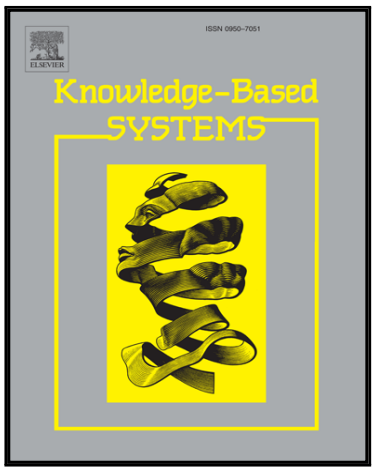

A Novel Bayes Defect Predictor Based on Information Diffusion Function

Yaning Wu , Song Huang , Haijin Ji , Changyou Zheng , Chengzu Bai

PII: S0950-7051(17)30587-7
DOI: 10.1016/j.knosys.2017.12.015
Reference: KNOSYS 4151

To appear in: *Knowledge-Based Systems*

Received date: 7 January 2017
Revised date: 9 December 2017
Accepted date: 13 December 2017

Please cite this article as: Yaning Wu , Song Huang , Haijin Ji , Changyou Zheng , Chengzu Bai , A Novel Bayes Defect Predictor Based on Information Diffusion Function, *Knowledge-Based Systems* (2017), doi: 10.1016/j.knosys.2017.12.015

Highlights

- A new Bayes software defect predictor is proposed.
- The predictor is to improve Naïve Bayes classifier with VSDF.
- The predictor is to solve problems of insufficient and non-normal distributed data.

A Novel Bayes Defect Predictor Based on Information Diffusion Function

Yaning Wu[1], Song Huang[1], Haijin Ji[1], Changyou Zheng[1], Chengzu Bai[2]

[1] Research Center of Software Engineering, Institute of Information System, PLA University of Science and Technology, Nanjing 211101, China

[2] Research Center of Ocean Environment Numerical Simulation, Institute of Meteorology & Oceanography, PLA University of Science and Technology, Nanjing 211101, China

Abstract

Software defect prediction plays a significant part in identifying the most defect-prone modules before software testing. Quite a number of researchers have made great efforts to improve prediction accuracy. However, the problem of insufficient historical data available for within- or cross- project still remains unresolved. Further, it is common practice to use the probability density function for a normal distribution in Naïve Bayes (NB) classifier. Nevertheless, after performing a Kolmogorov-Smirnov test, we find that the 21 main software metrics are not normally distributed at the 5% significance level. Therefore, this paper proposes a new Bayes classifier, which evolves NB classifier with non-normal information diffusion function, to help solve the problem of lacking appropriate training data for new projects. We conduct three experiments on 34 data sets obtained from 10 open source projects, using only 10%, 6.67%, 5%, 3.33% and 2% of the total data for training, respectively. Four well-known classification algorithms are also included for comparison, namely Logistic Regression, Naïve Bayes, Random Tree and Support Vector Machine. All experimental results demonstrate the efficiency and practicability of the new classifier.

Keywords: Cross-project defect prediction; Naïve Bayes; information diffusion function; software metrics

1. Introduction

In the area of software engineering, software defect prediction usually focuses on estimating precisely the faulty software modules, and helps software practitioners allocate limited testing resources to those parts which are most likely to contain defects. During the phase of predicting defects, the most important step is building the defect predictors (also known as defect prediction models), which has motivated plenty of researchers to design and define different types of prediction models [1-10]. In general, these studies could be divided into two groups, i.e. Within-Project Defect Prediction (WPDP) and Cross-Project Defect Prediction (CPDP). WPDP means the predictors are trained from data of historical releases in the same project and used to predict defects in the new releases [11]. Zimmermann et al. [12] illustrated that once there are sufficient data available to train any models, the defect predictors would perform well within projects. However, it is not always able to collect such kind of historical data for new projects although there are many public defect data sets on-line, such as PROMISE[1], Apache[2] and Eclipse[3]. Thus, in some cases, defect prediction based on within-project data is not very easy to implement.

An attemptable method to predict defects in projects which are lack of historical data is to utilize public data sets as training data. This way is commonly called CPDP. In the other word, CPDP refers to predicting defects in a project using prediction models trained from the historical data from different projects [5]. Many researchers have tried very hard to demonstrate the practicability of CPDP. Kamei et al. [10] empirically evaluated the performance of Just-In-Time (JIT) defect prediction models in a cross-project context, in order to address the limitation of few training data. They found that JIT models learned using other projects could effectively deal with projects with limited historical data, especially when the data were carefully selected. Zhang et al. [13] brought a new insight to tackle this limitation

1 http://promisedata.org.
2 http://www.apache.org/.
3 http://eclipse.org.

using connectivity-based unsupervised classifiers, which assume that defective entities tend to cluster around the same area. Based on the assumption, they proposed a spectral classifier and conducted a series of experiments in both cross-project and with-project contexts. The results showed that the proposed spectral classifier could achieve impressive performance. However, the studies were based on the hypothesis that all the software metrics were normalized. After we conducted a normal distribution hypothetical test on 34 data sets (the referenced data sets are illustrated in Appendix B), we found that the 21 most commonly used attributes follow non-normal distribution. Thus, despite all these empirical studies, defect prediction in CPDP still faces a lot of challenges [12].

Since the proposed predictors are mostly built on machine learning algorithms, a predictor usually depends on two factors: the training data and the learning algorithm. The training data from other projects seem to be abundant for predicting the defects of the local project. However, the heterogeneous distribution of software metric values between the training projects and the target project often makes the case of inadequate data available quite likely to occur [14, 15]. Moreover, the data could be too sparse to meet the requirements for sample size using traditional methods to implement defect prediction. In terms of learning algorithms (classifiers), there are many classifiers have been employed in CPDP, such as Logistic Regression (LR), Naïve Bayes (NB), Random Tree (RT) and Support Vector Machine (SVM) [7-9, 16]. Compared with LR, RT, and SVM (more detailed information is shown in Appendix A), Menzies et al. [17] illustrated that Naïve Bayes may be more suitable than the other classifiers, especially in dealing with incomplete data [18]. However, it is still common practice to use the probability density function for a normal distribution in NB, which is not consistent with the hypothetical test results. The details of normal distribution hypothetical test are illustrated as follows.

Firstly, we draw a random sample of size n from the combination of all data sets and performed a Kolmogorov-Smirnov test [19-21] of the default null hypothesis that each attribute comes from a

distribution in the normal family based on a “Lillietest” function in Matlab. The test returns the logical value h=1 if it rejects the null hypothesis at the 0.05 significance level, and h=0 if it cannot. For each *n*, we repeat this procedure 10000 times. The results are given in Table 1. From Table 1, it can be easily obtained that it is not appropriate to estimate the possibility density at each of the values in numeric attributes using the normal distribution for NB (the 21 attributes shown in Table 1, including 20 feature attributes and 1 labeled attribute, are explained in detail in Appendix B).

Table 1 Mean Values for the Kolmogorov-Smirnov Test

Attribute	n=30%		n=60%		n=95%	
	h=0	h=1	h=0	h=1	h=0	h=1
WMC	7	9993	1	9999	2	9998
DIT	8	9992	6	9994	4	9996
LCOM	13	9987	8	9992	4	9996
RFC	5	9995	6	9994	1	9999
CBO	13	9987	5	9995	3	9997
NOC	8	9992	7	9993	3	9997
CA	1	9999	7	9993	3	9997
CE	3	9997	7	9993	4	9996
DAM	12	9988	7	9993	1	9999
NPM	8	9992	10	9990	2	9998
MFA	3	9997	2	9998	1	9999
CAM	6	9994	8	9992	5	9995
MOA	10	9990	3	9997	3	9997
IC	3	9997	1	9999	2	9998
CBM	13	9987	7	9993	3	9997
AMC	4	9996	5	9995	3	9997
LCOM3	16	9984	5	9995	3	9997
MAX_CC	5	9995	7	9993	3	9997
AVG_CC	13	9987	8	9992	4	9996
LOC	3	9997	4	9996	3	9997
Bug	5	9995	7	9993	5	9995

Aiming at solving the shortcomings of the existing classifiers mentioned above, the diffusion function based on the vibration of string (VSDF) is introduced to improve the Naïve Bayes classifier in this paper. The VSDF method is usually used for hazard forecasting in meteorology. And this is the first attempt to apply this interdisciplinary approach to software defect prediction. VSDF is an effective method of dealing with the non-normal issue in the condition of incomplete data [22]. Based on the VSDF, an incomplete dataset can be regarded as a piece of fuzzy information; through some diffusion methods, some additional information can be extracted by spreading the observations. Finally, we obtained a novel classifier and named it as the Diffused Bayes (DB) classifier.

The rest of this paper is structured as follows: Sect. 2 gives a brief summary of some related work and describes the approach of our new model. Sect. 3 and Sect. 4 illustrate the detailed experimental setups and results, respectively. Finally, we conclude this study in Sect. 5.

2. Methodology

2.1 Related Work

The main purpose of our research is to evolve Naïve Bayes classifier using a diffusion function based on the vibration of string. For readers' easy understanding, we first review these related work in this section.

2.1.1 Naïve Bayes Classifier

The classifier is based on the Bayes' theorem. Informally, the theorem means that next depends on how new evidence affects old beliefs. The object unit of Naïve Bayes classifier to be trained and predicted is the software module, which is defined by IEEE as: (1) A program unit that is discrete and identifiable with respect to compiling, combining with other units; (2) A logically separable part of a program [23].

More formally, let $E = \{E_1, E_2, \cdots, E_m\}$ be a set of software metrics, $M = \{(E_1, y_1), (E_2, y_2), \cdots, (E_m, y_m)\}$ be a software module, and $X = \{(E_1, X_1), (E_2, X_2), \cdots, (E_m, X_m)\}$ be given fragments of evidence. Here, *m* represents the number of software metrics; y_k $(k = 1,2, \dots, m)$ is the associated value of E_k; $X_k = \{x_1, x_2, \dots, x_n\}$ is the value set of training data for E_k, where *n* refers to the number of software modules in the training dataset.

Then we define the category notion of software module is $H \in \{H_D, H_N\}$, where H_D is defective category and H_N is non-defective one. According to Bayesian theory [24], the probability that software module is defective or non-defective will be computed by Eq. (1):

$$P(H|M) = \frac{P(H)}{P(M)} \prod_i P[(E_i, x_i)|H] \quad (1)$$

We use evidence sets to train the classifier, and compute the defective probability of a new module,

which will make an alert when it exceeds a threshold. As shown by Eq. (2), classifier will classify software module M to H_D when the ratio greater than λ,

$$\log\{\frac{P(H_D|E)}{P(H_N|E)}\} > \lambda \quad (2)$$

$P(H)$ in Eq. (1) will be computed by Eq. (3):

$$P(H) = \frac{N_H}{N} \quad (3)$$

where N_H is the number of training modules in category H (i.e. H_D *or* H_N) and N is the total number of training modules. The estimation of likelihood function $P[(E_k, X_k)|H]$ is a key problem of NB classifier. In NB classifier, the metrics are usually assumed to be as normally distributed. Then

$$P[(E_k, X_k)|H] = \frac{1}{\sqrt{2\pi}\sigma_i} e^{\frac{-(y_k-\mu_i)^2}{2\sigma_i^2}} \quad (4)$$

where the mean μ_i and the standard deviation σ_i are estimated from the training data sets X_i.

The above has briefly described the principle of the Naïve Bayes classifier. However, in practice, the software metrics are not always in normal distribution, which has been experimentally confirmed above.

2.1.2 Vibration of String Based Diffusion Function

Information diffusion refers to making an affirmation: when a knowledge sample is given, it can be used to compute a relationship. Let $T = \{t_1, t_2, \cdots, t_w\}$ be a given sample and let the universe of discourse be U. If and only if T is incomplete, there must be a reasonable information diffusion function $\mu(t_i, u), u \epsilon U$, which can accurately estimate the real relation $\mathbb{R}^1$. This is called the principle of information diffusion [25]. Let t_i $(i = 1,2, \cdots, w)$ be w independent identically distributed attributes drawn from a population with density $p(t), x \epsilon \mathbb{R}^1$. Suppose $\mu(t)$ is a Borel measurable function in $(-\infty, +\infty)$.

$$\hat{f}(t) = \frac{1}{wd}\sum_{i=1}^{w} \mu[\frac{t-t_i}{d}] \quad (5)$$

is called an information diffusion estimator about $p(t)$, where $\mu(t)$ is the diffusion function and d is the diffusion coefficient [25]. Motivated by acoustic wave propagation, i.e. the equation of the vibrating string, and nearby criteria [25], Bai et al. [22] obtained the diffusion function as

$$u(t) = \frac{1}{2} e^{-\frac{|t|}{d}} \tag{6}$$

and

$$d = \frac{k}{w-1} [\max(t) - \min(t)] \tag{7}$$

where k is listed in the Table 2.

Table 2 The Values of k

w	2	3	4	5	6	7
k	1.75260	1.55885	1.49338	1.46630	1.45405	1.44826
w	8	9	10	11	12	>12
k	1.44544	1.44406	1.44338	1.44303	1.44270	1.44270

With sparse training data, the VSDF is an appropriate tool for analyzing the data that follow either non-normal or normal distribution. Another advantage is that VSDF can obtain an acceptable output without prior knowledge. In next section, we propose a novel Bayes classifier in coordination with VSDF.

2.2 Diffused Bayes Classifier

As discussed in Sect. 1, the main software metrics do not always follow normal distribution. Thus, we suggest a new Bayes Classifier, called as the Diffused Bayes (DB) Classifier, to improve the universality and efficiency of NB using VSDF in this section.

Let $E = \{E_1, E_2, \cdots, E_m\}$ be a set of software metrics, $M = \{(E_1, y_1), (E_2, y_2), \cdots, (E_m, y_m)\}$ be a software module, and $X = \{(E_1, X_1), (E_2, X_2), \cdots, (E_m, X_m)\}$ be given fragments of evidence. Here, m represents the number of software metrics; y_k $(k = 1,2, \dots, m)$ is the associated value of E_k; $X_k = \{x_1, x_2, \dots, x_n\}$ is the value set of training data for E_k, where *n* refers to the number of software modules in the training dataset. According to Bayesian theory [24], the probability that software module is defective or non-defective will be computed based on Eq. (1).

And classifier will classify software module *M* to H_D when the ratio greater than λ,

$$\log\{\frac{P(H_{\mathrm{D}}|E)}{P(H_{\mathrm{N}}|E)}\} > \lambda \tag{8}$$

$\lambda\epsilon[0.8, 1.2]$ is found to be experimentally fine. $P(H)$ can be obtained according to Eq. (3).

The main difference between the NB and DB is the likelihood function $P[(E_k, X_k)|H]$. We introduce the VSDF in the computation of $P[(E_k, X_k)|H]$. Let $T = \{t_1, t_2, \cdots, t_w\}$ equals to $X_k = \{x_1, x_2, \ldots, x_n\}$, and $w = n$; then $\hat{f}(t) = \hat{f}(x) = \frac{1}{wd}\sum_{i=1}^{w} \mu\left[\frac{x-x_i}{d}\right] = \frac{1}{nd}\sum_{i=1}^{n} \mu\left[\frac{x-x_i}{d}\right] = \frac{1}{2nd}\sum_{i=1}^{n} e^{\frac{-|x-x_i|}{d^2}}$. Let $P[(E_k, X_k)|H] = \widehat{f_k}(y_k)$, then we can get the likelihood function $P[(E_k, X_k)|H]$ as

$$P[(E_k, X_k)|H] = \frac{1}{2nd_k}\sum_{i=1}^{n} e^{\frac{-|y_k-x_i|}{2{d_k}^2}} \tag{9}$$

where d_k is the diffusion coefficient of the metric E_k. To facilitate readers' understanding, we decompose the DB classifier into the Algorithm 1.

Algorithm 1:

Input: A software module M to be tested; A set of given fragments of evidence X.

Step 1: Compute the diffusion coefficients d_k with respect to each software metric E_k based on Eq. (7).

Step 2: Calculate the prior probability $P(H)$ using Eq. (3).

Step 3: Estimate the likelihood function $P[(E_k, X_k)|H]$ by Eq. (9).

Step 4: Compute the ratio

$$r = \log\{\frac{P(H_{\mathrm{D}}|E)}{P(H_N|E)}\} \tag{10}$$

If $r > \lambda$, the module M is classified to H_D; otherwise, M is classified to H_N.

Output: The judgment of whether the module M is faulty or non-faulty.

3. Experimental Setup

In order to conduct an empirical study on software defect prediction, there are many primary work to be done, such as the collection of representative data sets, the selection of feature attributes, the evaluation methods and the evaluation criteria.

3.1 Data Collection

The training data sets in our experiments are the most suitable training data for each testing data set, whose detailed information can be seen in Table 3 in reference [11] and Table 1 in Appendix B. For example, for the given data set of release Ant-1.7, the combinations of data sets outside the Ant project are used as training data (e.g., <Camel-1.0>, <Camel-1.0, Ivy-1.4>, ..., <Ivy-1.4, Poi-1.5, Xalan-2.5>, …). If the use of <Camel-1.0, Lucene-2.0, Synapse-1.2> as the training data produces the best prediction result for release Ant-1.7, then <Camel-1.0, Lucene-2.0, Synapse-1.2> is the most suitable training data for Ant-1.7 in the cross-project context. Table 3 gives out the most suitable training data for each release.

Table 3 The most suitable training data for each release

No.	Release	Most suitable training data	No.	Release	Most suitable training data
1	Ant-1.3	<13, 14, 22>	18	Poi-1.5	<13, 30, 34>
2	Ant-1.4	<19, 26, 27>	19	Poi-2.0	<13, 26, 30>
3	Ant-1.5	<15, 28, 29>	20	Poi-2.5	<11, 29, 34>
4	Ant-1.6	<9, 13, 29>	21	Poi-3.0	<10, 17, 29>
5	Ant-1.7	<6, 15, 24>	22	Synapse-1.0	<4, 5, 19>
6	Camel-1.0	<12, 16, 18>	23	Synapse-1.1	<1, 13, 17>
7	Camel-1.2	<16, 21, 34>	24	Synapse-1.2	<10, 13, 21>
8	Camel-1.4	<16, 22>	25	Synapse-1.4	<2, 16, 34>
9	Camel-1.6	<3, 10, 17>	26	Synapse-1.5	<31, 32, 34>
10	Ivy-1.1	<2, 16, 22>	27	Synapse-1.6	<10, 15, 20>
11	Ivy-1.4	<2, 6, 31>	28	Xalan-2.4	<4, 7, 8>
12	Ivy-2.0	<3, 5, 16>	29	Xalan-2.5	<16, 18, 26>
13	Jedit-3.2	<12, 16, 24>	30	Xalan-2.6	<16, 21, 26>
14	Jedit-4.0	<15, 31, 32>	31	Xerces-init	<10, 20, 25>
15	Lucene-2.0	<21>	32	Xerces-1.2	<16, 25, 26>
16	Lucene-2.2	<13, 25, 34>	33	Xerces-1.3	<13, 23, 27>
17	Lucene-2.4	<21, 31, 34>	34	Xerces-1.4	<17, 25>

In order to verify the feasibility of the novel Diffused Bayes classifier in small sample issues, we chose respectively 1/10 (10%), 1/15 (6.67%), 1/20 (5%), 1/30 (3.33%) and 1/50 (2%) of the most suitable training data for each testing data set. The entire framework of our experiments is illustrated in Fig. 1.

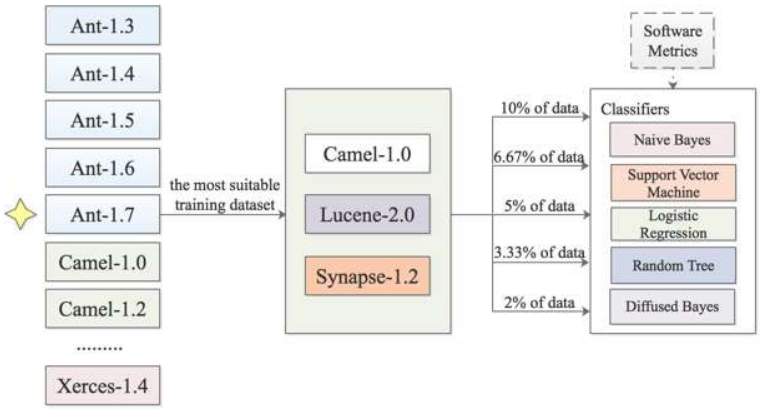

Fig.1 The entire framework of experiments - an example of release *Ant-1.7*

3.2 Feature Attributes

As described in Fig. 1, each instance represents a class file and includes two parts: independent variables including 20 static code metric attributes and a dependent variable *bug* labeling how many bugs are in this class. Table 2 in Appendix B presents all of the variables involved in our study.

However, before using *bug* as the dependent variable in the following experiments, we should do a preprocessing that transforms *bug* into a binary classification. There are two reasons why we use such a preprocessing: one is that the majority of class files in the 34 data sets have no more than 3 defects; the other is that the ratio of the instances with more than 10 defects to the total instances is less than 0.2% [26]. Furthermore, many previous studies have used the preprocessing in predicting software defect proneness, such as [11, 27-30]. In a word, a class is non-buggy only if its number of bugs is equal to zero. Otherwise, it is buggy. A defect prediction model typically labels each class as either buggy or non-buggy.

Besides this preprocessing, there should also be a simplification of these attributes for the reason that it would be easy to generate information redundancy and increase the complexity of the prediction models. Sometimes, useless or correlative attributes could even do harm to the accuracy of predictor [26].

In our experiments, we use the top-5 representative metric attributes to form a general feature subset to predict defects for all projects, which are determined by the times of occurrences. The occurrences of

these five metric attributes are more than 17 compared with the total number of occurrences 34: CE (17), LCOM (18), LOC (20), RFC (20), and CBO (21). Meanwhile, the five representative metric attributes perform as well as the entire metric attribute set in defect predicting. For more details, please refer to Sect. 3.3 in [26].

3.3 Evaluation Measures

In our research, the DB classifier is a binary classification technique, which could predict classes to be faulty or non-faulty. A binary classifier can make two possible errors: false negative (FN) and false positive (FP). In addition, the other two categories of defect prediction results are true negative (TN) and true positive (TP).

FN means that non-buggy classes are wrongly classified to be faulty, while FP means buggy classes are wrongly classified to be non-faulty. TN and TP refer to correctly classified non-buggy and buggy classes, respectively.

In order to verify the correctness of the prediction model, we use two accuracy indicators: Recall and Precision, to assess the results, and the F-measure to evaluate the overall performance.

Recall and Precision will be calculated by Eq. (11), while F-measure will be computed by Eq. (12).

$$Recall = \frac{TP}{TP+FN} \quad Precision = \frac{TP}{TP+FP} \tag{11}$$

$$F - measure = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{12}$$

The values of Recall, Precision and F-measure range from 0 to 1 and higher values indicate better prediction results.

Precision refers to how many of the classes judged buggy by the model are actually defect-prone. The false positives decrease as the Precision increases.

Recall addresses how many of the defect-prone classes are actually predicted by the model. The higher the Recall is; the fewer the false negatives exist.

F-measure computes the accuracy based on both Precision and Recall, which can be explained as a weighted average of Precision and Recall.

4. Experimental Results

In this section, we report the experimental results to validate the feasibility and effectiveness of the DB classifier we proposed.

4.1 Experiment 1: using 1/10 of the most suitable training sets

In order to ensure the reliability of the experimental results, all the five experiments are repeated 50 times using different data subsets selected from the most suitable training sets randomly. In this experiment, we use 1/10 of the most suitable training sets for each testing data set as training data. Table 4 and Fig. 2 below are the results for Experiment 1.

Table 4 Performance indices of different classifiers

(10% data, 50 times, where boldface font indicates the best performance)

Classifier	Mean Values		
	Recall	Precision	F-measure
NB	0.6525	0.4607	0.5134
SVM	0.6460	0.4387	0.4695
LR	0.5726	**0.5024**	0.4684
RT	0.5501	0.4191	0.4452
DB	**0.6814**	0.4512	**0.5215**

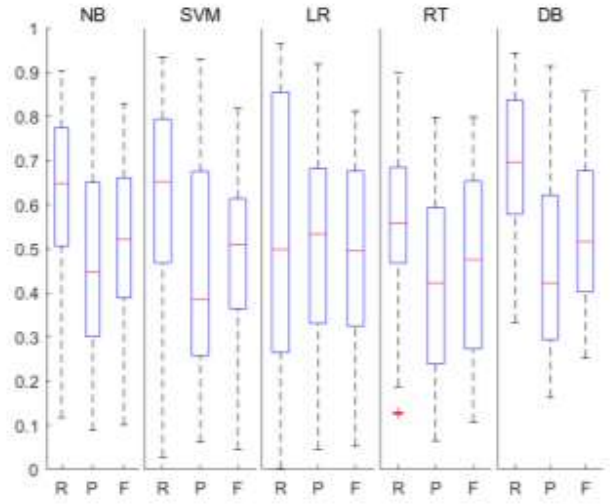

Fig.2 The standardized boxplots of the performance achieved in Experiment 1 by different predictors based on NB, SVM, LR, RT and DB, respectively. From the bottom to the top of a standardized box plot: minimum, first quartile, median, third quartile, and maximum. Any data not included between the box is plotted as a small cross

When compared with NB, SVM, LR, and RT in predicting defects, our DB classifier outperforms in Recall and F-measure, with values of 0.6814 and 0.5215, respectively. Although LR is more prominent in Precision than DB, Menzies et al. in literature [17] illustrated that Recall and false alarm rates are more stable than Precision when assessing their detectors.

4.2 Experiment 2: using 1/15 of the most suitable training sets

In the second experiment, we reduce the training data to 1/15 of the most suitable training sets. The experimental results are shown in Table 5 and Fig. 3.

Table 5 Performance indices of different classifiers

(6.67% data, 50 times, where boldface font indicates the best performance)

Classifier	Mean Values		
	Recall	Precision	F-measure
NB	0.6507	**0.4523**	0.4814
SVM	0.6584	0.4405	0.4676
LR	0.5120	0.4501	0.4472
RT	0.5365	0.4306	0.4323
DB	**0.7155**	0.4510	**0.5202**

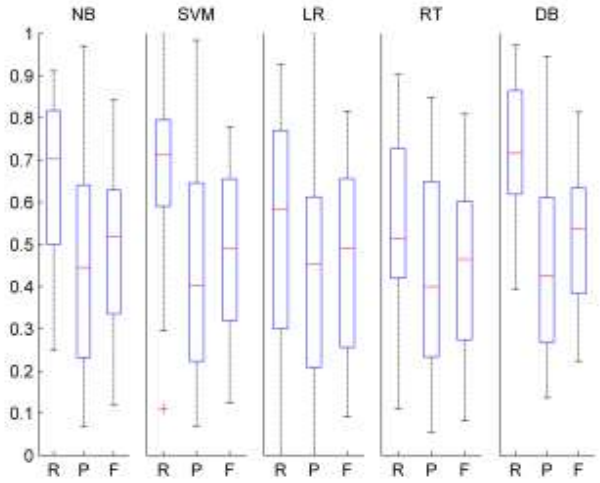

Fig.3 The standardized boxplots of the performance achieved in Experiment 2 by different predictors based on NB, SVM, LR, RT and DB, respectively.

Similar with Experiment 1, in terms of Recall and F-measure, our DB classifier performs better than the others. However, in this experiment, NB owns the highest Precision with the slightest advantage.

4.3 Experiment 3: using 1/20 of the most suitable training sets

Subsequent to Experiment 1 and 2, we further reduce the training data to only 5% in this experiment. Table 6 and Fig. 4 present a comparison for using different classifiers in terms of various performance indices on average and can be interpreted as follows.

Table 6 Performance indices of different classifiers

(5% data, 50 times, where boldface font indicates the best performance)

Classifier	Mean Values		
	Recall	Precision	F-measure
NB	0.6353	0.4649	0.5057
SVM	0.6796	0.4347	0.4746
LR	0.5724	**0.4664**	0.4996
RT	0.4929	0.3995	0.4329
DB	**0.7104**	0.4440	**0.5261**

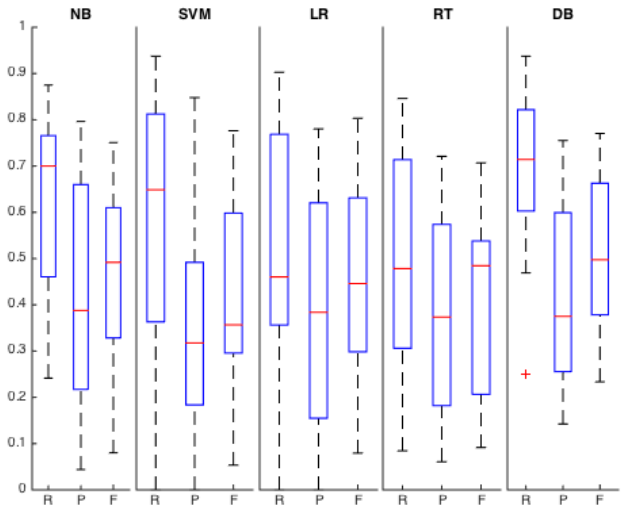

Fig.4 The standardized boxplots of the performance achieved in Experiment 3 by different predictors

In this experiment, although LR obtained the highest Precision, and NB and SVM could achieve relatively acceptable performance, the overall performance of DB is still better than the others, with the Recall of 0.7104 and F-measure of 0.5261 in the condition of decreased size of training data.

4.4 Experiment 4: using 1/30 of the most suitable training sets

In this experiment, the size of training data is decreased to only 3.33% of the most suitable training data set. The same as experiments before, the result is also the average of those obtained from 50 times repeated experiments using different data subsets. Table 7 and Fig. 5 show the performance of different classifiers.

In this experiment, although NB obtained the highest Precision (0.4459), DB still performs better than the others with a Recall of 0.7145 and F-measure of 0.5021 overall.

Table 7 Performance indices of different classifiers

(3.33% data, 50 times, where boldface font indicates the best performance)

Classifier	Mean Values		
	Recall	Precision	F-measure
NB	0.6528	**0.4459**	0.4898
SVM	0.6598	0.4416	0.4893
LR	0.5788	0.4269	0.4773
RT	0.5747	0.4343	0.4648
DB	**0.7145**	0.4443	**0.5021**

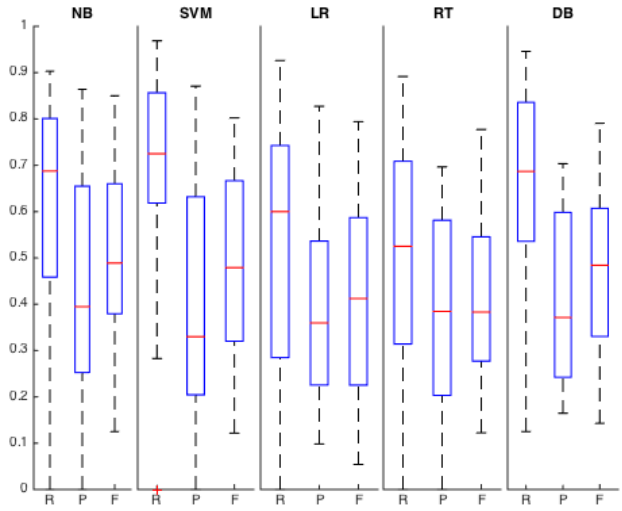

Fig.5 The standardized boxplots of the performance achieved in Experiment 4 by different predictors

4.5 Experiment 5: using 1/50 of the most suitable training sets

In Experiment 5, the size of training data is further decreased to 2%. Table 8 and Fig. 6 below give out the overall performance of different classifiers. In this experiment, we can find that our DB classifier has achieved the best in all the three respects, with the Recall of 0.6862, Precision of 0.4230 and F-measure of 0.5006 in the case where the training data are really scarce.

Table 8 Performance indices of different classifiers

(2% data, 50 times, where boldface font indicates the best performance)

Classifier	Mean Values		
	Recall	Precision	F-measure
NB	0.5842	0.4101	0.4517
SVM	0.6235	0.4034	0.4391
LR	0.5674	0.4121	0.4347
RT	0.4967	0.4132	0.4298
DB	**0.6862**	**0.4230**	**0.5006**

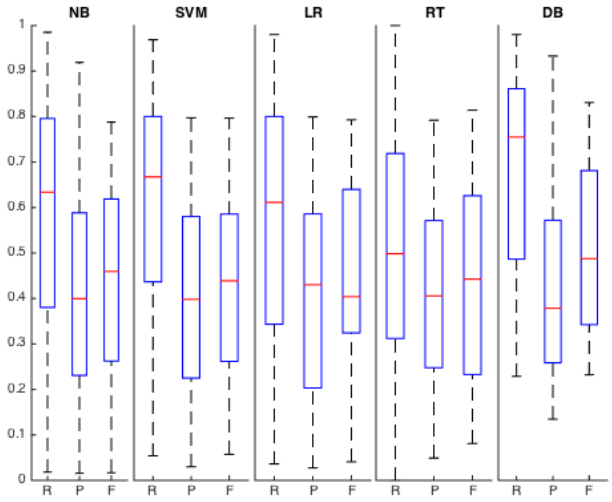

Fig.6 The standardized boxplots of the performance achieved in Experiment 5 by different predictors

When the results of the five experiments are summarized, we can find that, in the case of the same training data, DB performs best and stable in Recall and F-measure, while NB and LR are comparable in terms of Precision. Especially when the size of training data is reduced to a certain extent, the advantage of DB classifier is more obvious. The reason may be that DB classifier is more appropriate to deal with data that are scarce and follow non-normal distribution.

5. Discussion and Conclusion

This paper reports an empirical study to evolve the Naïve Bayes classifier with the diffusion function based on the vibration of string (VSDF), which is called Diffused Bayes (DB) classifier. The experimental results showed that the DB classifier can be used to estimate the probability distributions of the

non-normal distributed data especially in small samples. Our main contributions to the current research status can be summarized as follows:

(1) We proposed a novel Diffused Bayes classifier, which is to improve the traditional Naïve Bayes classifier with a new diffusion function based on the vibration of string. The novel one can help solve the problem of insufficient cross-project training data as well as the case of non-normal distributed attributes.

(2) We also validated the optimized performance of the classifier by comparing with other 4 main classifiers in different experiments, and found that it was competent enough when the training data from other projects are insufficient and follow non-normal distribution.

In summary, the experimental results show that our DB classifier for defect prediction is practical and viable. The defect prediction model built by DB classifier can have a satisfactory performance. We believe that our DB classifier can be of practical use especially when fewer training data are provided for predicting new projects, and expect our findings could improve the development activities.

We will focus mainly on two aspects in the future: (1) evolve the diffusion function to improve the generality and (2) simplify the metric attribute sets to further enhance the predictive performance of our approach.

Acknowledgements

This paper was supported in part by the Natural Science Foundation, China (No. 61702544) and the Natural Science Foundation of Jiangsu Province, China (No. BK20160769).

References

[1] C. Tantithamthavorn, S. Mcintosh, A. Hassan, K. Matsumoto, An Empirical Comparison of Model Validation Techniques for Defect Prediction Models, IEEE Transactions on Software Engineering, 43 (2017) 1-18.
[2] F. Zhang, A. Mockus, I. Keivanloo, Y. Zou, Towards building a universal defect prediction model with rank transformed predictors, Empirical Software Engineering, 21 (2016) 2107-2145.
[3] T. Jiang, L. Tan, S. Kim, Personalized defect prediction, in: Ieee/acm International Conference on Automated Software Engineering, 2013, pp. 279-289.
[4] E. Arisholm, L.C. Briand, E.B. Johannessen, A systematic and comprehensive investigation of methods to build and evaluate fault prediction models, Journal of Systems & Software, 83 (2010) 2-17.
[5] T. Menzies, Z. Milton, B. Turhan, B. Cukic, Y. Jiang, A. Bener, Defect prediction from static code features: current results, limitations, new approaches, Automated Software Engineering, 17 (2010) 375-407.
[6] Y. Ma, G. Luo, X. Zeng, A. Chen, Transfer learning for cross-company software defect prediction, Information & Software Technology, 54 (2012) 248-256.
[7] C. Jin, J.A. Liu, Applications of Support Vector Mathine and Unsupervised Learning for Predicting Maintainability Using Object-Oriented Metrics, in: Second International Conference on Multimedia and Information Technology, 2010.
[8] C. Catal, B. Diri, Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem, Information Sciences, 179 (2009) 1040-1058.
[9] Q. Song, M. Shepperd, M. Cartwright, C. Mair, Software Defect Association Mining and Defect Correction Effort Prediction, IEEE Transactions on Software Engineering, 32 (2006) 69-82.
[10] Y. Kamei, T. Fukushima, S. Mcintosh, K. Yamashita, N. Ubayashi, A.E. Hassan, Studying just-in-time defect prediction using cross-project models, Empirical Software Engineering, 21 (2016) 2072-2106.
[11] Z. He, F. Shu, Y. Yang, M. Li, Q. Wang, An investigation on the feasibility of cross-project defect prediction, Automated Software Engineering, 19 (2012) 167-199.
[12] Zimmermann, Thomas, Nagappan, Nachiappan, Harald, Giger, Emanuel, Murphy, Brendan, Cross-project defect prediction: a large scale experiment on data vs. domain vs. process, (2009).
[13] F. Zhang, Q. Zheng, Y. Zou, A.E. Hassan, Cross-Project Defect Prediction Using a Connectivity-Based Unsupervised Classifier, in: Ieee/acm International Conference on Software Engineering, 2017, pp. 309-320.
[14] J. Nam, S.J. Pan, S. Kim, Transfer defect learning, in: International Conference on Software Engineering, 2013, pp. 382-391.
[15] J. Nam, S. Kim, Heterogeneous defect prediction, IEEE Transactions on Software Engineering, PP (2015) 1-1.
[16] J. Xuan, J. Lu, G. Zhang, R.Y.D. Xu, X. Luo, Bayesian Nonparametric Relational Topic Model through Dependent Gamma Processes, IEEE Transactions on Knowledge & Data Engineering, 29 (2015) 1357-1369.
[17] T. Menzies, J. Greenwald, A. Frank, Data Mining Static Code Attributes to Learn Defect Predictors, IEEE Transactions on Software Engineering, 33 (2007) 2-13.
[18] T. Wang, W.H. Li, Naive Bayes Software Defect Prediction Model, in: Computational Intelligence and Software Engineering (CiSE), 2010 International Conference on, 2010, pp. 1-4.
[19] F.J.M. Jr., Taylor & Francis Online :: The Kolmogorov-Smirnov Test for Goodness of Fit - Journal of the American Statistical Association - Volume 46, Issue 253, Journal of the American Statistical Association.
[20] H.W. Lilliefors, On the Kolmogorov-Smirnov Test for Normality with Mean and Variance Unknown, Journal of the American Statistical Association, 62 (1967) 399-402.
[21] N.M. Razali, Y.B. Wah, Power Comparisons of Shapiro-Wilk, Kolmogorov-Smirnov, Lilliefors and Anderson-Darling Tests, 2 (2011).
[22] C.Z. Bai, R. Zhang, M. Hong, L. Qian, Z. Wang, A new information diffusion modelling technique based on vibrating string equation and its application in natural disaster risk assessment, International Journal of General Systems, 44 (2014) 161-165.
[23] I.C. Society., IEEE standard glossary of software engineering terminology, Institute of Electrical and Electronics Engineers, 1983.
[24] I.H. Witten, E. Frank, Data Mining: Practical Machine Learning Tools and Techniques (Third Edition), (2005).
[25] C. Huang, Principle of information diffusion, Fuzzy Sets & Systems, 91 (1997) 69-90.
[26] P. He, B. Li, X. Liu, J. Chen, Y. Ma, An empirical study on software defect prediction with a simplified metric set, Information & Software Technology, 59 (2015) 170-190.
[27] F. Peters, T. Menzies, A. Marcus, Better cross company defect prediction, (2013) 409-418.
[28] B. Turhan, T. Menzies, A.B. Bener, J.D. Stefano, On the relative value of cross-company and within-company data for defect prediction. Empir Softw Eng, Empirical Software Engineering, 14 (2009) 540-578.
[29] B. Turhan, T. M, A., A. Bener, Empirical evaluation of the effects of mixed project data on learning defect predictors, Information & Software Technology, 55 (2013) 1101-1118.
[30] S. Herbold, Training data selection for cross-project defect prediction, in: International Conference on Predictive MODELS in Software Engineering, 2013, pp. 1-10.