National Research Council Canada
Institute for Information Technology
Conseil national de recherches Canada
Institut de Technologie de l'information
ERB-1058

NRC·CNRC

Comparing Case-based Reasoning Classifiers for Predicting High Risk Software Components

Khaled El Emam, Saida Benlarbi,
and Nishith Goel
September 1999

Canada
NRC 43602

National Research Council Canada
Institute for Information Technology

Conseil national de recherches Canada
Institut de Technologie de l'information

Comparing Case-based Reasoning Classifiers for Predicting High Risk Software Components

Khaled El Emam, Saida Benlarbi,
and Nishith Goel
September 1999

Comparing Case-Based Reasoning Classifiers for Predicting High Risk Software Components

Khaled El Emam

National Research Council, Canada
Institute for Information Technology
Building M-50, Montreal Road
Ottawa, Ontario
Canada K1A OR6
Khaled.El-Emam@iit.nrc.ca

Saida Benlarbi
Nishith Goel
Cistel Technology
210 Colonnade Road
Suite 204
Nepean, Ontario
Canada K2E 7L5
{benlarbi, ngoel}@cistel.com

Abstract

Case-based reasoning (CBR) has been proposed for predicting the risk class of software components. Risky components can be defined as those that are fault-prone, or those that require a large amount of effort to maintain. Thus far evaluative studies of CBR classifiers have been promising, showing that their predictive performance is as good as or better than other types of classifiers. However, a CBR classifier can be instantiated in different ways by varying its parameters, and it is not clear which combination of parameters provides the best performance. In this paper we evaluate the performance of a CBR classifier with different parameters, namely: (a) different distance measures, (b) different standardization techniques, (c) use or non-use of weights, and (d) the number of nearest neighbors to use for the prediction. In total, we compared 30 different CBR classifiers. The study was conducted with a data set from a large real-time system, and the objective was to predict the fault-proneness of its components. Our results indicate that there is no difference in prediction performance when using any combination of parameters. Based on these results, we recommend using a simple CBR classifier with Euclidean distance, z-score standardization, no weighting scheme, and selecting the single nearest neighbor for prediction. The advantage of such a classifier is its intuitive appeal to nonspecialists, and the fact that it performs as well as more complex classifiers.

1 Introduction

Mounting evidence demonstrates that most faults are found in only a few of a system's components (Fenton and Ohlsson, 2000; Moller, 1993; Kaaniche and Kanoun, 1996; Ohlsson and Alberg, 1996). This makes the early identification of the few faulty components an important research endeavor in that it allows an organization to take mitigating actions, such as focus defect detection activities on components that are likely to be faulty, for example by optimally allocating testing resources (Harrison, 1988), or by redesigning components.

Early identification of faulty components is commonly achieved through a binary *quality model* that classifies components into either a *faulty* or *not-faulty* category (Briand et al., 1993a; Lanubile and Visaggio, 1997). These binary classification models almost always utilize product metrics as predictor variables.[1] The theoretical basis for developing such binary classifiers has been articulated in (Briand et

[1] In some cases aggregates of the product metrics are used, such as domain metrics (Khoshgoftaar et al., 1996a; Khoshgoftaar et al., 1997a) derived from principal components analysis.

al., 1998). There it is hypothesized that the structural properties of a software component (such as its coupling) have an impact on its cognitive complexity. Cognitive complexity is defined as the mental burden of the individuals who have to deal with the component, for example, the developers, testers, inspectors, and maintainers. High cognitive complexity leads to a component exhibiting undesirable external qualities, such as increased fault-proneness and reduced maintainability.

A promising modeling technique that has been applied to this quality modeling problem is case-base reasoning (henceforth CBR).[2] For example, one empirical evaluation found a CBR classifier to be competitive to a discriminant analysis model (Khoshgoftaar et al., 1997a). Another study that considered models for predicting the number of faults found a CBR system to have superior predictive performance to an ordinary least squares regression model (Ganesan et al., 1999). Furthermore, it has been argued that CBR is in a class of modeling techniques more suited to the analysis of software engineering data (compared with classical statistical techniques, such as least squares regression) (Gray and MacDonell, 1997).

However, there are many different ways that one can construct a CBR classifier by choosing its parameters. Specifically:

- Which distance measure should be used ?
- Should weights be used or should all independent variables be considered equally important ?
- If the equal importance assumption is invoked, which standardization technique should be used ?
- How many neighbors should be used for making a prediction ?

While one can make some theoretical arguments for selecting the appropriate parameters, it remains of particular importance to evaluate the predictive performance of the different CBR classifiers empirically.[3]

In this paper we evaluate the parameters of a CBR classifier in the context of predicting the risk of software components using *source code* metrics. A high risk component is defined as one that has one or more faults detected during acceptance testing. Therefore, we focus on the prediction of the *fault-proneness* of software components. We use a large data set from an operational real-time system for performing the comparison of CBR classifier parameters with respect to prediction accuracy.

Briefly, our results indicate that there is no difference in predictive performance among all of the CBR classifiers that were considered. These results suggest that the use of a simple CBR classifier that is intuitive to nonspecialists would be preferred since it would achieve similar performance. This preferred

[2] In non-software domains, at least one previous study that constituted an extensive evaluation of CBR compared with a discriminant analysis model demonstrated the superiority of the CBR classifier, for example see (Althoff et al., 1995).

[3] Prediction performance is not the only factor that can be used to evaluate binary classifiers. For example, it has been noted that ease of interpretation of the resultant output of a prediction model and the ability to handle missing data are also important features (Briand et al, 1992; Gray and MacDonell, 1997). However, prediction performance is still the feature of a classifier that is of primary importance. For example, if a classifier had an elegant solution to the missing value problem but could not predict high risk components better than a guessing classifier does, then it is doubtful that it will be applied in practice.

classifier would use Euclidean distance, z-score standardization, no weighting scheme, and selecting the single nearest neighbor for prediction.

The paper is organized as follows. Section 2 provides background on the CBR classifiers that we evaluate and the criterion that we use for evaluating predictive performance. In Section 3 our research method is described. This is followed in Section 4 by the results of the study, and a discussion. We conclude the paper in Section 5 with a summary and directions for future research.

2 Background

2.1 Case-Based Reasoning Classifiers

A CBR classifier uses previous "similar" cases as the basis for predicting the risk class of an unseen software component. Previous cases are stored in a *case base*. Similarity (or equivalently, distance) is defined in terms of a set of product metrics. The rationale is that components in the case base that have a similar structure to an unseen case will also exert a similar mental burden on the development staff, and consequently will have a similar fault-proneness.

Khoshgoftaar et al. (Khoshgoftaar et al., 1997a) noted a number of advantages of a CBR classifier. Most notably in our context is that the detailed characterization of the "similar" cases can help one interpret the automatic classification results. In principle, as well, a CBR classifier would provide a straight forward approach for dealing with missing values. However, in the context of quality prediction using product metrics, there are rarely missing values.[4]

When evaluating the predictive performance of a CBR classifier, one first constructs a *case base* of previous components where the source code metrics and the fault-proneness are known. A different data set is then used as the *test* set. This can be achieved in a number of different ways, such as a holdout sample, cross-validation, bootstrapping, multiple releases, or random subsets. For our study we use random subsets, and this is explained further below.

2.2 Notation

We first present some notation to help explain the different CBR parameters and our evaluation method.

A classification algorithm is used to develop a classifier that assigns a binary value to unseen observations. We restrict ourselves to binary risk classes: High and Low (i.e., the risk of having a fault). Let x_i be the vector of product metrics of the i^{th} component in the test data set, and x_{ij} be the j^{th} product metric measured on the component. Further, let c_k be the vector of product metrics of the k^{th} component in the case base, and let c_{kj} be the j^{th} product metric measured on that component.

[4] The exception is when, for example, "include" files are considered.

Furthermore, let the case base be defined by the set $D_X = \{c_1, c_2, \ldots, c_k, \ldots, c_{m-1}, c_m\}$ and the test set be defined as $D_h = \{x_1, x_2, \ldots, x_i, \ldots, x_{N-1}, x_N\}$.

2.3 CBR Parameters

In setting up a CBR classifier, a number of different parameters can be specified that can potentially have a nonnegligible impact on the classifier's prediction accuracy. Specifically:

- Which distance measure should be used ?
- Should weights be used or should all independent variables be considered equally important ?
- If the equal importance assumption is invoked, which standardization technique should be used ?
- How many neighbors should be used for making a prediction ?

Below we describe these parameters and the particular values that we evaluated, including justifications for the selections made.

2.3.1 Distance Function

A multitude of different distance functions can reasonably be used in a CBR classifier. We limit ourselves to the ones that have been actually used in previous software engineering studies, which also happen to be common distance functions in other disciplines. We also limit ourselves to the context where the predictor variables are continuous, since in almost all quality modeling studies this is the case.

Kaufman and Rousseeaw (Kaufman and Rousseeuw, 1990) define the Minkowski distance between a component in the case base and a component in the test set as follows:

$$d_{ik} = \left(\sum_j w_j \left| c_{kj} - x_{ij} \right|^q \right)^{\frac{1}{q}} \qquad \textbf{Eqn. 1}$$

The w_j value is a weight and is discussed below. The most commonly used distance functions for continuous variables are the Euclidean and Manhattan distances. In previous quality modeling studies that used CBR for analysis, both the Euclidean (Khoshgoftaar et al., 1997a) and the Manhattan (Ganesan et al., 1999) distance functions were used. We therefore evaluate these.

The Euclidean distance between a component i in the test data set and a component k in the case base is given by[5] (i.e., $q = 2$):

[5] Here, the weighted Euclidean distance function is different from that defined in (Khoshgoftaar et al, 1997a), where the weight is also squared, suggesting that the weight is actually part of the distance rather than weighting the distance. We follow the more common formulation in (Kaufman and Rousseeuw, 1990).

$$Euclidean_{ik} = \sqrt{\sum_j \left(w_j \left(c_{kj} - x_{ij} \right)^2 \right)} \qquad \textbf{Eqn. 2}$$

The Manhattan distance is defined by (i.e., $q = 1$):

$$Manhattan_{ik} = \sum_j \left(w_j \left| c_{kj} - x_{ij} \right| \right) \qquad \textbf{Eqn. 3}$$

A priori, there is no compelling reason to prefer one distance function over another, and therefore, it is prudent to evaluate them both empirically.

2.3.2 Weights

Some previous studies with CBR classifiers used weights (Khoshgoftaar et al., 1997a), while others assumed equal weights (Ganesan et al., 1999). The rationale for applying weights is that not all product metrics are equally related to fault-proneness. A weighting method suggested in (Kolodner, 1993; Watson, 1997), and applied in (Khoshgoftaar et al., 1997a) is to use the estimated parameters from a regression model. Given that our dependent variable is binary, we use the estimated parameters from a logistic regression model (Hosmer and Lemeshow, 1989) as weights. [6,7]

2.3.3 Standardization

Standardization rescales the predictor variables (i.e., the product metrics), and accounts for the fact that not all variables are measured on the same units. We consider four different standardization approaches.

A simulation study in the allied area of cluster analysis found one type of standardization scheme to be superior in recovering the underlying cluster structure under different conditions, including error free data, and data with noise and with outliers (Milligan and Cooper, 1988). However, the parameters of that simulation are not necessarily reflective of software product metrics data, and therefore this standardization scheme, referred to as Z_5, is evaluated here:[8]

$$Z_5 = \frac{c_{kj} - \min(c_k)}{\max(c_k) - \min(c_k)} \qquad \textbf{Eqn. 4}$$

As will be noted, this has a non-robust denominator in that it will be easily affected by even a single outlier. A more traditional standardization scheme that makes the unit of the variables the sample standard deviation is the z-score. This makes the mean equal to zero and the standard deviation equal to

---

[6] The weights are derived from the regression using the case base.

[7] It should be noted that there are substantial differences between using the logistic regression parameter estimates to weight the distances, and using the logistic regression parameter estimates directly for prediction. In the former the parameter estimates are augmented with further information from the distance function and therefore they can produce quite different results from the latter. One can consider this as a hybrid of a logistic regression and a CBR model.

[8] All standardization equations are defined here in terms of the case base. However, they can also be easily reformulated in terms of the test set by changing the c to an x.

one, and, in the context of standardization for the allied problem of cluster analysis, has been suggested by (Dubes and Jain, 1980; Everitt, 1980). It is defined as:

$$Z-score_j = \frac{c_{kj} - \frac{1}{n}\sum_k c_{kj}}{\sqrt{\frac{1}{n-1}\sum_k \left(c_{kj} - \frac{1}{n}\sum_k c_{kj} \right)^2}} \qquad \textbf{Eqn. 5}$$

The z-score has been used in previous software engineering studies (Khoshgoftaar et al., 1997a; Ganesan et al., 1999).

A more robust approach for standardization is to use the *mean absolute deviation* (Weisberg, 1992) instead of the standard deviation in the denominator. This is robust because the deviations are not squared, therefore atypical points do not exaggerate it (Kaufman and Rousseeuw, 1990). Robustness is desirable because product metrics data typically have a few extreme values that may exert strong influence on the analysis results, for example, see (Myrvold, 1990).

$$Mean\ Absolute_j = \frac{c_{kj} - \frac{1}{n}\sum_k c_{kj}}{\frac{1}{n}\sum_k \left| c_{kj} - \frac{1}{n}\sum_k c_{kj} \right|} \qquad \textbf{Eqn. 6}$$

An even more robust measure of dispersion that can be used is the *median absolute deviation* (Hoaglin et al., 1983), which is:

$$Median\ Absolute_j = \frac{c_{kj} - med(c_j)}{med\left(\left| c_{kj} - med(c_j) \right| \right)} \qquad \textbf{Eqn. 7}$$

2.3.4 Nearest Neighbor

It can be expected that the number of nearest neighbors that are used as the basis for prediction will have an impact on the prediction performance. Previous software engineering studies with CBR systems (Khoshgoftaar et al., 1997a; Ganesan et al., 1999) used the single nearest neighbor as the basis for prediction.[9] We evaluate one, three, and five nearest neighbors.

The CBR classifier that we implemented uses a majority vote to make the prediction in the cases with 3 and 5 nearest neighbors. If there are ties in the kth nearest neighbor, all candidates are included in the vote. This is the approach used by (Venables and Ripley, 1997) in their implementation of a nearest neighbor classifier.

[9] (Ganesan et al., 1999) also used a scoring algorithm for selecting and aggregating the values from the nearest neighbours. However, this particular algorithm is implementable directly only if the dependent variable is continuous rather than binary.

2.4 Evaluation Criterion

The coefficient that we use for evaluating and comparing the prediction accuracy is Youdon's J coefficient (Youdon, 1950), which is used for evaluating diagnostic tests in medical methodology texts (Armitage and Berry, 1994). This coefficient addresses problems in current binary classifier evaluation criteria used in the software engineering literature, such as proportion correct accuracy, type I and type II error rates, correctness, completeness, and the Kappa coefficient. The justification for J is detailed in the appendix. The logic of the J coefficient is described below.

When evaluating a binary classifier, one compares the predicted risk class with the actual risk class. Table 1 shows the notation in obtained frequencies when a classifier is used to predict the class of unseen observations in a confusion matrix.

		Predicted Risk Class for Test Set		
		Low	**High**	
Real Risk Class for Test Set	**Low**	n_{11}	n_{12}	N_{1+}
	High	n_{21}	n_{22}	N_{2+}
		N_{+1}	N_{+2}	N

Table 1: Notation for a confusion matrix.

A measure of success of a classifier in identifying low risk components is:

$$\frac{n_{11} - n_{12}}{n_{11} + n_{12}} \qquad \textbf{Eqn. 1}$$

A measure of success of a classifier in identifying high risk components is:

$$\frac{n_{22} - n_{21}}{n_{22} + n_{21}} \qquad \textbf{Eqn. 2}$$

The J coefficient is defined as the average of these two values:

$$J = \frac{1}{2}\left(\frac{n_{11} - n_{12}}{n_{11} + n_{12}} + \frac{n_{22} - n_{21}}{n_{22} + n_{21}}\right) \qquad \textbf{Eqn. 3}$$

The J coefficient can vary from minus one to plus one, with plus one being perfect accuracy and –1 being the worst accuracy. A guessing classifier (i.e., one that guesses High/Low risk with a probability of 0.5) would have a J value of 0. Therefore, J values greater than zero indicate that the classifier is

performing better than would be expected from a guessing classifier (i.e., by chance). Youdon provides estimates of the standard error of J, which can be used to construct confidence intervals.

3 Research Method

3.1 Data Source

Our data set comes from a large real-time system. It was written by professional programmers in a large commercial organization. This embedded computer application included numerous finite state machines, and interfaces to other kinds of equipment. It was written in a procedural language similar to C. The focus of our study was one subsystem.

The subsystem had a total of 13577 components. Of these, 11012 components were not "include" files, and therefore it was possible to extract the relevant source code product metrics. These latter components amount to more than 10 million lines of code. A total of 8585 components (78%) did not have faults in them, and the remaining 2427 components did. All faults were discovered during acceptance testing.

For the purpose of our study we extracted two random samples to serve as the case base and the test set respectively. Each consisted of 500 observations. The procedure followed was as follows. A simple random sample without replacement of 500 observations was taken from all 2427 observations with faults found in them. Another sample of 500 observations was similarly drawn from the 8585 non-faulty components. For each of the faulty and non-faulty sample, we assigned half the observations to the case base, and the other half to the test data set. Therefore we end up with a case base of 500 observations, 250 of which are faulty and 250 are not faulty. This also applies to the test data set. While the prevalence of faulty components in our case base and test data set do not match the prevalence in the whole system, we did use an evaluative coefficient that is not influenced by prevalence (see the appendix).

3.2 Product Metrics

Below we describe each of the 9 product metrics that were used in our analysis, and provide references to previous studies that have used the same or similar metrics in the past. Since our objective in the current study was to compare CBR classifier parameters, we used metrics that were developed in the past and that have been used in previous studies rather than develop a new or different set of metrics. Furthermore, this set of metrics is collected by the organization with whom the study was conducted, and are currently used for project decision making. In addition, all of these metrics are actively used to evaluate software products during the acquisition of software at Bell Canada (Coallier et al., 1999; Mayrand and Coallier, 1996).

The unit about which the metrics were collected is a routine (or procedure). The unit of analysis is a module, which may contain more than one routine.

Metric	Description
Arcs	The total number of arcs found in the control graph. This metric was used in (Khoshgoftaar et al., 1997b).
FanOut	The number of distinct calls to other modules. This is similar to the fan-out metric of Kafura and Henry (Kafura and Henry, 1981), and has been used in (Khoshgoftaar at al., 1996a; Khoshgoftaar et al., 1997b; Khoshgoftaar et al., 1995b).
Breach	Number of control structure breaches. This represents the number of arc crossings in the control graph that violate the principles of structured programming (e.g., usage of one-entry, one-exit control structures such as loops and conditional statements). This has been used in (Khoshgoftaar et al., 1995b).
Knots	Number of knots. This represents the number of arc crossings in the control graph, and includes crossings that violate the principles of structured programming (breaches of structure).
LOC	Total number of lines of code. This excludes lines in included files, preprocessor directives, and empty lines. Comment lines are included. This value has a Pearson correlation of 0.967 with number of source lines of code. Therefore, it does not matter which one is used. This is a commonly used metric in quality models since size is expected to be an important contributor to fault-proneness.
Entry	The number of entry nodes in the control graph.
Pending	Number of pending nodes. A pending node can be seen as a not attainable path in the module. Even though every possibility of the conditional statements are tested, a pending node cannot be accessed. When programming style conventions require that there be no “dead code”, a positive value on this metric indicates that code intended to be functional is not executable (Khoshgoftaar et al., 1995a; Khoshgoftaar et al., 1995b).
Cyclomatic	The cyclomatic complexity number (McCabe, 1976).
Global	The number of global variables that are used. This metric was used in (Navlakha, 1987; Khoshgoftaar et al., 1997b).

Table 2: Product metrics collected.

The above source code metrics were specifically selected since previous work has identified that they are not necessarily redundant (Khoshgoftaar et al., 1995a).[10]

3.3 Dependent Variable

The dependent variable that we used in our study was the detection or non-detection of a fault during acceptance testing. If a fault is detected then the component is considered high risk, and if not then it is considered low risk.

4 Results

4.1 Descriptive Statistics

Table 3 shows the mean and standard deviation of all the product metrics that were considered during our study.

	Case Base		Test Data Set	
Metric	**Mean**	**Std. Dev.**	**Mean**	**Std. Dev.**
Arcs	373.8	590.2	355.8	617
FanOut	96.65	148	92.2	144
Breach	15.4	121.5	6.4	37
Knots	152.4	731	100.8	437
LOC	1256	1878.6	1202	1950
Entry	22	31.4	21.5	34
Pending	7.8	21	6.2	14.3
Cyclomatic	148	228	141	236.7
Global	894.5	1442	880	1518

Table 3: Descriptive statistics for the product metrics.

The fact that the standard deviation is much larger than the mean in all cases, and that none of these metrics can go below zero (i.e., they are counts) indicates heavily skewed distributions. There is a large number of observations with low values on these product metrics, and few with large values.

[10] Although the use of principal components analysis was a possibility for reducing the number of variables, previous work does indicate that this may not necessarily improve the accuracy of classifiers (Lanubile and Visaggio, 1997). Furthermore, (Fenton and Neil, 1999) have criticized the use of principal components in the quality modeling literature because they are difficult to interpret in practical terms.

In general, it can also be stated that there is a good concordance between the case base and the test set (some variations would be expected). The biggest discrepancy was in the Breach and Knots variables. Further examination indicated that the case base had a small number of extreme points, hence exaggerating the mean and standard deviations in that set.

4.2 Results from Varying CBR Parameters

The results from varying the CBR parameters are summarized in Table 4, Table 5, and Table 6 for nearest neighbor, nearest three neighbors, and nearest five neighbors respectively. The tables show the J coefficient values.

It would seem from these results that there is a steady progression of improvement as the number of nearest neighbors increases. However, it is more difficult to draw conclusions from these tables on which particular combination of parameters produces the best predictive performance since the differences in the J coefficient are not marked.

	Equal Weight (standardization method)				**Weighted**
Distance Measure	**Z score**	**Mean Absolute**	**Median Absolute**	**Z_5**	
Euclidean	0.172	0.204	0.208	0.176	0.176
Manhattan	0.208	0.212	0.184	0.182	0.18

Table 4: Evaluation results in terms of the J-coefficient for the different parameters with one nearest neighbor. The values in the cells are the obtained J values.

	Equal Weight (standardization method)				**Weighted**
Distance Measure	**Z score**	**Mean Absolute**	**Median Absolute**	**Z_5**	
Euclidean	0.248	0.224	0.216	0.256	0.196
Manhattan	0.224	0.236	0.224	0.284	0.224

Table 5: Evaluation results in terms of the J-coefficient for the different parameters with the three nearest neighbors. The values in the cells are the obtained J values.

	Equal Weight (standardization method)				**Weighted**
Distance Measure	**Z score**	**Mean Absolute**	**Median Absolute**	**Z_5**	
Euclidean	0.256	0.28	0.256	0.208	0.288
Manhattan	0.28	0.272	0.292	0.193	0.236

Table 6: Evaluation results in terms of the J-coefficient for the different parameters with the five nearest neighbors. The values in the cells are the obtained J values.

The charts in Figure 1, Figure 2, and Figure 3 show the 95% confidence intervals for all the classifiers that we considered. The notation that is used is “E” or “M” for either the Euclidean distance or the

Manhattan distance, and followed by either the standardization method's name (z-score, mean absolute, median absolute, or Z_5) or that it was weighted.

These charts make clear two interesting conclusions. First, all of the binary classifiers considered here have lower tails for the J values greater than zero, indicating that all perform better than one would expect from a guessing classifier. This is somewhat contrary to the strong conclusions drawn in (Lanubile and Visaggio, 1997) about the utility (or otherwise) of binary classifiers.

The second conclusion that can be drawn from our results, given that all confidence intervals overlap, is that the predictive performance of all the 30 classifiers is essentially the same.[11] There is no compelling reason to prefer a set of parameters in preference to another set of parameters when implementing a CBR classifier.

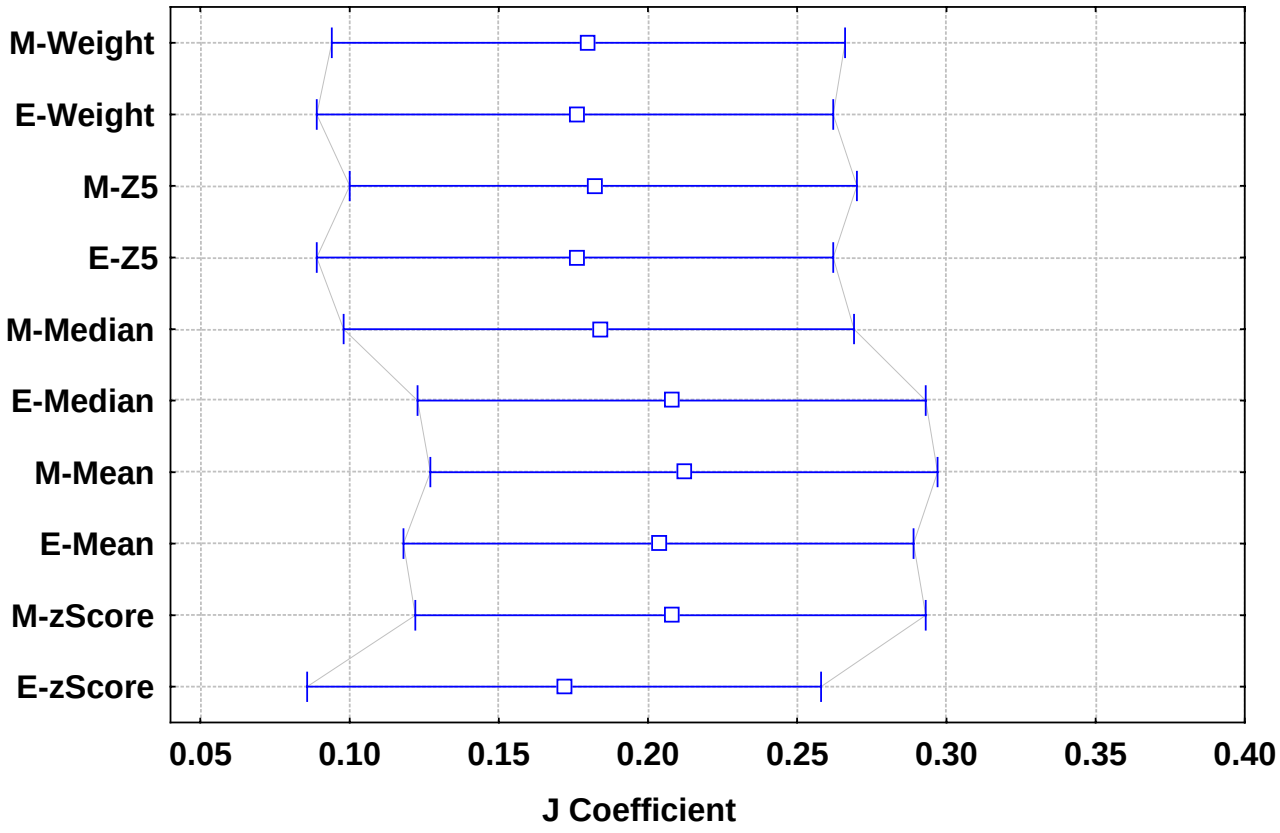

Figure 1: 95% confidence intervals for the J coefficient with the nearest neighbor CBR classifiers.

[11] More formal tests of hypotheses with Bonferonni adjustments confirm this.

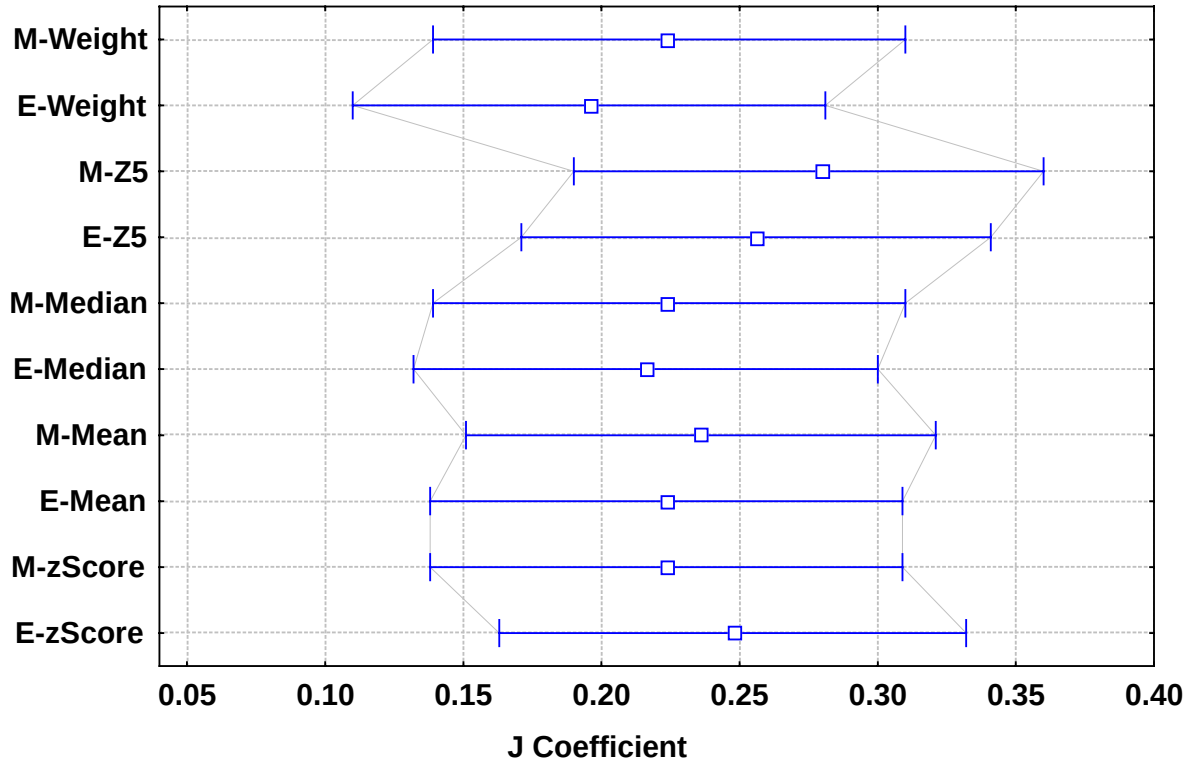

Figure 2: 95% confidence intervals for the J coefficient with the three nearest neighbor CBR classifiers.

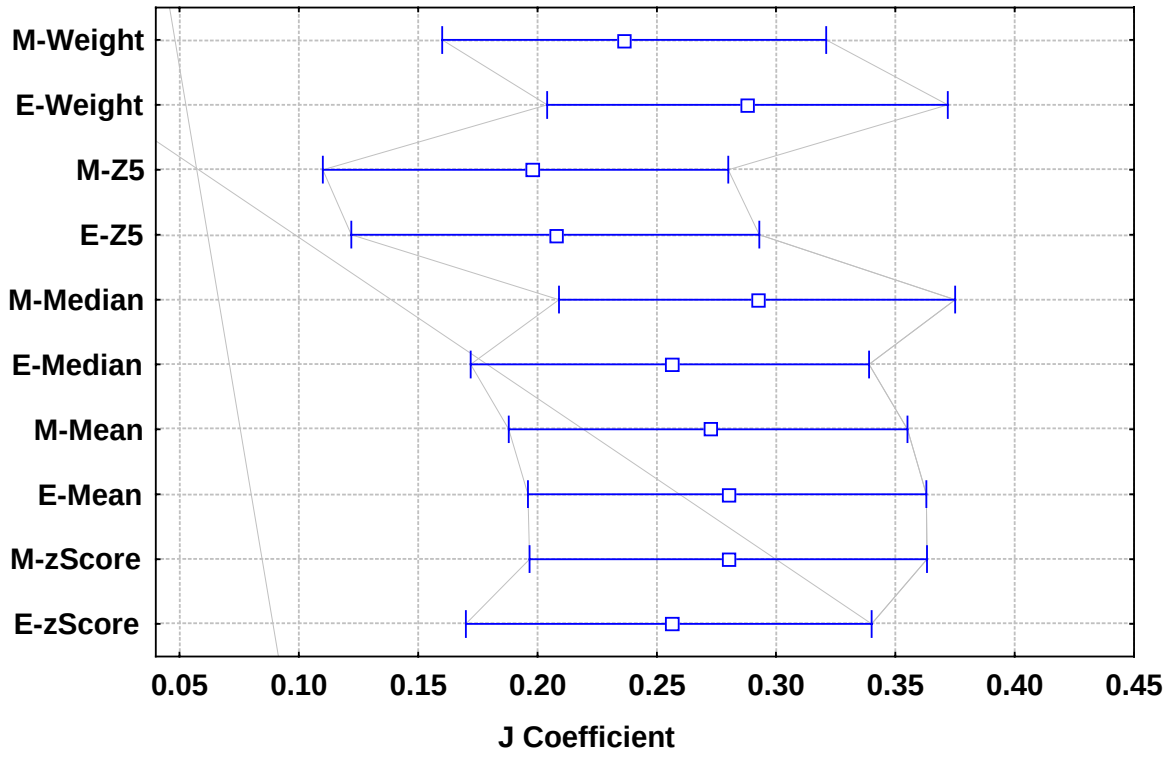

Figure 3: 95% confidence intervals for the J coefficient with the five nearest neighbor CBR classifiers.

4.3 Discussion

The results that we obtained would support the recommendation that, when one is constructing a CBR classifier for quality modeling, it would be preferred to use the simplest CBR classifier. The reason for this being that their predictive performance is the same. The simplest CBR classifier is one that uses a Euclidean distance and the z-score for standardization, and that selects the nearest single neighbor for making the prediction. This is exactly the classifier that was used in an earlier study by Khoshgoftaar et al. (Khoshgoftaar et al., 1997a). The reason for using the simplest one is that it is quite easy to implement and is intuitive in its mechanisms, requiring no special knowledge to understand.

Two further methodological notes that ought to be emphasized are: (a) proper accuracy coefficients should be used that are resilient to prevalence, such as the J coefficient (see the appendix), and (b) it would be of value for future research to consider the confidence intervals when comparing binary classifiers. By just considering the obtained J values, one would have been tempted to conclude that a more complex classifier (to build and to understand) has the best predictive performance. Where, in fact, the evidence when one considers sampling variability does not support that conclusion.

5 Conclusions

The objective of this paper was to evaluate the predictive performance of case-based reasoning classifiers with different parameters. An empirical evaluation was conducted using a large data set from a large real-time system. Our results indicate that there is no difference among all of the different CBR classifiers. We therefore recommended the use of a simple CBR classifier due to its intuitive appeal to nonspecialists and the fact that our evidence shows it performs as well as other classifiers.

Furthermore, our methodology re-enforces the need to consider prevalence independent measures of prediction accuracy and the use of confidence intervals in the comparison of binary classifiers.

It would be informative for future research to first focus on confirmatory studies of our results, and secondly to compare the predictive performance of other classification techniques with this simple CBR classifier to determine if any of them really are superior. While such comparisons may not unearth a truly "best" classifier across all data sets, it may help discount classifiers that do not seem to add much prediction performance value. Furthermore, additional research on the interpretation of the J coefficient would be beneficial for future studies that are not comparative in nature.

6 Appendix: A Coefficient for Evaluating Binary Classifiers

The purpose of this appendix is to show that current criteria that are used in software engineering to evaluate the accuracy of binary classifiers have some important deficiencies. The deficiencies surface when one is either evaluating a binary classifier on a single data set with the hope that the accuracy results will generalize to real projects, and when one is comparing binary classifiers on the same data set. We then propose a new coefficient that addresses these deficiencies. This new coefficient was originally proposed in medicine to evaluate the accuracy of diagnostic tests, and is used in our study.

6.1 Definitions and Notation

A classification algorithm is used to develop a classifier that assigns a risk class to unseen observations. We restrict ourselves to binary risk classes: High and Low. If we let our case base be D_X, and let Q be the space of unseen observations, then a classification algorithm maps a given data set D into a classifier C, and a classifier C maps an unseen instance $q \in Q$ to a binary value $y \in$ {High, Low}. We use the notation Z(D,q) to denote the binary value assigned to an unseen case q by the classifier built by the algorithm Z using data set D.

D_R is a new data set that is obtained from a real project that wishes to use the classifier after it has been evaluated. The analyst does not have access to D_R during the evaluation[12], but hopes that the results of the evaluation using D_X are indicative of the performance of the classifier when it is used in the future on D_R. Therefore, we make a distinction between the *evaluation* phase of using a classifier, and the *application* phase.

An intuitively obvious way to evaluate the overall "goodness" of a classifier is to calculate the proportion of correct classifications. The proportion correct has been used in a number of previous studies to evaluate classifiers (Almeida et al., 1998; Lanubile and Visaggio, 1997; Schneidewind, 1994).

One commonly used sampling approach for estimating the accuracy of a classifier on unseen cases is to use a holdout sample. With the holdout sample method, accuracy is calculated as:

$$Accuracy = A = \frac{1}{N} \times \sum_{(q_i, y_i) \in D_h} \lambda(Z(D_X, q_i), y_i) \qquad \textbf{Eqn. 4}$$

where D_h is the hold-out sample (the test set in our study), $D_h \cap D_X \cap D_R = \Theta$, $|D_h| = N$, and $\lambda(i, j) = 1$ if $i = j$ and zero otherwise.

Table 7 shows the notation in obtained frequencies when a classifier is used to predict the class of unseen observations in a confusion matrix.

		Predicted Risk Class for D_h		
		Low	**High**	
Real Risk Class in D_h	**Low**	n_{11}	n_{12}	N_{1+}
	High	n_{21}	n_{22}	N_{2+}
		N_{+1}	N_{+2}	N

Table 7: Notation for a confusion matrix.

If the obtained accuracy on D_h is high enough, the analyst would then apply the classifier on real projects (i.e., on data set D_R).

[12] Or if s/he does, the values of y in D_R are unknown.

Such a confusion matrix also appears frequently in the medical sciences in the context of evaluating diagnostic tests, for example, see (Gordis, 1996). Two important parameters have been defined on such a matrix that will be used for our exposition, namely sensitivity and specificity.[13]

The *sensitivity* of a classifier is defined as:

$$s = \frac{n_{22}}{n_{21} + n_{22}} \qquad \textbf{Eqn. 5}$$

This is the proportion of high risk components that have been correctly classified as high risk components.

The *specificity* of a classifier is defined as:

$$f = \frac{n_{11}}{n_{11} + n_{12}} \qquad \textbf{Eqn. 6}$$

This is the proportion of low risk components that have been correctly classified as low risk components.

Ideally, both the sensitivity and specificity should be high. A low specificity means that there are many low risk components that are classified as high risk. Therefore, the organization would be wasting resources reinspecting or focusing additional testing effort on these components. A low sensitivity means that there are many high risk components that are classified as low risk. Therefore, the organization would be passing high risk components to subsequent phases or delivering them to the customer. In both cases the consequences may be expensive field failures or costly defect correction later in the life cycle.

Finally, the hold-out prevalence[14] of high risk components is defined as:

$$p_h = \frac{n_{21} + n_{22}}{N} \qquad \textbf{Eqn. 7}$$

This is the proportion of high risk components in the hold-out sample (i.e., the prevalence of "High" risk components in the test sample D_h that was used as the basis for evaluating the classifier).

[13] The terms sensitivity and specificity were originally used by Yerushalmy (Yerushalmy, 1947) in the context of comparing the reading of X-rays, and have been used since.

[14] This is also sometimes referred to as the "base rate".

Furthermore, let π be the (unknown) proportion of observations in D_R that have a "High" risk class, and consequently $1-\pi$ is the proportion of observations with a "Low" risk class. We will refer to π as *actual prevalence* (i.e., the prevalence of high risk classes in contexts where the classifier is actually going to be used for making decisions).

6.2 Generalization and Comparison

Software engineering studies that evaluate binary classifiers tend to be of two kinds:

- Evaluating the accuracy of a classifier on some data set where the actual risk classes of the components are known (i.e., build the classifier on D_X and evaluate it on D_h). The results of this evaluation are assumed to be a reflection of how well the classifier will perform when used on D_R (i.e., the evaluation results will generalize).
- The accuracy of two or more classifiers are compared on the same data set. The classifier that has the largest accuracy is assumed to be better. It is further assumed that the same high accuracy of the best classifier will be obtained when it is used on D_R.

It is therefore necessary that studies of classifiers use accuracy measures that will not be conducive to misleading conclusions under both of the above contexts.

6.3 Defining a "Good" Classifier

Intuitively appealing and necessary conditions for a binary classifier to be considered "good" during evaluation are twofold:

- It must perform better than would be expected by simple guessing (i.e., assigning unseen observations to classes at random with prior probabilities of 0.5)
- It must do better than always assigning observations to the most frequent class (i.e., if $p_h > 0.5$ then always classify as "High" risk, and if $p_h < 0.5$ then always classify as "Low" risk).

With a binary classification, one would expect to obtain 50% proportion correct accuracy by guessing with no prior information[15]. As the accuracy increases above 50%, the more confidence one gains about the usefulness of the classifier for prediction[16]. In many cases, software engineering researchers take 50% as an implicit minimal accuracy for a binary classifier[17]. This threshold has also been made explicit in the

[15] With no prior information means assigning observations to either of the Low or High risk classes with a probability of 0.5.

[16] Most research in quality modeling does not take misclassification costs into account - except for example (Khoshgoftaar and Allen, 1997). This is partly due to the difficulty in obtaining data on the cost of misclassification. Improved estimates about the usefulness of classifiers could be made if such costs were available.

[17] No evaluative judgement is made here about whether this is the best accuracy value to use. Only that it is commonly used, indicating acceptance amongst researchers as a reasonable threshold for binary classifiers.

past (Lanubile and Visaagio, 1997). Therefore, classifiers that have accuracies above 50% are considered to be "good".

In many cases it is possible to obtain rather good proportion correct accuracies by simply classifying observations in the most frequent category. This is particularly true when there are few "High" risk components and therefore classfiying everything as "Low" risk would still give a good proportion correct accuracy result.

		Predicted Risk Class		
		Low	High	
Real Risk Class	Low	80	0	80
	High	5	0	5
		85	0	85

Table 8: An example of a classifier that always predicts all risk classes as "Low".

Table 8 gives an example of such a classifier. Here the classifier always predicts "Low" risk, however its proportion correct accuracy is a respectable 94%. Few would argue that such a classifier is very useful, despite its high proportion correct accuracy.

While this specific problem has not been articulated explicitly in the software engineering literature, in other areas where validation coefficients for nominal scales were considered, corrections for this kind of "accidental" accuracy have been proposed (Brennan and Prediger, 1981).

We therefore expect that any criterion for evaluating and comparing binary classifiers to account for both of the above, otherwise misleading conclusions may ensue.

6.4 Software Engineering Contexts for Evaluating Classifiers

In this subsection we show that in software engineering studies, it is common that $p_h \neq \pi$. This has important implications on the types of coefficients that are used to evaluate binary classifiers.

It is frequent in software engineering studies that the dependent variable is not binary in nature, but is dichotomized by the researchers. For example, the dependent variable may be the number of field defects found. This would then be dichotomized into "High" and "Low" categories.[18]

The dichotomization may be based on expert opinion, for example, as in (Khoshgoftaar et al., 1996b), or may be based on the data itself. When dichotomization is based on the data itself, one of the following options is frequently followed:

- The authors perform a median split of the dependent variable, and then use a leave-one-out cross-validation approach to estimate the accuracy values. For example, see (Almeida et al.,

1998; Basili at al., 1997). The consequence of this is that $p_h \approx 0.5$. It is unrealistic to assume that in D_R exactly half of the components will be high risk, and therefore $p_h \neq \pi$. In other studies a sample is constructed to ensure equal numbers of high and low risk components (Briand et al., 1993b).

- Some studies performed a median split on the training data set, and the *same* cutoff value is used on the hold-out sample which can be a different system within the same organization. As is clear in (El Emam, 1998), median splits on one system do not result in the same prevalence for another system even within the same organization, i.e., this does not necessarily result in a $\pi = 0.5$.

- Some authors perform splits on quartiles, such as (Khoshgoftaar et al., 1997a) where splits were performed on the third quartile. In addition to the fact that quartile splits do not guarantee any more than do median splits that $p_h = \pi$, in this particular study the quartile split was performed on the hold-out sample as well (i.e., the cutoff point was different for the training and the hold-out samples rather than being the same). This will ensure that $\pi = 0.25$ only if the dependent variable in D_R is already known, which is not the case.[19]

- Student programs are used and the holdout sample is selected randomly. However, notwithstanding any other deficiencies in using student programs, it is not clear to what extent p_h is representative of any realistic π in an application context, irrespective of which of the above techniques is used for splitting.

While the above scenarios do not characterize all studies that evaluate binary classifiers, they still do represent a significant proportion of the studies that are conducted that any evaluation criterion must be able to accommodate them.

6.5 Evaluative Measures in Software Engineering

A number of different measures are used to evaluate binary classifiers in software engineering research. Some of these are descriptive and some are inferential. We review these briefly below.

6.5.1 Chi-Square Test

Some software engineering authors have used a chi-square inferential test for the 2x2 confusion matrix as the basis for determining predictive validity (Basili et al., 1997; Almeida et al., 1998; Lanubile and

[18] Where dichotomization is not performed and the prediction is made of the number of defects found directly, then this is outside the scope of our paper since we are only interested in binary classifiers.

[19] If the dependent variable values are known in D_R then there is no point in building a classifier.

Vissagio, 1997; Schneidewind, 1997). It has also been suggested that classifiers that do not pass the chi-square test of significance should be discarded (Lanubile and Vissagio, 1997).

The current criterion is to first calculate a χ^2 value. This value compares the observed confusion matrix with the expected confusion matrix under the null hypothesis of independence between the column and row variables. If the value of an obtained χ^2 is larger than a critical value then it is claimed that the classifier has predictive validity. The critical value can be obtained from tables of the chi-square distribution with one degree of freedom for a given alpha level.

It has been noted that this test evaluates the capability of a classifier to predict future behavior from past behavior, or the ability of a classifier to discriminate between high risk and low risk software components (Lanubile and Vissagio, 1997). Furthermore, it has been such that a classifier that does not meet the criterion of predictive validity using this test is rejected from further consideration (Lanubile and Vissagio, 1997).

6.5.2 Sensitivity and Specificity

Some authors, such as (Almeida et al., 1998), report the sensitivity and specificity values directly.

6.5.3 Proportion Correct

Most authors will report the proportion correct value, for example (Almeida et al., 1998; Lanubile and Vissagio, 1997; Schneidewind, 1994), also called correctness in (Porter and Selby, 1990; Porter, 1993). This is an intuitively appealing measure of prediction performance since it is easy to interpret.

6.5.4 Type I and Type II Misclassifications

These two evaluative measures are used in a number of different studies, such as (Khoshgoftaar et al, 1995b; Koshgoftaar et al., 1996b; Khoshgoftaar et al., 1999; Koshgoftaar et al., 1997a). The Type I misclassification rate is $1-f$, and the Type II misclassification rate is $1-s$.

6.5.5 Completeness and Correctness

Correctness and completeness are also frequently used measures for evaluating binary classifiers. Completeness, as defined in (Briand et al., 1993a; Porter and Selby, 1990), is equal to the senstivity. Correctness is the true positive rate (Briand et al., 1993a; Briand et al., 1998; Briand et al., 1999; Briand et al., 2000), also called consistency in (Porter and Selby, 1990; Porter, 1993), and is defined as:

$$Corr = \frac{n_{22}}{N_{+2}} \qquad \textbf{Eqn. 8}$$

The logic of considering correctness is that it gives an indication of how many of the "High" risk predictions made by the classifier are actually correct. If correctness is low then there will be extensive

wasted effort since many of the components identified by the classifier as high risk will in actuality be low risk.

6.5.6 Kappa

In some recent studies the Kappa coefficient has been used as a measure of prediction performance of binary classifiers (Briand et al., 1998; Briand et al., 2000). The Kappa coefficient was originally devised by Cohen to evaluate inter-observer agreement in their classifications on a nominal scale (Cohen, 1960)[20]. It is defined as follows:

$$\kappa = \frac{P_O - P_e}{1 - P_e} \qquad \textbf{Eqn. 9}$$

where P_O is proportion correct accuracy as defined above (i.e., equivalent to obtained agreement) , and:

$$P_e = \sum_{i=1}^{2} \frac{N_{i+} N_{+i}}{N^2} \qquad \textbf{Eqn. 8}$$

The above marginal proportions are maximum likelihood estimates of the population proportions under a multinomial sampling model. If a classifier assigns risk classes at random according to the marginal proportions, then the above is chance agreement (derived using the multiplication rule of probability and assuming independence between the row and column variables).

The observed "agreement" that is in excess of chance "agreement" is given by P_O - P_e. The maximum possible excess over chance agreement is 1 - P_e. The definition of Kappa is then the "agreement" obtained beyond that which would be expected by chance compared to the maximum possible "agreement" that could be obtained.

When there is complete "agreement" between the classifier and the actual risk status, P_O will take on the value of 1. In this case, the coefficient is 1. If observed "agreement" is greater than chance, then the coefficient is greater than zero. If observed "agreement" is less than would be expected by chance, then the coefficient is less than zero.

The Kappa coefficient can therefore be expressed as:

$$\kappa = \frac{\frac{n_{11} + n_{22}}{N} - \sum_{i=1}^{2} \frac{N_{+i} N_{i+}}{N^2}}{1 - \sum_{i=1}^{2} \frac{N_{+i} N_{i+}}{N^2}} \qquad \textbf{Eqn. 10}$$

[20] Cohen also defined a weighted version of Kappa (Cohen, 1968), but the characteristics that are of interest to us are the same for Kappa and its weighted version.

6.5.7 Summary

In addition to the fact that different authors use different measures, they also frequently report *multiple* measures in a single study. While multiple measures for a single study may be useful for getting an overall intuitive picture of prediction performance, they also present a difficulty in drawing conclusions. For example, if in a comparison study the classifier U was found to better than classifier W on one measure, but the opposite was true on another measure, drawing objective conclusions becomes more difficult. To illustrate this point we refer, for example, to the results in (Khoshgoftaar et al., 1997a). Here the authors were comparing nonparametric discriminant analysis with case-based reasoning classification. The Type I misclassification rate was lower for discriminant analysis, but the Type II misclassification rate and the overall proportion correct accuracy were lower for case-based reasoning. From these results it is not clear which modeling technique is superior.

Therefore, ideally, there ought to be a *single* evaluative measure that can be used for evaluation and comparison, which in turn would lead to consistently objective conclusions. Below we highlight deficiencies in most of the above measures individually. We divide this into two parts, the inferential measure, namely the chi-square test, and the descriptive measures. This is then followed by a proposed new single measure that addresses these deficiencies.

6.6 An Appraisal of the Chi-Square Test

The chi-square testing approach has two disadvantages:

- This test, when used in the context of evaluating classifiers, does not provide useful information about the utility of the classifier. First, when the null hypothesis is rejected, this does not signify that a classifier is necessarily “good”. Second, when the null hypothesis is not rejected, this does not necessarily signify that the classifier is “bad”.
- In practice, the test is not used alone but in conjunction with another criterion: classification accuracy. Therefore, the real decision process consists of two criteria. The Type I error rate (in a statistical sense) of the whole decision process is conservative and results in the analyst actually operating at a lower alpha level than s/he stipulates. This consequently results in loss of power (i.e., inability to identify “good” classifiers).

Here we show that the chi-square approach does not actually provide useful information *the way it is applied in practice*. It is shown below that a classifier that *does* pass the chi-square criterion should not necessarily be accepted as “good”. Conversely, if the chi-square criterion can identify classifiers that should be rejected (i.e., they do not pass the chi-square criterion), then perhaps it still has some value. However, it also shown that this criterion does reject classifiers that are “good”.

The chi-square test is non-directional. This means that association on the left diagonal is as good as association on the right diagonal of the confusion matrix[21]. For example, the confusion matrix in Figure 4 shows a classifier that has a high chi-square value (approximately 15 after a correction for continuity) that is statistically significant at an alpha level of 0.05. The confusion matrix in Figure 5 gives exactly the same result in terms of chi-square. It is clear that the classifier of Figure 4 has greater accuracy (approximately 74%) than that the one in Figure 5 (approximately 26%). Therefore, the chi-square criterion by itself does not tell us whether the classifier is "good" or not.

		Predicted Class		
		LOW	**HIGH**	
Real Class	**LOW**	25	10	35
	HIGH	10	31	41
		35	41	76

Figure 4: Confusion matrix with significant chi-square and high concordance.

		Predicted Class		
		LOW	**HIGH**	
Real Class	**LOW**	10	25	35
	HIGH	31	10	41
		41	35	76

Figure 5: Confusion matrix with significant chi-square and low concordance.

Proponents of the chi-square testing approach may argue that the test is not used in a blind fashion. If patterns such as those shown above are obtained, and the test is statistically significant, then it is obvious whether this is "good" or not because of high or low concordance between the actual and predicted values.

However, it is plausible that even though the chi-square test is significant, it is not possible to tell whether this is because of high or low concordance between the actual and predicted values. Consider the confusion matrix in Figure 6. The chi-square test here is statistically significant at an alpha level of 0.05 (with a χ^2 value of 4.96). Exactly half of the classifications are on the right diagonal, and half on the left diagonal, giving an accuracy of 50%.

		Predicted Class		
		LOW	**HIGH**	
Real Class	**LOW**	19	37	56
	HIGH	1	19	20
		20	56	76

Figure 6: Confusion matrix with unclear concordance.

[21] In all examples we use Pearson's chi-square statistic with Yates' continuity correction. Furthermore, all examples use N's that are common in the software engineering literature , for example, see (Almeida et al., 1998; Basili et al., 1997).

Let's say that the criterion is changed so that the accuracy has to be greater than 50% for a classifier to be claimed to have predictive validity. Therefore, the whole decision process is actually a two stage one:

- determine whether the chi-square value is statistically significant, then
- if it is, determine whether the accuracy is greater than 50%

If both of the above criteria are satisfied, then it can be claimed that the classifier has predictive validity. In such a case, the null hypothesis that is being tested is not that of independence, but of *independence and accuracy less than or equal to 50%*.

However, this approach would lead to a test that is conservative. This means that the Type I error rate (the probability of rejecting the null hypothesis when it is true) would be lower than the nominal alpha rate. First we illustrate this. We conducted a Monte Carlo simulation to determine the actual Type I error rate if the above decision process was implemented. We chose observation sizes (NOBS) ranging from 10 to 100 in increments of 10. For each number of observations we chose the number of actual Low risk components to be $x \times \frac{NOBS}{10}$ where x was varied from 1 to 5. The classifier we implemented was a guessing classifier that would assign observations to each of the two risk classes with a probability of 0.5. The above scheme for evaluating predictive validity was implemented for an alpha level of 0.1 and 0.05. For each combination of NOBS, x, and alpha value we performed 5000 iterations. Under this simulation we would hope that the proportion where the null hypothesis of *independence and accuracy less than or equal to 50%* to be rejected at the same rate as the alpha level.

The results are shown in Table 9 for an alpha level of 0.1, and Table 10 for an alpha level of 0.05. It is clear from these tables that the null hypothesis is rejected at much smaller proportions than the nominal alpha level. In general, it is rejected half the number of times that you would expect. This means that this approach is conservative, and assigns a lower risk to the test than one was willing to accept. In some cases, it is rejected ten times less. By implication, this also means that the power of this test is reduced.

NOBS/x	1	2	3	4	5
10	0.010	0.039	0.088	0.070	0.059
20	0.056	0.051	0.059	0.066	0.057
30	0.064	0.061	0.059	0.060	0.052
40	0.049	0.046	0.052	0.055	0.044
50	0.045	0.057	0.050	0.049	0.063
60	0.047	0.058	0.051	0.053	0.044
70	0.054	0.051	0.052	0.053	0.060
80	0.044	0.051	0.051	0.055	0.046
90	0.051	0.056	0.049	0.047	0.052
100	0.047	0.050	0.054	0.050	0.045

Table 9: Proportion of times the null hypothesis is rejected for alpha equal to 0.1.

NOBS/x	1	2	3	4	5
10	0.010	0.012	0.032	0.021	0.034
20	0.010	0.029	0.020	0.030	0.017
30	0.013	0.026	0.028	0.026	0.022
40	0.029	0.021	0.028	0.024	0.022
50	0.025	0.030	0.023	0.026	0.038
60	0.024	0.030	0.028	0.031	0.026
70	0.028	0.024	0.025	0.025	0.024
80	0.022	0.025	0.022	0.025	0.028
90	0.025	0.027	0.024	0.025	0.020
100	0.021	0.028	0.027	0.028	0.027

Table 10: Proportion of times the null hypothesis is rejected for alpha equal to 0.05.

The reason is that the chi-square test would normally reject the null hypothesis of no association a proportion of times that is almost equal to the nominal alpha rate. However, because of the decision to only reject the overall null hypothesis when accuracy is greater than 50%, this means that the null hypothesis would not be rejected in some of the cases where the chi-square test rejects the independence (sub-)hypothesis. The simulation shows the extent to which this would happen.

The above discussion illustrates that when the chi-square test gives a statistically significant result, this by itself does not provide information about whether the classifier is "good" or not. When combined with another criterion, the whole decision process has a rather conservative Type I error rate.

However, when the chi-square test is *not* significant, does it provide useful information ? Can it be used to eliminate classifiers that have no predictive power ? If this is the case, then the test still has merit. Unfortunately, it can be shown that the chi-square criterion by itself would eliminate classifiers that are "good" by the accepted criterion of accuracy greater than 50%.

Consider the confusion matrix in Figure 7. This has a chi-square value that is *not* statistically significant, even though this classifier has an accuracy of 75%, which is quite large. The probability of getting 75% correct classifications for 100 unseen cases is less than 0.0001 for a classifier that guesses at 0.5 probabilities (this can be calculated using a Monte Carlo simulation), which *is* statistically significant[22].

[22] The reason this happens is because the chi-square test does not assume under the null hypothesis of independence that a classifier classifies randomly at probabilities of 0.5. Expected frequencies are calculated from the observed marginal totals. In the example, if a random classifier assigns cases to "Low" with 0.75 probability and "High" with 0.25 probability, it would be expect to have an accuracy of 70% in the long run. Therefore, a different threshold is effectively used compared to the one that is assumed for evaluating how "good" a classifier is. Actually, the probability of getting 75 correct classifications or more for the "Real Class" marginal totals in Figure 7 using a guessing classifier that classifies to the Low risk category at a 0.75 probability and the High risk category at a 0.25 probability is approximately 0.137 (this value can calculated using a Monte Carlo run). This is not statistically significant.

		Predicted Class		
		LOW	**HIGH**	
Real Class	**LOW**	70	20	90
	HIGH	5	5	10
		75	25	100

Figure 7: Confusion matrix for a "good" classifier that provides a chi-square test that is not statistically significant.

In the above exposition we have presented scenarios illustrating the difficulty in interpreting the chi-square test for evaluating binary classifiers. While there will be instances where the interpretation of the chi-square test results will be clear, one cannot recommend an evaluative criterion that sometimes works and at other times does not. Furthermore, the Monte Carlo simulation highlighted the conservatism of the test, suggesting that one should not rely on its conclusions. Finally, the chi-square test, as described above, is not appropriate for comparing classifiers. Therefore, we consider some of the alternatives below.

6.7 An Appraisal of the Descriptive Measures

In this subsection we provide a critical appraisal of the descriptive measures of accuracy. We first present a series of examples that we will use throughout to illustrate our points.

6.7.1 Definition of Examples

We define two sets of examples, the first are used to illustrate the generalizability problems, and the second to demonstrate the comparison problems.

Example No.	f	s	p_h	π
1	0.9	0.7	0.9	0.2
2	0.9	0.7	0.2	0.9
3	0.95	0.1	0.5	0.05

Table 11: Examples used to illustrate generalizability problems.

Table 11 shows the examples to illustrate generalizability problems. Example 1 is a classifier with a specificity of 0.9 and a sensitivity of 0.7. The hold-out sample has a prevalence of 0.9 and the actual data set has a prevalence of 0.2 (i.e., the actual data set's prevalence is much smaller than the hold-out sample). The second example is for the same classifier except that the prevalances have been switched. The third example is a classifier that can classify "Low" risk components very well, with a specificity of 0.95, but that is rather bad at classifying "High" risk components. The hold-out sample has 50% of its components high risk, but the actual sample has very few "High" risk components (only 5%).

Example No.	f	s	p_h
4	0.8	0	0.2
5	0.6	0.8	0.2

Table 12: Examples used to illustrate comparison problems.

Table 12 shows examples of two classifiers to be compared. In both cases the hold-out prevalence is the same since we assume that the classifiers are compared on the same data set. In example 4, the classifier classifies all components in the "Low" risk category, hence its senstivity is always zero. The fifth example shows a more balanced classifier.

In all our examples we take it for granted that sensitivity and specificity estimated on the hold-out sample will be stable on a real project data set. If this assumption is not tenable than there is no value to any evaluative study of quality models since then one can never generalize from evaluative studies to actual practice.

6.7.2 Proportion Correct Accuracy

Below we show that the use of proportion correct classifications as a measure of accuracy, and consequently any inferential test based on it (such as that proposed in (El Emam, 1998)), can produce quite erroneous results in the context of identifying risky software components. The reason is that A has a strong dependence on the prevalence p_h. We will first show this dependency and then illustrate its detrimental effects with reference to the examples.

We can make the following definitions:

$$n_{22} = s \times p_h \times N \qquad \textbf{Eqn. 11}$$

$$n_{11} = f \times (1 - p_h) \times N \qquad \textbf{Eqn. 12}$$

And therefore

$$A = \frac{(s \times p_h \times N) + (f \times (1 - p_h) \times N)}{N} = (s \times p_h) + (f \times (1 - p_h)) \qquad \textbf{Eqn. 13}$$

Now, consider that a classifier has been developed and it has been evaluated on the holdout sample D_h. From this evaluation, the proportion correct accuracy using D_h, say A_h, has been calculated. Let's say that the value A_h was sufficiently high, and therefore now we want to use it on a real project to make predictions. We expect that the accuracy A_h obtained during the evaluation will be reflective of the accuracy of the predictions that are made on the data set D_R, such that if we were able to calculate A_R, the accuracy on the real project, we would have $A_R \approx A_h$.

In fact, this expectation is quite misguided as, by definition, it would only occur under some rather restrictive conditions.

Let's consider our first example in Table 11. This was for a classifier with $f = 0.9$ and $s = 0.7$ respectively. For this example, we can calculate $A_h = 0.72$ and $A_R = 0.86$. Thus, during the evaluation on data set D_h we have substantially underestimated the proportion correct accuracy of our classifier. A different situation would occur if we reverse the prevalences as in example 2 in Table 11. Then we have $A_h = 0.86$ and $A_R = 0.72$. In such a case, the evaluation on data set D_h overestimates the accuracy of the classifier when it is used in a real project. The only condition when the two accuracies are the same is when $p_h = \pi$ (i.e., the evaluation data set has the same prevalence of high risk components as actual projects).

The reason is that this classifier does a good job of classifying low risk components, and a less proficient job at classifying high risk components. Therefore, if the sample has more low risk components, the proportion correct accuracy will be better. But, if the sample has more high risk components, the proportion correct accuracy will deteriorate.

It is quite easy to conclude from this that the proportion correct accuracy is not a reflection of the performance of a classifier on a project with different prevalences from the evaluation data set.[23] This problem is further exacberated because some researchers artificially construct their evaluation data sets so that 50% of the observations are in each class. Example 3 in Table 11 is an extreme case where the classifier does not do a good job in identifying "High" risk components. In such a case, $A_h = 0.525$ and $A_R = 0.91$. This indicates that on the hold-out sample with 50% "High" risk components the classifier performs barely better than our 50% acceptable accuracy, but on the real project data set it has a remarkably high accuracy. Therefore, making the reasonable assumption that the prevalence of high risk components on real systems is considerably less than 50%, it is likely that many past evaluation studies

[23] It is plausible that the prevalences between the evaluation data set and the actual project are the same or almost the same in situations where a classifier is constructed using release k, and then it is used to make predictions for release k+1.

underestimated the proportion correct accuracy of their classifiers through the practice of creating artificial samples, and that many useful classifiers were discarded.

It is also easy to conclude that efforts to construct null hypothesis tests for the proportion correct accuracy by comparing obtained values with the "guessing" null distribution, such as (El Emam, 1998), do not solve the problem since they still rely on the proportion correct accuracy as the test statistic.

It can also be demonstrated that when *comparing* classifiers, misleading conclusions can be drawn if one uses the proportion correct accuracy as the comparison criterion. Consider the two classifiers represented by examples 4 and 5 in Table 12.

Classifier 4 has absolutely no capability in classifying any of the high risk components correctly. The proportion correct accuracy of both of these classifiers is 0.64 for a prevalence of 0.2. Therefore, on the same data set with the same prevalence, both classifiers look the same in terms of proportion correct accuracy. However, intuitively, the first classifier is quite useless.

Therefore, we can conclude that even for comparing different classifiers, the use of proportion correct accuracy can provide quite misleading results, and should therefore be abandoned.

The problem with using proportion correct accuracy is known in the medical sciences (Gordis, 1996; Vecchio , 1966). For example, If one is evaluating a diagnostic test in a sample of the population that is at high-risk of having a disease, then prevalence is likely to be higher than if evaluating it in a sample from the general population. The results from both of these evaluation contexts can give dramatically different results as to the accuracy of the test, and certainly one cannot justify applying the test using only the accuracy information from the high risk sample with the intent of using it on the general population.

6.7.3 Correctness

Correctness can be reformulated as follows:

$$Corr = \frac{sp_h}{(1-p_h)(1-f)+sp_h} \qquad \textbf{Eqn. 14}$$

This equation makes clear that correctness is highly dependent on prevalence. Let us consider example 1 in Table 11. We have an appealing correctness of 0.98 on the evaluation data set, but a correctness of only 0.636 on the real project data set, D_R. This is a substantial reduction in correctness that would occur in practice compared to the evaluation results. The relationship between prevalence and correctness is clearly seen in Figure 8. A situation more congruent with current practice is exemplified by example 3. Here we have a seemingly respectable correctness of 0.67 for the evaluation data set, but this drops to a negligible 0.09 for the actual data set. Therefore, from these examples it can be seen that correctness is extremely sensitive to prevalence.

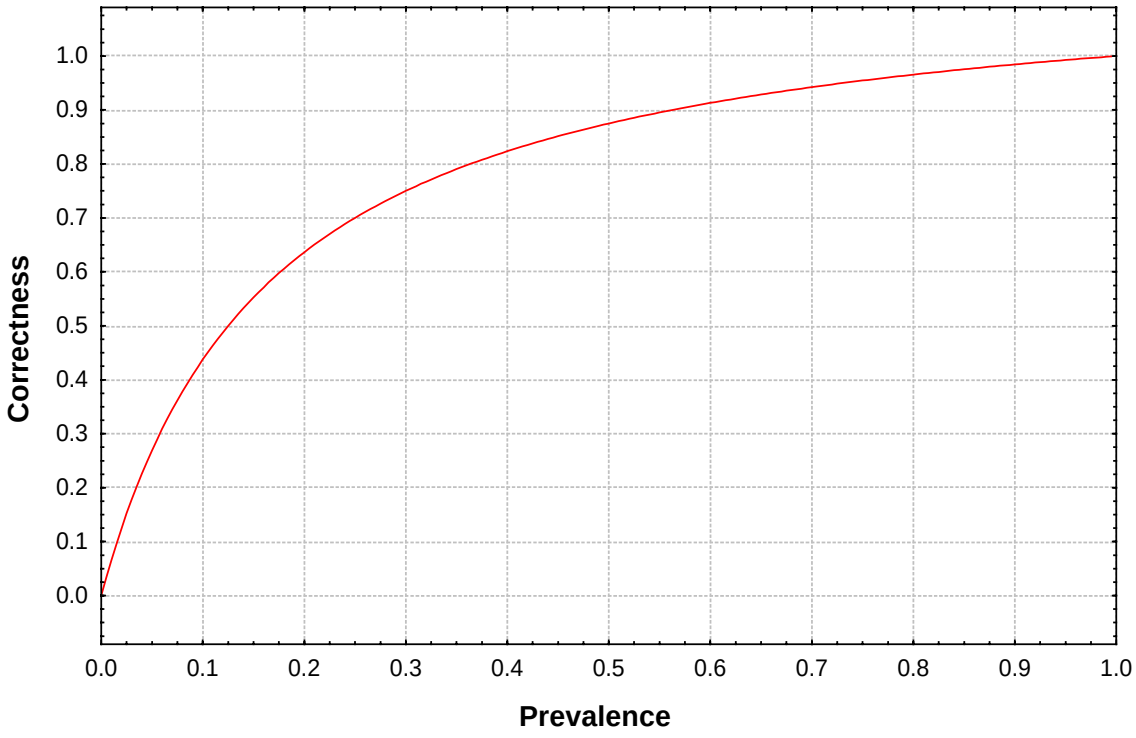

Figure 8: Plot of prevalence versus correctness for s=0.7 and f=0.9 (examples 1 and 2 in Table 11).

6.7.4 The Kappa Coefficient

It is known that the Kappa coefficient has a strong dependence on prevalence (Grove et al., 1981; Spitznagel and Helzer, 1985). The Kappa coefficient can be expressed in terms of prevalence, the sensitivity, and the specificity as follows:

$$\kappa = \frac{2p_h(p_h - 1)(1 - f - s)}{(1 - p_h) - (1 - 2p_h)[f(1 - p_h) + p_h(1 - s)]} \qquad \textbf{Eqn. 15}$$

It is clear from this equation that Kappa depends strongly on the prevalence value. The problem with this dependence can be seen in Figure 9. At extremes of prevalance Kappa tends to decrease for a fixed sensitivity and specificity. Therefore, if the prevalence during evaluation, p_h, is set at 0.5 then Kappa is approximately 0.6, which would seem to be a high value. However, if during the application of the classifier prevalence, π, is lower than 0.1 then Kappa dips quite fast. Therefore, it is clear that the use of Kappa with the common practice of median splits can lead to rather optimistic conclusions about the value of a classifier.

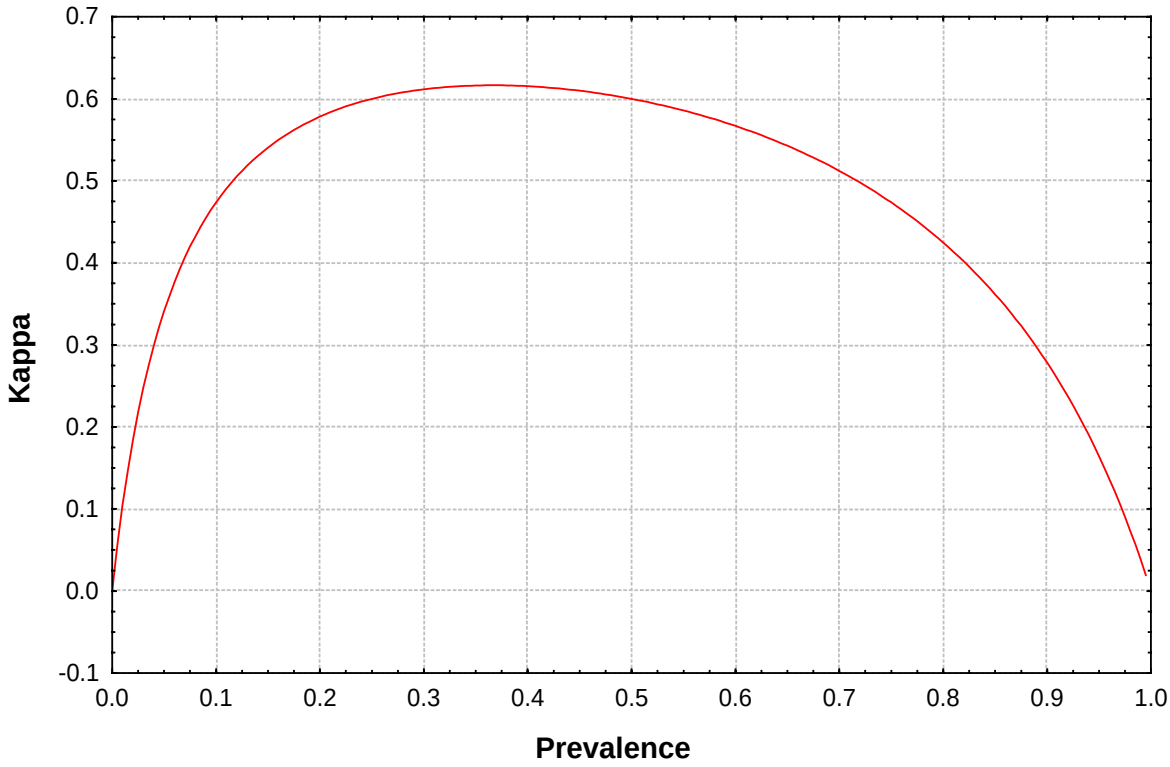

Figure 9: Plot of prevalence versus Kappa for s=0.7 and f=0.9 (examples 1 and 2 in Table 11).

It has been suggested that when one of the marginals is fixed (which is the case with a confusion matrix) that the chance "agreement" should be defined as proportion correct obtained by guessing, in this case with a prior probability of 0.5. However, as noted in (El Emam, 1998), this results in a linear transformation of the proportion correct coefficient, and therefore will still suffer from the same problems.

Another suggestion was to evaluate improvement over a "best" a priori strategy exemplified by $\max(p_h, 1-p_h)$ (Brennan and Prediger, 1981). Therefore, a new Kappa coefficient would be defined as:

$$\kappa_b = \frac{sp_h + f(1-p_h) - \max(p_h, 1-p_h)}{1 - \max(p_h, 1-p_h)} \qquad \textbf{Eqn. 16}$$

In the case where $\max(p_h, 1-p_h) = p_h$, we have:

$$\kappa_b = \frac{p_h(s-1) + f(1-p_h)}{1-p_h} \qquad \textbf{Eqn. 17}$$

and when $\max(p_h, 1-p_h) = 1-p_h$, we have:

$$\kappa_b = \frac{p_h(s+1) + f(1-p_h) - 1}{p_h} \qquad \textbf{Eqn. 18}$$

As can be seen, both of these equations also exhibit a strong dependence on the prevalence, and therefore the results obtained from the evaluation phase are unlikely to generalize to the application phase.

6.7.5 The Information Score

Recognizing the problems with criteria for evaluating classifiers, Kononenko and Bratko (Kononenko and Bratko, 1991) propose a new coefficient based on information theory. While this coefficient has not, to our knowledge, been used in software engineering, we include it here to pre-empt its application in a software engineering context. Their coefficient however explicitly takes into account the prevalence p_h. Therefore, given that in software engineering contexts $p_h \neq \pi$, it follows that such a coefficient would not be useful in our context. We illustrate this below.

In an evaluation context, the prior probability of a risk class $C \in \{High, Low\}$ is given by the prevalence p_h. For every observation i in D_h the posterior probability of being in a "High" risk class is denoted by p_i' and of being in a "Low" risk class by $1-p_i'$. Some classifiers, such as decision trees (Breiman et al., 1984; Quinlan, 1993) return a subset at the leaves and take the most frequent class as the predicted value. However, each leaf has a distribution and therefore p_i' can take on any value between 0 and 1. Other classifiers, such as k nearest neighbor with k=1, return a single similar observation from the case base, and therefore p_i' can have values of either 0 or 1.

The expected amount of information for classifying a single observation is given by the entropy:

$$E = (p_h \times \log_2(p_h)) + ((1-p_h) \times \log_2(1-p_h)) \qquad \textbf{Eqn. 19}$$

Let us say that the correct risk class of a particular observation is "High". The authors then define three cases:

1. If $p_i' > p_h$ then the probability of being in a "High" class has moved in the right direction.

2. If $p_i' < p_h$ then the probability of being in a "High" risk class has moved in the wrong direction.

3. If $p_i' = p_h$ then this classification contains no information.

An information score I is then defined as shown below.

If $p_i' \geq p_h$ then:

$$I_C(i) = -\log_2(p_h) + \log_2(p_i') \qquad \textbf{Eqn. 20}$$

which is the information necessary to correctly classify an instance into the "High" risk class, minus the remainder of information necessary to correctly classify that instance.

If $p_i' < p_h$ then:

$$I_M(i) = -\log_2(1-p_h) + \log_2(1-p_i') \qquad \textbf{Eqn. 21}$$

which is the information necessary to decide that an instance does not belong to the "High" risk class, minus the remainder of information necessary to make the decision.

The average information score is then given by:

$$I_a = \frac{1}{|D_h|}\sum_{i}^{|D_h|} I(i) \qquad \textbf{Eqn. 22}$$

where $I(i)$ is $I_C(i)$ or $I_M(i)$ depending on the case. The relative information score is defined as:

$$I_r = \left(\frac{I_a}{E}\right) \times 100 \qquad \textbf{Eqn. 23}$$

It is suggested that I_a be used to compare different classifiers on the same data set, and I_r be used to compare classifiers on different data sets.

Now, consider the scenario where we have a perfect classifier that always classifies in the correct class with a probability of 1. Furthermore, assume that our prevalence on the holdout sample, p_h, is 0.5. The average information score obtained during evaluation for this classifier is $I_a = 1$. However, if the actual prevalence $\pi = 0.2$, then we have the actual average information score of $I_a = 0.72$. Therefore, in this particular example, the performance of the classifier was exaggerated during the evaluation phase

because the evaluation prevalence was larger than the actual prevalence. By reversing the prevalence values we can get an example of an evaluation that underestimates the performance of a classifier.

6.7.6 Summary

The above exposition makes clear that the chi-square, proportion correct, correctness, Kappa, and information score measures are not adequate in a software engineering context. Furthermore, we have argued that multiple measures, such as Type I and Type II error rates when used together in comparing classifiers have been known to give conflicting results, making it more difficult to draw objective conclusions. Below we propose a single measure of accuracy that alleviates these problems.

6.8 The J Coefficient

To address deficiencies in evaluative measures in the medical sciences, Youdon has proposed the J coefficient[24] (Youden, 1950). The calculation of the J coefficient is described in the main body of the paper, but it can be expressed as:

$$J = s + f - 1 \qquad \textbf{Eqn. 24}$$

This coefficient has a number of desirable properties. First, it is prevalence independent. This can be illustrated with reference to our examples above. If our classifier has specificity and sensitivity equal to $f = 0.9$ and $s = 0.7$, then its J value is 0.6 irrespective of prevalence in the evaluation sample and in the real project. For our two classifiers in the second example set, the values would be $J_1 = -0.2$ and $J_2 = 0.4$ respectively, which makes intuitive sense since the first classifier uses only one class in its classifications.

The J coefficient is used in our study to evaluate and compare classifiers on the same data set.

7 Acknowledgements

We wish to thank Anatol Kark, James Miller, and Barbara Kitchenham for their comments and suggestions on an earlier version of this paper.

8 References

Almeida, M., Lounis, H., Melo, W., 1998. An Investigation on the Use of Machine Learned Models for Estimating Correction Costs. *Proceedings of the 20th International Conference on Software Engineering*, 473-476.

[24] Clearly, as noted in (Spitznagel and Helzer, 1985), it is plausible to devise a number of evaluative measures that are independent of prevalence. Amongst these, there is no compelling reason to prefer one over the other, and therefore we use the J coefficient due to its obvious relationship to specificity and sensitivity.

Althoff, K-D, Auriol, E., Barletta, R., Manago, M., 1995. *A Review of Industrial Case-Based Reasoning Tools*. AI Intelligence.

Armitage, P., Berry, G., 1994. *Statistical Methods in Medical Research*. Blackwell Science.

Basili, V., Condon, S., El Emam, K., Hendrick, R., Melo, W., 1997. Characterizing and Modeling the Cost of Rework in a Library of Reusable Software Components. *Proceedings of the 19th International Conference on Software Engineering,* 282-291.

Brennan, R., Prediger, D., 1981. Coefficient Kappa: Some Uses, Misuses, and Alternatives". *Educational and Psychological Measurement*, 41, 687-699.

Briand, L., Basili, V., Thomas, W., 1992. A Pattern Recognition Approach for Software Engineering Data Analysis. *IEEE Transactions on Software Engineering*, 18(11), 931-942.

Briand, L., Basili, V., Hetmanski, C., 1993a. Developing Interpretable Models with Optimized Set Reduction for Identifying High-Risk Software Components". *IEEE Transactions on Software Engineering*, 19(11), 1028-1044.

Briand, L., Thomas, W., Hetmanski, C., 1993b. Modeling and Managing Risk Early in Software Development. *Proceedings of the International Conference on Software Engineering*, 55-65.

Briand, L., Wuest, J., Ikonomovski, S., Lounis, H., 1998. A Comprehensive Investigation of Quality Factors in Object-Oriented Designs: An Industrial Case Study. International Software Engineering Research Network, technical report ISERN-98-29.

Briand, L., Wuest, J., Ikonomovski, S., Lounis, H., 1999. Investigating Quality Factors in Object-Oriented Designs: An Industrial Case Study. *Proceedings of the International Conference on Software Engineering*.

Briand, L., Wuest, J., Daly, J., Porter, V., 2000. Exploring the Relationships Between Design Measures and Software Quality in Object Oriented Systems. *Journal of Systems and Software* (to appear).

Breiman, L., Friedman, J., Olshen, R., Stone, C., 1984. *Classification and Regression Trees*. Wadsworth.

Coallier, F., Mayrand, J., Lague, B., 1999. Risk Management in Software Product Procurement. In: El Emam, K., Madhavji, N. H. (eds.). *Elements of Software Process Assessment and Improvement*, IEEE CS Press.

Cohen, J., 1960. A Coefficient of Agreement for Nominal Scales. *Educational and Psychological Measurement*, XX(1), 37-46.

Cohen, J., 1968. Weighted Kappa: Nominal Scale Agreement with Provision for Scaled Agreement or Partial Credit. *Psychological Bulletin*, 70, 213-220.

Dubes, R., Jain, A., 1980. Clustering Methodologies in Exploratory Data Analysis. Advances in Computers, 19, 113-228.

El Emam, K., 1998. The Predictive Validity Criterion for Evaluating Binary Classifiers. *Proceedings of the 5th International Symposium on Software Metrics*, 235-244.

Everitt, B., 1980. Cluster Analysis (2nd ed.), Heinemann.

Fenton, N., Neil, M., 1999. A Critique of Software Defect Prediction Models. *IEEE Transactions on Software Engineering*, 25(5), 675-689.

Fenton, N., Ohlsson, N., 2000. Quantitative Analysis of Faults and Failures in a Complex Software System. *IEEE Transactions on Software Engineering* (to appear).

Ganesan, K., Khoshgoftaar, T., Allen, E., 1999. Case-based Software Quality Prediction. *International Journal of Software Engineering and Knowledge Engineering* (to appear).

Gordis, L., 1996. *Epidemiology*. W. B. Saunders Company.

Gray, A., MacDonell, S., 1997. A Comparison of Techniques for Developing Predictive Models of Software Metrics. *Information and Software Technology*, 39, 425-437.

Grove, W., Andreasen, N., McDonald-Scott, P., Keller, M., Shapiro, R., 1981. Reliability Studies of Psychiatric Diagnosis: Theory and Practice. *Archives of General Psychiatry*, 38, 408-413.

Harrison, W., 1988. Using Software Metrics to Allocate testing Resources. *Journal of Management Information Systems*, 4(4), 93-105.

Hoaglin, D., Mosteller, F., Tukey, J. (eds.), 1983. *Understanding Robust and Exploratory Data Analysis*. Wiley.

Hosmer, D., Lemeshow, S., 1989. *Applied Logistic Regression*. John Wiley & Sons.

Kaaniche, M., Kanoun, K., 1996. Reliability of a Commercial Telecommunications System. *Proceedings of the International Symposium on Software Reliability Engineering*, 207-212.

Kafura, D., Henry, S., 1981. Software Quality Based on Interconnectivity. *Journal of Systems and Software*, 2, 121-131.

Kaufman, L., Rousseeuw, P., 1990. *Finding Groups in Data*. John Wiley & Sons.

Khoshgoftaar, T., Allen, E., Hudepohl, J., Aud, S., Mayrand, J., 1995a. Selecting Software Metrics for a Large Telecommunications System. *Proceedings of the Fourth Software Engineering Research Forum*, 221-229.

Khoshgoftaar, T., Allen, E., Kalaichelvan, K., Goel, N., Hudepohl, J., Mayrand, J., 1995b. Detection of Fault-Prone Program Modules in a Very Large Telecommunications System. *Proceedings of the 6th International Symposium on Software Reliability Engineering*, 24-33.

Khoshgoftaar, T., Allen, E., Kalaichelvan, K., Goel, N., 1996a. The Impact of Software Evolution and Reuse on Software Quality. *Empirical Software Engineering: An International Journal*, 1, 31-44.

Khoshgoftaar, T., Allen, E., Bullard, L., Halstead, R., Trio, G., 1996b. A Tree Based Classification Model for Analysis of a Military Software System. *Proceedings of the IEEE High-Assurance Systems Engineering Workshop*, 244-251.

Khoshgoftaar, T., Ganesan, K., Allen, E., Ross, F., Munikoti, R., Goel, N., Nandi, A., 1997a. Predicting Fault-Prone Modules with Case-Based Reasoning. *Proceedings of the Eighth International Symposium on Software Reliability Engineering*, 27-35.

Khoshgoftaar, T., Allen, E., Lanning, D., 1997b. An Information Theory-Based Approach to Quantifying the Contribution of a Software Metric. *Journal of Systems and Software*, 36, 103-113.

Khoshgoftaar, T., Allen, E., 1997. The Impact of Costs of Misclassification on Software Quality Modeling. *Proceedings of the 4th International Symposium on Software Metrics*, 54-62.

Khoshgoftaar, T., Allen, E., Jones, W., Hudepohl, J., 1999. Classification Tree Models of Software Quality Over Multiple Releases. *Proceedings of the International Symposium on Software Reliability Engineering*.

Kolodner, J., 1993. *Case-Based Reasoning*. Morgan Kaufmann Publishers Inc.

Kononeko, I., Bratko, I., 1991. Information-Based Evaluation Criterion for Classifier's Performance. *Machine Learning*, 6:67-80.

Lanubile, F., Visaggio, G., 1997. Evaluating Predictive Quality Models Derived from Software Measures: Lessons Learned. *Journal of Systems and Software*, 38, 225-234.

Mayrand, J., Coallier, F., 1996. System Acquisition Based on Software Product Assessment. *Proceedings of the 18th International Conference on Software Engineering*, 210-219.

McCabe, T., 1976. A Complexity Measure. *IEEE Transactions on Software Engineering*, 2(4), 308-320.

Milligan, G., Cooper, M., 1988. A Study of Standardization of Variables in Cluster Analysis. *Journal of Classification*, 5, 181-204.

Moller, K-H, 1993. An Empirical Investigation of Software Fault Distribution. *Proceedings of CSR*, 82-90.

Myrvold, A., 1990. Data Analysis for Software Metrics. *Journal of Systems and Software*, 12, 271-275.

Navlakha, J., 1987. Measuring the Effect of External and Internal Interface on Software Development. *Proceedings of the 20th Annual International Conference on the System Sciences*, 127-136.

Ohlsson, N., Alberg, H., 1996. Predicting Fault-Prone Software Modules in Telephone Switches. *IEEE Transactions on Software Engineering*, 22(12), 886-894.

Porter, A., Selby, R., 1990. Evaluating Techniques for Generating Metric-Based Classification Trees. *Journal of Systems and Software*, 12, 209-218.

Porter, A., 1993. Using Measurement-Driven Modeling to Provide Empirical Feedback to Software Developers. *Journal of Systems and Software*, 20, 237-243.

Quinlan, J., 1993. *C4.5: Programs for Machine Learning*. Morgan Kaufmann.

Schneidewind, N., 1994. Validating Metrics for Ensuring Space Shuttle Flight Software Quality. *IEEE Computer*, 50-57, August.

Schneidewind, N., 1997. Software Metrics Model for Integrating Quality Control and Prediction. *Proceedings of the 8th International Symposium on Software Reliability Engineering*, 402-415.

Spitznagel, E., Helzer, J., 1985. A Proposed Solution to the Base Rate Problem in the Kappa Statistic. *Archives of General Psychiatry*, 42, 725-728.

Vecchio, T., 1966. Predictive Value of a Single Diagnostic Test in Unselected Populations. *The New England Journal of Medicine*, 274(21), 1171-1173.

Venables, W., Ripley, B., 1997. *Modern Applied Statistics with S-Plus*. Springer.

Watson, I., 1997. *Applying Case-Based Reasoning: Techniques for Enterprise Systems*. Morgan Kaufmann Publishers Inc.

Weisberg, H., 1992. *Central Tendency and Variability*. Sage Publications.

Yerushalmy, J., 1947. Statistical Problems in Assessing Methods of Medical Diagnosis, with Special Reference to X-Ray Techniques. *Public Health Report*, 62, 1432-1449.

Youden, W., 1950. Index for Rating Diagnostic Tests. *Cancer*, 3, 32-35.