A Unified Framework for Defect Data Analysis Using the MBR Technique

Venkata U.B. Challagulla, Farokh B. Bastani, I-Ling Yen
Department of Computer Science
University of Texas at Dallas
{uday, bastani, ilyen}@utdallas.edu

Abstract

Failures of mission-critical software systems can have catastrophic consequences and, hence, there is strong need for scientifically rigorous methods for assuring high system reliability. To reduce the V&V cost for achieving high confidence levels, quantitatively based software defect prediction techniques can be used to effectively estimate defects from prior data. Better prediction models facilitate better project planning and risk/cost estimation.

Memory Based Reasoning (MBR) is one such classifier that quantitatively solves new cases by reusing knowledge gained from past experiences. However, it can have different configurations by varying its input parameters, giving potentially different predictions. To overcome this problem, we develop a framework that derives the optimal configuration of an MBR classifier for software defect data, by logical variation of its configuration parameters. We observe that this adaptive MBR technique provides a flexible and effective environment for accurate prediction of mission-critical software defect data.

1. Introduction

A wide range of industries are becoming increasingly dependent on complex software-intensive systems. This include nuclear power plant control systems, defense systems, command and control systems and embedded systems in medical devices. A common attribute of these systems is that they perform real-time complex and critical tasks that require extremely high-dependability, where a single failure may result in catastrophic loss of life or property. Hence, to prevent such failures, techniques capable of assessing the dependability of these systems to a high degree of confidence are critically required.

Generally, dependability assessment is performed using either quantitative (statistical and machine learning) prediction techniques or verification and validation methods (including testing and formal methods). However, assuring the dependability of the entire software system using testing or formal methods can be expensive. Formal verification is difficult to apply to complex programs and cannot cope with specification faults. Testing can require a long time (estimated to be over 10,000 years) and huge costs to achieve a reasonable confidence in the correctness of a safety-critical system. Applying quantitative based defect prediction techniques on stored failure data provides a very useful and important tool to gauge the likely quality and maintenance effort for software systems before they are developed and even after their deployment. The task of eliminating defects, and thereby improving dependability, becomes simpler if we could predict them at an early stage in the software development cycle.

Memory based reasoning technique [Sta86] [Wal87] is one such quantitative method that predicts a new case by retrieving similar cases from the past. It uses the past cases to predict the solution to the current problem. If necessary, it revises the proposed solution and saves it to form a new case. The following steps describe the Memory based learning process [Kas95]:

1) Measurement Repository. This is a stored database of all the previous records to be used for analysis. The user specifies the features to be selected for prediction.

2) Distance function. This function calculates the dissimilarity between the attributes (or an associated function) of the training and the target record. While calculating dissimilarity, the relative influence of each attribute is regulated by using an appropriate attribute weighting function.

3) Number of Neighbors. In this step, the number of neighbors to be used for analysis is determined.

4) Prediction using a Combination function. This function aggregates the results from the nearest neighbors to predict the output.

The memory-based classifier can be instantiated in a number of ways by varying its configuration parameters as described above, including using different distance methods, different weights, different combination techniques, and different number of nearest neighbors. However, there is no systematic method for choosing an appropriate value for these parameters that gives the best prediction for the task at hand.

In this paper, we derive a framework that provides a flexible and effective environment for mission-critical software defect prediction using the MBR method.

Proceedings of the 18th IEEE International
Conference on Tools with Artificial Intelligence (ICTAI'06)
0-7695-2728-0/06 $20.00 © 2006 IEEE

Essentially, the framework guides the user through the logical process of selecting the appropriate MBR configuration that gives the best prediction depending on the software defect data being analyzed. The framework is constructed using real-time defect data sets obtained from NASA's MDP (Metrics Data Program) data repository.

The rest of this paper is organized as follows. Section 2 provides a more in-depth description of the MBR technique and the related works on using different MBR classifiers for prediction. Section 3 describes the datasets used for constructing the framework. Section 4 describes the assessment criteria used to compare the different classifiers. Section 5 describes the results of the study and the evolution of the MBR framework. Section 6 summarizes the paper.

2. MBR classifications and related work

Memory-Based Reasoning (MBR) is a problem solving and learning paradigm similar to other AI approaches, such as the use of Fuzzy Logic or Neural Networks. MBR classification methods are widely used in different domains including natural language parsing [Sta86], computational biology [Cos93], and software agents [Mae93], to name a few. In this paper, we follow the approach of applying MBR to the prediction of fault-prone modules for mission-critical systems.

Software defect prediction models predominantly focus on either predicting the number of faults ("Error Count") in a module or predicting whether the module is faulty or not. We define a module to be "faulty" if it has one or more faults. While the "Error Count" attribute can be approximated to a continuous or discrete variable, we believe that both these assumptions might not hold in practice. The range of values of the attribute is not very large, making the continuous assumption invalid. Also, the number of data points in each of the discrete intervals is abnormally variant, making the discrete distribution widely skewed. Hence, a better way would be to analyze the modules as faulty or faultless. In this way, we have only two classes of data, which is the ideal case for the application of classification prediction technique such as MBR. It has also been shown that for all practical purposes, it is sufficient to predict whether a module is faulty or not for software quality assessment [Mun92].

While there has been some work on the application of MBR to software quality analysis [Pau02], the majority of software defect prediction analysis is carried out using Case-based reasoning (CBR) technique. MBR and CBR both belong to the class of Instance-based learning methods. While CBR primarily focuses on indexing selection for case retrieval process, MBR focuses on parallel retrieval without domain knowledge [Aam94]. However, using the classic Nearest Neighbor technique (NN) [Das91] with variant distance functions for retrieving cases close to the target case is common to all the Instance-based learning methods. CBR and MBR methods differ in the usage of the number of input features for the NN calculation. While MBR uses all the features in a case for NN calculation, CBR determines the relevant features for NN based on the indexing method used. In this paper, we use all the features for NN calculation and build prediction models with no domain knowledge, which is the essence of the MBR technique. Because of the above similarities between CBR and MBR techniques, we provide the literature survey of both these techniques with regard to software quality prediction.

Khoshgoftar [Kho97] applied a CBR model with Euclidean distance function for predicting fault-prone modules. The attributes are not assumed to be of equal weights. Instead, weights are chosen to be coefficients of a multiple linear regression model. [Kho97] showed that CBR performs better than discriminant analysis for type II misclassifications, which assigns fault-prone modules to the not fault-prone group. Emam, et al., [Ema99] compared different case-based reasoning classifiers for predicting high risk software components and concluded that there is no added advantage in varying the combination of parameters (including varying the number of nearest neighbors and using different weight functions) of the classifier to make the prediction accuracy better. Shepperd, et al., [She01] evaluated regression, rule induction, case-based reasoning, and neural networks for building software prediction models using simulated data and concluded that it is better to build a prediction system based on the "context" at hand rather than a universal "best" prediction system. They showed that the CBR method performed the best if the software data has outliers and collinearity.

3. Characteristics of Software Data

The real-time defect data sets used in this paper are shown in Table 1. They can be accessed from NASA's MDP (Metric Data Program) data repository, available online at http://mdp.ivv.nasa.gov. All these data sets varied in the percentage of defect modules.

The prediction confidence of any software prediction analysis is proportional to the number of data sets available for analysis. We also notice that the data size of JM1 data set is substantially larger than other data sets. So, we decided to create more data sets by random splitting of the JM1 data set into multiple data sets of different sizes. Random splitting ensured that the ratio of faulty to non-faulty modules of the subset JM1 datasets remained nearly the same as the original JM1 dataset. Table 2 shows the data sets considered for our analysis with their sizes and percentages of defective modules. The varying data sizes helps in better analysis of the

performance of MBR prediction process across a range of small to very large data sets. Each of those data sets is further divided in a ratio of 7:3, with 70% used for training and the remaining 30% used as testing data.

Table 3 shows the different types of predictor software metrics (independent variables) used in our analysis. These complexity and size metrics [Fen97] include well known metrics, such as Halstead, McCabe, Line count, Operator/operand count, and Branch count.

4. Assessment Criteria

The "Confusion Matrix" [Koh98] and its associated metrics are used to assess the performance of the MBR technique. The "Confusion Matrix" describes the classifications and misclassifications of a classifier in a tabular format. Table 4 describes the confusion matrix in the context of analyzing software defects.

The associated metrics used in our analysis are,

pd = Probability of detection. This is the conditional probability of correctly detecting that a module is defective, given that the module is actually defective.

$$pd = tp/(tp+fn)$$

pf = Probability of false alarm. This is the conditional probability of incorrectly detecting that a module is defective, given that the module is actually non defective.

$$pf = fp/(fp+tn)$$

pt = Probability of missing a true alarm. This is the conditional probability of incorrectly detecting that a module is non defective, given that the module is actually defective.

$$pt = fn/(tp+fn)$$

acc = Accuracy. This is probability of correctly predicting that a module is defective or not.

$$acc = (tn+tp)/(tn+fn+tp+fp)$$

In an ideal defect prediction case, all the faulty modules should be classified as faulty and no non-faulty be misclassified as faulty i.e., %pf = 0 and %pd = 100. So, we expect the learner to maximize "pd", minimize "pf" and give a high "acc". However, since we evaluate MBR prediction process for mission-critical software systems and considering the fact that any failures of these systems are catastrophic, maximizing "pd" becomes a priority even at the cost of higher "pf".

Table 1. NASA defect data sets

Project	# modules	% defects	Lang	Notes
CM1	505	9.5 %	C	spacecraft instrument
JM1	10878	19.3 %	C++	Real-time ground system
KC1	2107	15.4 %	C++	Science data processing
PC1	1107	6.8 %	C	Flight software

Table 2. Data sets used in our analysis

Data	# modules	% with defects	Data	# modules	% with defects
CM1	505	9.5%	KC1	2107	15.4%
JM1a	774	18.1%	JM1e	2324	18.5%
PC1	1107	6.8%	JM1f	2828	19.2%
JM1b	1191	21.8%	JM1g	3480	19.8%
JM1c	1668	19.4%	JM1h	5003	19.1%
JM1d	2093	19.3%			

Table 3. Metric groups used in analysis

Metric Type	Metric	Definition
McCabe	CC	Cyclomatic Complexity
	EC	Essential Complexity
	DC	Design Complexity
	ELOC	Lines of Code
Halstead	N	Length
	V	Volume
	D	Difficulty
	I	Intelligent Content
	E	Effort
	B	Error Estimate
	L	Level
	T	Programming Time
Line Count	BLOC	Lines of Blank
	CLOC	Lines of Comment
	CCLOC	Lines of Code and Comment
Operator / Operand	UNOD	Unique Operands
	UNOT	Unique Operators
	NOD	Total Operands
	NOT	Total Operators
Branch	BC	Branch Count

Table 4. Confusion Matrix for defect data

		Actual	
		Non-Defective	Defective
Predicted	Non-Defective	tn	fn
	Defective	fp	tp

tn is the number of true negatives, i.e., predicting non-defective modules as non-defective

fn is the number of false negatives, i.e. predicting defective modules as non-defective

fp is the number of false positives, i.e., predicting non-defective modules as defective

tp is the number of true positives, i.e. predicting defective modules as defective

5. Results

As observed in the earlier sections, there are many independent configuration parameters that influence the prediction capabilities of the MBR classification method. In this section, we will evaluate these configuration parameters in a logical order that helps us in eventually building a generalized MBR framework.

Any MBR data analysis should begin with some default values for the number of nearest neighbors, combination function, weighted function, and the distance metric. These parameters will be incrementally tuned on

the basis of the results obtained as we evolve through building the framework. As the first step, we assume that all the attributes are weighted equally and the combination function weights the nearest neighbors by their distances, with weights decreasing as the distance of the neighbor increases from the target.

5.1. Choose the distance measures

There is lot of literature [Wil97][Lia98] on using NN with variant distance functions for improving the accuracy of Instance-based learning methods. Of particular interest is the application of NN to MBR process. If ***x*** and ***y*** are the two input vectors representing training and test data respectively and m is the number of attributes, then Table 5 shows the definitions of distances used in our analysis.

Table 5. Distance functions

Distance $\delta(\mathbf{x},\mathbf{y})$ with equal weights for all attributes	
Manhattan (MaD) $\sum_m \lvert x_i - y_i \rvert$	Chebychev (ChD) $\max_m \lvert x_i - y_i \rvert$
Canberra (CaD) $\sum_m \left(\lvert x_i - y_i \rvert / \lvert x_i + y_i \rvert \right)$	Clark (ClD) $\sum_m \left(\lvert x_i - y_i \rvert^2 / \lvert x_i + y_i \rvert^2 \right)$
Sigmoid (SiD) $\sum_m \left(1 \Big/ \left(1 + e^{-\left(d - \frac{k}{2}\right)} \right) \right)$ k=max affordable dist.	Exponential (ExD) $\sum_m e^d$ $d = \sqrt{(x_i - y_i)^2}$
Euclidean (EuD) $\sqrt{\sum_m (x_i - y_i)^2}$	

It is expected that the best choice of the distance metric depends on the type of attributes and on the distribution of these attributes. Given the initial "as is" data repository of MBR process, we choose to evaluate a number of different distance metrics. As we continue building the framework, we will identify the distance metrics that are most effective in the software defect prediction domain.

Figures 1 and 2 graphically show the accuracy comparisons of different distance metrics for nearest neighbor count of three and five, respectively. We see that, except for the exponential method, all the distance metrics perform nearly the same. A preliminary observation is that using raw defect data with no weighting functions and data pre-processing, the standard Euclidean measure is a good choice for software defect data prediction. A more rigorous analysis of nearest neighbor selection will be presented in Section 5.2.

5.2. Initial Choice of Nearest Neighbors with different distance metrics

We conducted experiments using nearest neighbor numbers in the range 1-10. From these results, we are able to deduce the effect of increasing/decreasing the numbers of nearest neighbors on the accuracy of prediction. Due to space constraints, we show the results for Manhattan and Euclidean distance metrics. Figures 3 and 4 graphically show the accuracy for different nearest neighbor count using Manhattan and Euclidean distances, respectively.

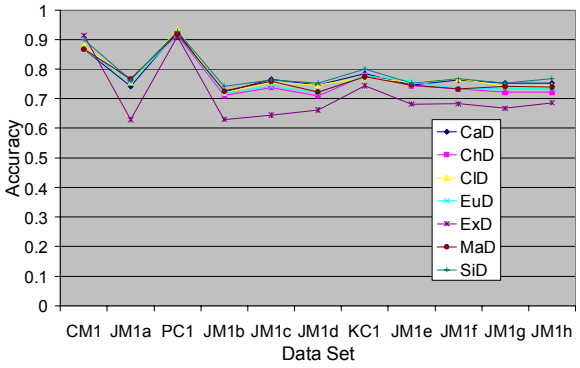

Figure 1. Accuracy prediction for nearest neighbor count of three

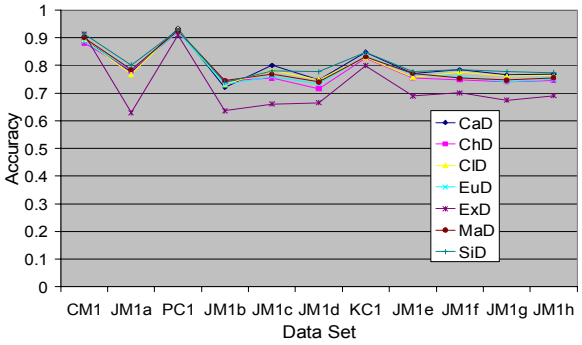

Figure 2. Accuracy prediction for nearest neighbor count of five

We notice that as the number of nearest neighbors increase, the accuracy increases. Hence, as a preliminary conclusion, it appears that using a larger number of nearest neighbors is always beneficial. However, we notice that the classifier tends to classify more cases as zero, thus decreasing the value for true positives and overall reduction in "pd", the probability of detection. This is evident from the following results obtained for 20 nearest neighbors with Euclidean distance metric as shown in Table 6. In most of the cases, we notice there is

not much difference in the percentage of non-defective modules and the predicted accuracy.

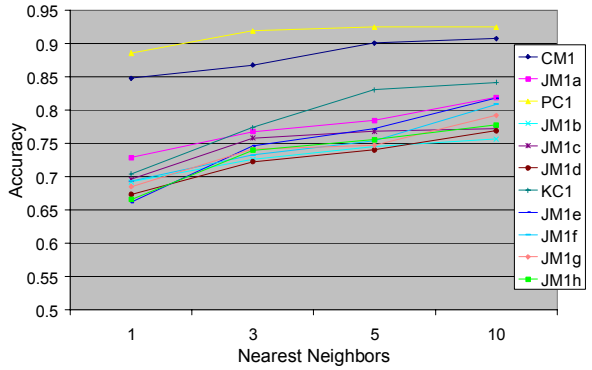

Figure 3. Accuracy using Manhattan distance for different nearest neighbors

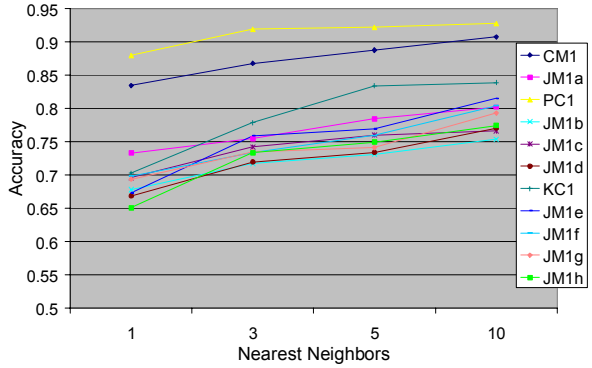

Figure 4. Accuracy using Euclidean distance for different nearest neighbors

The particular reason for the above behavior stems from the fact that there is a large difference in the proportion of non-defective modules compared to the defective ones. Hence, as we increase the number of nearest neighbors, there is more chance that the number of negatives neighbors increases at a rate faster than the positives, giving rise to increased non-defective predictions. Thus, it appears that by just increasing the number of nearest neighbors decreases the probability of detection, thus increasing the probability of missing true alarms. At the same time, we see a decreased rate of false alarms. Hence, the determination of the optimal number of nearest neighbors for a given MBR analysis should be based on the tolerance of "pd" and "pf" that we can withstand. This leads to the next section that describes the assessment criteria chosen for software defect analysis. Also, another important observation is that even with the different range of nearest neighbor counts from 1 to 10, we did not notice much difference in the prediction accuracies for Manhattan and Euclidean distances. We conducted the experiments for other distance functions and noticed that all predicted nearly the same for different nearest neighbors. Hence, if we use the raw data with no weighting on the attributes, and employ a simple combination function, we can use the most widely employed Euclidean distance metric for accuracy prediction. In the following sections, we will do the analysis using the Euclidean distance metric.

5.3. Better Assessment Criteria to determine the optimal number of nearest neighbors

The "Confusion Matrix" presents a more detailed insight into factors important than just the accuracy, such as the number of false negatives and false positives. The data we used for analysis is collected from mission-critical systems which typically should have very low failure rates. As discussed earlier in section 4, for these systems, the assessment criteria for an MBR learner should be based on maximizing "pd", apart from the accuracy. Table 7 shows the confusion matrix for KC1 and JM1e data sets with Euclidean distance metric.

We see that as the number of nearest neighbors increases, the "pd" starts to decrease. We also notice that "pf" also starts declining, with more false positives being converted to true negatives. But as our priority is to increase "pd", we concentrate on "pd" analysis for this study. In less critical applications, the number of nearest neighbors is chosen based on a "balancing function" that chooses optimum values for "pd" and "pf". Figure 5 shows the variations of "pd" with an increase in the number of nearest neighbors for the Euclidean distance.

Table 6. Comparison between % of non-defective modules and accuracy for NN = 20

Data	% positives	Accuracy	Data	% positives	Accuracy
CM1	0.92053	0.9072	KC1	0.8497	0.8496
JM1a	0.8103	0.8103	JM1e	0.8278	0.8278
PC1f	0.9247	0.9277	JM1f	0.8066	0.7334
JM1b	0.7843	0.7535	JM1g	0.8181	0.8278
JM1c	0.808	0.786	JM1h	0.81333	0.8013
JM1d	0.7926	0.783			

From the graph, we notice that for all data sets, an increase in the number of nearest neighbors decreases the "pd". Due to space constraints, we do not show the results for all distance metrics other than Euclidean. But we did notice a similar trend even for other distance metrics. Hence, for mission-critical software data, we need to choose a lower number of nearest neighbors. Therefore, at this stage of our framework, we will continue our analysis with the number of nearest neighbors equal to 1, 3, and 5.

The reason for not choosing the nearest neighbor number to be just 1 is that it does not give us further scope for applying different combination functions.

Table 7. Confusion Matrix for KC1 and JM1e data sets for Euclidean distance and different NNs

	K = 1	K = 5	K = 10	K = 20
KC1	acc:0.7025 tn:409,fn:60, fp:128,tp:35 pd:0.368, pf:0.238	acc:0.8338 tn:496,fn:64, fp:41,tp:31 pd:0.326, pf:0.076	acc:0.8386 tn:500,fn:65, fp:37,tp:30 pd:0.316, pf:0.069	acc:0.8496 tn:510,fn:68, fp:27,tp:27 pd:0.284, pf:0.05
JM1e	acc:0.6728 tn:431,fn:82, fp:146,tp:38 pd:0.317, pf:0.253	acc:0.769 tn:507,fn:91, fp:70,tp:29 pd:0.242, pf:0.121	acc:0.8149 tn:544,fn:96, fp:33,tp:24 pd:0.2, pf:0.057	acc:0.8278 tn:555,fn:98, fp:22,tp:22 pd:0.183, pf:0.038

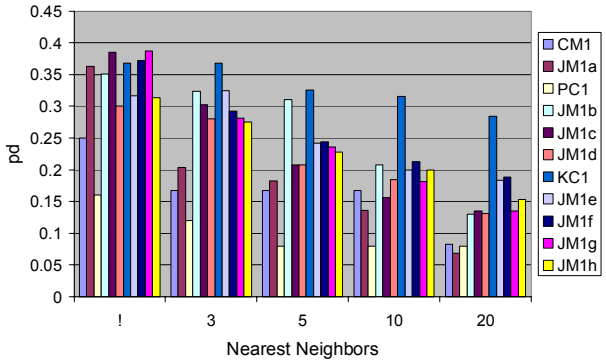

Figure 5. Probability of detection for Euclidean distance with varying nearest neighbors

5.4. Data Preparation

One of the distinct features of MBR is to be able to predict using the data "as is". However, the accuracy of the prediction might be affected by noise present in the raw data. Noise includes excessive outliers, missing data, inconsistent data, and irrelevant data. In this section, we consider techniques that minimize the noise in the data and investigate their impact on the classification.

5.4.1. Characteristics of the defect data. Skewness and Z-scores are the statistical measures used to determine the presence of outliers in the data. Depending on the positive or negative nature of the skewness, we employ necessary transformations to correct the problem. Some argue that transformations "modify" the data, but a transformation applied to the whole data set preserves the relationships between the attributes. Although the real differences between attributes are varied, the relative differences between the attributes remain the same.

We noticed that positive skewness, which is typical for software defects, is very dominant in these datasets. Log transformation is typically used for data with positive skewness. This has the effect of reducing the right tail of the distribution that contributes to positive skewness. Tables 8 and 9 show the accuracy and probability of detection after using log transformation on the different raw data sets respectively.

We see that while log transformation seems to improve the accuracy and probability of detection for most for the data sets, there is no correlation between improvement of accuracy and corresponding improvement of probability of detection for the data sets. However, variations in nearest neighbor showed a highly inconsistent behavior when it came to improving "pd" than improving accuracy. So, at this juncture, we say that transformation can be a good option, when accuracy is the criteria, but is a volatile option for improving "pd".

Table 8. Accuracy after Log transformation using different NN

Data	K = 1	K=1 (Log)	K = 3	K=3 (Log)	K = 5	K=5 (Log)
CM1	0.834	**0.874**	0.867	**0.894**	0.887	**0.894**
JM1a	0.732	0.711	0.754	0.732	0.784	0.767
PC1	0.879	**0.930**	0.918	**0.939**	0.921	**0.927**
JM1b	0.677	0.658	0.717	**0.719**	0.731	0.708
JM1c	0.696	0.688	0.742	**0.746**	0.76	**0.782**
JM1d	0.668	**0.712**	0.719	**0.759**	0.733	**0.771**
KC1	0.702	**0.742**	0.778	**0.806**	0.833	0.833
JM1e	0.672	**0.674**	0.758	0.741	0.769	**0.774**
JM1f	0.698	**0.706**	0.733	**0.764**	0.759	**0.766**
JM1g	0.693	0.683	0.734	0.728	0.741	**0.744**
JM1h	0.650	**0.696**	0.733	**0.75**	0.749	**0.77**

Table 9. Probability of detection after Log transformation using different NN

Data	K = 1	K=1 (Log)	K = 3	K=3 (Log)	K = 5	K = 5 (Log)
CM1	0.25	0.076	0.167	**0.23**	0.167	0.076
JM1a	0.363	0.142	0.204	0.102	0.182	0.081
PC1	0.16	**0.42**	0.12	**0.44**	0.08	**0.32**
JM1b	0.351	**0.452**	0.324	**0.342**	0.311	0.205
JM1c	0.385	0.375	0.302	**0.312**	0.208	**0.27**
JM1d	0.3	**0.351**	0.28	0.236	0.208	**0.213**
KC1	0.368	**0.42**	0.368	**0.37**	0.326	0.26
JM1e	0.317	**0.335**	0.325	0.304	0.242	0.265
JM1f	0.372	0.352	0.292	0.287	0.244	0.241
JM1g	0.387	0.341	0.281	**0.308**	0.236	**0.271**
JM1h	0.314	**0.379**	0.275	**0.329**	0.228	**0.311**

5.4.2. Standardization and Normalization of data. Standardization and normalization involves rescaling the data, so that all the attributes are comparable on the same units. While normalization involves scaling the attribute

data to be in the range 0..1 by dividing the attribute value by the range of that attribute, standardization involves scaling the attribute data to a normal distribution, such that each attribute has zero mean and unit variance. Standardization is predominantly used when there is little information about the data. Normalization is particularly used in case of distance functions, so that all the attributes contribute in the same units of distance.

In this section, we investigate the appropriateness of standardization and normalization in case of software defect data. Tables 10 and 11 show the accuracy and "pd" after applying standardization on the different raw data sets respectively. We observe that standardization of data improved the accuracy in general for most of the data sets. It also improved "pd" for most of the data sets when the nearest neighbor number is one. It did not perform well for trials with larger nearest numbers. One of the important observations is that standardization technique either improved or decreased "pd" for a data set along various different nearest neighbor selections. This is important as it amounts to improving the accuracy and "pd" of the data without really "fudging" the data.

We have also experimentally determined the accuracy and "pd" after applying normalization on the different raw data sets. We observed that normalization performed poorly in predicting "pd" for most data sets. Upon analysis, this behavior is attributed to the presence of outliers in the data. The presence of extremely large and small values in the data set causes the data range to be a large value, resulting in most of the normalized data having very low values. As a result, false "non-faulty" clusters are easily formed which reduces the "pd" of the trials. From the above results, we conclude that standardization is an option to be considered for defect data preprocessing. Also, in almost all the cases, as the nearest neighbor count increased, the accuracy increased, while "pd" decreased. Hence, we should use low neighbor counts for defect prediction. So, our further analysis use nearest neighbor counts as one and three.

5.5. Weights of Attributes

Till now, we consider all the attributes to be contributing equal weights. However, not all attributes are related equally to the defective target field. Hence, weights [Die97] should be assigned corresponding to the attribute relevance to the target field. We consider two weighting techniques here, weighted parameters from a multinomial logistic regression model and weighting from discretizing the continuous attribute and following the classic MBR weight assignment for discrete data.

5.5.1. Discretization technique. We applied the discretization weighting technique to the original data and did not notice appreciable deviation in either the accuracy prediction or "pd". The reason is that larger valued attributes dominate the distance function so highly that even after any weighting, the relative distances are unchanged. Hence, we apply the weighting techniques to the pre-processed standardized data and evaluate its performance compared to the original. The discretization intervals are chosen to be 10, 20 and 40. In this case also, we noticed that applying weights through discretization did not improve the "pd".

Table 10. Accuracy after applying standardization using different NN

Data	K = 1	K=1 (Std)	K = 3	K = 3 (Std)	K = 5	K = 5 (Std)
CM1	0.834	**0.854**	0.867	**0.874**	0.887	**0.894**
JM1a	0.732	**0.754**	0.754	**0.767**	0.784	**0.793**
PC1	0.879	**0.894**	0.918	0.909	0.921	0.912
JM1b	0.677	**0.714**	0.717	**0.747**	0.731	**0.756**
JM1c	0.696	**0.7**	0.742	**0.764**	0.76	**0.786**
JM1d	0.668	**0.717**	0.719	**0.748**	0.733	**0.763**
KC1	0.702	**0.716**	0.778	**0.780**	0.833	0.821
JM1e	0.672	**0.684**	0.758	0.736	0.769	**0.774**
JM1f	0.698	**0.734**	0.733	**0.752**	0.759	**0.781**
JM1g	0.693	**0.707**	0.734	**0.751**	0.741	**0.758**
JM1h	0.650	**0.689**	0.733	**0.739**	0.749	**0.752**

Table 11. Probability of detection after applying standardization using different NN

Data	K = 1	K=1 (Std)	K = 3	K = 3 (Std)	K = 5	K = 5 (Std)
CM1	0.25	0.187	0.167	0.125	0.167	0.062
JM1a	0.363	0.295	0.204	0.136	0.182	0.113
PC1	0.16	**0.354**	0.12	**0.354**	0.08	**0.225**
JM1b	0.351	**0.373**	0.324	0.265	0.311	0.253
JM1c	0.385	0.333	0.302	0.288	0.208	**0.244**
JM1d	0.3	**0.39**	0.28	**0.289**	0.208	**0.265**
KC1	0.368	**0.444**	0.368	0.322	0.326	0.255
JM1e	0.317	0.309	0.325	0.26	0.242	0.232
JM1f	0.372	**0.394**	0.292	0.299	0.244	0.235
JM1g	0.387	**0.426**	0.281	**0.37**	0.236	**0.284**
JM1h	0.314	**0.444**	0.275	**0.367**	0.228	**0.301**

5.5.2. Multinomial Logistic Regression technique. We experimentally determined the accuracy and "pd" on the raw data sets by using estimated parameters from a multinomial logistic regression as weights to the attributes. We observed that while accuracy was drastically reduced in most of the cases, the "pd" increased. More careful analysis showed that "pd" is almost 1 or 0, implying that all the cases are either classified as 1 or 0. This is unacceptable since there seems to be no classification or, in other words, there is

classification to only one category. But, any learner should provide a satisfactory accuracy before one starts proceeding to analyze "pd". Hence, we do not consider multinomial logistic regression weighting approach.

5.6. Combination function

The combination function was varied to use simple majority voting instead of weighting the neighbors according to their distances. We observed that for nearest neighbor of count one, the accuracy and "pd" remained the same, while for nearest neighbor count of three, "pd" decreased for all the data sets. Hence, we stress the need to associate combination function with distance.

6. Conclusions

In this paper, we performed different experiments to build a framework to predict mission-critical software defect data using MBR analysis. We identified the following sequence of steps in the framework: First, establish the assessment criteria for evaluating the performance of the different learners. In our cases, we evaluated our learners both on accuracy and probability of detection. If probability of detection is the criteria, then the following steps can be used: (1) Use Majority Voting combination function along with Euclidean distance and nearest neighbor count of one to obtain the preliminary estimates. (2) Modify the combination function to be weighted according to the distance of neighbors and run the analysis for nearest neighbor count of 1 and 3. Take the better of the two trials. (3) Standardize the data and run step 2 and pick the best of all trials.

Predicting the probability of detection improves as we progress through those steps. The steps increase in complexity and prediction capabilities. Hence, depending on the resources and prediction accuracy required, we can conduct all the steps or terminate and pick a particular trial that provides the "pd" desired. By following the sequence of steps described above, we can construct the best MBR learner for mission-critical software defect data.

Apart from that, we also notice that removing outliers through transformations and applying weights to the attributes through discretization did not improve the prediction capabilities of the learner. Hence, these steps can be avoided. Finally, we conclude that if accuracy is the only criteria, then simple MBR with Euclidean distance and one nearest neighbor should suffice as the learning method.

7. References

[Aam94] A. Aamodt, E. Plaza, "Case-Based reasoning: foundational issues, methodological variations, and system approaches", *AI Communications*, Vol. 7, No. 1, Mar 1994, pp. 39-59.

[Cos93] S. Cost, and S. Salzberg, "A weighted nearest neighbor algorithm for learning with symbolic features", *Machine learning*, Vol. 10, No. 1, 1993, pp. 57-78.

[Das91] B.V. Dasarathy, "Nearest neighbor norms: NN pattern classification techniques", *IEEE Computer Society Press*, Los Alamitos, CA, 1991.

[Die97] W. Dietrich, D.W. Aha, and T. Mohri, "A review and empirical evaluation of feature weighting methods for a class of lazy learning algorithms", *In Artificial Intelligence Review*, Vol. 11, 1997, pp. 173-314.

[Ema99] K. E. Emam, S. Benlarbi, N. Goel, and S. N. Rai, "Comparing case-based reasoning classifiers for predicting high risk software components", *The Journal of Systems and Software*, Vol. 55, 2001, pp. 301-320.

[Fen97] N.E. Fenton, and S.L. Pfleeger, "Software metrics: A rigorous and practical approach", *PWS Publishing Company*, Second Edition, 1997.

[Kas95] S. Kasif, S. Salzberg, D. Waltz, J. Rachlin, and D. Aha, "Towards a Framework for Memory-Based Reasoning", *NECI Technical Report*, Dec. 1995, pp.95-132.

[Kho97] T.M. Khoshgoftaar, K. Ganesan, E.B. Allen, F.D. Ross, R. Munikoti, N. Goel, and A. Nandi, "Predicting fault-prone modules with case-based reasoning", *Eighth International Symposium on Software Reliability Engg*, 1997, pp. 27-35.

[Koh98] R. Kohavi, and F. Provost, "Glossary of terms", *Machine Learning*, Vol. 30 (2/3), pp. 271-274, 1998.

[Lia98] T.W. Liao, and Z. Zhang, "Similarity measures for retrieval in case-based reasoning systems," *Applied Artificial Intelligence*, Vol. 12, 1998, 267-288.

[Mae93] P. Maes and R. Kozierok, "Learning interface agents", *In Proc. of the Eleventh National Conf. on Artificial Intelligence*, MIT Press , 1993, pp. 459-465.

[She01] M. Shepperd, and G. Kadoda, "Comparing software prediction techniques using simulation", *IEEE Transactions of Software Engg*, Vol. 27, No. 11, Nov. 2001, pp. 1014-1022.

[Mun92] J.C. Munson, and T.M. Khoshgoftaar, "The detection of fault prone modules", *IEEE transactions on Software Engineering*, Vo. 18, No. 5, May 1992, pp. 423-433.

[Pau02] R. Paul, F. Bastani, V. Challagulla, I.-L. Yen, "Software measurement data analysis using memory-based reasoning", *Proc. 14th IEEE Intl. Conf. Tools with AI*, Nov. 2002, pp. 261-267.

[Sta86] C. Stanfill and D. Watlz, "Towards memory-based reasoning", *Comm. of the ACM*, Dec. 1986, pp. 1213-1228.

[Wal87] D.L. Waltz, "Applications of the connection machine", *IEEE Computer*, Vol. 20, No. 1, Jan. 1987, pp. 85-97.

[Wil97] D.R. Wilson, T.R. Martinez, "Improved heterogeneous distance functions", *Journal of Artificial Intelligence Research*, Vol. 6, 1997, pp. 1-34.