lasso regression models for cross-version defect prediction. *IEEE Transactions on Reliability*, 67(3):885–896, 2018.