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

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.

* Corresponding author. Tel. +966 3 860 1150; Fax. +966 3 860 2174
Email addresses: kelish@kfupm.edu.sa (K. Elish), elish@kfupm.edu.sa (M. Elish).
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 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 nonparametric 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 non-linear 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). Figure 1(a) 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 \( f: \mathbb{R}^p \mapsto \{\pm 1\} \) using training data. Let us represent the class \( A \) with \( y = 1 \) and class \( B \) with \( y = -1 \); \((x_i, y_i) \in \mathbb{R}^p \times \{\pm 1\}\). If the training data are linearly separable by hyperplane in the \( p \) dimensional space then there exists a pair \((\mathbf{w}, b)\) such that:

\[
\begin{align*}
\mathbf{w}^T \mathbf{x}_i + b &\geq +1; \quad \text{for all } \mathbf{x}_i \in A \\
\mathbf{w}^T \mathbf{x}_i + b &\leq -1; \quad \text{for all } \mathbf{x}_i \in B
\end{align*}
\]

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:

\[
y_i (\mathbf{w}^T \mathbf{x}_i + b) \geq 1; \quad \text{for all } \mathbf{x}_i \in A \cup B
\]

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

\[
\begin{align*}
\text{Minimize } & \frac{1}{2} \|\mathbf{w}\|^2 \\
\text{Subject to } & y_i (\mathbf{w}^T \mathbf{x}_i + b) \geq 1, \quad i = 1, \ldots, n.
\end{align*}
\]

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:

\[
F(\lambda) = \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \|\mathbf{w}\|^2 = \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j
\]
where $\lambda = (\lambda_1, \ldots, \lambda_n)^T$ are the Lagrange multipliers and are nonzero 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 $\lambda_i \geq 0$. Therefore, we optimize the following problem:

\[
\begin{align*}
\text{Maximize} & \quad \left\{ \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j x_i^T x_j \right\} \\
\text{Subject to} & \quad \sum_{i=1}^{n} \lambda_i y_i = 0; \quad \text{and} \quad \lambda_i \geq 0; \quad i = 1, 2, \ldots, n.
\end{align*}
\]

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

\[
f(x) = \text{sign} \sum_{i=1}^{n} \lambda_i y_i (x_i^T x) + b
\]

### 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. Figure 1(b) 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 $\xi_i \geq 0$ in the constraints to measure how much the margin constraints are violated (Cortes and Vapnik, 1995):

\[
\begin{align*}
\text{Minimize} & \quad \frac{1}{2} \|w\|^2 + C \sum_{i=1}^{n} \xi_i \\
\text{Subject to} & \quad y_i (w^T x_i + b) \geq 1 - \xi_i; \quad \text{for} \quad i = 1, \ldots, n.
\end{align*}
\]

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, Equation (7) is changed to:

\[
\begin{align*}
\sum_{i=1}^{n} \lambda_i y_i = 0; \quad \text{and} \quad 0 \leq \lambda_i \leq C; \quad i = 1, 2, \ldots, n
\end{align*}
\]
Thus the only difference from the separable case is that the $\lambda_i$ 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).

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

3.2 Two-Class: Non-Linear Support Vector Machines

In case that SVM cannot linearly separate two classes, SVM extends its applicability to solve this problem by mapping input data into higher dimensional feature spaces using a nonlinear mapping $\phi$ (Burges, 1998), such that $x \mapsto \phi(x)$, where $\phi : \mathbb{R}^n \rightarrow \mathbb{R}^m$ 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. Figure 2 shows the transformation from lower to higher dimensional feature spaces using $\phi$. This transformation can be done using a kernel function. Therefore, the kernel function is an important parameter in SVM. The kernel function $K(x_i, x_j)$ is defined as follows (Abe, 2005):

$$K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$$

(12)
Therefore, the optimization problem of the equation (6) becomes:

\[
\text{Maximize } \{ \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j k(x_i, x_j) \} \tag{13}
\]

Subject to \( \sum_{i=1}^{n} \lambda_i y_i = 0; \text{ and } \lambda_i \geq 0; \ i = 1, 2, ..., n. \tag{14} \)

where \( K(x_i, x_j) \) is the kernel function performing the non-linear mapping into feature space. Once the solution has been found, the decision can be constructed as:

\[
f(x) = \text{sign} \left( \sum_{i=1}^{n} \lambda_i y_i k(x_i, x) + b \right) \tag{15}\]

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

1. Linear: \( K(x_i, x_j) = x_i^T x_j \)
2. Polynomial: \( K(x_i, x_j) = (x_i^T x_j + 1)^d \)
3. Gaussian (RBF): \( K(x_i, x_j) = \exp(-\gamma \| x_i - x_j \|^2) \)
4. Sigmoid (MLP): \( K(x_i, x_j) = \tanh(\gamma (x_i^T x_j) - r) \)

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

![Figure 2. Mapping input data into higher dimensional space.](image)
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 twenty-one 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.

<table>
<thead>
<tr>
<th>Project</th>
<th>Language</th>
<th># of Modules</th>
<th>% of Defective Modules</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CM1</td>
<td>C</td>
<td>496</td>
<td>9.7%</td>
<td>NASA spacecraft instrument</td>
</tr>
<tr>
<td>PC1</td>
<td>C</td>
<td>1107</td>
<td>6.9%</td>
<td>Flight software for an earth orbiting satellite</td>
</tr>
<tr>
<td>KC1</td>
<td>C++</td>
<td>2107</td>
<td>15.4%</td>
<td>Storage management for receiving and processing ground data</td>
</tr>
<tr>
<td>KC3</td>
<td>Java</td>
<td>458</td>
<td>6.3%</td>
<td>Collection, processing and delivery of satellite metadata</td>
</tr>
</tbody>
</table>

---

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

\(^2\) http://mdp.ivv.nasa.gov/index.html
4.3 Independent Variables

The independent variables are twenty-one 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.

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

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.
Table 3. Best subset of independent variables in each dataset

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Best subset of independent variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>CM1</td>
<td>LOC, IV(g), D, I, LOComment, LOBlank,</td>
</tr>
<tr>
<td>PC1</td>
<td>V(g), I, LOCCodeAndComment, LOComment, LOBlank, UniqOp</td>
</tr>
<tr>
<td>KC1</td>
<td>V, D, V(g), LOCCode, LOComment, LOBlank, UniqOpnd</td>
</tr>
<tr>
<td>KC3</td>
<td>N, EV(g), LOCCodeAndComment</td>
</tr>
</tbody>
</table>

4.4 Dependent Variable

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

Table 4. A confusion matrix

<table>
<thead>
<tr>
<th>Actual</th>
<th>Not Defective</th>
<th>Defective</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not Defective</td>
<td>TN = True Negative</td>
<td>FP = False Positive</td>
</tr>
<tr>
<td>Defective</td>
<td>FN = False Negative</td>
<td>TP = True Positive</td>
</tr>
</tbody>
</table>

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:

\[
\text{Precision} = \frac{TP}{TP + FP}
\]

### 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:

\[
\text{Recall} = \frac{TP}{TP + FN}
\]

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, 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:

\[
F\text{-measure} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}
\]

### 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 Table 5, Table 6, Table 7, and Table 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 Table 5, Table 6, Table 7, and Table 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.

Table 5. Prediction performance measures: CM1 Dataset

<table>
<thead>
<tr>
<th>Prediction Model</th>
<th>Accuracy</th>
<th>Precision</th>
<th>Recall</th>
<th>F-Measure</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean</td>
<td>StDev</td>
<td>Sig.?</td>
<td>Mean</td>
</tr>
<tr>
<td>SVM</td>
<td>90.69</td>
<td>1.074</td>
<td>No(-)</td>
<td>90.66</td>
</tr>
<tr>
<td>LR</td>
<td>90.17</td>
<td>1.987</td>
<td>No(+)</td>
<td>91.10</td>
</tr>
<tr>
<td>KNN</td>
<td>83.27</td>
<td>4.154</td>
<td>Yes(+)</td>
<td>91.36</td>
</tr>
<tr>
<td>MLP</td>
<td>89.32</td>
<td>2.066</td>
<td>No(+)</td>
<td>90.75</td>
</tr>
<tr>
<td>RBF</td>
<td>89.91</td>
<td>1.423</td>
<td>No(+)</td>
<td>90.32</td>
</tr>
<tr>
<td>BBN</td>
<td>76.83</td>
<td>6.451</td>
<td>Yes(+)</td>
<td>94.17</td>
</tr>
<tr>
<td>NB</td>
<td>86.74</td>
<td>3.888</td>
<td>Yes(+)</td>
<td>92.49</td>
</tr>
<tr>
<td>RF</td>
<td>88.62</td>
<td>2.606</td>
<td>Yes(+)</td>
<td>90.93</td>
</tr>
<tr>
<td>DT</td>
<td>89.82</td>
<td>1.526</td>
<td>No(+)</td>
<td>90.35</td>
</tr>
</tbody>
</table>

Table 6. Prediction performance measures: PC1 Dataset

<table>
<thead>
<tr>
<th>Prediction Model</th>
<th>Accuracy</th>
<th>Precision</th>
<th>Recall</th>
<th>F-Measure</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean</td>
<td>StDev</td>
<td>Sig.?</td>
<td>Mean</td>
</tr>
<tr>
<td>SVM</td>
<td>93.10</td>
<td>0.968</td>
<td>No(-)</td>
<td>93.53</td>
</tr>
<tr>
<td>LR</td>
<td>93.19</td>
<td>1.088</td>
<td>No(-)</td>
<td>93.77</td>
</tr>
<tr>
<td>KNN</td>
<td>91.82</td>
<td>2.080</td>
<td>No(+)</td>
<td>95.54</td>
</tr>
<tr>
<td>MLP</td>
<td>93.59</td>
<td>1.212</td>
<td>No(-)</td>
<td>94.20</td>
</tr>
<tr>
<td>RBF</td>
<td>92.84</td>
<td>0.948</td>
<td>No(+)</td>
<td>93.40</td>
</tr>
<tr>
<td>BBN</td>
<td>90.44</td>
<td>4.534</td>
<td>No(+)</td>
<td>94.60</td>
</tr>
<tr>
<td>NB</td>
<td>89.21</td>
<td>2.606</td>
<td>Yes(+)</td>
<td>94.95</td>
</tr>
<tr>
<td>RF</td>
<td>93.66</td>
<td>1.665</td>
<td>No(-)</td>
<td>95.15</td>
</tr>
<tr>
<td>DT</td>
<td>93.58</td>
<td>1.499</td>
<td>No(-)</td>
<td>94.59</td>
</tr>
</tbody>
</table>
Table 7. Prediction performance measures: KC1 Dataset

<table>
<thead>
<tr>
<th>Prediction Model</th>
<th>Accuracy</th>
<th>Precision</th>
<th>Recall</th>
<th>F-Measure</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean</td>
<td>StDev</td>
<td>Sig.?</td>
<td>Mean</td>
</tr>
<tr>
<td>SVM</td>
<td>84.59</td>
<td>0.714</td>
<td></td>
<td>84.95</td>
</tr>
<tr>
<td>LR</td>
<td>85.55</td>
<td>1.394</td>
<td>No(−)</td>
<td>87.08</td>
</tr>
<tr>
<td>KNN</td>
<td>83.99</td>
<td>2.144</td>
<td>No(+)</td>
<td>89.58</td>
</tr>
<tr>
<td>MLP</td>
<td>85.68</td>
<td>1.535</td>
<td>Yes(−)</td>
<td>86.75</td>
</tr>
<tr>
<td>RBF</td>
<td>84.81</td>
<td>1.107</td>
<td>No(−)</td>
<td>85.81</td>
</tr>
<tr>
<td>BBN</td>
<td>75.99</td>
<td>2.925</td>
<td>Yes(+)</td>
<td>91.98</td>
</tr>
<tr>
<td>NB</td>
<td>82.86</td>
<td>2.210</td>
<td>Yes(+)</td>
<td>88.59</td>
</tr>
<tr>
<td>RF</td>
<td>85.20</td>
<td>1.798</td>
<td>No(−)</td>
<td>88.12</td>
</tr>
<tr>
<td>DT</td>
<td>84.56</td>
<td>1.580</td>
<td>No(+)</td>
<td>86.79</td>
</tr>
</tbody>
</table>

Table 8. Prediction performance measures: KC3 Dataset

<table>
<thead>
<tr>
<th>Prediction Model</th>
<th>Accuracy</th>
<th>Precision</th>
<th>Recall</th>
<th>F-Measure</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean</td>
<td>StDev</td>
<td>Sig.?</td>
<td>Mean</td>
</tr>
<tr>
<td>SVM</td>
<td>93.28</td>
<td>1.099</td>
<td></td>
<td>93.65</td>
</tr>
<tr>
<td>LR</td>
<td>93.42</td>
<td>1.892</td>
<td>No(−)</td>
<td>94.37</td>
</tr>
<tr>
<td>KNN</td>
<td>92.50</td>
<td>2.979</td>
<td>No(+)</td>
<td>95.23</td>
</tr>
<tr>
<td>MLP</td>
<td>93.50</td>
<td>2.215</td>
<td>No(−)</td>
<td>94.74</td>
</tr>
<tr>
<td>RBF</td>
<td>93.59</td>
<td>1.557</td>
<td>No(−)</td>
<td>94.10</td>
</tr>
<tr>
<td>BBN</td>
<td>93.21</td>
<td>2.684</td>
<td>No(−)</td>
<td>95.72</td>
</tr>
<tr>
<td>NB</td>
<td>92.81</td>
<td>3.134</td>
<td>No(+)</td>
<td>95.89</td>
</tr>
<tr>
<td>RF</td>
<td>92.63</td>
<td>3.073</td>
<td>No(+)</td>
<td>95.49</td>
</tr>
<tr>
<td>DT</td>
<td>93.11</td>
<td>1.636</td>
<td>No(+)</td>
<td>93.85</td>
</tr>
</tbody>
</table>

4.9 Discussion of Results

Figure 3 and Figure 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 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 Figure 5 to Figure 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.
Figure 3. Mean vs. standard deviation of accuracy: CM1 dataset

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

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

Figure 6. Mean vs. standard deviation of F-measure: PC1 dataset
Figure 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 4 out of the 8 compared models in accuracy, i.e., there are four Yes(+). It can also be observed that SVM is significantly outperformed by 2 models in precision, i.e., there are two Yes(-), and so on. Figure 12, Figure 13, and Figure 14 provide similar histograms for PC1, KC1, and KC3 datasets respectively.
4.9.1 Results from CM1 Dataset

From Table 5 and Figure 11, it is observed that SVM outperforms all the compared eight models in accuracy. SVM is outperformed in precision by 6 out of the 8 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 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 5 out of the 8 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%).
4.9.2 Results from PC1 Dataset

From Table 6 and Figure 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 5 out of the 8 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 Figure 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 6 out of the 8 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 Figure 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 5 out of the 8 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 4 out of the 8 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 2 out of the 4 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 4 out of the 8 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 5 out of the 8 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% to 93.3%; its precision is within 84.9% to 93.6%; its recall is within 99.4% to 100%; and its F-measure is within 0.916 to 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 5 out of the 8 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.


