http://dx.doi.org/10.5516/NET.04.2011.023

RISKY MODULE PREDICTION FOR NUCLEAR I&C SOFTWARE

YOUNG-MI KIM[1] and HYEON SOO KIM[2*]
[1]Korea Institute of Nuclear Safety
P.O.Box 114, Yuseong-Gu, Daejeon, Korea, 305-600
[2]Dept. of Computer Science & Engineering, Chungnam Nat'l Univ
220 Gung-dong, Yuseong-Gu, Daejeon, Korea
[*]Corresponding author. E-mail : hskim401@cnu.ac.kr

Received April 06, 2011
Accepted for Publication October 09, 2011

As software based digital I&C (Instrumentation and Control) systems are used more prevalently in nuclear plants, enhancement of software dependability has become an important issue in the area of nuclear I&C systems. Critical attributes of software dependability are safety and reliability. These attributes are tightly related to software failures caused by faults. Software testing and V&V (Verification and Validation) activities are hence important for enhancing software dependability. If the risky modules of safety-critical software can be predicted, it will be possible to focus on testing and V&V activities more efficiently and effectively. It should also make it possible to better allocate resources for regulation activities. We propose a prediction technique to estimate risky software modules by adopting machine learning models based on software complexity metrics. An empirical study with various machine learning algorithms was executed for comparing the prediction performance. Experimental results show SVMs (Support Vector Machines) perform as well or better than the other methods.

KEYWORDS : Machine Learning, Safety-critical Software, Software Complexity Metrics, Software Dependability, Software Testing

1. INTRODUCTION

Software dependability is one of the most important features for nuclear I&C systems. Dependability can have several important attributes such as safety, reliability, security, etc [1]. Safety and reliability are tightly related to software failures that are caused by faults. High dependability often implies high investments for software testing, verification and validation.

A number of software complexity metrics are highly correlated with measures of quality such as fault count [2]. The likelihood that a software component will fail is directly related to the complexity of that software module. Software complexity metrics can thus be used as good predictors of software dependability. It is also important that software complexity metrics can be obtained early in the software life cycle.

Numerous studies focused on searching relationships between software metrics and software faults have been carried out. There are two categories for estimating software faults using software metrics, using statistical techniques and using machine learning. The models using statistical classification techniques include Discriminant Analysis[2] and Factor Analysis[3] and those using machine learning classification techniques include decision trees[4], artificial neural networks[5], support vector machines[6][7], etc. Many of these studies have dealt with finding the subsets of the software metrics that are most likely to predict the existence of faults. However, none of the metrics defined thus far can fully capture all aspects of software complexity and fault distribution. Researchers have recognized that the sets of metrics can capture software complexity better than any single metrics [8].

Early prediction of the risky modules of safety critical software makes it possible to focus on testing activities and regulation activities more efficiently. Many faults of software systems occur in only a few software components, and as such software verification and validation activities should focus on identifying and eliminating high risky problems that may be encountered during the software project [9].

SVM (Support Vector Machine) is known to generalize well even in high dimensional spaces under small training sample conditions and it is adaptive to model nonlinear functional relationships that are difficult to model with other techniques. It has been used successfully in many object recognition applications [10].

In this paper, we present a prediction model of risky modules during the early software lifecycle using machine

learning techniques and software complexity metrics. Complexity metrics of each software module are used as input variables for the prediction model. An input variable selection technique that uses a sensitivity analysis and correlation coefficients is suggested. An empirical study with various machine learning algorithms was executed using two NASA software projects, CM1 and PC1 datasets. The results show that the SVMs perform as well or better than the other machine learning algorithms.

The remainder of this paper is organized as follows. In chapter 2, we present related research for our modeling approach. SVM and other machine learning algorithms are introduced. In chapter 3, we present a methodology for risky module prediction. In chapter 4, we show the experimental results and discuss them. Finally, some concluding remarks are given in chapter 5.

2. RELATED RESEARCH

2.1 Support Vector Machines (SVMs)

Support Vector Machines (SVMs) are kernel based learning machines introduced by Vapnik [11]-[13]. SVMs use the structural risk minimization principle (SRM) for minimizing generalization error, and they can generalize high dimensional feature spaces using small training sample data sets

2.1.1 The Case When the Data Are Linearly Separable

Figure 1 shows the linear separation of two classes by SVM in two-dimensional spaces. The hyperplane corresponding to $w \cdot x + b = 0$ is the optimal hyperplane. The distance between $w \cdot x + b = 1$ and $w \cdot x + b = -1$ is the margin. Intuitively, however, we expect that a hyperplane with a larger margin will be more accurate at classifying future data tuples than a hyperplane with a smaller margin. The optimal hyperplane corresponds to the one that minimizes the training error and has the maximal margin. If the data are linearly separable, the separating hyperplane that creates the maximum distance between the plane and the nearest data is the optimal separating hyperplane. This is called the maximum marginal hyperplane.

The optimal hyperplane must satisfy the following constrained minimization:

$$\min\left\{\frac{1}{2}\|\mathrm{w}\|^2\right\} \tag{1}$$

s.t. $y_i(\mathrm{w} \cdot \mathrm{x}_i + b) \geq 1,\ i = 1, 2, \cdots, l$

where (x_i, y_i) is the training set and l is the number of training sets.

2.1.2 The Case When the Data Are Linearly Inseparable

If the data are not linearly separable, as in Figure 2, no straight line that would separate the classes can be found. In order to generalize to the case where the input spaces cannot be separated into the two classes properly, a hyperplane is established in the high dimensional feature space and the nonlinear classification is replaced by a linear classification problem.

If the dimensionality of the new feature space is sufficiently high, the data will always be linearly separable. The maximum marginal hyperplane found in the new space corresponds to a nonlinear separating hypersurface in the original space. For supporting nonlinear mapping into feature space, the kernel function is used. The kernel function $K(x_i, y_j)$ is defined as follows:

$$K(\mathrm{x}_i, \mathrm{x}_j) = \phi(\mathrm{x}_i) \cdot \phi(\mathrm{x}_j) \tag{2}$$

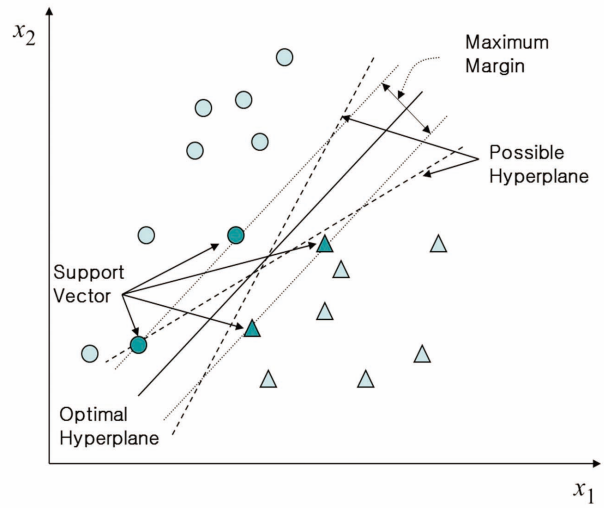

Fig. 1. Support Vectors

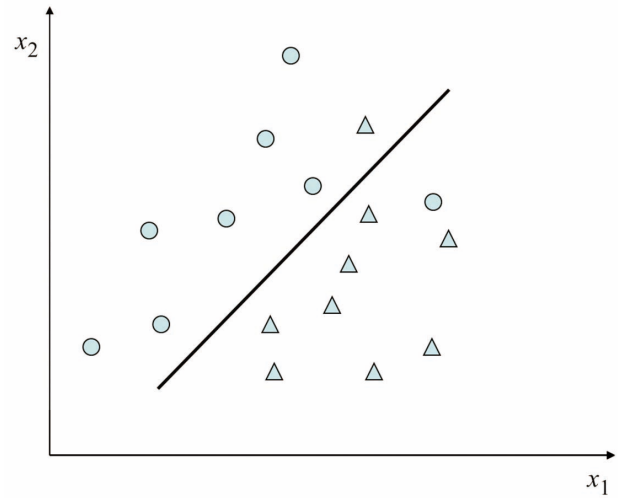

Fig. 2. Linearly Inseparable Data

Properties of the kinds of kernel functions that could be used to replace the dot product have been studied. The most common kernel functions are linear, polynomial, gaussian, and sigmoid.

These kernel functions are as follows:

$$\text{Polynomial: } K(\mathrm{x}_i, \mathrm{x}_j) = \left[(\mathrm{x}_i, \mathrm{x}_j) + 1\right]^q \tag{3}$$

$$\text{Radial basis (RBF): } K(\mathrm{x}_i, \mathrm{x}_j) = \exp\left(-\left\|\mathrm{x}_i - \mathrm{x}_j\right\|^2 / 2\sigma^2\right) \tag{4}$$

$$\text{Sigmoid: } K(\mathrm{x}_i, \mathrm{x}_j) = \tanh(v(\mathrm{x}_i, \mathrm{x}_j) + c) \tag{5}$$

There are no golden rules for determining which admissible kernel will result in the most accurate SVM. In practice, the chosen kernel does not generally have a large impact on the resulting accuracy. SVM training always finds a global solution, unlike neural networks such as backpropagation, where many local minima usually exist. Experiments for these three kernel functions will be performed in this research.

2.2 Other Machine Learning Algorithms

There are many classification and prediction methods such as decision tree induction, Bayesian classification, rule-based classification, and lazy classifier in addition to support vector machines. In this paper, we compare the performance of machine learning algorithms among support vector machines, *k*-Nearest Neighbor, naïve-Bayesian, and ensemble methods.

2.2.1 Naïve Bayesian Classification

Bayesian classifiers are statistical classifiers. They can predict class membership probabilities, such as the probability that given data belong to a particular class. Bayesian classification is based on Bayesian's theorem. One highly practical Bayesian learning method is the Naïve Bayesian learner, often called the Naïve Bayesian classifier. Bayesian classifiers also exhibit high accuracy and speed when applied to large datasets. Naïve Bayesian classifiers assume that the effect of an attribute value on a given class is independent of the values of the other attributes [14][15].

The naïve Bayesian classifier is based on the simplifying assumption that the attribute values are conditionally independent given the target value. In other words, the assumption is that, given the target value of the instance, the probability of observing the conjunction $a_1, a_2 \ldots a_n$ is just the product of the probabilities for the individual attributes: $P(a_1, a_2 \ldots a_n \mid v_j) = \prod_i P(a_i \mid v_j)$. Substituting this into Equation (6), we derive the approach used by the naïve Bayesian classifier.

$$v_{NB} = \arg\max_{v_v \in V} P(v_j) \prod_i P(a_i \mid v_j) \tag{6}$$

Various empirical studies of this classifier in comparison to the decision tree and the neural network classifiers have revealed it to be comparable in some domains. In theory, Bayesian classifiers provide the minimum error rate in comparison to all other classifiers. However, in practice this is not always true owing to inaccuracies in the assumptions made for its use, such as class conditional independence, and the lack of available probability data.

2.2.2 k-Nearest Neighbor

The classification methods discussed previously—support vector machines and Naïve Bayesian classification—are all eager learners. Eager learners will construct a generalization model before receiving new test data to classify. On the other hand, lazy learners simply store training data set and wait until test data arrives. When it obtains the test data, it then performs generalization in order to classify the data based on their similarity to the stored training data set [14][15].

The *k*-nearest neighbor method is an example of the lazy learner. It was first described in the early 1950s. Nearest neighbor classifiers are based on learning by comparing given test data with training data sets that are similar to the data. Given unknown data, a *k*-nearest neighbor classifier searches the pattern space for the *k* training data that are closest to the unknown data. The *k* training data are the k "nearest neighbors" of the unknown data. Nearest neighbor classifiers use distance-based comparisons that intrinsically assign equal weight to each attribute.

2.2.3 Ensemble Methods

Techniques for improving classification accuracy by aggregating the predictions of multiple classifiers are referred to as ensemble or classifier combination methods. Table 1 shows the general procedure for an ensemble method. An ensemble method constructs a set of base

Table 1. Algorithm for an Ensemble Method

General procedure for an ensemble method.
1. Let D denote the original training data, k denote the number of base classifiers, and T be the test data.
2. **for** $i = 1$ to k **do**
3. Create training set, D_i from D.
4. Build a base classifier C_i from D_i.
5. end for
6. **for** each test record $x \in$ T **do**
7. $C^*(x) = Vote(C_1(\mathrm{x}), C_2(\mathrm{x}), \ldots, C_k(\mathrm{x}))$
8. end for

classifiers from the training data and performs classification by voting on the predictions made by each base classifier [16].

Bagging is a typical ensemble method. Bagging, also known as bootstrap aggregating, is a technique that repeatedly samples from a data set according to a uniform probability distribution. Each bootstrap sample has the same size as the original data. Because the sampling is done with replacement, some instances may appear several times in the same training set, while others may be omitted from the training set.

The bagged classifier often has significantly greater accuracy than a single classifier derived from the original training data. It is also more robust to the effects of noisy data. The increased accuracy occurs because the composite model reduces the variance of the individual classifiers [15].

2.3 PCA and Sensitivity Analysis

The selection of software metrics that will be used an input variables is important for the performance of a prediction model. Software complexity metrics have been shown to be closely related to the distribution of faults in program modules [2]. Also, there is a phenomenon in which performance degrades as the number of inputs increases [17][18]. Dimensionality reduction techniques have been used to obtain better prediction models. One of the most common methods for dimensionality reduction is Principal Components Analysis (PCA) [19]. Many software metrics have a high correlation with each other. PCA transforms raw data into variables that are not correlated to each other. While PCA is very useful, its weakness lies in the difficulty of intuitive understanding about induced dimensions.

Another method for dimension reduction is sensitivity analysis [20]. Sensitivity analysis methods estimate the rate of change in the output of a model as a result of varying the input values. The estimated values can be used to decide which value will be selected. However, in case of when using a sensitivity analysis, it is possible to have a set of highly related complexity metrics. The sensitivity analysis method which is used in Gondra's research [21] analyzes the criticality of each input variable. Elish used a selection method based on correlation among input variables [22]. This is called the correlation based feature selection technique (CFS) [23]. All combinations of input variables are used for selecting the best choice of the estimated results.

3. METHODOLOGY

In this, we are going to predict the risky modules using software complexity metrics. Figure 3 shows the basic concept of this methodology.

The software metric repository has various sets of software metrics. It provides a training data set for a

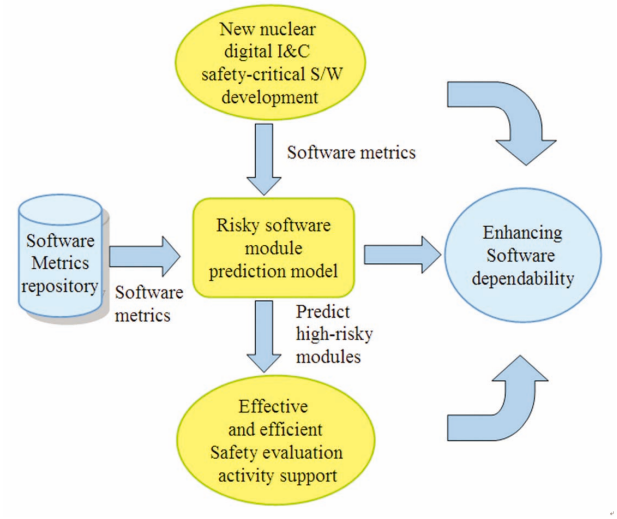

Fig. 3. Basic Concept

prediction model. If new nuclear safety-critical software is developed, we can predict the risky modules using its complexity metrics and the similar data sets which were collected from the previous projects. Using the software modules predicted with to be highly risky, it is possible to execute effective and efficient safety evaluation activities. The final aim of this research is to enhance the dependability of safety-critical software.

The main procedure for our study is as follows:

- select input variables from software metrics
- determine risk levels for software modules (ex, High Risk Module or Low Risk Module)
- predict software module's risk level
- evaluate prediction results with performance measures

There are many kinds of software metrics that can be used input variables for machine learning. As there is a phenomenon in which performance degrades as the number of inputs increases, the selection of software metrics is important for the performance of a prediction model. Risk level is defined as the value that presents risk that defines the importance of the software module to the user. It is necessary to determine the risk level for each software module. Next, we can predict a software module's risk level with selected input variables. Lastly, we evaluate execution results with performance measures.

3.1 Selection of Input Variables

In our research, complexity metrics of each software module are used as input variables for prediction model. As stated previously, both PCA[19] and sensitivity analysis[20] have some weak points for the risky module classification. We use sensitivity values for F1 measures and correlation coefficients of input variables. The sensitivity analysis for F1 values quantifies the importance of each

input variable. The F1 measure is selected for the sensitivity analysis because it is a composite measure wherein precision and recall are equally considered by taking their harmonic mean. This sensitivity analysis method estimates the rate of change of the F1 measure as a result of varying the input values. The rate of change can be used to determine the importance of each input variable. If the rate of change of the F1 measure is large, the relevant input variable is an important feature for the prediction model. Original data will be normalized before calculating the sensitivity value. Given input vectors $X = \{x_i\}_1^n$, where $x_i \in \Re^d$, for a SVM with the F1 measure $y = f(X)$, $X = \{x_i\}_1^n$, the sensitivity of F1(*SoF*) measure for each input variable x_j is defined as

$$SoF_j = 100 \times \left| f(X) - f(X + \Delta_{ij}) \right| \quad (7)$$

subject to $X + \Delta_{ij} = \{x_i + \Delta_{ij}\}_1^n$; $i = 1, 2, ..., n$

where n is the number of data sets, d is dimension of input variables, and Δ_{ij} is a small value added to the x_j of x_i.

After executing the sensitivity analysis, we calculate the correlation coefficient matrix of input variables. The correlation coefficient represents the normalized measure of the strength of the linear relationship between input variables. The correlation coefficients range from -1 to 1, where

- Values close to 1 suggest that there is a positive linear relationship between two input variables.
- Values close to -1 suggest that there is a negative linear relationship between two input variables.
- Values close to or equal to 0 suggest there is no linear relationship between two input variables.

Using the correlation coefficient, we can determine the redundancy of input variables. If two input variables are significantly correlated to each other, we can discard one of them. First, we choose the first input variable from the START list, which contains the input variables in decreasing order of sensitivity. This variable has the largest sensitivity among the total input variables. It is added to the SELECTED list and is removed from the START list. Next, we remove the next largest one from the START list. If the absolute values of correlation coefficients between the next largest one and pre-selected input variables in the SELECTED list do not exceed the threshold β, then it is added to the SELECTED list. Otherwise it is discarded. This process is repeated until the START list is empty. Table 2 shows the basic steps of the selecting input variables. By modifying the threshold, we can adjust the number of input variables for the prediction model. In our research, we conducted empirical studies that evaluated several machine learning algorithms with threshold β values of 0.4, 0.5, 0.6, and 0.7.

Table 2. Basic Algorithm for Selecting Input Variables

1. Begin with the **START** list containing the input variables in decreasing order of sensitivity.
2. Calculate the correlation coefficients matrix $A_{n \times n}$ of input variables
3. Let **SELECTED** is an empty list.
4. Let v_j is the highest one in **START**. If $\forall v_k \in$**SELECTED**, $\left| A = [a_{jk}] \right| < \beta$, then add it to **SELECTED**. Remove v_j from **START**.
5. If **START** is not empty, go to 4.
6. Return **SELECTED**.

3.2 Determination of a Risk level

Risk is the combination of the probability of an abnormal event or failure and the consequence(s) of that event or failure to a system's components, operators, users, or environment [24]. In this paper, we define risk level. It is a value representing risk that defines the importance of the software module to the user. The modules predicted as a high risk level will be called high risk modules. Severity presents the degree of impact that a requirement has on a system. Severity level of the module quantifies the impact of a fault on the overall system. Fault count is the number of faults that occurred in the module. For example, severity level 5 may imply that the defect caused a loss of functionality without a workaround, whereas severity 1 may mean that the impact is superficial and does not cause any major disruptions to the system.

The value of the severity level should be determined by domain or application experts. For example, in the nuclear field, graded requirements for nuclear software can be used for determination of severity level. There are several standards and regulatory guidance for determining software grade.

The risk level of a software module m is defined as follows:

$$Risk\_Level_m = Severity\_Level_m * (1 + Fault\_Count_m) \quad (8)$$

where,

- $Risk\_Level_m$: A value representing risk that defines the importance of the software module to the user.
- $Severity\_Level_m$: Severity presents the degree of impact that a requirement has on a system. The severity level of the module quantifies the impact of a fault on the overall system.
- $Fault\_Count_m$: Fault count is the number of faults that occurred in the module.

Fault count is the number of faults that occurred in the module. Even if the fault is corrected, the fault count is increased by one. Table 3 shows the relationship between risk level and a risky module. The value of α is related to the number of faults and the severity level of each module.

Table 3. Estimated Risk Level

$Risk\_Level_m$	Estimated Risk Level
$Risk\_Level_m \geq \alpha$	*High Risk Module*
$Risk\_Level_m < \alpha$	*Low Risk Module*

Table 4. Confusion Matrix

		Predicted	
		Low Risk	High Risk
Actual	Low Risk	TN	FP
	High Risk	FN	TP

Table 5. The Number of Risky Modules

Dataset	Total # of modules	*# of High Risk Modules*	*# of Low Risk Modules*
CM1	505	*105*	*175*
PC1	1107	*400*	*932*

For example, if a module's fault count is 1 and the severity level is 1, the risk level is 2. Then, is the module risky? We cannot answer this question with an unqualified "yes". Rather, it is dependent on the characteristics of the domain and the application for which relevant software will be used. The value of α should be determined by domain or application experts. In our research, the experiments are performed with α =2.

3.3 Measurements of Performance

In this study, we used the following prediction performance measures: accuracy, precision, recall and F-measure. These are commonly used measures for evaluating and comparing prediction models quantitatively [25]. A confusion matrix is used for deriving these measures. The confusion matrix contains information about the actual and the predicted classification done by a machine learning algorithm. Table 4 shows the confusion matrix.

The entries in the confusion matrix have the following meaning in the context of our study:

- TN is the number of correct predictions that an instance is low risk,
- FP is the number of incorrect predictions that an instance is high risk,
- FN is the number of incorrect of predictions that an instance low risk, and
- TP is the number of correct predictions that an instance is high risk.

The accuracy (*A*) is the proportion of the total number of predictions that were correct. It is calculated using the following equation:

$$\text{Accuracy(A)} = \frac{TP + TN}{TP + TN + FP + FN} \tag{9}$$

The precision (*P*) is the proportion of the predicted high risk modules that were correct, as calculated using the following equation:

$$\text{Precision}(P) = \frac{TP}{TP + FP} \tag{10}$$

The recall (*R*) is the proportion of high risk modules that were correctly identified, as calculated using the following equation:

$$\text{Recall}(R) = \frac{TP}{TP + FN} \tag{11}$$

F1 is a composite measure wherein precision (*P*) and recall (*R*) are equally considered, as calculated using the following equation:

$$F1 = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}} \tag{12}$$

4. EXPERIMENTS

4.1 Preparation

Two NASA datasets acquired from NASA IV&V Facility Metrics Data Program (MDP) are used for this study [26]. We selected two safety-critical NASA software projects, CM1 and PC1 datasets. CM1 is NASA spacecraft instrument software consisting of 20KLOC. PC1 is flight software for an earth orbiting satellite consisting of 40 KLOC. Both are written in C language. They collected metric values of each software module, which have been gathered using McCabe IQ. MDP provides software metrics and associated fault information for general users. Table 5 provides the number of high risk modules and low risk modules with α =2 that are obtained from the CM1 and PC1 datasets. The CM1 dataset has 26.3% *high risk modules* and the PC1 dataset has 18.8% *high risk modules*.

The sensitivities of F1 measures over the normalized data sets are calculated. SVM with a RBF kernel is used for calculating F1 measures. Average the values of sensitivity, results using three Δ values (0.05, 0.1, and 0.2) are used for executing the algorithm, which is described in Table 2. Table 6 and Table 7 show the results of the input

Table 6. The Results of Input Variable Selection (CM1 Dataset)

Selected input variable	β=0.4	β=0.5	β=0.6	β=0.7
LOC_EXECUTABLE	✓	✓	✓	✓
PARAMETER_COUNT	✓	✓	✓	✓
CYCLOMATIC_DENSITY	✓	✓	✓	✓
LOC_BLANK				✓
SEVERITY_LEVEL	✓	✓	✓	✓
PATHOLOGICAL_COMPLEXITY	✓	✓	✓	✓
MAINTENANCE_SEVERITY	✓	✓	✓	✓
HALSTEAD_LEVEL			✓	✓
ESSENTIAL_DENSITY	✓	✓	✓	✓
LOC_CODE_AND_COMMENT				✓
DECISION_DENSITY	✓	✓	✓	✓

Table 7. The Results of Input Variable Selection (PC1 Dataset)

Selected input variable	β=0.4	β=0.5	β=0.6	β=0.7
CYCLOMATIC_DENSITY	✓	✓	✓	✓
LOC_COMMENT	✓	✓	✓	✓
HALSTEAD_EFFORT	✓	✓	✓	✓
MODIFIED_CONDITION_CNT			✓	✓
MAINTENANCE_SEVERITY	✓	✓	✓	✓
DECISION_DENSITY			✓	✓
DESIGN_DENSITY	✓	✓		✓
PATHOLOGICAL_COMPLEXITY			✓	✓
LOC_BLANK			✓	✓
CALL_PAIRS	✓	✓	✓	✓
HALSTEAD_LEVEL	✓	✓	✓	✓
LOC_CODE_AND_COMMENT			✓	✓
NUM_UNIQUE_OPERATOR				✓
ESSENTIAL_DENSITY	✓	✓	✓	✓
HALSTEAD_CONTENT		✓	✓	✓
SEVERITY_LEVEL	✓	✓	✓	✓
PARAMETER_COUNT	✓	✓	✓	✓

variables selecting algorithm. β is the threshold for determining the dependency between two complexity metrics. When $\beta = 0.7$, CM1 dataset has 11 selected input variables, such as LOC_EXECUTABLE, PARAMETER_COUNT, CYCLOMATIC_DENSICY, LOC_BLACK, etc. In the case of $\beta = 0.4$, it has 8 selected input variables, because it excludes one of the two complexity metrics which have a correlation coefficient between them above the β value. Experiments are performed with various β values (0.4, 0.5, 0.6, and 0.7). The reduced dataset is used to implement the risky module prediction model. These data sets are shuffled and normalized before classification in order to enhance performance.

We use 5-folder cross validation to prevent over-fitting. Over-fitting occurs when a statistical model describes random error or noise instead of the underlying relationship. Generally, a model that has over-fitting will have poor predictive performance. The initial data are randomly partitioned into 5 mutual exclusive "folds," each being of approximately equal size. Training and testing are performed 5 times, respectively. Each sample (fold) is used the same number of times for training and once for testing.

4.2 Discussion

The major goals of this paper are to evaluate the capability of our SVM based prediction model in predicting high risk software modules. Table 8 and Figure 4 show the prediction results of the CM1 dataset. When $\beta = 0.6$, bagged SVM ensemble outperforms all the other machine learning algorithms in accuracy and F1. However, when $\beta = 0.4$, it is observed that single SVM achieves higher performance than the other machine learning algorithms. In this case, the accuracy and F1 measure of the CM1 dataset are 95.1% and 81.8%, respectively. This means that 95.1% of the software modules are classified correctly in the CM1 dataset. F1 is a composite measure of the proportion of the predicted high risk modules that were correct and the proportion of the high risk modules that were correctly identified. The composite measure here is 81.8%. From Table 9 and Figure 5, we can show the prediction results of the PC1 dataset. When $\beta = 0.4$, it is observed that the single SVM shows the best performance in terms of accuracy and F1. In this case, the percentage of correct classification is 95.4% and F1 is 88.1%.

In general, the results show that bagged SVM ensembles are not always better than the single SVM classifier. In some cases, single SVMs are better than bagged SVM ensembles. However, when we consider the average accuracies and the F1 measures, the experimental results show the performances of single SVMs and bagged SVM ensembles are better than those of other machine learning algorithms such as kNN and naïve Bayesian.

β is a threshold for determining dependency between two complexity metrics. Usually, if β has a small value, the number of selected input variables is decreased. The empirical results show that the prediction performance is not significantly affected by a specific β value. When $\beta = 0.4$ or $\beta = 0.5$, it is observed that the performances of the single SVM and bagged SVM are slightly better than with

other β values. This means that the procedure for input variable reduction is useful for risky module prediction.

Figure 6 and Figure 7 summarize the performance of various kernel functions with the single SVM as measured by selected software metrics. In the case of the RBF kernel, the performances are significantly better than those of

Table 8. Accuracy and F1 Value for CM1 Dataset

	β=0.4		β=0.5		β=0.6		β=0.7	
	Accuracy	F1	Accuracy	F1	Accuracy	F1	Accuracy	F1
Single SVM	95.1%	81.8%	95.1%	81.4%	95.0%	81.2%	95.0%	81.1%
Bagged SVM	94.6%	80%	95.3%	80%	95.3%	83.9%	95.3%	83.7%
kNN	93.5%	77.3%	93.8%	77.0%	93.6%	79.2%	94.2%	80.0%
Naïve Bayesian	93.0%	77.2%	93.4%	78.2%	92.8%	77.2%	91.7%	73.8%

Table 9. Accuracy and F1 Value for PC1 Dataset

	β=0.4		β=0.5		β=0.6		β=0.7	
	Accuracy	F1	Accuracy	F1	Accuracy	F1	Accuracy	F1
Single SVM	95.4%	88.1%	95.1%	87.1%	94.7%	86.1%	94.7%	86.4%
Bagged SVM	95.0%	87.1%	95.2%	88.7%	94.8%	87.8%	94.1%	86.0%
kNN	94.4%	85.3%	94.0%	84.3%	93.4%	82.7%	93.2%	82.2%
Naïve Bayesian	91.5%	79.8%	90.2%	77.0%	87.3%	69.7%	87.5%	69.4%

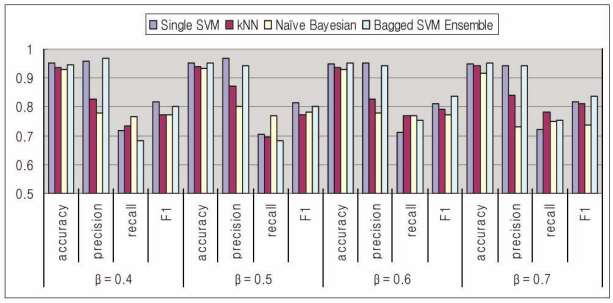

Fig. 4. Performance of Various Machine Learning Methods (CM1 Dataset)

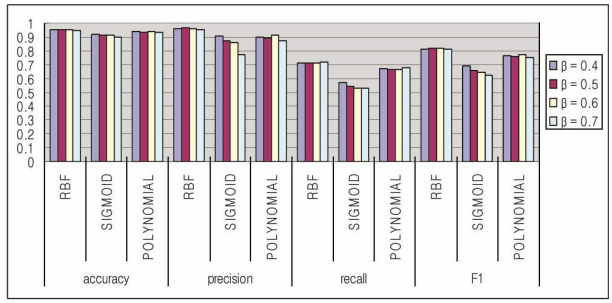

Fig. 6. Performance of Various Kernel Functions with SVM (CM1 Dataset)

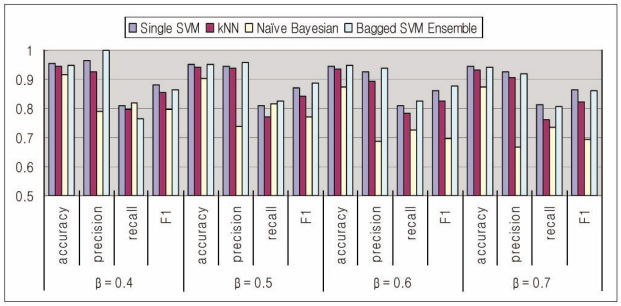

Fig. 5. Performance of Various Machine Learning Methods (PC1 Dataset)

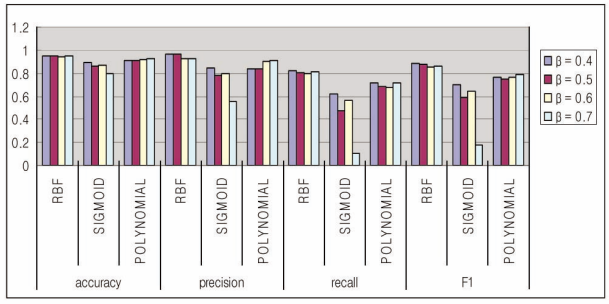

Fig. 7. Performance of Various Kernel Functions with SVM (PC1 Dataset)

any other kernels for the two datasets. We find that the RBF kernel is very useful for classifying software risky modules.

5. CONCLUSION

In this paper, we proposed a prediction model for detecting high risk modules based on machine learning techniques and software complexity metrics. The major goal of this work is to evaluate the capability of SVM based prediction models in predicting high risk software modules. Software modules that are predicted as being of high risk can be targets for intensive software testing, V&V, and regulatory activity.

The basic technique we have developed in this study relates to a classification problem: risky software modules are classified using the observed meaningful predictors in terms of the number of errors that these modules might contain and the severity levels that were assigned by the developer during the requirement phase. Software complexity metrics would be useful as numerical measures and could be obtained prior to the test and verification of a program.

The proposed model has two phases. In the first phase, input features are selected. The sensitivity analysis for F1 measures and correlation coefficients of complexity metrics are used in the first phase. The main advantages of this approach are that the use of selected metrics is intuitive and that the redundancy between software metrics can be reduced. Selected metrics also can be used to enhance software dependability during the overall software lifecycle. In the second phase, the risky modules are predicted using various machine learning algorithms and the selected software complexity metrics.

An empirical study was performed to evaluate the capability of various machine learning algorithms in predicting risky software modules and to compare their prediction performance. It was performed using software metrics data of two NASA software projects; CM1 and PC1. Contrary to our expectations, bagged SVM ensemble methods did not always yield better performance than the single SVM. However, when we consider average accuracies and F1 measures, the performances of single SVMs and bagged SVM ensembles were better than those of other machine learning algorithms. Also, we found that the RBF kernel is very useful for classifying risky software modules.

For all classification error rates, we expect that the results predicted by machine learning algorithms can be useful and practical for software testing, V&V, and activities for regulatory reviews. The main drawbacks of the proposed model are the lack of fault data and a relevant complexity data set. The present experimental study for our model was performed using restricted datasets from NASA. To enhance practicality, it is essential to test the model using various application software used in the nuclear sector. These are possible research directions for future work.

REFERENCES

[1] Michael R. Lyu, *Handbook of Software Reliability Engineering*, McGraw-Hill

[2] Briand, L.T., Basili, V.R., Hetmanski, C., *Developing interpretable models for optimized set reduction for identifying high-risk software components*. IEEE Transactions on Software Engineering SE-19(11), 1028-1034, 1993.

[3] Khoshgoftaar, T.M., Munson, J.C., *Prediction software development errors using complexity metrics*. IEEE Journal on Selected Areas in Communications 8 (2), 153-261, 1990

[4] Porter, A., Selby, R., Empirically guided software development using metric-based classification trees. IEEE Software 7(2), 46-54, 1990.

[5] Khoshgoftaar, T.M., Lanning, D.L., Pandya, A.S., *A comparative study of pattern recognition techniques for quality evaluation of telecommunications software*, IEEE Journal on Selected Areas in Communications 12 (2), 208-217, 1994.

[6] Xing, F., Guo, P., Lyu, M.R., *A novel method for early software quality prediction based on support vector machine*, Proceedings of IEEE International Conference on Software Reliability Engineering, pp. 213-222, 2005

[7] Young-Mi Kim, Choong-Heui Jeong, A-Rang Jeong and Hyeon Soo Kim, Risky Module Estimation in Safety-Critical Software, IEEE/ACIS International Conference on Computer and Information Science, 2009,6

[8] T.J. McCabe. *A complexity measure*. IEEE Transactions on Software Engineering, 2(4):308-320, Dec. 1976.

[9] B. Boehm. *Software Engineering Economics*. Prentice-Hall, 1981.

[10] H. Drucker, D. Wu, and V. N. Vapnik, *Support vector machines for spam categorization*, IEEE Transactions on Neural Networks, vol. 10, no. 5, pp. 1048-1054, 1999.

[11] Burges, C., *A Tutorial on Support Vector Machines for Pattern Recognition*, Data Mining and Knowledge Discovery, 1998

[12] T. Joachims, *Making large-Scale SVM Learning Practical. Advances in Kernel Methods - Support Vector Learning*, B. Schölkopf and C. Burges and A. Smola (ed.), MIT-Press, 1999

[13] Steve R, Gunn, *Support Vector Machines for Classification and Regression*, Technical Report, University of Southampton, 10 May 1998

[14] Tom M. *Mitchell, Machine Learning*, McGRAW-Hill, 1997

[15] Jiawei Han, *Data Mining: Concepts and Techniques*, University of Illinois at Urnaba-Champaign, Elsevier

[16] Pang-Ning Tan, Michael Steinbach and Vipin Kumar, *Introduction to Data Mining*, Addison Wesley

[17] Taghi M. Khoshgoftaar and John C. Monson, *Prediction Software Development Errors Using Software Complexity Metrics*, IEEE Journal on Selected Areas in Communications. Vol. 8, No. 2, Feb 1990

[18] Bellman, *Adaptive Control Processes*, Princeton University Press.

[19] Taghi M. Khoshgoftaar and Kalai S. Kalaichelvan, *Detection of Fault-Prone Program Modules in a Very Large Tele-*

communications System, 16th International Symposium on Software Reliability Engineering, 1995
[20] Tiong-Hwee Goh, *Semantic Extraction Using Neural Network Modelling and Sensitivity Analysis*, Proceedings of International Joint Conference on Neural Networks, 1993
[21] I. Gondra, *Applying machine learning to software fault-proness prediction*, The Journal of Systems and Software 81 (2008) 186-195
[22] Karim O. Elish, Mahmoud O. Elish, *Predicting defect-prone software modules using support vector machines*, The Journal of Systems and Software 81(2008) 649-660
[23] Mark A. Hall, *Correlation-based Feature Selection for Descrete and Numeric Class Machine Learning*, Proceedings of the 17'th International Conference on Machine Learning, 2000
[24] IEEE Std. 1012, *IEEE Standard for Software Verification and Validation*, 2004
[25] Witten, I., Frank, E., Data Mining: Practical Machine Learning Tools and Techniques, Morgan Kaufmann, San Francisco, 2005.
[26] http://sw-assurance.gsfc.nasa.gov/disciplines/reliability/index.php