SOFTWARE DEFECT PREDICTION USING IMPROVED SUPPORT VECTOR MACHINE CLASSIFIER

M. Karthikeyan
Research Scholar, Department of Computer Science, Karpagam University, Coimbatore, India

Dr. S. Veni
Research Supervisor, Department of Computer Science, Karpagam University, Coimbatore, India

ABSTRACT
This paper proposes new algorithm for software defect prediction using improved Support Vector Machine Classifier. Support Vector Machine Classifier (SVM) and Improved Support Vector Machine Classifier (ISVM) are trained with 50% dataset size and implemented using MATLAB. From the results, it is evident that ISVM attains better software defect prediction accuracy than that of SVM and false alarm rate is reduced considerably.

Key words: Support Vector Machine Classifier (SVM), Improved Support Vector Machine Classifier (ISVM), True positive (TP), True negative (TN), False positive (FP), False negative (FN), Accuracy, False Alarm Rate


http://www.iaeme.com/ijmet/issues.asp?JType=IJMET&VType=7&IType=5

1. INTRODUCTION

1.1. Introduction to Support Vector Machine (SVM)
SVMs are commonly implemented among the machine learning methods. SVM is based on statistical learning theory proposed by Vapnic and co-workers. SVM is a classifier based system which was originally designed for binary classification, can be used to classify the software defects. Binary classification problems can be solved using SVM. It belongs to the family of generalized linear classifier and can be interpreted as an extension of the linear perceptron. It is also considered a special case of regularization. A special property is that they simultaneously minimize the empirical classification error and maximize geometric margin and hence they are also known as maximum margin classifiers.

Support Vector Machines (SVM) has recently gained importance in the field of machine learning and pattern classification. Classification is a process that can be defined as a tool for categorization. SVM is a
Software Defect Prediction Using Improved Support Vector Machine Classifier classification technique that is achieved by realizing a linear or non-linear separation surface in the input space.

1.2. Steps Performed in SVM
The following are the steps performed while executing SVM. At first the training dataset is selected from the datasets. The SVM training is performed for the training data and thus structured fields are produced. Then the weighted structures are obtained and the trained dataset is loaded for testing. The SVM data classification is performed based on the trained structure using the structured fields. Finally, classified results are obtained that contains the software defects and software in defects. The step-by-step execution of SVM algorithm is shown below.

1.3. SVM Algorithm for Software Defect Prediction
Step 1: Select the training dataset.
Step 2: The SVM training is carried for the training data and produces structure of fields which contains Support Vectors, Alpha, Bias, Group Names and Scale Data.
Step 3: Weighted structures are obtained.
Step 4: The trained dataset is loaded for testing.
Step 5: The testing data, structured fields are given for classification of test data.
Step 6: The SVM classifier works based upon the trained structure.
Step 7: The classified results are obtained.
Step 8: The classified result contains defective / non-defect prone modules.

2. PROPOSED METHODOLOGY

2.1. Proposed Improved Support Vector Machine (ISVM) Classifier
ISVM which is an extension of conventional SVM has a set of supervised learning methods used for classification and regression. Nonnegative matrix factorization is an unsupervised learning method that is being used in ISVM. Supervised learning is a technique in which the algorithm uses predictor and target attribute value pairs to learn the predictor and target value relation. Improved Support Vector Machine is a supervised learning technique for creating a decision function with a training dataset. The training data consist of pairs of predictor and target values. Each predictor value is tagged with a target value. If the algorithm predicts a categorical value for a target attribute, it is called classification function. Class is an example of a categorical variable. Positive and negative can be two values of the categorical variable class. Categorical values do not have partial ordering. If the algorithm can predict a numerical value, then it is called regression. Numerical values have partial ordering.

ISVM maps linear algorithms into non-linear space. It uses a feature called, kernel function, for this mapping. Kernel functions like polynomial, radial basis function are used to divide the feature space by constructing a hyper plane. The kernel functions can be used at the time of training the classifiers which selects support vectors along the surface of this function. ISVM classifies the data by using these support vectors that outline the hyper plane in the feature space.

2.2. Steps performed in ISVM Classifier
The following are the steps performed while executing ISVM. The training dataset is selected from software defect prediction dataset.
Step 1: Select the training dataset.
Step 2: RBF kernel is employed.
Software Defect Prediction Using Improved Support Vector Machine Classifier

Step 3: The ISVM training is carried for the training data and produces structure of fields which contains Support Vectors, Alpha, Bias, Kernel Function Arguments, Group Names and Scale Data.
Step 4: Then serialized learning mechanism of ISVM simplifies and weighted structure are obtained.
Step 5: The trained dataset is loaded for testing.
Step 6: The testing data, structured fields are given for classification of test data.
Step 7: The ISVM classifier works based upon the trained structure.
Step 8: The classified results are obtained.
Step 9: The classified result contains the predicted list of defect prone / non-defect prone modules.

3. EXPERIMENTAL RESULTS

3.1. Dataset
The PROMISE Software Engineering Repository dataset which has been made publicly available in order to encourage repeatable, verifiable, refutable, and/or improvable predictive models of software engineering such as CM1, KC1 and PC1 datasets \(^{[a]−[c]}\). This is a PROMISE Software Engineering Repository data set made publicly available to encourage repeatable, verifiable, refutable, and/or improvable predictive models of software engineering. CM1 \([1]\) is a NASA spacecraft instrument written in "C". KC1 \([2]\) is a "C++" system implementing storage management for receiving and processing ground data. PC1 \([3]\) dataset contains data from C functions. These data come from McCabe and Halstead features extractors of source code. These features are an attempt to objectively characterize code features that are associated with software quality. The nature of association is under dispute. The McCabe and Halstead measures are "module"-based where a "module" is the smallest unit of functionality. In C or Smalltalk, "modules" would be called "function" or "method" respectively. The McCabe metrics are a collection of four software metrics namely essential complexity, Cyclomatic complexity, design complexity and LOC, Lines of Code.

- Cyclomatic Complexity measures the number of "linearly independent paths". A set of paths is said to be linearly independent if no path in the set is a linear combination of any other paths in the set through a program's "flowgraph". A flowgraph is a directed graph where each node corresponds to a program statement, and each arc indicates the flow of control from one statement to another.

- Essential Complexity is the extent to which a flowgraph can be "reduced" by decomposing all the sub flowgraphs that are "D-structured primes". Such "D-structured primes" are also sometimes referred to as "proper one-entry one-exit sub flowgraphs".

- Design Complexity is the Cyclomatic complexity of a module's reduced flowgraph. The flowgraph of a module is reduced to eliminate any complexity which does not influence the interrelationship between design modules. Per McCabe, this complexity measurement reflects the modules calling patterns to its immediate subordinate modules.

- Lines of code are measured per McCabe's line counting conventions.

The details of the datasets are given in Table 4.1.

<table>
<thead>
<tr>
<th>Dataset Name</th>
<th>Number of Attributes</th>
<th>Number of Total Instances</th>
<th>Number of Defect Prone Modules</th>
<th>Number of non-defect prone modules</th>
</tr>
</thead>
<tbody>
<tr>
<td>CM1 ([a])</td>
<td>22</td>
<td>498</td>
<td>49</td>
<td>449</td>
</tr>
<tr>
<td>KC1 ([b])</td>
<td>22</td>
<td>2109</td>
<td>326</td>
<td>1783</td>
</tr>
<tr>
<td>PC1 ([c])</td>
<td>22</td>
<td>1109</td>
<td>1032</td>
<td>77</td>
</tr>
</tbody>
</table>

Software Defect Prediction Using Improved Support Vector Machine Classifier

http://www.iaeme.com/IJMET/index.asp 419  
editor@iaeme.com
3.2. Performance Metrics

- True positive (TP): defective module correctly identified as defective
- True negative (TN): non-defective module correctly identified as non-defective
- False positive (FP): non-defective module incorrectly identified as defective
- False negative (FN): Defective module incorrectly identified as non-defective
- Accuracy = (TP + TN) / (TP + TN + FP + FN)
- False Alarm Rate = FP / (FP + TN)

3.3. Results

Table 2.0 Performance Analysis of SVM and ISVM

<table>
<thead>
<tr>
<th>Dataset</th>
<th>TP</th>
<th>TN</th>
<th>FP</th>
<th>FN</th>
<th>Accuracy</th>
<th>False Alarm Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>S</td>
<td>IS</td>
<td>S</td>
<td>IS</td>
<td>S</td>
<td>IS</td>
</tr>
<tr>
<td>M</td>
<td>V</td>
<td>V</td>
<td>V</td>
<td>V</td>
<td>M</td>
<td>V</td>
</tr>
<tr>
<td></td>
<td>39</td>
<td>41</td>
<td>325</td>
<td>352</td>
<td>46</td>
<td>90</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>88</td>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>73.09</td>
<td>78.92</td>
<td>26.91</td>
<td>21.08</td>
</tr>
<tr>
<td>K</td>
<td>S</td>
<td>IS</td>
<td>S</td>
<td>IS</td>
<td>S</td>
<td>IS</td>
</tr>
<tr>
<td>C-1</td>
<td>V</td>
<td>V</td>
<td>V</td>
<td>V</td>
<td>M</td>
<td>V</td>
</tr>
<tr>
<td></td>
<td>294</td>
<td>296</td>
<td>1339</td>
<td>1478</td>
<td>252</td>
<td>157</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>224</td>
<td>178</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>77.43</td>
<td>84.12</td>
<td>22.57</td>
<td>15.88</td>
</tr>
<tr>
<td>P</td>
<td>S</td>
<td>IS</td>
<td>S</td>
<td>IS</td>
<td>S</td>
<td>IS</td>
</tr>
<tr>
<td>C-1</td>
<td>V</td>
<td>V</td>
<td>V</td>
<td>V</td>
<td>M</td>
<td>V</td>
</tr>
<tr>
<td></td>
<td>39</td>
<td>43</td>
<td>325</td>
<td>400</td>
<td>46</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>88</td>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>73.09</td>
<td>88.86</td>
<td>26.91</td>
<td>11.04</td>
</tr>
</tbody>
</table>

The MATLAB result graphs of accuracy and false alarm ratio are shown in Fig.3.1. and Fig.3.2 respectively.

Figure 3.1 MATLAB Result Graph for Software Defect Prediction Accuracy
4. CONCLUSION

SVM and ISVM are implemented using MATLAB. The CM-1, KC-1 and PC-1 datasets are given as input. SVM and ISVM are trained with 50% dataset size. From the results, it is evident that ISVM attains better software defect prediction accuracy than that of SVM. Also, false alarm rate is reduced considerably.

REFERENCES