PAPER

Empirical Studies of a Kernel Density Estimation Based Naive Bayes Method for Software Defect Prediction*

Haijin JI^{†,††}, Song HUANG^{††a)}, Xuewei LV^{†,††}, Yaning WU^{††}, *Nonmembers*, and Yuntian FENG^{††}, *Member*

SUMMARY Software defect prediction (SDP) plays a significant part in allocating testing resources reasonably, reducing testing costs, and ensuring software quality. One of the most widely used algorithms of SDP models is Naive Bayes (NB) because of its simplicity, effectiveness and robustness. In NB, when a data set has continuous or numeric attributes, they are generally assumed to follow normal distributions and incorporate the probability density function of normal distribution into their conditional probabilities estimates. However, after conducting a Kolmogorov-Smirnov test, we find that the 21 main software metrics follow non-normal distribution at the 5% significance level. Therefore, this paper proposes an improved NB approach, which estimates the conditional probabilities of NB with kernel density estimation of training data sets, to help improve the prediction accuracy of NB for SDP. To evaluate the proposed method, we carry out experiments on 34 software releases obtained from 10 open source projects provided by PROMISE repository. Four well-known classification algorithms are included for comparison, namely Naive Bayes, Support Vector Machine, Logistic Regression and Random Tree. The obtained results show that this new method is more successful than the four well-known classification algorithms in the most software releases.

key words: software defect prediction, naive Bayes, kernel density estimation, software metrics

1. Introduction

In recent years, software defect prediction (SDP) has attracted the attentions of a growing number of researchers in the field of software engineering [1]–[7]. It usually focuses on estimating the defect proneness of software modules, and helps software practitioners allocate limited testing resources to those parts which are most likely to contain defects. This effort is particularly useful when the whole software system is too large to be tested exhaustively or the project budget is limited. At present, many machine learning and statistical methods have been investigated for defect prediction [8]–[11], and one of the most widely used approaches is classification. Its main task is to classify software modules into two types: defective or non-defective. Therefore, prediction accuracy is the key to a defect prediction model. For a specific defect prediction model, it usually consists of two components:

training data and classifier [12]. The training data could be obtained from bug reports, emails of developers, change logs and so on [3], which can be then used to train the classifiers. At present, many software metric data sets from real-world projects are available for public use, such as PROMISE (available at <http://openscience.us/repo>), Apache (available at <http://www.apache.org>) and Eclipse (available at <http://eclipse.org>) repositories, which allow researchers to build repeatable, comparable models across studies. In terms of classifiers, many classification algorithms (classifiers) have been employed in SDP, such as Naive Bayes (NB) [7], Support Vector Machine (SVM) [9], Logistic Regression (LR) [10] and Random Tree (RT) [11]. After comparing with SVM, LR and RT, Menzies et al. [7] demonstrated that NB may be more suitable than the other classifiers in SDP.

NB is one of the most widely used classifiers of SDP models because of its simplicity, effectiveness and robustness [13]. In NB, when a data set has continuous or numeric attributes, they are generally assumed to follow normal distributions and incorporate the probability density function of normal distribution into their conditional probabilities estimates [20]. However, after conducting a Kolmogorov-Smirnov (KS) test on the 34 software releases (the 34 software releases are explained in detail in Sect. 4.1) [14], we find that the 21 main software metrics follow non-normal distribution at the 5% significance level. More specifically, for each numeric attribute in a specific data set (i.e. each software release is a data set), the default null hypothesis is that all the values of this attribute are from a normal distribution; we do the KS test 10000 times for each numeric attribute of a data set based on the “kstest” function in Matlab; the return value h equals 1 if it rejects the null hypothesis at the default significance level 5%, otherwise, h equals 0 if it cannot. Taking Lucene-2.4 as an example, the results are shown in Table 1. From this Table, it can be easily obtained that it is not appropriate to estimate the conditional probability of a specific value in numeric attributes utilizing the probability density function of normal distribution.

Aiming at shortcomings of the existing classifiers mentioned above, we introduce kernel density estimation (KDE) method to improve the NB classifier in this paper. To the best of our knowledge, no study about KDE has been reported in SDP literature. KDE is an effective method to estimate the underlying probability density function of a data set. It is a nonparametric density estimator requiring no assumption that the underlying density function is from a

Manuscript received May 17, 2018.

Manuscript publicized October 3, 2018.

[†]The authors are with Huaiyin Normal University, Huaian, China.

^{††}The authors are with Army Engineering University of PLA, Nanjing, China.

*This work is supported by the National Natural Science Foundation of China (Grant No. 61702544), Natural Science Foundation of Jiangsu Province of China (Grant No. BK20160769).

a) E-mail: songh.jfj@163.com (Corresponding author)

DOI: 10.1587/transinf.2018EDP7177

Table 1 The results of Lucene-2.4 for the KS test.

Attribute	h=0	h=1	Attribute	h=0	h=1
WMC	1	9999	LOC	1	9999
DIT	3	9997	DAM	2	9998
NOC	2	9998	MOA	3	9997
CBO	3	9997	MFA	3	9997
RFC	2	9998	CAM	4	9996
LCOM	1	9999	IC	1	9999
CA	2	9998	CBM	3	9997
CE	3	9997	AMC	4	9996
NPM	1	9999	MAX_CC	2	9998
LCOM3	2	9998	AVG_CC	1	9999
BUG	1	9999			

parametric family [15]. Based on the KDE, the probability density function can be learned from the observations, whether they follow normal distribution or not. As a result, we obtained an improved NB method based on KDE and named it as KDENB.

The rest of this paper is structured as follows: Section 2 gives a brief overview of some related work. Section 3 presents our new proposed method KDENB for SDP. Sections 4 and 5 give the detailed experimental setup and the primary results analysis respectively. Finally, we close this paper with some summaries in Sect. 6.

2. Related Work

To facilitate readers' easy understanding, we review several basic concepts of Naive Bayes and kernel density estimation in this section.

2.1 Naive Bayes

NB is one of the most effective classifier methods. Compared with other machine learning algorithms, it is easy to understand and usually superior to the more complex classifiers especially in small data sets [7], [16]–[19].

Let $A = \{A_1, A_2, \dots, A_m\}$ be a set of software attributes, and $C = \{C_0, C_1\}$ be the category notion of a software module, where C_0 denotes non-defective category and C_1 is defective one. Let $Y = \{(A_1, a_1), (A_2, a_2), \dots, (A_m, a_m)\}$ be a testing software module, and $X = \{x_i | i = 1, 2, \dots, n\}$ be training data set of Y , where a_i and x_i are the attribute value and a software module, respectively.

According to Bayesian theory [20], the posterior probability of an instance is proportional to prior probability and likelihood. Thus, the NB formula of a specific category to which the testing software module Y belongs will be computed by Eq. (1):

$$P(C_k | Y) = \frac{P(C_k)P(Y | C_k)}{P(Y)} \quad (1)$$

Since the denominator $P(Y)$ in Eq. (1) is same for all categories (i.e. C_0 and C_1 in this paper), it does not affect the classification. Thus, it can be removed from Eq. (1) and written as the following expression:

$$P(C_k | Y) = P(C_k)P(Y | C_k) \quad (2)$$

In NB, the attributes are assumed to be independent [20]. Therefore, Eq. (2) can be deduced to Eq. (3):

$$P(C_k | Y) = P(C_k) \prod_{i=1}^m P(a_i | C_k) \quad (3)$$

After training the NB classifier with training data set X , the C_k ($k = 0$ or 1) with higher $\arg\max_k (P(C_k | Y))$ is the category of testing software module Y . In other words, as shown in Eq. (4), the classifier will classify the software module Y to the category with higher $V(Y)$.

$$V(Y) = \arg \max_k \left(P(C_k) \prod_{i=1}^m P(a_i | C_k) \right) \quad (4)$$

In Eq. (4), $P(C_k)$ will be computed by Eq. (5):

$$P(C_k) = \frac{N_k}{N} \quad (5)$$

where N is the total number of software modules in training data set X , and N_k refers to the number of software modules which belong to the category C_k .

In SDP, as most of the attributes of software metric data sets are numeric ones, they are generally assumed to follow normal distributions and incorporate the probability density function of normal distribution into their conditional probabilities estimates (i.e. the likelihood $P(a_i | C_k)$) [20]. Therefore, the likelihood $P(a_i | C_k)$ will be computed by Eq. (6):

$$P(a_i | C_k) = \frac{1}{\sqrt{2\pi} \sigma_i} e^{-\frac{(a_i - \mu_i)^2}{2\sigma_i^2}} \quad (6)$$

where the parameters μ_i and σ_i are the mean and standard deviation of the values of attribute A_i in training data set X belonging to the category C_k .

The above derivation has briefly described the principle of NB. However, in SDP, the software metric attributes often follow non-normal distribution, which will eventually do harm to the classification performance of the NB classifier. In other words, the Eq. (6) is not appropriate in this case.

2.2 Kernel Density Estimation

Kernel density estimation refers to a nonparametric statistical modeling method that only uses given data to build a statistical model [15]. In other words, it can estimate a probabilistic distribution for given data without requiring the specific probability density functions. The probability density function of given data can be obtained by combining kernel functions, which are generated by each value in the given data [21], [22]. Therefore, KDE is an effective way to estimate the probability density function when we do not know the distribution of given data and this characteristic of KDE will help solve the problem of non-normal distribution for NB, when a data set has continuous or numeric attributes.

Before presenting the detailed principle of KDE, it could be helpful to get some intuitive feeling on KDE

Fig. 1 The frequency description of attribute “DAM”.

Fig. 2 The probability density function of normal distribution (i.e. “normal distribution pdf”) according to mean = 0.4758 and standard deviation = 0.44442, and the one estimated by KDE estimator with Gaussian kernel (i.e. “KDE pdf”).

through an experiment. This experiment is conducted on the attribute “DAM” of software release “Lucene-2.4” (the software release “Lucene-2.4” is explained in Sect. 4.1). The attribute “DAM” has 340 values and the frequency description is shown in Fig. 1. We plot the probability density function of normal distribution (i.e. “normal distribution pdf”) in Fig. 2) according to mean = 0.4758 and standard deviation = 0.44442, and the one estimated by KDE estimator with Gaussian kernel (i.e. “KDE pdf” in Fig. 2). From Fig. 1 and Fig. 2, it can easily be concluded that “normal distribution pdf” does not reflect the actual probability density of data following non-normal distribution because it only considers the mean and standard deviation, while “KDE pdf” can reflect the actual probability density of data well. Therefore, we intend to use the probability density estimated by KDE to solve the problem of non-normal distribution in NB. Now, let us give several basic concepts of KDE as follows.

Let $X_1, X_2, \dots, X_n$ in the set $\mathbb{R}$ be a given univariate random sample from a distribution with probability density function f which we wish to estimate. Then the probability

Table 2 Common second-order kernels.

Kernel	Equation	Equation number
Gaussian	$K(u) = \frac{1}{\sqrt{2\pi}} \exp(-\frac{u^2}{2})$	(9)
Epanechnikov	$K(u) = \frac{3}{4} (1-u^2) I(u \leq 1)$	(10)
Biweight	$K(u) = \frac{15}{16} (1-u^2)^2 I(u \leq 1)$	(11)
Triweight	$K(u) = \frac{35}{32} (1-u^2)^3 I(u \leq 1)$	(12)

Table 3 $C_v(k)$ in Silverman’s rule.

Kernel	$v=2$	$v=4$	$v=6$
Gaussian	1.06	1.08	1.08
Epanechnikov	2.34	3.03	3.53
Biweight	2.78	3.39	3.84
Triweight	3.15	3.72	4.13

density function estimated by KDE $\hat{f}: \mathbb{R} \rightarrow \mathbb{R}$ is defined as [21], [22]:

$$\hat{f} = \frac{1}{nh} \sum_{i=1}^n K\left(\frac{x - X_i}{h}\right) \quad (7)$$

where n is the sample size, h is the smoothing parameter of the KDE estimator $\hat{f}$ and named as bandwidth. $K: \mathbb{R} \rightarrow \mathbb{R}$ is the kernel function that satisfies the condition in Eq. (8):

$$\int_{-\infty}^{+\infty} K(x) dx = 1 \quad (8)$$

An important factor that affects the accuracy of KDE estimator $\hat{f}$ is the types of kernel functions. Table 2 lists the common second-order kernels [23]. The most commonly used kernels are the Gaussian and the Epanechnikov [23]. In this paper, we will explore using them to evolve Naive Bayes classifier.

Another important factor that affects the accuracy of KDE estimator $\hat{f}$ is bandwidth h . The smaller the value of h , the sharper the probability density function estimated by KDE $\hat{f}$; the larger the value of h , the smoother the probability density function estimated by KDE $\hat{f}$. An inappropriate bandwidth h may result in under-smoothing or over-smoothing. Therefore, an optimum h is very important to a kernel density function.

Among bandwidth selection methods, Silverman’s rule of thumb is the most popular one [24]. The more detailed information about Silverman’s rule of thumb can be seen in reference [21] and [23]. It defines optimum bandwidth h for common kernel functions as follows.

$$h = \hat{\sigma} C_v(k) n^{-1/(2v+1)} \quad (13)$$

where n is the number of samples, v is the order of kernel functions ($v = 2$ in this paper). $C_v(k)$ is a constant shown in Table 3 that depends on the order v . $\hat{\sigma}$ is the standard deviation estimated from data samples. As Silverman’s rule of thumb is sensitive to outliers, it is not appropriate to use the standard deviation that square root of variance of values from its mean. In this paper, a more robust corrected standard deviation is used, and the Eq. (14) shows the formula

definition [25].

$$\hat{\sigma} = \frac{\text{Median}(|X_i - \text{Median}(X_i)|)}{0.6745} \quad (14)$$

With the training data, the KDE is an appropriate method for analyzing the probability density function either normal or non-normal distribution. In the next section, we will propose the improved Naive Bayes based on KDE.

3. A Kernel Density Estimation Based Naive Bayes Method for Software Defect Prediction

As discussed in previous sections, the software metric attributes are usually numeric attributes and they do not follow normal distribution. Therefore, we propose an improved Naive Bayes method, named as KDENB, to improve the performance of NB for SDP using KDE.

Let $A = \{A_1, A_2, \dots, A_m\}$ be a set of software attributes, and $C = \{C_0, C_1\}$ be the category notion of a software module, where C_0 denotes non-defective category and C_1 is defective one. Let $Y = \{(A_1, a_1), (A_2, a_2), \dots, (A_m, a_m)\}$ be a testing software module, and $X = \{x_i | i = 1, 2, \dots, n\}$ be training data set of Y , where a_i and x_i are the attribute value and a software module, respectively.

As discussed in Sect. 2, according to Eq. (4), the testing software module Y will be classified to C_0 or C_1 . The prior probability $P(C_k)$ in Eq. (4) can be obtained by Eq. (5). The conditional probability $P(a_i | C_k)$ (i.e. the likelihood), which is the main difference between NB and our proposed method KDENB, can be computed by Eq. (15).

$$P(a_i | C_k) = \frac{1}{nh} \sum_{j=1}^n K\left(\frac{a_i - x_{ji}}{h}\right) \quad (15)$$

where x_{ji} is the i_{th} metric attribute of the software module x_j in the training data set X . The bandwidth h can be obtained by Eq. (13) and Eq. (14). There are 4 common kernel functions $K(\cdot)$ available, i.e. Gaussian (Eq. (9)), Epanechnikov (Eq. (10)), Biweight (Eq. (11)) and Triweight (Eq. (12)), which have been discussed in Sect. 2. Therefore, in this paper, we conduct experiments to make comparisons among these common kernel functions, and select the most appropriate one based on *F-measure*. For readers' easy understanding, the improved Naive Bayes method based on kernel density estimation KDENB can be described as the following algorithm.

Algorithm 1 the proposed method KDENB

Input: The testing software module Y ; the training data set $X = \{x_i | i = 1, 2, \dots, n\}$.

Output: The category of Y .

1. Compute the prior probability $P(C_k)$ based on Eq.(5).
 2. Calculate the bandwidth h using Eq.(13) and Eq.(14) for each software metric attribute.
 3. Select the most appropriate common kernel functions between Eq.(9), Eq.(10), Eq.(11) and Eq.(12) based on *F-measure* for Eq.(15).
 4. Compute the conditional probabilities $P(a_i | C_k)$ (i.e. the likelihood) using Eq.(15) for each software metric attribute value in the testing software module Y .
 5. Calculate the category of the testing software module Y using Eq.(4).
-

4. Experimental Setup

In order to apply the proposed method KDENB to software defect prediction, there are some work to be done, such as the data sets used in the experiments, the learner model and the experiment design.

4.1 Data Collection

There are 34 software releases from 10 real-world software applications in our experiments. They are public available in PROMISE repository. Table 4 presents the details of these software releases, where #MDs represents the number of instances, #DP is the number of defects, and %DP is the ratio of defective modules to all modules. The 21 software metric attributes are listed in Table 5, which contains 20 software attributes describing structural characteristics of each module from the selected software applications, and a labeled attribute BUG. Since BUG is the number of bugs in the module, we should transform BUG into a binary classification in our experiments. More specifically, if the BUG of a module is 0, this module is non-defective. Otherwise, it

Table 4 Details of the 34 software releases.

Release	#MDs	#DP	%DP	Release	#MDs	#DP	%DP
Ant-1.3	125	20	0.16	Poi-1.5	237	141	0.595
Ant-1.4	178	40	0.225	Poi-2.0	314	37	0.118
Ant-1.5	293	32	0.109	Poi-2.5	385	248	0.644
Ant-1.6	351	92	0.262	Poi-3.0	442	281	0.636
Ant-1.7	745	166	0.223	Synapse-1.0	157	16	0.102
Camel-1.0	339	13	0.038	Synapse-1.1	222	60	0.270
Camel-1.2	608	216	0.355	Synapse-1.2	256	86	0.336
Camel-1.4	872	145	0.166	Velocity-1.4	196	147	0.750
Camel-1.6	965	188	0.195	Velocity-1.5	214	142	0.664
Ivy-1.1	111	63	0.568	Velocity-1.6	229	78	0.341
Ivy-1.4	241	16	0.066	Xalan-2.4	723	110	0.152
Ivy-2.0	352	40	0.114	Xalan-2.5	803	387	0.482
Jedit-3.2	272	90	0.331	Xalan-2.6	885	411	0.464
Jedit-4.0	306	75	0.245	Xerces-init	162	77	0.475
Lucene-2.0	195	91	0.467	Xerces-1.2	440	71	0.161
Lucene-2.2	247	144	0.583	Xerces-1.3	453	69	0.152
Lucene-2.4	340	203	0.597	Xerces-1.4	588	437	0.743

Table 5 The 21 software metric attributes.

Feature	Description
WMC	Weighted Methods per Class
DIT	Depth of Inheritance Tree
NOC	Number of Children
CBO	Coupling Between Object Classes
RFC	Response for a Class
LCOM	Lack of Cohesion in Methods
CA	Afferent couplings
CE	Efferent couplings
NPM	Number of Public Methods
LCOM3	Normalized version of LCOM
LOC	Lines of code
DAM	Data access metric
MOA	Measure of aggregation
MFA	Measure of function abstraction
CAM	Cohesion among methods of class
IC	Inheritance Coupling
CBM	Coupling between methods
AMC	Average method complexity
MAX_CC	Maximum values of methods in the same class
AVG_CC	Mean values of methods in the same class
BUG	Number of bugs in the module

Fig. 3 Learner model

Fig. 4 10 x 10 cross-validation

is defective.

4.2 Learner Model

The learner model used in experiments is shown in Fig. 3 referring to Song et al. [26]. As shown in Fig. 3, the learner model trains the proposed method KDENB in training data, and then applies the trained method to the test data to generate the performance report. The test data is not included in the training phase. Specifically, the training data is first pre-processed in the normalization and feature selection steps, and the obtained parameters are used to preprocess the test data. The parameters of KDENB are learned from the pre-processed training data and used in the proposed method. Each instance in the test data are then tested based on the proposed method KDENB, and finally the performance report is created. In order to get reliable experimental results, this learner model are repeated using 10×10 cross-validation, which is shown in Fig. 4. In more specific terms, the whole data set is divided into 10 bins, 9 bins are used for training data and 1 bin is used for test data. To ensure that each bin is used for test data once to minimize the sampling bias, the experiment is conducted 10 runs. On the other hand, to reduce the ordering effect, the above process is re-

peated 10 times and the ranking of data sets is randomized in each iteration. Thus, we perform $10 \times 10 = 100$ times for each experiment. Now let us introduce the normalization, feature selection and Evaluation Measures used in this paper.

4.2.1 Normalization

Since the range of software metric attributes may vary widely, this can lead standard deviation, minimum and maximum to be very different, which seriously affect the performance of the classifier. Therefore, it is necessary to normalize the data in preprocessing. In this paper, we use a min-max normalization procedure for each attribute. More specifically, the data is convert to the range of $[0, 1]$ as follows.

$$x' = \frac{x - \min}{\max - \min} \quad (16)$$

where x and x' are the original data and converted value respectively.

After normalization, standard deviation within and between attributes (i.e. features) is reduced and the negative impact on the classifier caused by outliers can be alleviated.

4.2.2 Feature Selection

The software metric attributes (i.e. features) are designed for different purposes, not all of them contribute to software defect prediction. In other words, the useless or correlative features may do harm to the performance of predictor. Therefore, feature selection is necessary in the preprocessing, and the correlation-based feature selection (CFS) is used in this paper [27].

CFS is widely used for feature selection [27]. CFS aims to select the good feature subsets, which are highly correlated with the class feature (i.e. labeled attribute), and uncorrelated with each other. CFS uses the best first strategy to search the feature subset space and uses the following equation to evaluate the merit of a feature subset S containing k features [27]:

$$Merit_s = \frac{k\overline{r_{cf}}}{\sqrt{k + k(k-1)\overline{r_{ff}}}} \quad (17)$$

where $\overline{r_{cf}}$ refers to the average feature-class correlation, and $\overline{r_{ff}}$ is the average feature-feature intercorrelation. It can be obtained that the heuristic merit aims to search the feature subset with bigger $\overline{r_{cf}}$ by removing irrelevant features and smaller $\overline{r_{ff}}$ by removing redundant features.

4.2.3 Evaluation Measures

In this paper, the prediction accuracy of proposed method is measured by *F-measure*, which is a weighted mean of *Precision* and *Recall*:

$$F - measure = \frac{2 \times Precision \times Recall}{Precision + Recall} \quad (18)$$

where *Precision* and *Recall* are defined as follows:

$$\text{Precision} = \frac{TP}{TP + FP} \quad (19)$$

$$\text{Recall} = \frac{TP}{TP + FN} \quad (20)$$

In software defect prediction, the defective modules are classified as positive category and the non-defective modules are classified as negative category. Therefore, *TP* (i.e. true positive) is the number of defective modules which are correctly classified, and *TN* (i.e. true negative) refers to the number of non-defective modules which are correctly classified. *FP* (i.e. false positive) is the number of modules which are wrongly classified from non-defective modules to defective, while *FN* (i.e. false negative) refers to the number of modules which are wrongly classified from defective modules to non-defective.

4.3 Experiment Design

In this section, we design experiments

based on three questions to validate the feasibility and effectiveness of the proposed method KDENB. All these experiments follow the process of learner model discussed in Sect. 4.2.

RQ1. Which kernel function used in KDE is more appropriate in SDP?

In this paper, KDENB use KDE to estimate the probability density function of given data to solve the problem of non-normal distribution of NB in SDP. The kernel function $K(\cdot)$ is very important to KDE. Therefore, to identify which kernel function is the most appropriate in KDE, we perform comparison experiments among Gaussian (Eq. (9)), Epanechnikov (Eq. (10)), Biweight (Eq. (11)) and Triweight (Eq. (12)) based on *F-measure*.

The descriptions of these four kernel functions have been discussed in Sect. 2.2. As shown in Table 6, after applying these kernel functions to KDE, the likelihood $P(a_i | C_k)$ in Eq. (15) can be represented as Eq. (21), Eq. (22), Eq. (23) and Eq. (24) respectively. The parameters n , h and x_{ji} are the same as in Eq. (15). In other words, the main task of this experiment is to identify which equation among Eq. (21), Eq. (22), Eq. (23) and Eq. (24) could obtain the best

Table 6 The likelihood $P(a_i | C_k)$ in Eq. (15) can be represented as Eq. (21), Eq. (22), Eq. (23) and Eq. (24) according to different kernel functions.

Kernel	Equation of $P(a_i C_k)$	Equation number
Gaussian	$P(a_i C_k) = \frac{1}{\sqrt{2\pi}nh} \sum_{j=1}^n \exp\left(-\frac{(a_i - x_{ji})^2}{2h^2}\right)$	(21)
Epanechnikov	$P(a_i C_k) = \frac{3}{4nh} \sum_{j=1}^n \left(1 - \frac{(a_i - x_{ji})^2}{h^2}\right) I\left(\left \frac{a_i - x_{ji}}{h}\right \leq 1\right)$	(22)
Biweight	$P(a_i C_k) = \frac{15}{16nh} \sum_{j=1}^n \left(1 - \frac{(a_i - x_{ji})^2}{h^2}\right)^2 I\left(\left \frac{a_i - x_{ji}}{h}\right \leq 1\right)$	(23)
Triweight	$P(a_i C_k) = \frac{35}{32nh} \sum_{j=1}^n \left(1 - \frac{(a_i - x_{ji})^2}{h^2}\right)^3 I\left(\left \frac{a_i - x_{ji}}{h}\right \leq 1\right)$	(24)

performance in KDENB.

RQ2. How is the classification performance affected by improper bandwidth h of KDE?

The bandwidth h of KDE is very important to the classification performance of the proposed method KDENB. An inappropriate bandwidth h may result in under-smoothing or over-smoothing. According to Silverman's rule of thumb [21], [23], [24], the bandwidth h is computed by Eq. (13). Since Silverman's rule of thumb is sensitive to outliers, $\hat{\sigma}$ in Eq. (13) is computed by Eq. (14), other than the standard deviation that square root of variance of values from its mean. Therefore, we design this experiment to verify that the bandwidth h used in this paper is appropriate, and that how classification performance is affected by improper bandwidth h . In this experiment, the best kernel function obtained in *RQ1* is used, and the values of bandwidth h used for comparison are listed in Table 7.

RQ3. What is the classification performance compared with other prediction methods?

In this experiment, we implement the proposed method KDENB using best kernel function obtained in *RQ1* and the bandwidth h , and the four well-known classification algorithms for SDP are included for comparison. They are Naïve Bayes (NB), Support Vector Machine (SVM), Logistic Regression (LR) and Random Tree (RT).

NB is one of the simplest classifier based on conditional probability [16]. The reason why the classifier is named as 'naïve' is that the features are assumed to be independent. In practice, the NB classifier can performs better than some more sophisticated classifiers, although the independence assumption is often violated [28]. The prediction model established by this classifier is a set of probabilities. The probability that a new instance is defective or not is computed from the product of the individual conditional probabilities for the feature values of the instance.

SVM is used as a kind of supervised learning algorithms. In common practice, it is utilized for classification and regression analysis. It could search the optimal hyper plane and maximally divide the instances into two categories [9].

LR is a type of probabilistic statistical regression model for categorical prediction by fitting data to a logistic curve [10], [29]. It could also be used as a binary classifier to predict a binary response. In SDP, the labeled feature (either defective or not) is binary, thus, it is suitable for SDP.

RT, as a hypothesis space for supervised learning algorithm, is one of the simplest hypothesis spaces possible [11]. It consists of two parts: a schema and a body. The schema is a set of features and the body is a set of labeled instances.

Table 7 The values of bandwidth h used for comparison.

Bandwidth	Value of bandwidth
h	The value used in KDENB, i.e. computed by Eq.(13) and Eq.(14).
h_1	Computed by Eq.(13), yet the $\hat{\sigma}$ in Eq.(13) is the standard deviation (i.e. square root of variance of values from its mean)
h_2	$h/2$
h_3	$3h/2$

5. Experimental Results

In this section, based on the experimental results, we study the three research questions.

RQ1. Which kernel function used in KDE is more appropriate in SDP?

In this experiment, we do experiments to make comparisons among the 4 kernel functions listed in Table 6. The bandwidth h used in these kernel functions is computed by Eq.(13) and Eq.(14). This experiment follows the learner model discussed in Sect. 4.2. KDENB with each kernel function runs after preprocessing (i.e. normalization and feature selection, more detailed information is shown in Sects. 4.2.1 and 4.2.2) and this process repeats according to 10×10 cross-validation. In other words, we perform $10 \times 10 = 100$ times for each data set and each reported result is the average *F-measure* of these 100 experiments. We take the release Ant-1.3 as an example to illustrate the experiment results as follows.

As shown in Fig. 5, considering the median line of the boxes, we find that, for the proposed methods with different kernel functions, the KDENB with Gaussian kernel performs better than the others in general. The similar results are shown in Table 8. Since each value in Table 8 is the average of 100 experimental results and the Gaussian kernel has the highest average, we perform a Student's t-test between the Gaussian and the others at a confidence level of 95%. The significance test results are listed in Table 9. However, all the three *p-value* are greater than 0.05. In other words, there is no significant difference between Gaussian kernel and the other three kernels for Ant-1.3.

All experiments for the 34 software releases are listed in Table 10. It can be easily obtained that there is no sig-

Fig. 5 The standardized boxplots of the performances of KDENB achieved by different kernel functions based on Gaussian, Epanechnikov, Biweight and Triweight, respectively. From the bottom to the top of a standardized box plot: minimum, first quartile, median, third quartile, and maximum.

Table 8 The performances of KDENB achieved by different kernel functions for Ant-1.3. Values in boldface are significantly better than the rest, and there is no significant difference between the boldface values.

Gaussian	Epanechnikov	Biweight	Triweight
0.5789	0.5467	0.5650	0.5760

nificant difference among the four kernels for all 34 software releases. Therefore, the most commonly used kernel Gaussian kernel is selected as the kernel function in the proposed method KDENB.

RQ2. How is the classification performance affected by improper bandwidth h of KDE?

The bandwidth h of KDE is another very important factor that may affect the classification performance of KDENB. Therefore, we do experiments to make comparisons among the 4 kinds of bandwidths listed in Table 7. In this experiment, the kernel function used in KDE is Gaussian kernel, which has been discussed in *RQ1*. The learner model discussed in Sect. 4.2 is followed. We take the release Ant-1.3 as an example to illustrate the experiment results below.

As shown in Fig. 6 and Table 11, the proposed method KDENB with bandwidth h has the best performance. The results of Student's t-test are shown in Table 12, and all the three *p-value* are less than 0.05. The statistical results reveal that the performance between bandwidth h and others is significantly different. In other words, this result prove that the proposed method with bandwidth h performs best

Table 9 The Student's t-test of KDENB achieved by different kernel functions for Ant-1.3.

	p-value
Gaussian vs Epanechnikov	0.3242
Gaussian vs Biweight	0.6779
Gaussian vs Triweight	0.9310

Table 10 The performances of KDENB achieved by different kernel functions for 34 software releases.

Release	Gaussian	Epanechnikov	Biweight	Triweight
Ant-1.3	0.5789	0.5467	0.5650	0.5760
Ant-1.4	0.3421	0.3743	0.3643	0.3626
Ant-1.5	0.4841	0.4713	0.4748	0.4801
Ant-1.6	0.6982	0.6967	0.6959	0.6947
Ant-1.7	0.6259	0.6197	0.6195	0.6179
Camel-1.0	0.3374	0.3392	0.3450	0.3451
Camel-1.2	0.4469	0.4459	0.4427	0.4436
Camel-1.4	0.4147	0.4211	0.4195	0.4184
Camel-1.6	0.3539	0.3451	0.3489	0.3569
Ivy-1.1	0.8040	0.8063	0.8087	0.8105
Ivy-1.4	0.2938	0.3024	0.2997	0.2992
Ivy-2.0	0.5178	0.5087	0.5111	0.5092
Jedit-3.2	0.7107	0.7083	0.7044	0.7081
Jedit-4.0	0.6282	0.6154	0.6195	0.6234
Lucene-2.0	0.7494	0.7409	0.7432	0.7443
Lucene-2.2	0.7830	0.7847	0.7831	0.7827
Lucene-2.4	0.7907	0.7921	0.7918	0.7907
Poi-1.5	0.8500	0.8467	0.8456	0.8495
Poi-2.0	0.4500	0.4417	0.4433	0.4496
Poi-2.5	0.8814	0.8797	0.8811	0.8820
Poi-3.0	0.8978	0.8990	0.8997	0.8993
Synapse-1.0	0.4925	0.4927	0.4903	0.4942
Synapse-1.1	0.6099	0.6049	0.6073	0.6084
Synapse-1.2	0.6555	0.6478	0.6541	0.6534
Velocity-1.4	0.9364	0.9409	0.9420	0.9433
Velocity-1.5	0.8567	0.8564	0.8577	0.8529
Velocity-1.6	0.5811	0.5772	0.5776	0.5798
Xalan-2.4	0.4944	0.4964	0.4937	0.4951
Xalan-2.5	0.6951	0.6935	0.6943	0.6950
Xalan-2.6	0.7453	0.7440	0.7450	0.7460
Xerces-init	0.7413	0.7451	0.7368	0.7342
Xerces-1.2	0.3289	0.3337	0.3268	0.3298
Xerces-1.3	0.5504	0.5648	0.5626	0.5515
Xerces-1.4	0.9671	0.9620	0.9634	0.9643

Fig. 6 The standardized boxplots of the performances of KDENB achieved by different bandwidths. Any data not included between the box is plotted as a small cross.

Table 11 The performances of KDENB achieved by different bandwidths for Ant-1.3.

h	h_1	h_2	h_3
0.5789	0.4917	0.4907	0.4805

Table 12 The Student's t-test of KDENB achieved by different bandwidths for Ant-1.3.

	p-value
h vs h_1	0.0303
h vs h_2	0.0088
h vs h_3	0.0026

for Ant-1.3.

For all the 34 software releases, the experimental results are listed in Table 13. The bandwidth h obtain the best performance on all 34 software releases, while the bandwidth h_1 , h_2 , h_3 obtain the best performance on 17, 4 and 3 software releases respectively. The bandwidth h also obtain the highest *F-measure* 0.6263 in average. These experiments demonstrate that the bandwidth h computed by Eq. (13) and Eq. (14) is the most appropriate in KDENB for SDP.

RQ3. What is the classification performance compared with other prediction methods?

To demonstrate the overall effectiveness of the proposed method KDENB, we do comparison experiments among KDENB, NB, SVM, LR and RT. The kernel function and bandwidth in KDENB are Gaussian kernel and bandwidth h , which have been discussed in *RQ1* and *RQ2* respectively. The experiments are conducted following the learner model which is discussed in Sect. 4.2. We take the release Ant-1.3 as an example to illustrate the experiment results as follows.

As shown in Fig. 7 and Table 14, the proposed method KDENB has the best performance for Ant-1.3. Table 15 shows that the performance of KDENB is significantly different from others. Therefore, our proposed method KDENB is the best classifier for Ant-1.3.

Table 16 shows the final results for all 34 releases. The proposed method KDENB performs better than the other classifiers or has no significant difference with the best classifiers, except for Ivy-1.4. That may be because the %DP

Table 13 The performances of KDENB achieved by different bandwidths for 34 software releases. Values in boldface are significantly better than the rest, and there is no significant difference between the boldface values.

Release	h	h_1	h_2	h_3
Ant-1.3	0.5789	0.4917	0.4907	0.4805
Ant-1.4	0.3421	0.3067	0.2311	0.2475
Ant-1.5	0.4841	0.4867	0.4244	0.3959
Ant-1.6	0.6982	0.6492	0.6462	0.6243
Ant-1.7	0.6259	0.5713	0.5632	0.5661
Camel-1.0	0.3374	0.3690	0.2872	0.2164
Camel-1.2	0.4469	0.3983	0.2987	0.3818
Camel-1.4	0.4147	0.3394	0.3415	0.3520
Camel-1.6	0.3539	0.3029	0.2689	0.3134
Ivy-1.1	0.8040	0.8096	0.7237	0.7526
Ivy-1.4	0.2938	0.2030	0.2524	0.2283
Ivy-2.0	0.5178	0.5064	0.4180	0.4588
Jedit-3.2	0.7107	0.6507	0.6468	0.6519
Jedit-4.0	0.6282	0.6106	0.5623	0.5582
Lucene-2.0	0.7494	0.6866	0.6708	0.6788
Lucene-2.2	0.7830	0.7380	0.7193	0.7279
Lucene-2.4	0.7907	0.7296	0.7252	0.7347
Poi-1.5	0.8500	0.7813	0.7982	0.7838
Poi-2.0	0.4500	0.3787	0.3671	0.3899
Poi-2.5	0.8814	0.8438	0.8211	0.8241
Poi-3.0	0.8978	0.8331	0.8311	0.8271
Synapse-1.0	0.4925	0.4644	0.4059	0.4244
Synapse-1.1	0.6099	0.5517	0.5394	0.5412
Synapse-1.2	0.6555	0.6054	0.5949	0.5858
Velocity-1.4	0.9364	0.9134	0.9171	0.9076
Velocity-1.5	0.8567	0.8370	0.8205	0.8268
Velocity-1.6	0.5811	0.5337	0.5787	0.5542
Xalan-2.4	0.4944	0.4370	0.4425	0.4465
Xalan-2.5	0.6951	0.6699	0.6728	0.6599
Xalan-2.6	0.7453	0.7412	0.7144	0.7129
Xerces-init	0.7413	0.7471	0.6791	0.6680
Xerces-1.2	0.3289	0.3072	0.2383	0.2511
Xerces-1.3	0.5504	0.4930	0.4500	0.4876
Xerces-1.4	0.9671	0.9525	0.9369	0.9410
Average (<i>F-measure</i>)	0.6263	0.5803	0.5612	0.5647

Fig. 7 The performances of different classifiers for Ant-1.3

Table 14 The performances of different classifiers for Ant-1.3.

KDENB	NB	SVM	LR	RT
0.5789	0.4465	0.4416	0.4487	0.4707

(i.e. the ratio of defective modules to all modules) of Ivy-1.4 is too small (i.e. 0.066) and KDENB could not estimate the probability density function appropriately. In addition, with regard to different classifiers (NB, SVM, LR and RT), our KDENB performs better than the other four methods with the highest *F-measure* in average.

Table 15 The Student's t-test between KDENB and other four classifiers for Ant-1.3.

	p-value
KDENB vs NB	0.0001
KDENB vs SVM	0.00004
KDENB vs LR	0.00005
KDENB vs RT	0.0010

Table 16 The performances of different classifiers for 34 software releases.

Release	KDENB	NB	SVM	LR	RT
Ant-1.3	0.5789	0.4465	0.4416	0.4487	0.4707
Ant-1.4	0.3421	0.2428	0.3880	0.2148	0.3034
Ant-1.5	0.4841	0.3857	0.3575	0.3428	0.3728
Ant-1.6	0.6982	0.6360	0.6513	0.5945	0.5638
Ant-1.7	0.6259	0.5520	0.5460	0.4926	0.4643
Camel-1.0	0.3374	0.3258	0.1692	0.4000	0.2738
Camel-1.2	0.4469	0.2437	0.2213	0.1734	0.4363
Camel-1.4	0.4147	0.3403	0.3989	0.1318	0.2961
Camel-1.6	0.3539	0.3165	0.0513	0.1657	0.3109
Ivy-1.1	0.8040	0.6982	0.6546	0.7095	0.7107
Ivy-1.4	0.2938	0.3449	0.2099	0.5111	0.3653
Ivy-2.0	0.5178	0.4540	0.3686	0.3041	0.2675
Jedit-3.2	0.7107	0.6312	0.6436	0.6373	0.5455
Jedit-4.0	0.6282	0.5793	0.5433	0.5081	0.4938
Lucene-2.0	0.7494	0.6666	0.6355	0.6364	0.5288
Lucene-2.2	0.7830	0.6945	0.7249	0.7090	0.6134
Lucene-2.4	0.7907	0.7063	0.5757	0.7239	0.6363
Poi-1.5	0.8500	0.7456	0.5261	0.7844	0.7516
Poi-2.0	0.4500	0.3451	0.2996	0.2866	0.3363
Poi-2.5	0.8814	0.7632	0.6602	0.8370	0.8324
Poi-3.0	0.8978	0.8154	0.5950	0.8150	0.8082
Synapse-1.0	0.4925	0.4945	0.4239	0.4556	0.3557
Synapse-1.1	0.6099	0.5424	0.5754	0.5161	0.5289
Synapse-1.2	0.6555	0.6013	0.5821	0.5430	0.5479
Velocity-1.4	0.9364	0.8499	0.8356	0.9045	0.8672
Velocity-1.5	0.8567	0.8077	0.6297	0.8030	0.7850
Velocity-1.6	0.5811	0.4862	0.5025	0.4803	0.4853
Xalan-2.4	0.4944	0.4423	0.3905	0.2814	0.3248
Xalan-2.5	0.6951	0.6064	0.5833	0.5811	0.6213
Xalan-2.6	0.7453	0.7163	0.2275	0.6941	0.6852
Xerces-init	0.7413	0.6706	0.6985	0.7015	0.7207
Xerces-1.2	0.3289	0.1984	0.2740	0.2012	0.3290
Xerces-1.3	0.5504	0.5038	0.4775	0.2360	0.4123
Xerces-1.4	0.9671	0.8764	0.4021	0.9441	0.9460
Average (<i>F-measure</i>)	0.6263	0.5509	0.4784	0.5226	0.5292

6. Discussion and Conclusion

In this paper, in order to solve the problem of non-normal distribution of the continuous or numeric attributes, kernel density estimation is used to evolve the Naive Bayes classifier, which is called KDENB classifier. The reported results showed that the KDENB classifier can be used to estimate the probability distributions appropriately, when the continuous or numeric attributes are non-normally distributed. The contributions of this study can be summarized as follows:

(1) An improved Naive Bayes method based on kernel density estimation (KDENB) is proposed. The KDENB can improve the performance of NB though estimating the probability distribution appropriately.

(2) We validated the performance of the KDENB based on four common kernels (i.e. Gaussian, Epanechnikov, Biweight and Triweight), and found that there is no significant difference among the four common kernels. Therefore, the most commonly used kernel Gaussian kernel is used in

KDENB. We also make comparison among the bandwidth h which is used in KDENB and other three kinds of bandwidth, and found that the bandwidth h used in KDENB obtained the best performance.

(3) We also validated the performance of the KDENB by making comparison with other 4 classifiers, and the KDENB was proved to be best classifier in the most software releases.

In conclusion, the reported results show that our KDENB classifier for SDP is practical and feasible. We expect our findings will promote the development of software testing. In our future work, we attempt to optimize the KDENB in two aspects. First, we plan to evolve the kernel function and its parameters to improve the generality. Second, we will explore more efficient approaches to simplify the metric attribute sets to further enhance the performance of KDENB.

References

- [1] K. Dejaeger, T. Verbraken, and B. Baesens, "Toward Comprehensive Software Fault Prediction Models Using Bayesian Network Classifiers," *IEEE Trans. Softw. Eng.*, vol.39, no.2, pp.237–257, 2013.
- [2] X.Y. Jing, S. Ying, Z.W. Zhang, S.S. Wu, and J. Liu, "Dictionary learning based software defect prediction," *Proc. 36th International Conference on Software Engineering*, pp.414–423, 2014.
- [3] W. Liu, S. Liu, Q. Gu, J. Chen, and X. Chen, "Empirical studies of a two-stage data preprocessing approach for software fault prediction," *IEEE Trans. Rel.*, vol.65, no.1, pp.38–53, 2016.
- [4] X. Yang, K. Tang, and X. Yao, "A learning-to-rank approach to software defect prediction," *IEEE Trans. Rel.*, vol.64, no.1, pp.234–246, 2015.
- [5] X. Xia, D. Lo, S.J. Pan, N. Nagappan, and X. Wang, "HYDRA: Massively Compositional Model for Cross-Project Defect Prediction," *IEEE Trans. Softw. Eng.*, vol.42, no.10, pp.977–998, 2016.
- [6] Ö.F. Arar and K. Ayan, "A feature dependent naive Bayes approach and its application to the software defect prediction problem," *Applied Soft Computing*, vol.59, pp.197–209, 2017.
- [7] 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.
- [8] B. Turhan and A. Bener, "Analysis of naive bayes' assumptions on software fault data: An empirical study," *Data & Knowledge Engineering*, vol.68, no.2, pp.278–290, 2009.
- [9] C. Jin and J.-A. Liu, "Applications of Support Vector Machine and Unsupervised Learning for Predicting Maintainability Using Object-Oriented Metrics," *Second International Conference on Multimedia and Information Technology*, pp.24–27, 2010.
- [10] H.M. Olague, L.H. Etzkorn, S. Gholston, and S. Quattlebaum, "Empirical validation of three software metrics suites to predict fault-proneness of object-oriented classes developed using highly iterative or agile software development processes," *IEEE Trans. Softw. Eng.*, vol.33, no.6, pp.402–419, 2007.
- [11] G. Jagannathan, K. Pillaipakkam, and R.N. Wright, "A Practical Differentially Private Random Decision Tree Classifier," *IEEE International Conference on Data Mining Workshops*, pp.114–121, 2009.
- [12] Z. He, F. Shu, Y. Yang, M. Li, and Q. Wang, "An investigation on the feasibility of cross-project defect prediction," *Automated Software Engineering*, vol.19, no.2, pp.167–199, 2012.
- [13] R. Malhotra, "A systematic review of machine learning techniques for software fault prediction," *Applied Soft Computing Journal*, vol.27, no.c, pp.504–518, 2015.

- [14] N.M. Razali and Y.B. Wah, "Power comparisons of Shapiro-Wilk, Kolmogorov-Smirnov, Lilliefors and Anderson-Darling tests," *Journal of Statistical Modeling and Analytics*, vol.2, no.1, pp.21–33, 2011.
- [15] E. Parzen, "On estimation of a probability density function and mode," *The Annals of Mathematical Statistics*, vol.33, no.3, pp.1065–1076, 1962.
- [16] D.J. Hand and K. Yu, "Idiot's Bayes: Not So Stupid after All?," *International Statistical Review*, vol.69, no.3, pp.385–398, 2001.
- [17] N.A. Zaidi, J. Cerquides, M.J. Carman, and G.I. Webb, "Alleviating naive Bayes attribute independence assumption by attribute weighting," *Journal of Machine Learning Research*, vol.14, no.1, pp.1947–1988, 2013.
- [18] C. Catal and B. Diri, "Investigating the effect of dataset size, metrics sets, and feature selection techniques on software," *Information Sciences*, vol.179, no.8, pp.1040–1058, 2009.
- [19] C. Catal, "Software fault prediction: A literature review and current trends," *Expert Syst. Appl.*, vol.38, no.4, pp.4626–4636, 2011.
- [20] L.H. Witten, E. Frank, and M.A. Hell, *Data Mining: Practical Machine Learning Tools and Techniques*, 3rd edition, ACM Sigsoft Software Engineering Notes, pp.90–99, Morgan Kaufmann, Burlington, 2011.
- [21] B.W. Silverman, *Density Estimation for Statistics and Data Analysis*, vol.26, CRC Press, London, 1986.
- [22] M.P. Wand and M.C. Jones, *Kernel Smoothing*, CRC Press, London, 1994.
- [23] B.E. Hansen, *Lecture notes on nonparametrics*, University of Wisconsin-Madison, WI, USA, <http://www.ssc.wisc.edu/~bhansen/718/NonParametrics1.pdf>, 2009.
- [24] A. Schindler, "Bandwidth selection in nonparametric kernel estimation," PhD Thesis, Göttingen, Georg-August Universität, Diss, 2011.
- [25] Analytical Methods Committee, "Robust statistics – How not to reject outliers. Part 1: Basic concepts," *Analyst*, vol.114, no.12, pp.1693–1697, 1989.
- [26] Q. Song, Z. Jia, M. Shepperd, S. Ying, and J. Liu, "A general software defect-proneness prediction framework," *IEEE Trans. Softw. Eng.*, vol.37, no.3, pp.356–370, 2011.
- [27] M.A. Hall, "Correlation-based Feature Selection for Discrete and Numeric Class Machine Learning," *Proc. 17th International Conference on Machine Learning*, pp.359–366, 2000.
- [28] I. Rish, "An empirical study of the naive Bayes classifier," *Journal of Universal Computer Science*, vol.1, no.2, 2001.
- [29] C.M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*, Springer-Verlag New York, 2006.

Song Huang is a professor of software engineering at Software Testing and Evaluation Center of Army Engineering University of PLA. His current research interests include software testing, quality assurance, empirical software engineering. He is a member of CCF and ACM.

Xuewei Lv is now a Doctor candidate in Army Engineering University of PLA. His current research interests include software testing, defects prediction and fuzzy information fusion.

Yaning Wu received her M.A.SC. in Information Science from Army Engineering University of PLA, Nanjing, China. Her current research interests include software testing, defects prediction, linguistic preference modelling and fuzzy information fusion.

Yuntian Feng received his M.A.SC. in Information Science from Army Engineering University of PLA, Nanjing, China. His current research interests include deep learning and natural language processing.

Haijin Ji received his M.A.SC. in Information Science from Jiangnan University, Wuxi, China. He is now a Doctor candidate in Army Engineering University of PLA. His current research interests include software testing, defects prediction and fuzzy information fusion.