Evaluating Defect Prediction Approaches Using A Massive Set of Metrics: An Empirical Study

Xiao Xuan[1], David Lo[2], Xin Xia[1*], and Yuan Tian[2]
[1]College of Computer Science and Technology, Zhejiang University, China
[2]School of Information Systems, Singapore Management University, Singapore
jackyxuan@zju.edu.cn, davidlo@smu.edu.sg, xxkidd@zju.edu.cn,
yuan.tian.2012@phdis.smu.edu.sg

ABSTRACT

To evaluate the performance of a within-project defect prediction approach, people normally use precision, recall, and F-measure scores. However, in machine learning literature, there are a large number of evaluation metrics to evaluate the performance of an algorithm, (e.g., Matthews Correlation Coefficient, G-means, etc.), and these metrics evaluate an approach from different aspects. In this paper, we investigate the performance of within-project defect prediction approaches on a large number of evaluation metrics. We choose 6 state-of-the-art approaches including naive Bayes, decision tree, logistic regression, kNN, random forest and Bayesian network which are widely used in defect prediction literature. And we evaluate these 6 approaches on 14 evaluation metrics (e.g., G-mean, F-measure, balance, MCC, J-coefficient, and AUC). Our goal is to explore a practical and sophisticated way for evaluating the prediction approaches comprehensively. We evaluate the performance of defect prediction approaches on 10 defect datasets from PROMISE repository. The results show that Bayesian network achieves a noteworthy performance. It achieves the best recall, FNR, G-mean1 and balance on 9 out of the 10 datasets, and F-measure and J-coefficient on 7 out of the 10 datasets.

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and Debugging

Keywords

Defect Prediction, Evaluation Metric, Machine Learning

1. INTRODUCTION

Due to the complexity of software systems, defects are inevitable. A previous study from NIST shows that software defects cost the US economy an estimated $59 billion every year [12]. Early detection or prediction of the defective classes/files/modules in a software system could not only enhance software dependability, but also reduce the cost of software development and maintenance. A number of defect prediction approaches have been proposed to identify defect-prone classes, files, or modules by leveraging machine learning techniques to build a prediction model from historical data [3, 6, 9]. Most of these approaches are trained and applied on classes/files/modules from the same project which are referred to as *within-project* defect prediction approaches.

To evaluate the performance of a within-project defect prediction approach, people normally use precision, recall, and F-measure scores [3, 6, 9]. However, in machine learning literature, there are massive evaluation metrics to evaluate the performance of an algorithm, e.g., accuracy [3, 6], specificity [3, 6], G-mean [3], and Area Under the ROC Curve (AUC) [3, 6]. These metrics evaluate an approach from different aspects.

In this paper, we investigate the performance of within-project defect prediction approaches on massive evaluation metrics. We choose 6 state-of-the-art approaches including naive Bayes, decision tree, logistic regression, kNN, random forest and Bayesian network which are widely used in defect prediction literature [3, 16, 11]. Then, we evaluate these 6 approaches on 14 evaluation metrics including accuracy, error rate, recall, specificity, precision, false positive rate (FPR), false negative rate (FNR), G-mean1, G-mean2, F-measure, balance, MCC, J-coefficient, and AUC. Our goal is to evaluate the prediction approaches comprehensively. To evaluate the performance of these 6 approaches on 14 metrics, we perform experiments on 10 defect datasets from PROMISE repository [10] containing a total of 5,305 instances. The results show that Bayesian network achieves a noteworthy performance. It achieves the best recall, FNR G-mean1 and balance on 9 out of the 10 datasets, and F-measure and J-coefficient on 7 out of the 10 datasets.

The main contributions of this work are listed as follows:

1. We empirically investigate performance of 6 most widely used classification algorithms for defect prediction with a large set of evaluation metrics. We perform experiments on a broad range of datasets containing a total of 5,305 instances.
2. Based on the experiment results, we present a detailed analysis on the performance of these defect prediction approaches with respect to the different metrics.

*Corresponding author.

SAC'15, April 13 - 17 2015, Salamanca, Spain.
Copyright 2015 ACM 978-1-4503-3196-8/15/04.
http://dx.doi.org/10.1145/2695664.2695959 ...$15.00.

Table 1: Confusion matrix

	Predicted Positive	Predicted Negative
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

2. WITHIN-PROJECT PREDICTION APPROACHES

Within-project defect prediction aims to predict if a particular class/file/module (aka. instance) is defective or not. Typically, a within-project defect prediction approach has the following steps:

Training Data Extraction. For each class/file/module, label it as defective or not by mining a project's revision history and bug tracking system. Defective class/file/module means there are some bugs in it.

Feature Extraction. Identify various features from the class/file/module. A great number of features have been used in many previous defect prediction work. In this paper, we use the datasets provided by Jureczko and Madeyski [5].

Model Building Phase. Build a model with a classification algorithm based on the extracted features.

Model Application Phase. For a new class/file/module, extract the values of its features, then input them to the learnt model to predict whether it is defective or not.

In this work, we evaluate 6 state-of-the-art within-project defect prediction approaches including naive Bayes (NB) [2], decision tree (DT) [2], logistic regression (LR) [4], k-nearest neighbor (kNN) [2], random forest (RF) [2] and Bayesian network (BN) [2].

3. MASSIVE EVALUATION METRICS

In this section, we elaborate the 14 evaluation metrics which would be used to evaluate the performance of defect prediction approaches.

The Confusion Matrix: Most of the performance evaluation metrics are based on the result of a Confusion Matrix [13, 14]. This matrix shows the numbers of correctly classified (i.e. the predicted outcome is the same as the observed one) and misclassified instances (i.e. the predicted outcome is contrary to the observation). As presented in Table 1, four categories of labelled instances will be concluded after a prediction has been performed:

- True Positive (TP): an instance is classified as defective when it truly is defective.
- False Positive (FP): an instance is classified as defective when it is not defective.
- True Negative (TN): an instance is classified as not defective when it is actually not defective.
- False Negative (FN): an instance is classified as not defective when it is actually defective.

Accuracy and Error Rate: Accuracy is the ratio of the number of correctly classified instances (TP and TN) to total number of instances (see Equation 1). Error rate is the number of correctly classified instances over the total number of instances, as shown in Equation 2.

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \quad (1)$$

$$ErrorRate = \frac{FP+FN}{TP+TN+FP+FN} \quad (2)$$

Both of these two metrics measure the overall performance of a classifier. However, neither of them inspect the data distribution and cost information.

Recall, Specificity & Precision: Recall, aka. sensitivity, true positive rate or probability of detection, is defined as the proportion of observed positive instances predicted as positive by a classifier (see Equation 3). Another metric, named specificity, is just complementary to recall, which is defined as the proportion of observed negative instances predicted as negative by a classifier (see Equation 4).

$$Recall = Sensitivity = \frac{TP}{TP+FN} \quad (3)$$

$$Specificity = \frac{TN}{FP+TN} \quad (4)$$

Precision is a metric that is easy to be confused with recall. It refers to the proportion of instances predicted as defective which actually are defective, as shown in Equation 5.

$$Precision = \frac{TP}{TP+FP} \quad (5)$$

Note that both recall and precision are crucial metrics for defect prediction. Jiang et al [3] elaborate that there is a trade-off existing between these two metrics. Zhang and Zhang [15] state that a good prediction model should achieve high recall and high Precision. That means single-focus metrics like recall, specificity and precision cannot provide comprehensive evaluation for software defect prediction approaches. To remedy this problem, some combination metrics were proposed, which are introduced later.

False Positive Rate (FPR) & False Negative Rate (FNR): False positive rate and false negative rate are additional single-focus metrics (see Equation 6 and 7), which are also known as Type I error rate and Type II error rate respectively. False positive rate is used for calculating balance and J-coefficient that are described later.

$$FPR = \frac{FP}{TN+FP} \quad (6)$$

$$FNR = \frac{FN}{FN+TP} \quad (7)$$

Geometric Mean (G-mean): Geometric mean was proposed by Kubat et al [7], which is motivated by two factors - the requirement for a metric that can tackle the class imbalance problem, and the requirement for a metric that is not only easy to compute but also more comprehensive than single-focus metrics. In software defect prediction, datasets usually are imbalanced. Defective instances are the minority, while faultless ones are the majority. As shown in Equation 8, G-mean is the square root (i.e. geometric mean) of the product of recall and specificity: one for the minority class and another for the majority class. Equation 9 denotes another definition of G-mean: square root of the product of recall and precision.

$$G-mean1 = \sqrt{Recall \times Specificity} \quad (8)$$

$$G-mean2 = \sqrt{Recall \times Precision} \quad (9)$$

F-Measure: F-measure is another combination metric, and it also looks at two positive metrics - recall and precision, the same as G-mean. Compared to G-mean, F-measure is more flexible and sophisticated by introducing a weight coefficient ω. The user is able to weigh the contribution of these two component metrics arbitrarily. To be specific, F-measure is defined as a weighted harmonic mean of precision and recall, see Equation 10. For any $\omega \in R$, we have:

$$F_{\omega} = \frac{(1+\omega)\,[Precision \times Recall]}{[\omega \times Precision] + Recall} \quad (10)$$

where, typically, ω is assigned the values 0.5, 1, or 2. In this work, we set $\omega = 1$ by default, considering that precision and recall are equal.

Balance: The metric balance is a combination of FPR and recall. According to the definition (see Equation 11), a high recall leads to a high balance; whereas a high FPR causes a low balance.

$$Balance = 1 - \sqrt{\frac{(0 - FPR)^2 + (1 - Recall)^2}{2}} \quad (11)$$

Matthews Correlation Coefficient (MCC): Matthews Correlation Coefficient is rarely used in the scenario of software defect prediction but widely used in medical science domain. MCC is a Chi-Square based metric which is defined in Equation 12.

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)\,(TP+FN)\,(TN+FP)\,(TN+FN)}} \quad (12)$$

J-coefficient (J-coeff): J-coefficient is another metric which is initially proposed for models evaluation in medical research. It was first used to assess the performance of software defect prediction approaches by El Emam et al in 2001 [1]. Equation 13 is J-coefficient's formal expression.

$$JCoeff = Recall + Specificity - 1 = Recall - FPR \quad (13)$$

Area Under the ROC Curve (AUC): Receiver Operating Characteristic (ROC) curve is a graphical evaluation method for classification approaches. This curve is plotted in a two-dimensional space with FPR as x-coordinate and recall as y-coordinate. Area Under the ROC Curve (AUC) is defined as what its name implies, which can also be computed in one simple way (see Equation 14).

$$AUC\,(f) = \frac{\sum_{i=1}^{|T_p|} (R_i - i)}{|T_p|\,|T_n|} \quad (14)$$

where $T_p \subset T$ and $T_n \subset T$ are the subsets of positive and negative instances in test set T, and R_i is the rank of the ith instance in T_p given by the classifier f.

4. EXPERIMENTS

Experiment Setup. The experimental environment of this work is a computer equipped with 2.4 GHz Intel Core 2 Duo CPU and 4GB 1067 MHz DDR3 RAM, running Mac OS X 10.8.2 (64-bit). We use Weka 3.6 to evaluate 6 classical software defect prediction approaches which consist of naive Bayes, decision tree, logistic regression, K-nearest neighbor, random forest and Bayesian network.

Table 2 shows the statistics of the 10 datasets that we collected from PROMISE repository [10]. The columns correspond to the dataset name and version (Dataset), the number of total instances (# Inst.), the number of defective instances (# Defect.), and the percentage of defective instances (% Defect.).

Table 2: Statistics of Collected Datasets

Dataset	*# Inst.*	*# Defect.*	*% Defect.*
ANT-1.7	745	166	22.3%
PROP-6	660	66	10.0%
IVY-2.0	352	40	11.4%
JEDIT-4.0	306	75	24.5%
LOG4J-1.0	135	34	25.2%
LUCENE-2.2	247	144	58.3%
POI-2.0	314	37	11.8%
CAMEL-1.6	965	188	19.5%
TOMCAT	858	77	9.0%
XALAN-2.4	723	110	15.2%
Total	5,305	937	17.7%

Results. We are interested in the following research question:

Which defect prediction approach achieves the best performance across the 14 evaluation metrics?

Motivation. In this paper, we propose 14 evaluation metrics, we would like to investigate whether exists a defect prediction approach which achieves the best performance across the massive set of evaluation metrics.

Method. We perform 10×10-fold cross-validation experiments on these 10 datasets. We repeat 10-fold cross-validation 10 times, and record the average scores across the 10 times. The parameters are set by default in Weka, except the k value for kNN classifier, which is configured as 5.

Results. Table 3 shows the overall performance of each defect prediction approach on the datasets. For each metric in this table, we show the average score of the metric over the 10 datasets. As shown in this table, in general, Bayesian network performs very well. It achieves the best scores in terms of recall (0.529), G-mean1 (0.653), G-mean2 (0.457), F-Measure (0.445), balance (0.640), J-coeff (0.351) and MC-C (0.305). Logistic regression performs the best in terms of accuracy (it scored 0.820). Random forest achieves the best AUC (it scored 0.750). kNN performs the best in terms of specificity (it scored 0.912).

Figure 1 presents the number of times each approach achieve the best performance on 14 evaluation metrics. We also notice that Bayesian network achieves the best performance; it achieves the best recall, FNR, G-mean1, and balance for 9 out of the 10 datasets, the best F-measure and J-coefficient for 7 out of the 10 datasets, the best G-mean2 for 6 out of the 10 datasets, and the best AUC for 4 out of the 10 datasets. Thus, for a new defect prediction approach, we need to compare it with Bayesian network since it achieves the best performance across a massive set of evaluation metrics.

Also, from the Table 3 and Figure 1, we need to use a range of metrics and not just one metric to evaluate a defect prediction approach. Using just one metric will not do well, since an approach that is good on one metric might be very poor on many others. For example, logistic regression achieves the best performance in terms of accuracy, but its recall, F-measure, G-mean1, G-mean2, and MCC scores are low.

5. RELATED WORK

There have been a number of empirical studies to evaluate the performance of defect prediction approaches. Jiang et al. perform an empirical study which investigates the performance of different defect prediction approaches on 8 datasets from NASA MDP repository [3]. They study the performance of defect prediction approaches by using cost curves. Li et al. also investigate the performance of different defect

Table 3: Classifiers Performance on Overall Datasets

	NB	*DT*	*LR*	*kNN*	*RF*	*BN*
Accuracy	0.790	0.807	**0.820**	0.816	0.818	0.762
Error Rate	0.210	0.193	**0.180**	0.184	0.182	0.238
Recall	0.383	0.333	0.289	0.269	0.307	**0.529**
Specificity	0.892	0.881	0.902	**0.912**	0.907	0.822
Precision	0.450	0.473	0.501	0.496	**0.513**	0.411
FPR	0.108	0.119	0.098	**0.088**	0.093	0.178
FNR	0.617	0.667	0.711	0.731	0.693	**0.471**
G-mean1	0.577	0.512	0.464	0.462	0.503	**0.653**
G-mean2	0.407	0.390	0.370	0.358	0.390	**0.457**
F-Measure	0.399	0.378	0.348	0.337	0.373	**0.445**
Balance	0.556	0.503	0.471	0.467	0.493	**0.640**
J-coeff	0.275	0.214	0.191	0.181	0.214	**0.351**
MCC	0.281	0.253	0.249	0.239	0.270	**0.305**
AUC	0.734	0.626	0.726	0.722	**0.750**	0.737

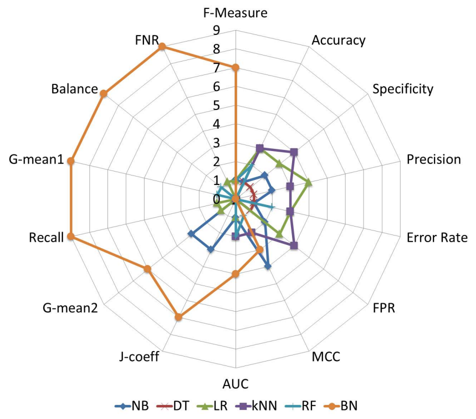

Figure 1: The number of times that each approach achieve the best performance on 14 evaluation metrics.

prediction approaches on 12 NASA MDP datasets [9]. Two evaluation metrics, namely recall and AUC, are used. Lessmann et al. evaluate the performance of 22 defect prediction approaches using AUC as the evaluation metric [8]. Our work complements the above studies. We investigate the performance of defect prediction approaches using a large number of evaluation metrics.

6. CONCLUSION AND FUTURE WORK

In this paper, we present an empirical study on evaluation of software defect prediction approaches with massive numeric metrics. Various classification techniques (e.g., naive Bayes, decision tree, logistic regression, K-nearest neighbor, random forest and Bayesian network) have been investigated. Besides, 14 performance metrics have been used to assess the performance of the 6 classifiers. In our experiment, we use 10 PROMISE datasets that contains 5,305 instances in total. From the experiment results, the performance of Bayesian network classifier is remarkable. It achieves the best performance in terms of recall, FNR, G-mean1 and balance on 9 out of the 10 datasets, and in terms of F-measure and J-coefficient on 7 out of 10 datasets. Furthermore, for 6 datasets, Bayesian network achieves the best value for more than 5 metrics.

In the future, we plan to evaluate the performance of defect prediction approaches with more evaluation metrics, and more datasets. We also plan to design a better approach which would achieve the best performance across all of the evaluation metrics.

Acknowledgement. This research was supported by the National Basic Research Program of China (the 973 Program) under grant 2015CB352201, National Key Technology R&D Program of the Ministry of Science and Technology of China under grant 2014BAH24F02, and the Fundamental Research Funds for the Central Universities.

7. REFERENCES

[1] K. El Emam, S. Benlarbi, N. Goel, and S. N. Rai. Comparing case-based reasoning classifiers for predicting high risk software components. *JSS*, pages 301–320, 2001.

[2] J. Han and M. Kamber. *Data Mining, Southeast Asia Edition: Concepts and Techniques.* Morgan kaufmann, 2006.

[3] Y. Jiang, B. Cukic, and Y. Ma. Techniques for evaluating fault prediction models. *EMSE*, pages 561–595, 2008.

[4] A. Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. 2002.

[5] M. Jureczko and L. Madeyski. Towards identifying software project clusters with regard to defect prediction. In *PROMISE*, page 9, 2010.

[6] A. Kaur and I. Kaur. Empirical evaluation of machine learning algorithms for fault prediction. *Lecture Notes on Software Engineering*, 2(2), 2014.

[7] M. Kubat, R. C. Holte, and S. Matwin. Machine learning for the detection of oil spills in satellite radar images. *Machine learning*, 30(2-3):195–215, 1998.

[8] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch. Benchmarking classification models for software defect prediction: A proposed framework and novel findings. *TSE*, pages 485–496, 2008.

[9] R. Li and S. Wang. An empirical study for software fault-proneness prediction with ensemble learning models on imbalanced data sets. *Journal of Software*, pages 697–704, 2014.

[10] T. Menzies, B. Caglayan, Z. He, E. Kocaguneli, J. Krall, F. Peters, and B. Turhan. The promise repository of empirical software engineering data, June 2012.

[11] J. Nam, S. J. Pan, and S. Kim. Transfer defect learning. In *ICSE*, pages 382–391, 2013.

[12] M. Newman. Software errors cost us economy 59.5 billion annually. *NIST Assesses Technical Needs of Industry to Improve Software-Testing*, 2002.

[13] T. J. Ostrand and E. J. Weyuker. How to measure success of fault prediction models. In *Fourth international workshop on Software quality assurance*, pages 25–30, 2007.

[14] N. J. Pizzi, R. Summers, and W. Pedrycz. Software quality prediction using median-adjusted class labels. In *Proceedings: International Joint Conference on Neural Networks*, volume 3, pages 2405–2409, 2002.

[15] H. Zhang and X. Zhang. Comments on data mining static code attributes to learn defect predictors. *TSE*, pages 635–636, 2007.

[16] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, and B. Murphy. Cross-project defect prediction: a large scale experiment on data vs. domain vs. process. In *ESEC/FSE*, pages 91–100, 2009.