Available online at www.sciencedirect.com
ScienceDirect
The Journal of Systems and Software 81 (2008) 649–660
The Journal of Systems and Software
www.elsevier.com/locate/jss

Predicting defect-prone software modules using support vector machines

Karim O. Elish, Mahmoud O. Elish *

Information and Computer Science Department, King Fahd University of Petroleum and Minerals, P.O. Box 1082, Dhahran 31261, Saudi Arabia

Received 26 February 2007; received in revised form 28 May 2007; accepted 27 July 2007
Available online 5 October 2007

Abstract

Effective prediction of defect-prone software modules can enable software developers to focus quality assurance activities and allocate effort and resources more efficiently. Support vector machines (SVM) have been successfully applied for solving both classification and regression problems in many applications. This paper evaluates the capability of SVM in predicting defect-prone software modules and compares its prediction performance against eight statistical and machine learning models in the context of four NASA datasets. The results indicate that the prediction performance of SVM is generally better than, or at least, is competitive against the compared models.

Keywords: Software metrics; Defect-prone modules; Support vector machines; Predictive models

1. Introduction

Studies have shown that the majority of defects are often found in only a few software modules (Fenton and Ohlsson, 2000; Koru and Tian, 2003). Such defective software modules may cause software failures, increase development and maintenance costs, and decrease customer satisfaction (Koru and Liu, 2005). Accordingly, effective prediction of defect-prone software modules can enable software developers to focus quality assurance activities and allocate effort and resources more efficiently. This in turn can lead to a substantial improvement in software quality (Koru and Tian, 2003).

Identification of defect-prone software modules is commonly achieved through binary prediction models that classify a module into either defective or not-defective category. These prediction models almost always utilize static product metrics, which have been associated with defects, as independent variables (Emam et al., 2001). Recently, support vector machines (SVM) have been introduced as an effective model in both machine learning and data mining communities for solving both classification and regression problems (Gun, 1998). It is therefore motivating to investigate the capability of SVM in software quality prediction.

The objective of this paper is to evaluate the capability of SVM in predicting defect-prone software modules (functions in procedural software and methods in object-oriented software) and compare its prediction performance against eight well-known statistical and machine learning models in the context of four NASA datasets. The compared models are two statistical classifiers techniques: (i) Logistic Regression (LR) and (ii) *K*-Nearest Neighbor (KNN); two neural networks techniques: (i) Multi-layer Perceptrons (MLP) and (ii) Radial Basis Function (RBF); two Bayesian techniques: (i) Bayesian Belief Networks (BBN) and (ii) Naïve Bayes (NB); and two tree-structured classifiers techniques: (i) Random Forests (RF) and (ii) Decision Trees (DT). For more details on these techniques see (Han and Kamber, 2001; Hosmer and Lemeshow, 2000; Duda et al., 2001; Webb, 2002; Breiman, 2001).

The rest of this paper is organized as follows. Section 2 reviews related work. Section 3 provides an overview of

* Corresponding author. Tel.: +966 3 860 1150; fax: +966 3 860 2174.
E-mail addresses: kelish@kfupm.edu.sa (K.O. Elish), elish@kfupm.edu.sa (M.O. Elish).

0164-1212/$ - see front matter
doi:10.1016/j.jss.2007.07.040

SVM. Section 4 discusses the conducted empirical evaluation and its results. Section 5 concludes the paper and outlines directions for future work.

2. Related work

A wide range of statistical and machine learning models have been developed and applied to predict defects in software. Basili et al. (1996) investigated the impact of the suite of object-oriented design metrics introduced by (Chidamber and Kemerer, 1994) on the prediction of fault-prone classes using logistic regression. Guo et al. (2004) proposed random forest technique to predict fault-proneness of software system. They applied this technique on NASA software defect datasets. The proposed methodology was compared with some machine learning and statistical methods. They found that the prediction accuracy of random forest is generally higher than other methods. Khoshgoftaar et al. (1997) investigated the use of the neural network as a model for predicting software quality. They used large telecommunication system to classify modules as fault-prone or not fault-prone. They compared the neural network model with a non-parametric discriminant model, and found that the neural network model had better predictive accuracy. Khoshgoftaar et al. (2002) applied regression trees with classification rule to classify fault-prone software modules using a very large telecommunications system as a case study. Fenton et al. (2002) proposed Bayesian Belief Networks for software defect prediction. However, the limitations of Bayesian Belief Networks have been recognized (Weaver, 2003; Ma et al., 2006).

Several other techniques have been developed and applied to software quality prediction. These techniques include: discriminant analysis (Munson and Khoshgoftaar, 1992; Khoshgoftaar et al., 1996), the discriminative power techniques (Schneidewind, 1992), optimized set reduction (Briand et al., 1993), genetic algorithms (Azar et al., 2002), classification trees (Selby and Porter, 1988), case-based reasoning (Emam et al., 2001; Mair et al., 2000; Shepperd and Kadoda, 2001), and Dempster–Shafer Belief Networks (Guo et al., 2003).

Recently, SVM has been applied successfully in many applications, for example in the field of optical character recognition (Burges, 1998; Cristianini and Shawe-Taylor, 2000), text categorization (Dumais, 1998), face detection in images (Osuna et al., 1997; Cristianini and Shawe-Taylor, 2000), speaker identification (Schmidt and Gish, 1996), spam categorization (Drucker et al., 1999), intrusion detection (Chen et al., 2005), cheminformatics and bioinformatics (Cai et al., 2002; Burbidge et al., 2001; Morris et al., 2001; Lin et al., 2003; Bao and Sun, 2002), and financial time series forecasting (Tay and Cao, 2001).

3. An overview of support vector machines

Support vector machines (SVM) are kernel based learning algorithm introduced by Vapnik (Vapnik, 1995) using the Structural Risk Minimization (SRM) principle which minimizes the generalization error, i.e., true error on unseen examples. The basic SVM classifier deals with two-class pattern recognition problems, in which the data are separated by the optimal hyperplane defined by a number of support vectors (Cristianini and Shawe-Taylor, 2000). Support vectors are a subset of training data used to define the boundary between the two classes. The main characteristics of SVM are (Abe, 2005; Cortes and Vapnik, 1995):

- It can be generalized well even in high-dimensional spaces under small training sample conditions. This means that the ability of SVM to learn can be independent of the feature space dimensionality.
- It gives a global optimum solution, since SVM is formulated as a quadratic programming problem.
- It is robust to outliers. It prevents the effect of outliers by using the margin parameter C to control the misclassification error.
- It can model nonlinear functional relationships that are difficult to model with other techniques.

These characteristics make SVM a good candidate model to apply in predicting defect-prone modules as such conditions are typically encountered. In the following subsections, we briefly discuss the binary SVM classifier for both linear and nonlinear separable data.

3.1. Two-class: linear support vector machines

3.1.1. The separable case

The set of vectors is said to be optimally separated by the hyperplane if it is separated without error and the distance (margin) between the closest vectors to the hyperplane is maximal (Abe, 2005). Fig. 1a shows the linear separation of two classes by SVM in two-dimensional space. Circles represent (class A) and squares represent (class B). The SVM attempts to place a linear boundary (solid line) between the two different classes and draw this line in such a way that the margin space between dotted lines is maximized.

In a binary category classification problem, we have to estimate a function $f : \Re^p \mapsto \{\pm 1\}$ using training data. Let us represent the class A with $x \in A$, $y = 1$ and class B with $x \in B, y = -1; (\mathbf{x}_i, y_i) \in \Re^p \times \{\pm 1\}$. If the training data are linearly separable by hyperplane in the p dimensional space then there exists a pair $(\mathbf{w}, b)$ such that:

$$\begin{aligned} \mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b &\geqslant +1 \quad \text{for all } \mathbf{x}_i \in A \\ \mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b &\leqslant -1 \quad \text{for all } \mathbf{x}_i \in B \end{aligned} \tag{1}$$

for all $i = 1, 2, \ldots, n$; where $\mathbf{w}$ is a p-dimensional vector orthogonal to the hyperplane and b is the bias term. The inequality constraints (1) can be combined to give:

$$y_i(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b) \geqslant 1 \quad \text{for all } \mathbf{x}_i \in A \cup B \tag{2}$$

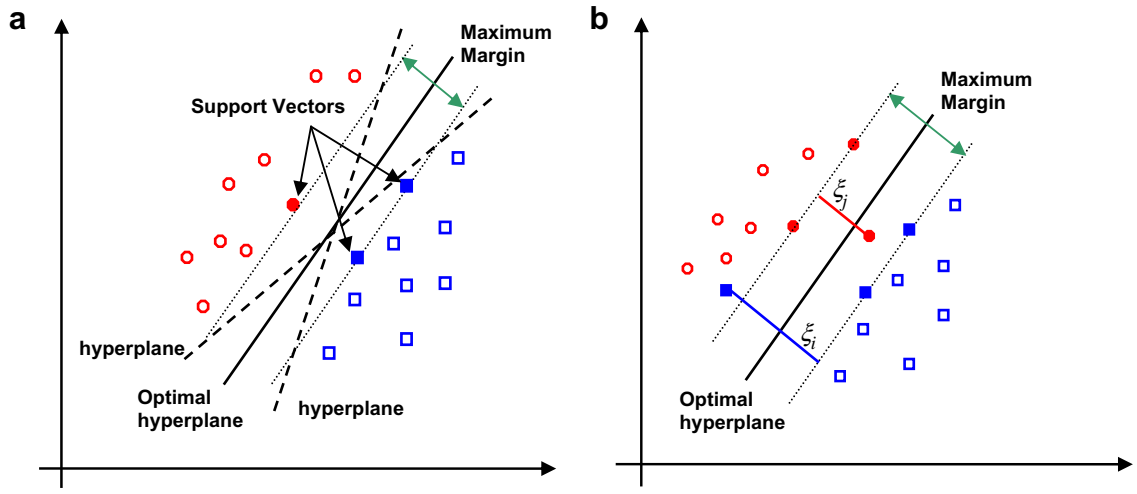

Fig. 1. (a) Optimal separating hyperplane for separable case in 2D space, (b) linear separating hyperplane for non-separable case in 2D space.

The maximal margin classifier optimizes this by separating the data with the maximal margin hyperplane. The learning problem is reformulated as: minimize $\|\mathbf{w}\|^2 = \mathbf{w}^{\mathrm{T}}\mathbf{w}$ subject to the constraints of linear separability (2). The optimization is now a quadratic programming (QP) problem:

$$\underset{\mathbf{w},b}{\text{Minimize}} \quad \left\{\frac{1}{2}\|\mathbf{w}\|^2\right\} \tag{3}$$

$$\text{Subject to} \quad y_i(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b) \geqslant 1, \quad i = 1, \ldots, n \tag{4}$$

where $(\mathbf{x}_i, y_i)$ is the training set, and n is the number of training sets. By using standard Lagrangian duality techniques, and after further simplification (See Vapnik (1995) or Burges (1998) for derivation details.), the following is the dual form of the optimization problem:

$$\begin{aligned} F(\lambda) &= \sum_{i=1}^{n}\lambda_i - \frac{1}{2}\|\mathbf{w}\|^2 \\ &= \sum_{i=1}^{n}\lambda_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\lambda_i\lambda_j y_i y_j \mathbf{x}_i^{\mathrm{T}}\mathbf{x}_j \end{aligned} \tag{5}$$

where $\lambda = (\lambda_1, \ldots, \lambda_n)^{\mathrm{T}}$ are the Lagrange multipliers and are non-zero only for the support vectors. The formulated support vector machine is called the hard-margin support vector machine (Abe, 2005). This function has to be maximized with respect to $\lambda_i \geqslant 0$. Therefore, we optimize the following problem:

$$\text{Maximize} \quad \left\{\sum_{i=1}^{n}\lambda_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\lambda_i\lambda_j y_i y_j \mathbf{x}_i^{\mathrm{T}}\mathbf{x}_j\right\} \tag{6}$$

$$\text{Subject to} \quad \sum_{i=1}^{n}\lambda_i y_i = 0; \quad \text{and} \quad \lambda_i \geqslant 0; \quad i = 1, 2, \ldots, n \tag{7}$$

Once the solution has been found in the form of a vector $\lambda = (\lambda_1, \ldots, \lambda_n)^{\mathrm{T}}$, the optimal separating hyperplane is found and the decision function is obtained as follows:

$$f(\mathbf{x}) = \operatorname{sign}\sum_{i=1}^{n}\lambda_i y_i(\mathbf{x}_i^{\mathrm{T}}\mathbf{x}) + b \tag{8}$$

3.1.2. The non-separable case

Consider the case where the training data are non-separable without error. In this case one may want to separate the training set with a minimal number of errors. Fig. 1b shows the non-separable case of two classes in two-dimensional space. Therefore, the minimization problem needs to be modified to allow misclassified data points. This can be done by introducing positive slack variables $\xi_i \geqslant 0$ in the constraints to measure how much the margin constraints are violated (Cortes and Vapnik, 1995):

$$\underset{\mathbf{w},\mathrm{b},\xi}{\text{Minimize}} \quad \left\{\frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\xi_i\right\} \tag{9}$$

$$\text{Subject to} \quad y_i(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b) \geqslant 1 - \xi_i; \quad \text{for } i = 1, \ldots, n \tag{10}$$

where C is the regularizing (margin) parameter that determines the trade-off between the maximization of the margin and minimization of the classification error (Gun, 1998; Cristianini and Shawe-Taylor, 2000). The solution to this minimization problem is similar to the separable case except for a modification of the bounds of the Lagrange multipliers. Therefore, Eq. (7) is changed to:

$$\sum_{i=1}^{n}\lambda_i y_i = 0; \quad \text{and} \quad 0 \leqslant \lambda_i \leqslant C; \quad i = 1, 2, \ldots, n \tag{11}$$

Thus the only difference from the separable case is that the λ_i now has an upper bound of C. The obtained support vector machine in this case is called the soft-margin support vector machine (Abe, 2005).

3.2. Two-class: nonlinear support vector machines

In case that SVM cannot linearly separate two classes, SVM extends its applicability to solve this problem by

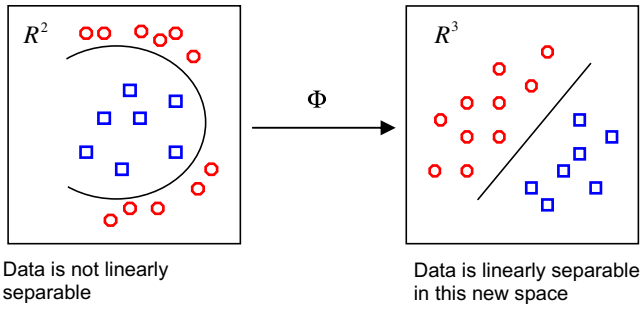

Fig. 2. Mapping input data into higher dimensional space.

mapping input data into higher dimensional feature spaces using a nonlinear mapping ϕ (Burges, 1998), such that $\mathbf{x} \mapsto \phi(\mathbf{x})$, where $\phi : \Re^n \to \Re^m$ is the feature map. It is possible to create a hyperplane that allows linear separation in high-dimensional space (Burges, 1998). This corresponds to a curved surface in the lower dimensional space. Fig. 2 shows the transformation from lower to higher dimensional feature spaces using ϕ. This transformation can be done using a kernel function. Therefore, the kernel function is an important parameter in SVM. The kernel function $K(\mathbf{x}_i, \mathbf{x}_j)$ is defined as follows (Abe, 2005):

$$K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^{\mathrm{T}} \phi(\mathbf{x}_j) \tag{12}$$

Therefore, the optimization problem of Eq. (6) becomes:

$$\text{Maximize} \quad \left\{ \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j k(\mathbf{x}_i, \mathbf{x}_j) \right\} \tag{13}$$

$$\text{Subject to} \quad \sum_{i=1}^{n} \lambda_i y_i = 0; \quad \text{and} \quad \lambda_i \geqslant 0; \quad i = 1, 2, \ldots, n \tag{14}$$

where $K(\mathbf{x}_i, \mathbf{x}_j)$ is the kernel function performing the nonlinear mapping into feature space. Once the solution has been found, the decision can be constructed as:

$$f(\mathbf{x}) = \text{sign} \sum_{i=1}^{n} \lambda_i y_i k(\mathbf{x}_i, \mathbf{x}) + b \tag{15}$$

The following are the most common kernel functions in literature (Abe, 2005; Gun, 1998; Burges, 1998):

1. Linear: $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^{\mathrm{T}} \mathbf{x}_j$.
2. Polynomial: $K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^{\mathrm{T}} \mathbf{x}_j + 1)^d$.
3. Gaussian (RBF): $K(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2)$.
4. Sigmoid (MLP): $K(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\gamma(\mathbf{x}_i^{\mathrm{T}} \mathbf{x}_j) - r)$.

Gaussian radial basis function (RBF) kernel was used in this study because it yields better prediction performance (Smola, 1998).

4. Empirical evaluation

This section discusses the conducted empirical study that evaluates the capability of SVM in predicting defect-prone software modules. We used the open source WEKA[1] machine learning toolkit to conduct this study.

4.1. Goal

Using GQM template (Basili and Rombach, 1988) for goal definition, the goal of this empirical study is defined as follows: *Evaluate* SVM *for the purpose of* predicting defect-prone software modules *with respect to* its prediction performance against the eight compared models (LR, KNN, MLP, RBF, BBN, NB, RF, and DT) *from the point of view of* researchers and practitioners *in the context of* four NASA datasets.

4.2. Datasets

The datasets used in this study are four mission critical NASA software projects, which are publicly accessible from the repository of the NASA IV&V Facility Metrics Data Program.[2] Two datasets (CM1 and PC1) are from software projects written in a procedural language (C) where a module in this case is a function. The other two datasets (KC1 and KC3) are from projects written in object-oriented languages (C++ and Java) where a module in this case is a method. Each dataset contains 21 software metrics (independent variables) at the module-level and the associated dependent Boolean variable: *Defective* (whether or not the module has any defects). Table 1 summarizes some main characteristics of these datasets.

4.3. Independent variables

The independent variables are 21 static metrics at the module-level including McCabe (McCabe, 1976; McCabe and Butler, 1989), Halstead (basic and derived) (Halstead, 1977), Line Count, and Branch Count. Table 2 lists these metrics.

Since some independent variables might be highly correlated, a correlation-based feature selection technique (CFS) (Hall, 2000) was applied to down-select the best predictors out of the 21 independent variables in the datasets. This involves searching through all possible combinations of variables in the dataset to find which subset of variables works best for prediction. CFS evaluates each subset of variables by considering the individual predictive ability of each variable along with the degree of redundancy between them. Table 3 provides the resulted best subset of independent variables in each dataset.

4.4. Dependent variable

This study focuses on predicting whether a module is defective or not, rather than how many defects it contains.

[1] WEKA (Waikato Environment for Knowledge Analysis). http://www.cs.waikato.ac.nz/~ml/weka/.

[2] http://mdp.ivv.nasa.gov/index.html.

Table 1
Characteristics of datasets

	Project	Language	# of Modules	% of Defective modules	Description
Procedural	CM1	C	496	9.7	NASA spacecraft instrument
	PC1	C	1107	6.9	Flight software for an earth orbiting satellite
Object oriented	KC1	C++	2107	15.4	Storage management for receiving and processing ground data
	KC3	Java	458	6.3	Collection, processing and delivery of satellite metadata

Table 2
Module-level metrics (independent variables)

Metric	Type	Definition
V (g)	McCabe	Cyclomatic Complexity
EV (g)	McCabe	Essential Complexity
IV (g)	McCabe	Design Complexity
LOC	McCabe	Total lines of code
N	Derived Halstead	Total number of operands and operators
V	Derived Halstead	Volume on minimal implementation
L	Derived Halstead	Program Length = V/N
D	Derived Halstead	Difficulty = 1/L
I	Derived Halstead	Intelligent count
E	Derived Halstead	Effort to write program = V/L
B	Derived Halstead	Effort Estimate
T	Derived Halstead	Time to write program = E/18 s
LOCode	Line Count	Number of lines of statement
LOComment	Line Count	Number of lines of comment
LOBlank	Line Count	Number of lines of blank
LOCodeAndComment	Line Count	Number of lines of code and comment
UniqOp	Basic Halstead	Number of Unique operators
UniqOpnd	Basic Halstead	Number of Unique operands
TotalOp	Basic Halstead	Total number of operators
TotalOpnd	Basic Halstead	Total number of operands
BranchCount	Branch	Total number of branch count

Table 3
Best subset of independent variables in each dataset

Dataset	Best subset of independent variables
CM1	LOC, IV(g), D, I, LOComment, LOBlank,
PC1	V(g), I, LOCodeAndComment, LOComment, LOBlank, UniqOp
KC1	V, D, V(g), LOCode, LOComment, LOBlank, UniqOpnd
KC3	N, EV(g), LOCodeAndComment

Accordingly, the dependent variable is a Boolean variable: *Defective* (whether or not the module has any defects). Predicting the number of defects is a possible future work if such data is available.

4.5. Prediction performance measures

The performance of prediction models for two-class problem (e.g. defective or not defective) is typically evaluated using a confusion matrix, which is shown in Table 4. In this study, we used the commonly used prediction performance measures (Witten and Frank, 2005): accuracy, precision, recall and *F*-measure to evaluate and compare prediction models quantitatively. These measures are derived from the confusion matrix.

4.5.1. Accuracy

Accuracy is also known as correct classification rate. It is defined as the ratio of the number of modules correctly predicted to the total number of modules. It is calculated as follows:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

4.5.2. Precision

Precision is also known as correctness. It is defined as the ratio of the number of modules correctly predicted as defective to the total number of modules predicted as defective. It is calculated as follows:

$$Precision = \frac{TP}{TP + FP}$$

4.5.3. Recall

Recall is also known as defect detection rate. It is defined as the ratio of the number of modules correctly predicted as defective to the total number of modules that are actually defective. It is calculated as follows:

$$Recall = \frac{TP}{TP + FN}$$

Both precision and recall are important performance measures. The higher the precision, the less effort wasted in testing and inspection; and the higher the recall, the fewer defective modules go undetected (Koru and Liu,

Table 4
A confusion matrix

		Predicted	
		Not defective	Defective
Actual	Not defective	TN = True Negative	FP = False Positive
	Defective	FN = False Negative	TP = True Positive

2005). However, there is a trade-off between precision and recall (Witten and Frank, 2005; Koru and Liu, 2005). For example, if a model predicts only one module as defective and this module is actually defective, the model's precision will be one. However, the model's recall will be low if there are other defective modules. As another example, if a model predicts all modules as defective, its recall will be one but its precision will be low. Therefore, *F*-measure is needed which combines recall and precision in a single efficiency measure (Witten and Frank, 2005).

4.5.4. F-measure

F-measure considers both precision and recall equally important by taking their harmonic mean (Witten and Frank, 2005). It is calculated as follows:

$$F\text{-}measure = \frac{2 * Precision * Recall}{Precision + Recall}$$

4.6. Parameters initialization

The parameters for each of the investigated prediction model were initialized mostly with the default settings of the WEKA toolkit as follows:

- *Support vector machines* (*SVM*): the regularization parameter (*C*) was set at 1; the kernel function used was Gaussian (RBF); and the bandwidth (γ) of the kernel function was set at 0.5.
- *Logistic regression* (*LR*): the method of optimization was the maximization of log-likelihood.
- *K-nearest neighbor* (*KNN*): the number of observations (*k*) in the set of closest neighbor was set at 3.
- *Multi-layer perceptrons* (*MLP*): a three layered, fully connected, feedforward multi-Layer perceptron (MLP) was used as network architecture. MLP was trained using backpropagation algorithm. The number of hidden nodes varied based on the size and nature of the datasets. Therefore, we used MLP with 4 hidden nodes for CM1 and PC1 datasets; 5 hidden nodes for KC1 dataset; and 3 hidden nodes for KC3 dataset. All nodes in the network used the sigmoid transfer function. The learning rate was initially 0.3 and the momentum term was set at 0.2. The algorithm was halted when there had been no significant reduction in training error for 500 epochs with a tolerance value to convergence of 0.01.
- *Radial basis function* (*RBF*): *k*-means clustering algorithm was used to determine the RBF center *c* and width σ. The value of *k* was set at 2.
- *Bayesian belief network* (*BBN*): the *SimpleEstimator* algorithm was used for finding the conditional probability tables and the hill climbing algorithm was used for searching the network.
- *Naïve Bayes* (*NB*): it does not require any parameters to pass.
- *Random forest* (*RF*): the number of trees to be generated was set at 10; the number of input variables randomly selected at each node was set at 2; and each tree grown to the largest extent possible, i.e. the maximum depth of the trees is unlimited.
- *Decision tree* (*DT*): it uses the well-known C4.5 algorithm to generate decision tree. The confidence factor used for pruning was set at 25% and the minimum number of instances per leaf was set at 2.

In addition to the above parameters initialization, a default threshold (cut-off) of 0.5 was used for all models to classify a module as defect-prone if its predicted probability is higher than the threshold.

4.7. Cross-validation

A 10-fold cross-validation (Kohavi, 1995) was used to evaluate the performance of the prediction models. Each dataset was randomly partitioned into 10 bins of equal size. For 10 times, 9 bins were picked to train the models and the remaining bin was used to test them, each time leaving out a different bin. This cross-validation process was run 100 times, using different randomization seed values for cross-validation shuffling in each run to ensure low bias. We then computed the mean and the standard deviation for each performance measure over these 100 different runs. The achieved results by each prediction model are reported in Tables 5–8 for the CM1, PC1, KC1 and KC3 datasets respectively.

4.8. Significance test

We performed the corrected resampled *t*-test (Nadeau and Bengio, 2003) at 0.05 level of significance (95% confidence level) to determine whether or not there is a significant difference between the prediction performance of SVM and the other compared models. The corrected resampled *t*-test is more appropriate than the standard *t*-test in the case of using *x*-fold cross-validation because the standard *t*-test may generate too many significant differences due to dependencies in the estimates (Dietterich, 1998). The results of the corrected resampled *t*-test are reported in the 'Sig?' columns of Tables 5–8. In these columns, *Yes* means that there is a significant performance difference between SVM and the corresponding model, and *No* means that there is no significant difference. In addition, a (+) means that SVM outperforms the corresponding model, and a (−) means that SVM is outperformed.

4.9. Discussion of results

Figs. 3 and 4 plot the mean versus the standard deviation of the accuracy and the *F*-measure that are achieved by each model using CM1 dataset respectively. A good prediction model should appear in the upper left corner of

Table 5
Prediction performance measures: CM1 dataset

Prediction model	Accuracy			Precision			Recall			*F*-measure		
	Mean	StDev	Sig.?	Mean	StDev	Sig.?	Mean	StDev	Sig.?	Mean	StDev	Sig.?
SVM	90.69	1.074		90.66	0.010		100.00	0.000		0.951	0.006	
LR	90.17	1.987	No(+)	91.10	0.013	No(−)	98.78	0.017	Yes(+)	0.948	0.011	No(+)
KNN	83.27	4.154	Yes(+)	91.36	0.020	No(−)	90.02	0.043	Yes(+)	0.906	0.025	Yes(+)
MLP	89.32	2.066	No(+)	90.75	0.012	No(−)	98.20	0.021	Yes(+)	0.943	0.012	Yes(+)
RBF	89.91	1.423	No(+)	90.32	0.008	No(+)	99.49	0.013	No(+)	0.947	0.008	No(+)
BBN	76.83	6.451	Yes(+)	94.17	0.026	Yes(−)	79.30	0.071	Yes(+)	0.859	0.044	Yes(+)
NB	86.74	3.888	Yes(+)	92.49	0.020	Yes(−)	92.90	0.038	Yes(+)	0.926	0.023	Yes(+)
RF	88.62	2.606	Yes(+)	90.93	0.014	No(−)	97.10	0.026	Yes(+)	0.939	0.015	Yes(+)
DT	89.82	1.526	No(+)	90.35	0.009	No(+)	99.34	0.017	No(+)	0.946	0.009	No(+)

Table 6
Prediction performance measures: PC1 dataset

Prediction model	Accuracy			Precision			Recall			*F*-measure		
	Mean	StDev	Sig.?	Mean	StDev	Sig.?	Mean	StDev	Sig.?	Mean	StDev	Sig.?
SVM	93.10	0.968		93.53	0.007		99.47	0.007		0.964	0.005	
LR	93.19	1.088	No(−)	93.77	0.008	No(−)	99.28	0.008	No(+)	0.964	0.006	No(+)
KNN	91.82	2.080	No(+)	95.54	0.012	Yes(−)	95.70	0.020	Yes(+)	0.956	0.011	Yes(+)
MLP	93.59	1.212	No(−)	94.20	0.009	No(−)	99.23	0.009	No(+)	0.966	0.006	No(−)
RBF	92.84	0.948	No(+)	93.40	0.007	No(+)	99.34	0.008	No(+)	0.963	0.005	No(+)
BBN	90.44	4.534	No(+)	94.60	0.011	Yes(−)	95.17	0.048	Yes(+)	0.948	0.027	No(+)
NB	89.21	2.606	Yes(+)	94.95	0.012	Yes(−)	93.40	0.025	Yes(+)	0.941	0.015	Yes(+)
RF	93.66	1.665	No(−)	95.15	0.012	Yes(−)	98.21	0.013	Yes(+)	0.967	0.009	No(−)
DT	93.58	1.499	No(−)	94.59	0.011	Yes(−)	98.77	0.012	No(+)	0.966	0.008	No(−)

Table 7
Prediction performance measures: KC1 dataset

Prediction model	Accuracy			Precision			Recall			*F*-measure		
	Mean	StDev	Sig.?	Mean	StDev	Sig.?	Mean	StDev	Sig.?	Mean	StDev	Sig.?
SVM	84.59	0.714		84.95	0.005		99.40	0.006		0.916	0.004	
LR	85.55	1.394	No(−)	87.08	0.010	Yes(−)	97.38	0.012	Yes(+)	0.919	0.008	No(−)
KNN	83.99	2.144	No(+)	89.58	0.014	Yes(−)	91.75	0.021	Yes(+)	0.906	0.013	Yes(+)
MLP	85.68	1.353	Yes(−)	86.75	0.010	Yes(−)	98.07	0.013	Yes(+)	0.921	0.008	No(−)
RBF	84.81	1.107	No(−)	85.81	0.007	Yes(−)	98.31	0.010	Yes(+)	0.916	0.006	No(+)
BBN	75.99	2.925	Yes(+)	91.98	0.017	Yes(−)	78.47	0.032	Yes(+)	0.847	0.021	Yes(+)
NB	82.86	2.210	Yes(+)	88.59	0.013	Yes(−)	91.53	0.021	Yes(+)	0.900	0.013	Yes(+)
RF	85.20	1.798	No(−)	88.12	0.012	Yes(−)	95.38	0.016	Yes(+)	0.916	0.010	No(+)
DT	84.56	1.580	No(+)	86.79	0.012	Yes(−)	96.46	0.021	Yes(+)	0.914	0.009	No(+)

Table 8
Prediction performance measures: KC3 dataset

Prediction model	Accuracy			Precision			Recall			*F*-measure		
	Mean	StDev	Sig.?	Mean	StDev	Sig.?	Mean	StDev	Sig.?	Mean	StDev	Sig.?
SVM	93.28	1.099		93.65	0.006		99.58	0.009		0.965	0.006	
LR	93.42	1.892	No(−)	94.37	0.013	No(−)	98.89	0.016	No(+)	0.966	0.010	No(−)
KNN	92.50	2.979	No(+)	95.23	0.017	Yes(−)	96.87	0.028	Yes(+)	0.960	0.016	No(+)
MLP	93.50	2.215	No(−)	94.74	0.015	Yes(−)	98.56	0.018	No(+)	0.966	0.012	No(−)
RBF	93.59	1.557	No(−)	94.10	0.011	No(−)	99.41	0.012	No(+)	0.967	0.008	No(−)
BBN	93.21	2.684	No(+)	95.72	0.017	Yes(−)	97.14	0.026	Yes(+)	0.964	0.015	No(+)
NB	92.81	3.134	No(+)	95.89	0.018	Yes(−)	96.50	0.029	Yes(+)	0.962	0.017	No(+)
RF	92.63	3.073	No(+)	95.49	0.018	Yes(−)	96.73	0.029	Yes(+)	0.961	0.017	No(+)
DT	93.11	1.636	No(+)	93.85	0.009	No(−)	99.15	0.019	No(+)	0.964	0.009	No(+)

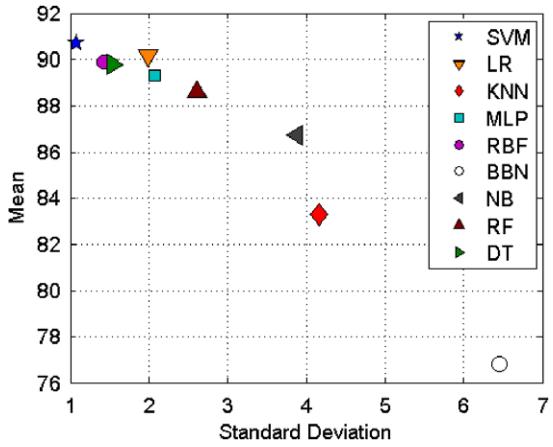

Fig. 3. Mean vs. standard deviation of accuracy: CM1 dataset.

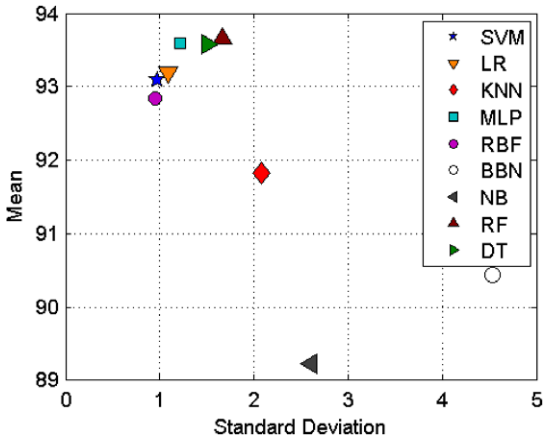

Fig. 5. Mean vs. standard deviation of accuracy: PC1 dataset.

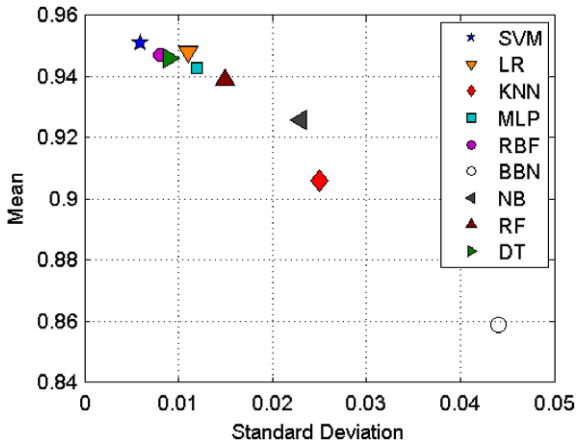

Fig. 4. Mean vs. standard deviation of *F*-measure: CM1 dataset.

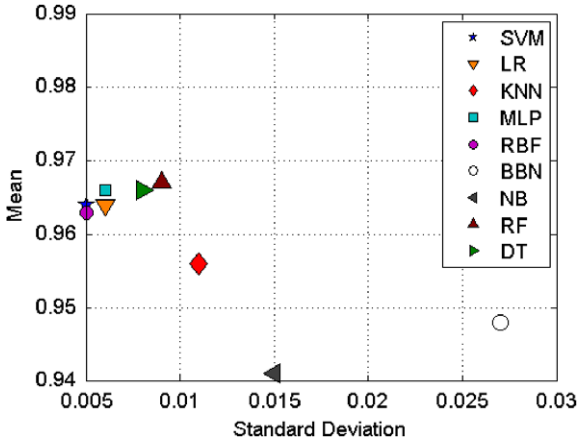

Fig. 6. Mean vs. standard deviation of *F*-measure: PC1 dataset.

these plots. It can be observed that SVM appears in the upper left corner of these plots as it achieves the highest mean with the lowest standard deviation in both accuracy and *F*-measure. Similar plots for PC1, KC1, and KC3 datasets are shown from Figs. 5–10. In all datasets except PC1, SVM achieves the lowest standard deviation in both accuracy and *F*-measure. In PC1 dataset, SVM achieves the lowest standard deviation in *F*-measure and the second lowest standard deviation in accuracy.

Fig. 11 provides a histogram that summarizes the results of the significance test between the prediction performance of SVM and the other eight compared models that are obtained from CM1 dataset. In other words, this histogram provides the numbers of Yes(+), No(+), No(−), and Yes(−) for each performance measure. The numbers of Yes(+) and No(+) are shown above the *x*-axis, whereas the numbers of Yes(−) and No(−) are shown below the *x*-axis. For example, it can be observed that SVM significantly outperforms four out of the eight compared models in accuracy, i.e., there are four Yes(+). It can also be observed that SVM is significantly outperformed by two models in precision, i.e., there are two Yes(−), and so on.

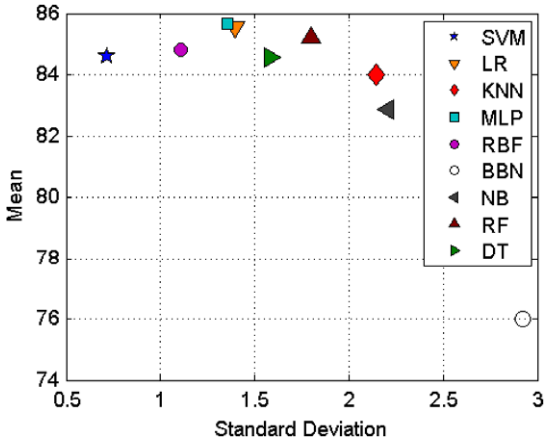

Fig. 7. Mean vs. standard deviation of accuracy: KC1 dataset.

Figs. 12–14 provide similar histograms for PC1, KC1, and KC3 datasets respectively.

4.9.1. Results from CM1 dataset

From Table 5 and Fig. 11, it is observed that SVM outperforms all the compared eight models in accuracy. SVM

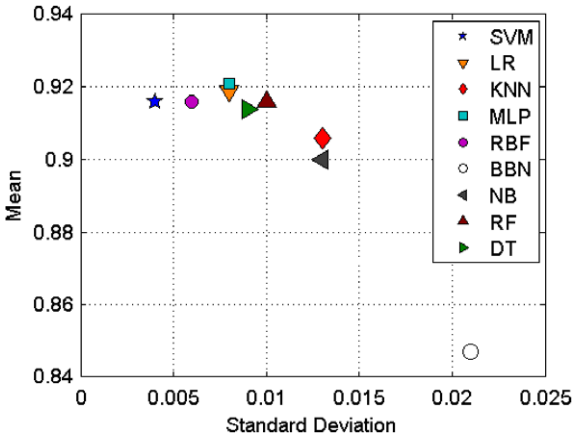

Fig. 8. Mean vs. standard deviation of *F*-measure: KC1 dataset.

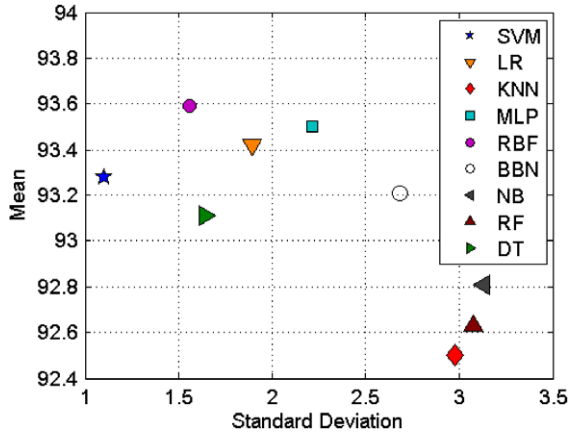

Fig. 9. Mean vs. standard deviation of accuracy: KC3 dataset.

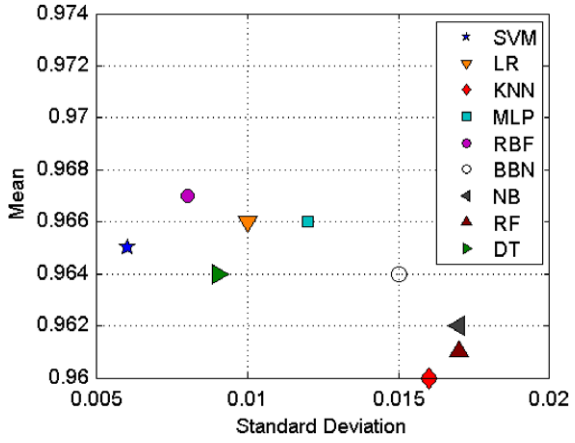

Fig. 10. Mean vs. standard deviation of *F*-measure: KC3 dataset.

is outperformed in precision by six out of the eight compared models, but this is not significant except for two models (BBN, NB). However, SVM achieves significantly higher recall than almost all the compared models. As explained earlier, there is trade-off between precision and

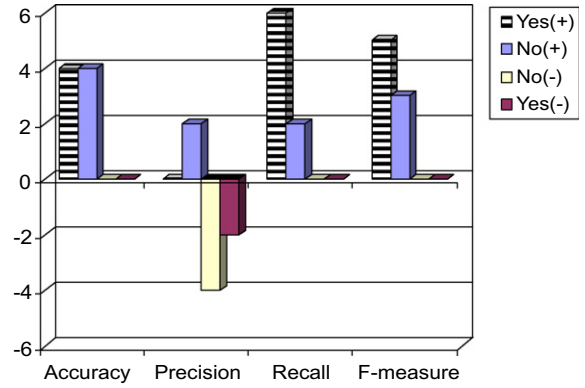

Fig. 11. Significance test results: SVM vs. other models: CM1 dataset.

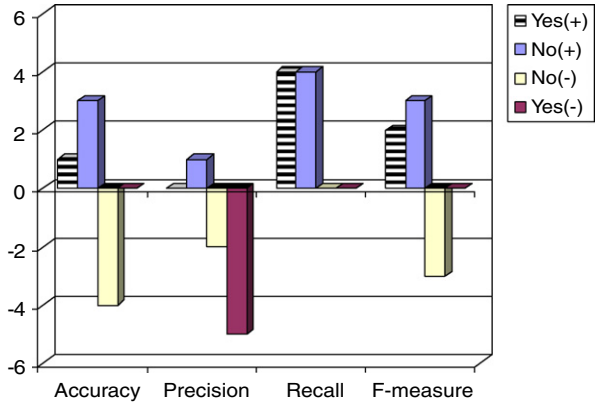

Fig. 12. Significance test results: SVM vs. other models: PC1 dataset.

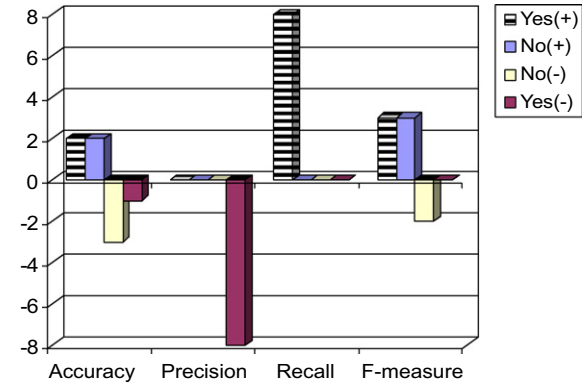

Fig. 13. Significance test results: SVM vs. other models: KC1 dataset.

recall, but the *F*-measure considers their harmonic mean, i.e. takes both of them equally into account. It can be observed that SVM achieves higher *F*-measure than all the compared models. Moreover, its *F*-measure is significantly higher than five out of the eight compared models.

In summary, SVM achieves the highest accuracy (90.69%), perfect recall (100%), and the highest *F*-measure (0.951). However, BBN achieves the highest precision (94.17%), and this is significantly higher than the SVM's precision (90.66%).

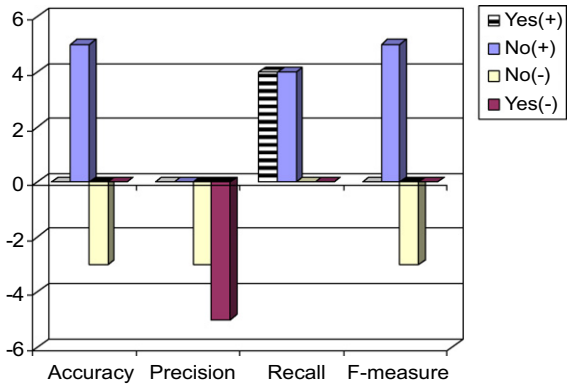

Fig. 14. Significance test results: SVM vs. other models: KC3 dataset.

4.9.2. Results from PC1 dataset

From Table 6 and Fig. 12, it is observed that SVM outperforms four models in accuracy. The other four models, however, do not outperform SVM significantly. SVM is outperformed in precision by almost all the compared models. By contrast, SVM outperforms all the compared models in recall. Considering the *F*-measure, SVM achieves higher *F*-measure than five out of the eight compared models. However, the remaining three models (MLP, RF, DT) do not outperform SVM significantly.

In summary, RF achieves the highest accuracy (93.66%), but this is not significantly higher than the SVM's accuracy (93.10%). KNN achieves the highest precision (95.54%), and this is significantly higher than the SVM's precision (93.53%). On the other hand, SVM achieves the highest recall (99.47%). RF also achieves the highest *F*-measure (0.967), but again this is not significantly higher than the SVM's *F*-measure (0.964).

4.9.3. Results from KC1 dataset

From Table 7 and Fig. 13, it is observed that SVM outperforms some models in accuracy, and is outperformed by some other significantly and non-significantly. SVM is outperformed significantly in precision by all the compared models. On the other hand, SVM significantly outperforms all the compared models in recall. SVM achieves higher *F*-measure than six out of the eight compared models, while the remaining two models (LR, MLP) do not outperform SVM significantly.

In summary, MLP achieves the highest accuracy (85.68%), and this is significantly higher than the SVM's accuracy (84.59%). BBN achieves the highest precision (91.98%), and this is significantly higher than the SVM's precision (84.95%). However, SVM achieves significantly the highest recall (99.40%). MLP achieves the highest *F*-measure (0.921), but this is not significantly higher than the SVM's *F*-measure (0.916).

4.9.4. Results from KC3 dataset

From Table 8 and Fig. 14, it is observed that SVM outperforms five models in accuracy. The other three models, however, do not outperform SVM significantly. SVM is outperformed in precision by all the compared models, whereas it outperforms all the compared models in recall. By looking at the *F*-measure, SVM achieves higher *F*-measure than five out of the eight compared models. There is no significance difference, however, between SVM and all the compared models in *F*-measure.

In summary, RBF achieves the highest accuracy (93.59%), but this is not significantly higher than the SVM's accuracy (93.28%). NB achieves the highest precision (95.89%), and this is significantly higher than the SVM's precision (93.65%). However, SVM achieves the highest recall (99.58%). RBF also achieves the highest *F*-measure (0.967), but again this is not significantly higher than the SVM's *F*-measure (0.965).

4.9.5. Overall observations

When considering the four datasets, we have obtained the following interesting observations regarding the prediction performance of SVM compared to the other eight models:

- SVM achieves higher accuracy than at least four out of the eight compared models in all datasets. Furthermore, no model significantly outperforms SVM in accuracy except only MLP in KC1 dataset.
- SVM's precision is outperformed by all models in two out of the four datasets, i.e., KC1 and KC3. In addition, SVM outperforms two models (RBF and DT) in CM1 dataset and only one model (RBF) in PC1 dataset.
- SVM outperforms all models in recall in all datasets. At least four out of the eight compared models are outperformed significantly by SVM in recall.
- SVM's *F*-measure is not significantly outperformed by any model in all datasets. Moreover, the number of models that are outperformed by SVM are more than the number of models that outperform SVM, i.e., SVM achieves higher *F*-measure than at least five out of the eight compared models in all datasets.

5. Conclusion

This paper has empirically evaluated the capability of SVM in predicting defect-prone software modules and compared its prediction performance against eight statistical and machine learning models in the context of four NASA datasets. The results indicate that the prediction performance of SVM is generally better than, or at least, is competitive against the compared models. In all datasets, the overall accuracy of SVM is within 84.6–93.3%; its precision is within 84.9–93.6%; its recall is within 99.4–100%; and its *F*-measure is within 0.916–0.965. Although the precision of SVM is outperformed by many models, SVM outperforms all models in recall. As explained earlier, there is trade-off between precision and recall, but the *F*-measure considers their harmonic mean, i.e. takes both of them

equally into account. When considering the *F*-measure, SVM achieves higher *F*-measure than at least five out of the eight compared models in all datasets, and is not significantly outperformed by any model.

The results reveal the effectiveness of SVM in predicting defect-prone software modules, and thus suggest that it can be useful and practical addition to the framework of software quality prediction. Moreover, the superior performance of SVM, especially in recall, can have a practical implication in the context of software testing by reducing the risks of defective modules go undetected.

One direction of future work would be conducting additional empirical studies with other datasets to further support the findings of this paper, and to realize the full potential and possible limitation of SVM. Another possible direction of future work would be considering additional independent variables such as coupling and cohesion metrics if information on such metrics is available. Finally, it would be interesting to apply SVM in predicting other software quality attributes in addition to defects.

Acknowledgements

The authors would like to thank the anonymous reviewers for their constructive comments. The authors also acknowledge the support of King Fahd University of Petroleum and Minerals in the development of this work.

References

Abe, S., 2005. Support Vector Machines for Pattern Classification. Springer, USA.

Azar, D., Bouktif, S., K'egl, B., Sahraoui, H., Precup, D., 2002. Combining and adapting software quality predictive models by genetic algorithms. In: Proceedings of the 17th IEEE International Conference on Automated Software Engineering (ASE2002), pp. 285–288.

Bao, L., Sun, Z., 2002. Identifying genes related to drug anticancer mechanisms using support vector machine. FEBS Letters 521, 109–114.

Basili, V., Rombach, H., 1988. The TAME project: towards improvement-oriented software environment. IEEE Transactions on Software Engineering 14 (6), 758–773.

Basili, V., Briand, L., Melo, W., 1996. A validation of object-oriented design metrics as quality indicators. IEEE Transactions on Software Engineering 22 (10), 751–761.

Breiman, L., 2001. Random forests. Machine Learning 45, 5–32.

Briand, L.C., Basili, V., Hetmanski, C., 1993. Developing interpretable models with optimized set reduction for identifying high-risk software components. IEEE Transactions on Software Engineering 19 (11), 1028–1044.

Burbidge, R., Trotter, M., Buxton, B., Holden, S., 2001. Drug design by machine learning: support vector machines for pharmaceutical data analysis. Computers and Chemistry 26, 5–14.

Burges, C., 1998. A tutorial on support vector machines for pattern recognition. Data Mining and Knowledge Discovery 2, 121–167.

Cai, Y.-D., Lin, X.-J., Xu, X.-B., Chou, K.-C., 2002. Prediction of protein structural classes by support vector machines. Computers and Chemistry 26, 293–296.

Chen, W., Hsu, S., Shen, H., 2005. Application of SVM and ANN for intrusion detection. Computers and Operations Research 32, 2617–2634.

Chidamber, S., Kemerer, C., 1994. A metrics suite for object-oriented design. IEEE Transactions on Software Engineering 20 (6), 476–493.

Cortes, C., Vapnik, V., 1995. Support-vector networks. Machine Learning 20, 273–297.

Cristianini, N., Shawe-Taylor, J., 2000. An Introduction to Support Vector Machines and Other Kernel-based Learning Methods. Cambridge University Press, Cambridge, UK.

Dietterich, T., 1998. Approximate statistical tests for comparing supervised classification learning algorithms. Neural Computation 10, 1895–1924.

Drucker, H., Wu, D., Vapnik, V., 1999. Support vector machines for spam categorization. IEEE Transactions on Neural Networks 10 (5), 1048–1054.

Duda, R., Hart, P., Stork, D., 2001. Pattern Classification, second ed. John Wiley & Sons, New York.

Dumais, S., 1998. Using SVMS for text categorization. IEEE Intelligent Systems 13 (4), 21–23.

Emam, K., Benlarbi, S., Goel, N., Rai, S., 2001. Comparing case-based reasoning classifiers for predicting high risk software components. Journal of Systems and Software 55 (3), 301–310.

Fenton, N., Ohlsson, N., 2000. Quantitative analysis of faults and failures in a complex software system. IEEE Transactions on Software Engineering 26 (8), 797–814.

Fenton, N., Neil, M., Krause, P., 2002. Software measurement: uncertainty and causal modeling. IEEE Software 19, 116–122.

Gun, S., 1998. Support vector machines for classification and regression. Technical Report, University of Southampton.

Guo, L., Cukic, B., Singh, H., 2003. Predicting fault prone modules by the Dempster–Shafer belief networks. In: Proceedings of the 18th IEEE International Conference on Automated Software Engineering (ASE 2003), pp. 249–252.

Guo, L., Ma, Y., Cukic, B., Singh, H., 2004. Robust prediction of fault-proneness by random forests. In: Proceedings of the 15th International Symposium on Software Reliability Engineering (ISSRE'04), pp. 417–428.

Hall, M., 2000. Correlation-based feature selection for discrete and numeric class machine learning. In: Proceedings of the 17th International Conference on Machine Learning, pp. 359–366.

Halstead, M., 1977. Elements of Software Science. Elsevier.

Han, J., Kamber, M., 2001. Data Mining: Concepts and Techniques, second ed. Morgan Kauffman.

Hosmer, D., Lemeshow, S., 2000. Applied Logistic Regression, second ed. John Wiley & Sons, New York.

Khoshgoftaar, T., Allen, E., Kalaichelvan, K., Goel, N., 1996. Early quality prediction: a case study in telecommunications. IEEE Software 13 (1), 65–71.

Khoshgoftaar, T., Allen, E., Hudepohl, J., Aud, S., 1997. Application of neural networks to software quality modeling of a very large telecommunications system. IEEE Transactions on Neural Networks 8 (4), 902–909.

Khoshgoftaar, T., Allen, E., Deng, J., 2002. Using regression trees to classify fault-prone software modules. IEEE Transactions on Reliability 51 (4), 455–462.

Kohavi, R., 1995. A study of cross-validation and bootstrap for accuracy estimation and model selection. In: Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI), pp. 1137–1143.

Koru, A., Liu, H., 2005. Building effective defect-prediction models in practice. IEEE Software, 23–29.

Koru, A., Tian, J., 2003. An empirical comparison and characterization of high defect and high complexity modules. Journal of Systems and Software 67, 153–163.

Lin, S., Patel, S., Duncan, A., Goodwin, L., 2003. Using decision trees and support vector machines to classify genes by names. In: Proceedings of the European Workshop on Data Mining and Text Mining for Bioinformatics, pp. 35–41.

Ma, Y., Guo, L., Cukic, B., 2006. A Statistical Framework for the Prediction of Fault-Proneness. Advances in Machine Learning Application in Software Engineering. Idea Group Inc..

Mair, C., Kadoda, G., Leflel, M., Phapl, L., Schofield, K., Shepperd, M., Webster, S., 2000. An investigation of machine learning based prediction systems. Journal of Systems and Software 53 (1), 23–29.

McCabe, T., 1976. A complexity measure. IEEE Transactions on Software Engineering 2 (4), 308–320.

McCabe, T., Butler, C., 1989. Design complexity measurement and testing. Communications of the ACM 32 (12), 1415–1425.

Morris, C., Autret, A., Boddy, L., 2001. Support vector machines for identifying organisms-a comparison with strongly partitioned radial basis function networks. Ecological Modeling 146, 57–67.

Munson, J., Khoshgoftaar, T., 1992. The detection of fault-prone programs. IEEE Transactions on Software Engineering 18 (5), 423–433.

Nadeau, C., Bengio, Y., 2003. Inference for the generalization error. Machine Learning 52, 239–281.

Osuna, E., Freund, R., Girosi, F., 1997. Training support vector machines: an application to face detection. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 130–136.

Schmidt, M., Gish, H., 1996. Speaker identification via support vector classifiers. In: Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP-96), pp. 105–108.

Schneidewind, N., 1992. Methodology for validating software metrics. IEEE Transactions on Software Engineering 18 (5), 410–422.

Selby, R., Porter, A., 1988. Learning from examples: generation and evaluation of decision trees for software resource analysis. IEEE Transactions on Software Engineering 14 (12), 1743–1756.

Shepperd, M., Kadoda, G., 2001. Comparing software prediction techniques using simulation. IEEE Transactions on Software Engineering 27 (11), 1014–1022.

Smola, A., 1998. Learning with kernels. Ph.D. dissertation, Department of Computer Science, Technical University Berlin, Germany.

Tay, F., Cao, L., 2001. Application of support vector machines in financial time series forecasting. Omega 29, 309–317.

Vapnik, V., 1995. The Nature of Statistical Learning Theory. Springer, New York.

Weaver, R., 2003. The safety of software – Constructing and assuring arguments. Ph.D. dissertation, Department of Computer Science, University of York.

Webb, R., 2002. Statistical Pattern Recognition, second ed. John Wiley & Sons, New York.

Witten, I., Frank, E., 2005. Data Mining: Practical Machine Learning Tools and Techniques, second ed. Morgan Kaufmann, San Francisco.

Karim O. Elish is a MS candidate and research assistant in the Information and Computer Science Department at King Fahd University of Petroleum and Minerals (KFUPM), Saudi Arabia. He received the BS degree in computer science from KFUPM in 2006. He is a member of Software Engineering Research Group (SERG) at KFUPM. His main research interests include software metrics and measurement, empirical software engineering, and application of machine learning and data mining in software engineering.

Mahmoud O. Elish received the PhD degree in computer science from George Mason University, USA, in 2005. He is an assistant professor in the Information and Computer Science Department at King Fahd University of Petroleum and Minerals, Saudi Arabia. His research interests include software metrics and measurement, object-oriented analysis and design, empirical software engineering, and software quality predictive models.