Support Vector Machine for Software Defect Prediction

P.A. Selvaraj\(^1\) and Dr.P. Thangaraj\(^2\)

\(^1\)Department of computer Applications, Kongu Engineering College, Erode, India
\(^2\)Department of Computer Science & Engineering, Bannari Amman Institute of Technology, Sathy.

ABSTRACT

Identifying/removing software defects is time consuming. In ill planned projects, rectifying defects consumes more time than for code development. Defective module prediction is essential, to recycle and preserve through system simplification, reusable parts localization. Software defect prediction improves quality building predictive classification models and software metrics to ensure fault-prone module identification. This paper is proposing to investigate the classification accuracy of Support Vector Machine (SVM) for software defect prediction using different kernels.

Keywords: Software defect prediction, Naive Bayes, Decision Stump, SVM, Polykernel and RBF.

1. INTRODUCTION

Software defects [1] are flaws, errors, failures, mistakes or faults in computer programs/systems which produce incorrect/unexpected results leading to the system behaving in unintended ways. Software defect prediction locates defective software modules to improve software quality/testing by constructing code attribute predictive models to ensure early fault-prone module identification and deduction. It aids planning, monitoring, controlling and predicting defect density to understand/control software quality better. Software Defect Prediction results, [defects in a software system] helps software developers control software process.

Classification locates a model set describing/distinguishing data classes/concepts. Data organization in classes called supervised learning provides training samples class labels which supervise classification model learning. Classification approaches use training sets where objects and class labels are associated. A model is built after the classification algorithm learns from a training set. New models are classified by the model. Fraud detection and credit risk applications suit this analysis. Numerous classification methods were touted for software defect prediction.

Software Defect Prediction [2], a learning problem attracts interest from academia/industry. Static code attributes from prior defect logs based software releases builds models to predict defective modules for next release. It locates defective software parts, useful with limited budgets or when system is too large for comprehensive testing. A defect predictor ensures that software engineers test defect-prone software parts.

Defect predictors ensure quality software, lowering system delivery costs, including defect prediction data sets from real-world projects for the public. It helps researchers build repeatable/comparable models. As of now, much research was devoted to metrics describing code modules and learning algorithms to construct software defect prediction models.
Support Vector Machine (SVM) [3] performs data classification useful in applications like medical diagnosis, identification, pattern recognition, text classification, organism’s identification and Chinese character classification. SVM constructs an N-dimensional hyperplane splitting data sets into 2 categories and aims to locate an optimal hyperplane, separating vector clusters to ensure that with one category of dependent variable on the plane’s one side, while cases with other category are on the other side. Support vectors are close to the hyperplane. SVM modeling locates hyperplane maximizing margin between support vectors. SVM handles non linear regions separating points through a kernel function to map data into varied space where separation is undertaken by the hyperplane. Radial basis Function (RBF) is the recommended kernel function. Hence, RBF is used in SVM modeling for faulty class prediction. Non-linear data are mapped by RBF kernels into higher dimensional space to control non linear relationships between dependent/ independent variables.

SVM and Gaussian Radial Basis Function (RBF) kernel [4] are integrated. SVM’s important part is hyper-parameters selection. RBF’s optimal spread and penalty parameter are achieved by pointing out large possibilities interval and through cross-validation result verification. Predictions bias’ manual adjustment and threshold was undertaken for test sets production and accurate results. A non-linear input function can be mapped into high dimensional feature spaces through use of SVM’s non linear RBF kernel, making calculating convex linear optimization problem easier.

Defects based on existing software metrics are identified by this paper using data mining techniques, improving software quality resulting in reduced software development cost in the developing/maintenance phases. This paper chose Decision Stump, Naïve Bayes, and SVM for defective modules prediction.

2. RELATED WORKS

Software defect prediction [5] aids understanding and controlling software quality. Current defect prediction techniques are historical project data based. This data is hard to obtain for new projects/organizations making it hard to achieve effective defect prediction. To overcome this, Li, et al., (2011) suggested a sample-based software defect prediction method which selects and tests a limited modules percentage for bigger software systems, to construct a defect prediction model for other modules. Sample prediction is of 3 types: random sampling with conventional machine learning, random sampling with a semi-supervised learner and active sampling with active semi-supervised learners. ACoForest a new, semi-supervised learning method sampled modules to help learn a good prediction model, was proposed. Experiments on PROMISE datasets proved the effectiveness of the proposed method and had potential for industrial applications.

Defect Prediction Models [6] locate error causing software system to ensure quality assurance activities like tests/code reviews. Research in this done for more than ten years resulted in more than 100 published papers. Most current models assume that quality assurance cost for all models are the same. When effort is considered, many classifiers performance is almost the same as randomly chosen modules. Mende and Koschke (2010) compared 2 strategies to include treatment in prediction and evaluated their predictive power. When compared to the effort taken for them, both the models did better.
Okutan and Yildiz (2012) used Bayesian networks [7] to determine probabilistic influential relationships between software metrics and defect proneness. Two metrics - NOD for developer’s number and LOCQ for source code quality - were defined. They were extracted through an inspection of the source code repositories of selected Promise data repository data sets. At the end of modeling, software system’s defect proneness probability, most effective metrics set, and relationships between metrics and defects was understood. Experiments with 9 open source Promise data repository data sets proved that for class (RFC), lines of code (LOC), and lack of coding quality (LOCQ) were effective metrics while coupling between objects (CBO), weighted method per class (WMC), and lack of cohesion of methods (LCOM) were not efficient on defect proneness. Limited effectiveness made number of children (NOC) and depth of inheritance tree (DIT) untrustworthy.

Software systems automated defective module detection lowers development costs making them more reliable. Static code metrics for module collection in 11 NASA data sets with a SVM classifier [8] was used by Gray, et al., (2009). Prior to classification data underwent rigorous pre-processing, including classes balancing (defective or otherwise) and removal of repeating instances. SVM demonstrated 70% accuracy on data unseen earlier.

Software quality improved through regression techniques and used software metrics to predict software modules defects helping developers allocate limited resources to defective modules. Yan, et al., (2010) suggested using Fuzzy Support Vector Regression (FSVR) [9] for predicting software defect numbers. Regressor fuzzification input takes care of unbalanced software metrics dataset. In contrast to support vector regression, experiments with MIS and RSDIMU datasets demonstrated that FSVR provided lower mean squared error and higher defects accuracy for many defective modules.

Software defect prediction is based on code attributes to assess likelihood of defective software modules. Numerous classification methods were suggested but association based classification methods were not touched in this context. Such a classification method CBA2 [10], was assessed by Ma, et al., (2011) comparing it to other rule based classification methods. Generated data rule sets were also investigated from a software project to predict software modules defects in similar software projects. Use of CBA2 algorithm resulted in accurate/comprehensible rule sets.

As predicting defect-prone software components are economically important they attracted much attention. As it was hard to act against seemingly inconsistent results, Song, et al., (2011) [11] proposed/evaluated a general software defect prediction framework that supported unbiased comparison between two prediction systems. Framework components included scheme evaluation/defect prediction. The former analyzed competing learning schemes prediction performances for historical data sets. The latter built models based on evaluated learning scheme, and predicting software defects with new data depending on a constructed model. Simulation and available software defect data sets were used to demonstrate performance of the proposed framework. Results revealed that different learning schemes were needed for differing data sets and that minor evaluation details could reverse findings. Also, the proposed framework proved effective and less biased compared to earlier approaches. Though failing to evaluate a learning scheme can be misleading, the proposed framework could offset such issues.
3. METHODOLOGY

KC1 dataset is NASA Metrics Data Program [12] which verifies/improves predictive software engineering models. KC1 is a C++ system implementing storage management for ground data receipt/processing. The dataset comprises of McCabe and Halstead features code extractors and module based measures.

Defect detectors are calculated as follows:

\[ a = \text{Classifier predicts no defects and the module has no error.} \]
\[ b = \text{Classifier predicts no defects and the module has error.} \]
\[ c = \text{Classifier predicts some defects and the module has no error.} \]
\[ d = \text{Classifier predicts some defects and the module has error.} \]

Accuracy, detection probability (pd) or recall, precision (prec), probability of false alarm (pf), and effort are calculated as

\[ \text{Accuracy} = \frac{a + d}{a + b + c + d} \]
\[ \text{recall} = \frac{d}{b + d} \]
\[ pf = \frac{c}{a + c} \]
\[ prec = \frac{d}{c + d} \]
\[ \text{effort} = \frac{c \cdot LOC + d \cdot LOC}{\text{Total LOC}} \]

KC1 dataset includes 2109 instances and 22 varied attributes which includes 5 different LOC, 3 McCabe metrics, 12 Halstead metrics, a branch count and 1 goal-field. Dataset’s attribute information is: design complexity, total operands, McCabe's line count of code (LOC), program length, cyclomatic complexity, effort, Halstead, class and others.

Examples from dataset:
Example 1 - 1.1, 1.4, 1.4, 1.4, 1.3, 1.3, 1.3, 1.3, 1.3, 1.3, 1.3, 1.3, 1.3, 1.3, 1.3, 2.2, 2.2, 1.2, 1.2, 1.2, 1.2, 1.4, false
Example 2 - 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, true
Example 3 - 83, 11, 1, 11, 171, 927.89, 0.04, 23.04, 40.27, 21378.61, 0.31, 1187.7, 65, 10, 6, 0, 18, 25, 107, 64, 21, true

4. CLASSIFIERS

Decision stumps are one level decision trees (DS) [13]. In a multiple layered tree, a stump stops after first split. Data population segmentation uses decision stumps. Occasionally, they are a yes/no decision model for limited and smaller data. Decision stumps are easily constructed when compared to decision trees. Best stumps are when decision tree nodes are located: features search splits them, and
thresholds are searched, attracted by values sorting. A decision stump node represents features in an instance to be classified, and branches represent node value. Classification and sorting of root node starting instances are based on feature values. Decision stumps reproduce common sense baseline, and continue to improve when selected feature is informative. For experiments, decision stumps were base learners.

Naive Bayes Classifier assigns instance $s_k$ with attribute values $(A_1=V_1, A_2=V_2, \ldots, A_n=V_n)$ to class $C_i$ with maximum probability $(C_i/(V_1,V_2,\ldots,V_n))$ for all $i$. It uses Bayes rule assuming attributes independently and used by Bayesian theorem, requiring discrete values for proper functioning [7]. A domain must be associated with each column that uses the following formula to calculate individual attribute’s probability:

$$P(H|E) = P(H)/P(E) \prod P(E_i|H)$$

Where, $E_i$ = fragments of evidence $E_i$

$P(H) = $ Prior Probability

$P(H|E) = $ Posterior Probability

For machine learning/data mining Naive Bayes [14] is an inductive learning algorithm whose classification is based on independence assumption which is rare in the real world applications. Naive Bayes is Bayesian network’s simpler form where independent attributes are given class variable values, known as conditional independence which is rare in real-world applications. A direct Naive Bayes limitation overcoming procedure is extending structure to represent attribute dependencies. An augmented Naive Bayesian network/augmented Naive Bayes (ANB), is extended Naive Bayes in which class node points to linked attribute nodes.

Advantages:
- It is quick to train and evaluate.
- It scales linearly with predictors/rows numbers.
- It is good for real-world problems

Disadvantages:
- Cannot solve complex problems.

5. **SUPPORT VECTOR MACHINE**

SVM classifier should be trained before use; thus reduced input data is partitioned $(y_i), i=1,\ldots,n$ into 2, $T \subset \{1,\ldots, n\}$ training set and $V \subset \{1,\ldots, n\}$ testing (or validation) set with $T \cup V = \{1,\ldots, n\}$ and $T \cap V = \emptyset$. Training data set $T$ is labeled manually into 2 classes with ground truth, $l(y_i) = \pm 1$. Once classifier is trained and decision function evaluation $d(y_i) = \pm 1$ yields classification of any data $y_i$.

In detail, SVM [15] tries to separate data $\varphi(y_i)$ mapped by selected kernel function $\varphi$ by a hyperplane $wT\varphi(y_i)+b=0$ with $w$ normal vector and $b$ translation. Decision function then is $d(y_i) = \text{sgn}(wT\varphi(y_i)+b)$. Maximizing margin and introducing slack variables $\xi = (\xi_i)$ for non-separable data, a primal optimization problem is received:
\[
\min_{w,b,\xi} = \frac{1}{2} w^T w + C \sum_{i \in T} \xi_i \\
\text{with constraints } l(y_i)(w^T \phi(y_i) + b) \geq 1 - \xi_i \\
\xi_i \geq 0 \text{ for } i \in T,
\]

where \( C \) is user-determined penalty parameter. Easier computation is possible when switched to dual optimization problem,

\[
\min_{\alpha} = \frac{1}{2} \alpha^T Q \alpha - e^T \alpha \\
\text{with constraints } 0 \leq \alpha_i \leq C \text{ for } i \in T \\
\sum_{i \in T} y_i \alpha_i = 0,
\]

where \( \alpha = (\alpha_i) \) are so-called support vectors, \( e = [1, \ldots, 1]^T \) and \( Q \) is positive semi-definite matrix formed by \( Q_{jk} = l(y_j)l(y_k)K(y_j, y_k) \), and \( K(y_j, y_k) = \phi(y_j)^T \phi(y_k) \) is kernel function from \( \phi \).

Once optimization problem is solved, hyperplane parameters \( w \) and \( b \), \( w \) are determined directly as

\[
w = \sum_{i \in T} \alpha_i l(y_i) \phi(y_i)
\]

and \( b \) via one of Karush-Kuhn-Tucker conditions as \( b = -l(y_i)y_iTw \), for those \( i \) with \( 0 < \alpha_i < C \). Thus, trained SVM classifier’s decision function ends up as

\[
d(y_i) = \text{sgn}(w^T \phi(y_i) + b) = \text{sgn}(\sum_{j \in T} \alpha_j l(y_j)K(y_j, y_i) + b).
\]

6. KERNEL FUNCTIONS

Inner feature space product has equal kernel in input space [16],

\[
K(x, x') = \langle \phi(x), \phi(x') \rangle,
\]

if certain conditions hold. When \( K \) is a symmetric positive definite function that satisfies Mercer’s Conditions,

\[
K(x, x') = \sum_{m=0}^{\infty} a_m \phi_m(x) \phi_m(x'), \quad a_m \geq 0,
\]

\[
\int \int K(x, x')g(x)g(x')dx'dx' > 0, \quad g \in L_2,
\]

then kernel represents a proper inner product of feature space. Valid functions satisfying Mercer’s conditions are given, which unless stated are valid for real \( x \) and \( x' \).
7. **POLYNOMIAL**

For non-linear modeling, a popular method is polynomial mapping,

\[ K(x, x') = \langle x, x' \rangle^d. \]

\[ K(x, x') = (\langle x, x' \rangle + 1)^d. \]

When hessian becomes zero, problems are avoided, so second kernel is usually preferred.

8. **GAUSSIAN RADIAL BASIS FUNCTION**

Radial basis functions received attention, commonly with a Gaussian of form,

\[ K(x, x') = \exp \left( -\frac{||x - x'||^2}{2\sigma^2} \right). \]

Classical techniques with radial basis functions are determiners of a centers subset. Clustering is used to select a centers subset first. SVM feature’ attraction is that selection with support vectors contributing one local Gaussian function centered on data point is implicit.

9. **RESULTS AND DISCUSSION**

The software complexity measures such as LOC measure, Cyclomatic complexity, Base Halstead measures and Derived Halstead measures of the KC1 (NASA) dataset are used to classify the software modules. Weka is a machine learning software written in Java. It supports several data mining process such as preprocessing, clustering, classification and so on. All classification in this study is carried out on Weka.

For the performance evaluation of the classifiers, 2107 samples from the KC1 Dataset is used, wherein 1391 samples are used as training set and 716 samples are used for testing. Weka was used on KC1 dataset for classification, and the result is summarized in Table 1 and Figure 1.

<table>
<thead>
<tr>
<th>Technique used</th>
<th>Correctly classified %</th>
<th>RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naïve Bayes</td>
<td>83.77</td>
<td>0.3967</td>
</tr>
<tr>
<td>Decision stump</td>
<td>86.1</td>
<td>0.3336</td>
</tr>
<tr>
<td>SVM-Polykernel</td>
<td>86.05</td>
<td>0.3735</td>
</tr>
<tr>
<td>SVM-RBF</td>
<td>86.05</td>
<td>0.3735</td>
</tr>
</tbody>
</table>

**Fig 1: Classification Accuracy on KC1 Dataset**
It is observed that decision stump provides the best classification accuracy which is negligibly higher than SVM. Both polykernel and RBF perform equally well in classifying the software defects. Table 2 tabulates the precision, recall and f Measure achieved by the classifiers. Figure 2 shows the precision and recall and Figure 3 shows the f Measure.

<table>
<thead>
<tr>
<th>Technique used</th>
<th>Precision</th>
<th>Recall</th>
<th>F-Measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naïve Bayes</td>
<td>0.833</td>
<td>0.838</td>
<td>0.836</td>
</tr>
<tr>
<td>Decision stump</td>
<td>0.741</td>
<td>0.861</td>
<td>0.797</td>
</tr>
<tr>
<td>SVM-Polykernel</td>
<td>0.804</td>
<td>0.86</td>
<td>0.8</td>
</tr>
<tr>
<td>SVM-RBF</td>
<td>0.804</td>
<td>0.86</td>
<td>0.8</td>
</tr>
</tbody>
</table>

Though decision stump achieves slightly higher classification accuracy, the precision and f measure is much lower. Further investigations for defect prediction based on SVM are required.

10. CONCLUSION

Prediction ensures predicting continuous/ordered values for input. But, classification techniques like Bayesian belief networks, neural network and genetic algorithms are adaptable for prediction. It is not enough to train a classifier/predictor; Software defect prediction traces defective components before testing phase starts. Also, use of various machine learning techniques for software fault prediction is presented here. Experiments reveal that SVM with Poly kernel achieves best performance. Further defect prediction experiments with a SVM base are required.
REFERENCES


