Cost Curve Evaluation of Fault Prediction Models

Yue Jiang, Bojan Cukic, Tim Menzies
Lane Department of Computer Science and Electrical Engineering
West Virginia University, Morgantown, WV 26506, USA
yue, cukic@csee.wvu.edu; tim@menzies.us

Abstract

Prediction of fault prone software components is one of the most researched problems in software engineering. Many statistical techniques have been proposed but there is no consensus on the methodology to select the "best model" for the specific project. In this paper, we introduce and discuss the merits of cost curve analysis of fault prediction models. Cost curves allow software quality engineers to introduce project-specific cost of module misclassification into model evaluation. Classifying a software module as fault-prone implies the application of some verification activities, thus adding to the development cost. Misclassifying a module as fault free carries the risk of system failure, also associated with cost implications. Through the analysis of sixteen projects from public repositories, we observe that software quality does not necessarily benefit from the prediction of fault prone components. The inclusion of misclassification cost in model evaluation may indicate that even the "best" models achieve performance no better than trivial classification. Our results support a recommendation to adopt cost curves as one of the standard methods for software quality model performance evaluation.

1 Introduction

Software quality models can provide guidance for software Verification and Validation (V&V) activities by predicting which modules are likely to hide faults. Many statistical models have been proposed to predict fault-proneness of program modules and various performance assessment techniques have been used to evaluate their effectiveness. Although the cost of V&V activities implied by the model are a primary concern to software project managers, model performance assessment techniques rarely address the implications of misclassification.

Fault prediction models offer a chance to identify faulty modules earlier in the development life cycle. Studies show that design and code metrics, possibly fused with the requirements metrics, offer sufficient information for effective modeling [9]. Boehm et. al [3] observed that fault removal is $50 - 200$ times less costly when performed in the design phase rather than after the deployment. NASA research [6] shows that a fault introduced in the requirements, which leaks into the design, code, test, integration, and operational phases, ensues the correction cost factors of 5, 10, 50, 130, and 368, respectively. Although these reports do not represent every software project, they indicate the benefits of fault prediction models. On the other hand, classifying a software module as fault-prone implies the application of some verification activities, typically design or code inspections, thus adding to the development cost. Using Boehm's result, if a fault prediction model misclassifies a 100 fault free modules as fault prone, the added unnecessary verification expense may offset the savings of correct identification of one faulty module.

There have been attempts to include cost factors into model evaluation. $F - measure$, for example, offers a cost factor to balance project specific cost concerns [12]. Khoshgoftaar and Allen [13] proposed the use of prior probabilities of misclassification to select classifiers which offer the most appropriate performance. In [14] they compare the return-on-investment in a large legacy telecommunication system when V&V activities are applied to software modules selected by a quality model vs. at random. Cost has been considered in test case selection for regression testing too [8].

In this paper, we analyze a new model evaluation technique called cost curve [7]. Cost curve was proposed by Drummond and Holte to evaluate machine learning classifiers in data mining. Cost curve is possibly the most informative chart for model comparison, because it offers visualization of a wide range of misclassification cost associated with classification performance as well as with the project specific quality assurance and economic needs. In comparison to $F - measure$ and return-on-investment approaches, cost curve evaluation provides model comparison results which are more general, yet easier to develop and understand. Our analysis includes sixteen fault prediction data sets publicly available from Metrics Data Program (MDP) [1] and PROMISE [4] repositories.

A cost curve represents many classifiers and thresholds in a simple informative chart. Each software system is unique in a sense that it is developed following specific processes, quality assurance techniques, and it operates in environments in which the occurrence of failures implies different consequences. The assessments of failure criticality or fault resolution priority have been a part of software $V\&V$ activities for a long time. Software development teams typically have a good understanding of the range of pertinent cost implications of failures, making cost curve evaluations practical.

The rest of paper is organized as follows. Section 2 introduces the cost curve. Section 3 explains the experimental setup and introduces statistical confidence test we use to evaluate classification results. Section 4 presents the experimental results. Section 5 summarizes our findings and points out possible directions for future work.

2 Cost Curves

2.1 Traditional Performance Indices

A confusion matrix serves as the foundation when evaluating software quality models. Figure 1 depicts two confusion matrices, which demonstrate the performance of two specific fault prediction models. The confusion matrix has four fields: True positives (TP) are modules correctly classified as faulty modules. False positives (FP) refer to fault-free modules incorrectly labeled as faulty. True negatives (TN) correspond to fault-free modules correctly classified as such. Finally, false negatives (FN) refer to faulty modules incorrectly classified as fault-free modules. In practice, model evaluation is based on combinations of these four values. The Probability of Detection (PD), also called recall or specificity [15]), is defined as the probability of correct classification of a faulty module. The Probability of False Alarm (PF) is defined as the ratio of false positives to all fault free modules. $Precision$ represents the proportion of correctly predicted faulty modules amongst all modules classified as faulty. Overall $accuracy$ is the overall probability of classifying modules correctly. The following expressions simplify the understanding of model evaluation metrics which emerge directly from a confusion matrix:

$PD = \frac{TP}{TP+FN}$

$PF = \frac{FP}{FP+TN}$

$precision = \frac{TP}{FP+TP}$

$accuracy = \frac{TP+TN}{TP+TN+FP+FN}$

Although the metrics are simple, selecting the "best" model is a multidimensional problem. In case of a high PF, many fault-free modules would be "tagged" for rigorous verification and validation, thus unnecessarily increasing the cost of the process and the time needed for project's completion. There is always a tradeoff between PD and PF. The $precision$ index measures the chance of correctly predicted faulty modules among the modules classified as fault-prone. A small number of correctly predicted faulty modules or a larger number of erroneously classified fault-free modules would result in a low precision. A tradeoff also exists between PD and $precision$ [15, 19] . In project PC1, Figure 1, less than 7% of $1,109$ modules contain fault(s). We predict faulty modules in PC1 using random forests [5], a tree ensemble classification method. These trees vote to classify a data instance. Using a simple majority voting scheme, the random forest algorithm produced the overall accuracy of 93.6% and PF of 1.5%, but PD is only 27.3% (Figure 1(a)). Only 15 out of $1,032$ fault-free modules were misclassified. With 21 fault-prone modules correctly classified, the precision is calculated to be 58.3%. By assigning a higher threshold to the majority class (thus changing the model), we obtained the confusion matrix in Figure 1(b). This model demonstrates a much higher PD (74%). Now, 57 out of 77 fault-prone modules were accurately predicted. Due to the trade-off between PD and PF, as well as the tradeoff between PD and $precision$, a higher PD is followed by a higher PF (17.1%). 176 failure free modules are incorrectly classified. The precision is 24.5%, but this does not necessarily imply worse classification.

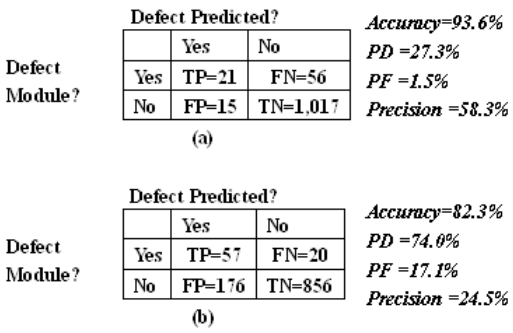

Figure 1. Confusion matrix of fault prediction on PC1. (a) Random forests with majority voting; (b) Random forests with modified voting cutoffs.

Receiver Operator Characteristic (ROC) curves visualize a classifier's performance. An ROC curve is a plot of the Probability of Detection (PD) as a function of the Probability of False alarm (PF) across all the experimental threshold settings. Many classification algorithms allow users to define and adjust the threshold parameter in order to chose an appropriate classifier. ROC curves have been frequently used for performance evaluation of fault prediction models in software engineering. Unfortunately, none of the performance indices presented in this subsection offer the opportunity to include misclassification cost implications in model evaluation.

2.2 F-measure

$F - measure$, defined by equation 1, has been used in evaluation of software quality models [12]. F-measure integrates PD and $precision$ in a single performance index. It offers more flexibility by including a weight factor β which allows us to manipulate the weight (cost) assigned to PD and $precision$. Parameter β may assume any non-negative value. If we set β to 1, equal weights are given to PD and $precision$; when $\beta > 1$, the weight of PD increases with the value of β; when $0 < \beta < 1$, the assigned weight of $precision$ is greater than that of PD. With the different values of β, F-measure can be adjusted to correspond to project's need. For instance, a project which wants to minimize the probability of failure in the field would increase the weight of PD to emphasize the importance of detecting many faulty modules even if that implies misclassifying many fault free modules. On the contrary, a project with severe budget limitations for V&V activities may want to emphasize $precision$. The problem with F-measure is the difficulty of assigning an appropriate value to β. This has proven to be the reason for its sporadic use.

$$F - measure = \frac{(\beta^2 + 1) * Precision * PD}{\beta^2 * Precision + PD} \quad (1)$$

2.3 Cost Curve Construction

Cost curve, proposed by Drummond and Holte [7], is a visual tool that describes classifier's performance based on the cost of misclassification. The x-axis represents probability cost function, denoted $PC(+)$. The y-axis represents normalized expected misclassification cost. Let us denote faulty modules with a "+" and fault-free modules as "-". $C(+|-)$ denotes the cost of incorrectly classifying a fault-free module as faulty. $C(-|+)$ represents the cost of misclassifying a faulty module as fault-free. $p(+)$ and $p(-)$ are the probabilities of a software module being faulty or fault free when the model is deployed. These two probabilities become known only after the deployment of the model, since the proportion of faulty modules in model's training and test sets are approximations. Cost curves support visualization of model's performance across all possible values of $p(+)$ and $p(-)$, thus offering performance predictions for various deployment environments. Equations 2 and 3 show how to compute the values of x-axis and y-axis:

$$x - axis = PC(+) = \frac{p(+) * C(-|+)}{p(+) * C(-|+) + p(-) * C(+|-)} \quad (2)$$

$$y - axis = (1 - PD - PF) * PC(+) + PF \quad (3)$$

Typical regions of a cost curve are shown on Figure 2. The diagonal line connecting (0,0) to (1,1) stands for a trivial classifier (TC) that classifies all the modules to fault-free. The points above this line indicate that the performance of a classifier is worse than the trivial classifier. The other diagonal line, connecting (0,1) to (1,0), stands for another trivial classifier which classifies all the modules as faulty. The points above this line also represent cases where the performance of a classifier is worse than the trivial classifier. The horizontal line connecting (0,1) to (1,1) stands for the extreme situation where the model misclassifies all the modules. The x-axis line connecting (0,0) to (0,1) represents the ideal model that correctly classifies all the modules. From this prototypical cost curve, we can see that the points in the range of (0,0.5) on y-axis are sufficient for model evaluation, because we are not interested in classifiers which perform worse than the trivial ones.

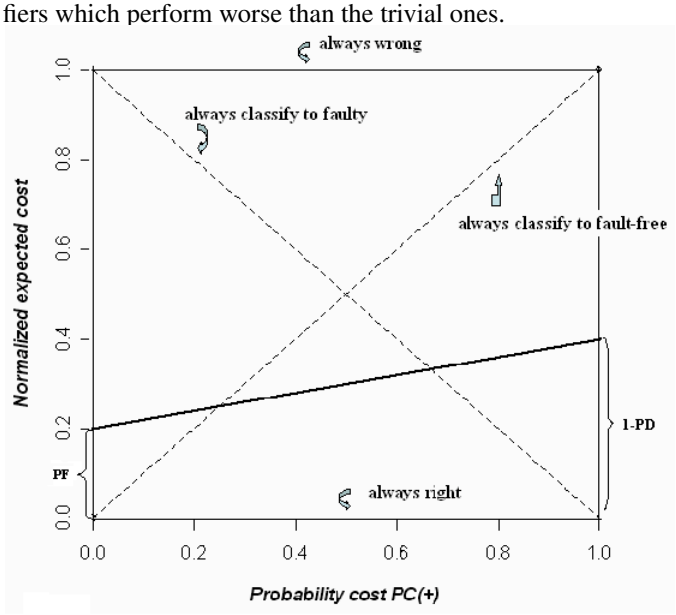

Figure 2. Typical regions in a cost curve.

An ROC curve connects a set of (PF, PD) pairs that characterize the performance of a model. Cost curves are closely related to ROC curves. To develop a cost curve, we draw straight lines connecting points (0,PF) to (1,1-PD). Each line, therefore, corresponds to point(PF,PD) in the ROC curve. After drawing all the straight lines with counterpart points in the ROC curve, the lower envelope of cost curve is formed by connecting all the intersection points from left to right. Figure 3 shows an example of a cost curve from the application of logistic classification model to project KC4. The lower envelope of the cost curve corresponds to the convex hull in the ROC curve.

2.4 The Meaning of Classification Results

Let us denote the misclassification cost ratio $\mu = C(+|-):C(-|+)$. Using μ, Equation 2 can be rewritten as:

$$x - axis = PC(+) = \frac{p(+)}{p(+) + p(-) * \mu}, \quad (4)$$

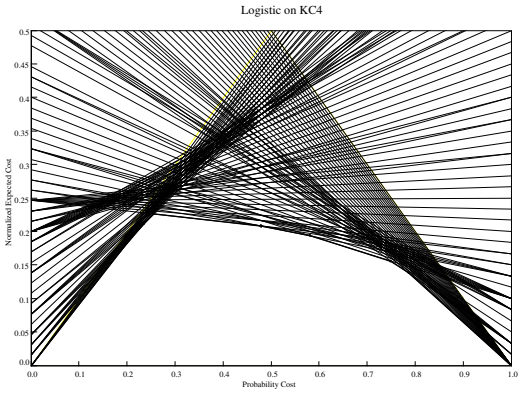

Figure 3. A cost curve of logistic classifier on KC4 project data.

Please note that $p(-) = 1 - p(+)$. If for the specific project we can estimate the adequate range or values of μ, we can calculate the corresponding PC(+) easily. Also, if we know the value of PC(+), we can derive the corresponding μ. For instance, in KC4 data set (described in Table 3), the proportion of faulty modules observed in development is very high, 48%. Therefore, $p(+) = 0.48$ and $p(-) = 0.52$. If $\mu = 1$, the cost of misclassifying a faulty module and the cost of misclassifying a fault-free module are the same, $C(+|-)$ = $C(-|+)$. From Equation 4, $PC(+) = 0.48$. When $\mu = 5$, the misclassification of a fault-free module as fault-prone is estimated to cost the project 5 times more. Equation 4 then offers $PC(+) = 0.156$. When $\mu = \frac{1}{5}$, $PC(+) = 0.82$. These points on the x-axis denote the stated misclassification costs.

Figure 4 shows the cost curves representing models generated by five classification algorithms: bagging, boosting, logistic, naiveBayes, and random forest. The values of $\mu = (100, 5, 1, \frac{1}{5}, \frac{1}{100})$ are indicated by vertical dashed lines at corresponding values of PC(+): 0.009, 0.156, 0.48, 0.82, and 0.99, respectively. If in KC4 we are interested in a particular misclassification cost ratio, for example ($\mu = 1$), we should choose a classifier that offers the minimal expected misclassification cost along the corresponding vertical line ($PC(+) = 0.48$), from Figure 4 that is boosting. When $PC(+) < 0.48$, misclassifying fault-free modules is more costly than misclassifying a faulty module ($\mu > 1$). The region $PC(+) < 0.48$ offers of models adequate for "$cost - adverse$" software projects. Intuitively, cost adverse projects have tight budgets and typically develop low criticality software. In cost-adverse environments, developers attempt to minimize the number of modules exposed to additional verification activities. Therefore, such projects prefer fault prediction models which rarely misclassify fault-free modules as fault-prone. The cost region where $PC(+) > 0.48$ offers model comparison adequate for so called "$risk - adverse$" projects. In risk averse development environments, eradicating as many faults as possible is a priority. This will imply unnecessarily analyzing some fault-free modules. Consequently, misclassifying a faulty module as fault-free is significantly more consequential ($0 < \mu < 1$). Cost curves offer a unique opportunity for cost-based software quality management. Misclassification costs can dictate the preference for selecting models and model parameters considered the most appropriate.

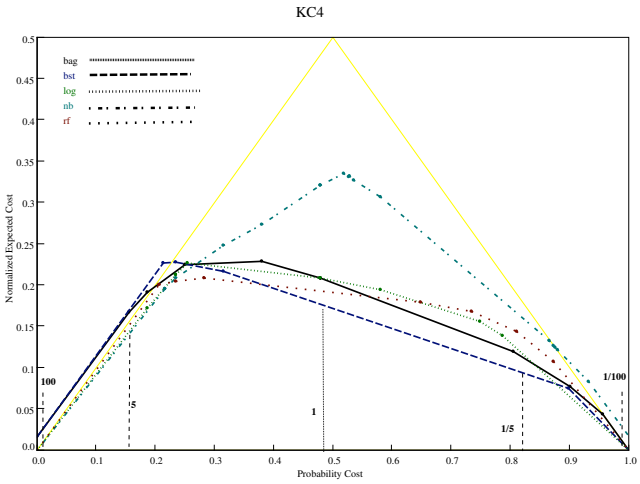

Figure 4. Cost curves of the five classifiers applied to KC4 data set.

The goal of cost curve analysis is to select the model which minimizes overall misclassification cost, that is, minimize the area under the lower envelope boundary. The smaller the area under the lower envelope boundary is, the lower the misclassification cost and, hence, the better the performance of the classifier. However, it is unlikely that a single model will be superior in all cost regions. It is equally unlikely that the entire range of misclassification costs would be of interest to a software project's fault prediction. If a project evolves to an understanding that the misclassification cost differential is not the same one used in the past, the project may decide to change the quality model it uses even though the underlying metric distributions in the data set remain the same. The model could be changed due to variations in observed probabilities $p(+)$ and $p(-)$. For instance, looking back at Figure 4, in the interval (0.22, 0.385) random forest offers the best performance; in the interval (0.385,0.88), boosting outperforms other classifiers; finally,in the range from 0.88 close to 1.0, logistic model has the minimal misclassification cost.

We want to emphasize another observation from Figure 4. Within interval $PC(+) = (0, 0.2)$, bagging and boosting models perform worse than the trivial classifier which simply classifies every module as faulty-free. Within $PC(+) = (0, 0.05)$ the other three models, logistic, NaiveBayes, and random forest, overlap with the trivial classi-

fier. In this misclassification cost range, their performance is no better than the trivial classifier's. At the other end, when $PC(+) > 0.87$, NaiveBayes performs worse than the trivial classifier which classifies each module as fault prone. After $PC(+) > 0.95$, bagging overlaps with the trivial classifier too. At both ends of x-axis where PC(+) is either close to 0 or 1, it becomes difficult to outperform trivial classifiers! The direct implication is that in extreme cost adverse or risk adverse environments, analyzing no modules (extreme cost averse) or all the modules (extreme risk averse) are the appropriate policies. Cost curves offer support for decisions regarding the most appropriate approach to the selection of software modules for verification.

Cost curves have the ability to reveal the differences among classification models that are not obvious in the corresponding ROC diagrams. Figure 5 demonstrates such an example using MC2 data set. The cost curve in Figure 5(b) provides a clear comparison between the two classifiers: Logistic classifier is better than random forest (rf) because its lower envelope is always closer to x-axis. Deriving this conclusion from the ROC curves would be difficult.

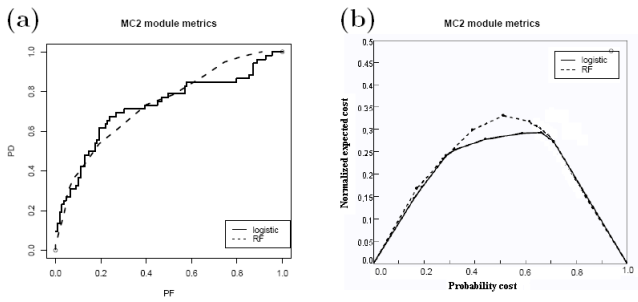

Figure 5. Comparing MC2 fault prediction models: (a) ROC curves and (b) the corresponding cost curves.

3 Experimental Set-up

In our experiments, we develop fault prediction models using five classification algorithms from Weka [18]. The five classifiers include random forest (rf), boosting (bst), logistic (log), NaiveBayes (nb), and bagging (bag). In our experience these classifiers demonstrate consistent good performance [9, 10, 15].

The characteristics of 16 fault prediction data sets are shown in Table 1. 13 NASA MDP [1] data sets offer design and code metrics for diverse projects. JM1 and KC1 offer 21 metrics used as predictor variables, MC1 and PC5 offer 39, the other data sets provide 40 software metrics. The three data sets from PROMISE [4] repository originated from a white-goods manufacturer that builds control software for washing machines, dishwashers and refrigerators. These data sets offer 29 software metrics used for fault prediction.

In practice, it is difficult to determine the exact value of misclassification cost ratio μ. But it should not be difficult to estimate project's approximate risk level in terms of a range of probability misclassification costs. For this paper, we divide projects into three groups based on their misclassification risk levels. We discussed this simplistic grouping with developers involved in projects we analyze and they found it reasonable. A project exhibits a low risk level if its misclassification ratio $\mu = C(+|-)$:$C(-|+)$ is in the range (100, 5). In simple terms, misclassifying a fault free module as fault prone is highly undesirable, 5 to 100 times more then misclassifying a faulty module as fault free. These are clearly cost averse projects, trying to optimize V&V activities and speed up product delivery. A project is assigned a $high$ risk level if the ultimate goal of analysis is to identify as many fault prone modules as possible. Such risk averse projects typically allocate a substantial portion of their budgets to verification activities. Their range of misclassification cost ratio $\mu = C(+|-)$:$C(-|+)$ is ($\frac{1}{5}$,$\frac{1}{100}$). $Medium$ risk projects are the remaining ones, for which the misclassification cost ratio μ is in the interval (5,$\frac{1}{5}$).

The risk levels for 16 data sets used in this study have been determined in discussions with software quality engineers at NASA, and the metrics collection experts at the white goods manufacturer. Seven NASA control and navigation software projects are categorized as having $high$ risk level. The three data sets from the white-goods manufacturer are classified as low risk, mostly due to strong time-to-market pressures. Two additional NASA data sets, KC4 and MC2, are also assigned a low risk level: KC4 is a ground-based subscription server and MC2 is a video guidance system. The remaining four projects are assigned $medium$ risk level. These data sets are associated with storage management for ground data, a zero gravity experiment, and a spacecraft instrument system. The misclassification cost ratio ranges (μ) are assigned accordingly. Since we know the ground truth about fault prone modules in all data sets, assigning probability parameters $p(+)$ and $p(-)$ for our experiments is simple. From Equation 2 we obtain probability costs $PC(+)$ and calculate the range of interest for each project, implied by its risk classification and probability parameters. Specific values of these parameters are shown in Table 2.

3.1 Statistical Significance in Cost Curves

Classifier's performance is derived from the confusion matrix. In cost curve, a confidence interval is generated from a series of confusion matrices. We illustrate the use of confidence bounds on cost curves following the procedure outlined in [7]. The procedure is based on the resampling of the confusion matrix at least 500 times using the bootstrap method. The 95% confidence interval for a specific $PC(+)$ value is obtained by eliminating the highest and the lowest

Table 1. Datasets used in this study

Data	mod.#	% faulty	project description	language	source	risk
JM1	10,878	19.3%	Real time predictive ground system	C	MDP	high
MC1	9466	0.64%	Combustion experiment of a space shuttle	(C)C++	MDP	high
PC2	5589	0.42%	**Dynamic simulator for attitude control systems**	C	MDP	high
PC5	17,186	3.00%	**Safety enhancement system**	C++	MDP	high
PC1	1109	6.59%	Flight software from an earth orbiting satellite	C	MDP	high
PC3	1563	10.43%	**Flight software for earth orbiting satellite**	C	MDP	high
PC4	1458	12.24%	**Flight software for earth orbiting satellite**	C	MDP	high
CM1	505	16.04%	Spacecraft instrument	C	MDP	med.
MW1	403	6.7%	**Zero gravity experiment related to combustion**	C	MDP	med.
KC1	2109	13.9%	Storage management for ground data	C++	MDP	med.
KC3	458	6.3%	Storage management for ground data	Java	MDP	med.
KC4	125	48%	Ground-based subscription server	Perl	MDP	low
MC2	161	32.30%	Video guidance system	C++	MDP	low
ar3	63	12.70%	Refrigerator	C	PROMISE	low
ar4	107	18.69%	Washing machine	C	PROMISE	low
ar5	36	22.22%	Dish washer	C	PROMISE	low

2.5% of the normalized expected cost values.

Although we deploy five classification algorithms, our goal is not to compare their performance. As we performed our experiments, it became clear to us that in many cases, some of these algorithms barely outperform trivial classifiers, especially in extreme high and low risk situations. For this reason, the goal of our analysis became understanding the basic utility of fault prediction modeling: Does the best prediction model outperform the trivial classifier in the probability cost range of interest? Therefore, for each project, we choose the algorithm that offers the lowest envelope boundary of the cost curve inside the probability cost range of interested. Then, we compare the best classifier and the trivial classifier (TC) to determine whether the difference is significant using 95% confidence interval (CI). For KC4, for example, of the five classification algorithms NaiveBayes (nb) offers the best performance. We test nb against the trivial classifier. The calculated 95% confidence interval of nb against TC cost curve is depicted in Figure 6.

The shaded area in Figure 6 represents the 95% confidence interval. There are three lines inside the shaded area. The middle line represents the difference between the means of nb and TC. The other two lines along the edges of the shaded area represent the 95% confidence interval of the difference between the means of these two classifiers. If the confidence interval contains the zero line, then there is no significant difference between the two classifiers; Otherwise, there is. The larger the distance of the confidence interval from the zero line, the more significant the difference in performance. In some ranges of the curve the performance difference is significant, but in the others it is not. Referring back to Figure 6, when PC(+) is in the ranges $(0.1, 0.21)$ and $(0.30, 0.73)$, the confidence interval indicates that nb performs better than the trivial classifier. However, the confidence interval is much closer to the zero line within $(0.1, 0.21)$ than within $(0.30, 0.73)$. The confidence band in cost curve reveals aspects which cannot be inferred from the statistical analysis of any other performance index, such as overall accuracy, error rate, or PD.

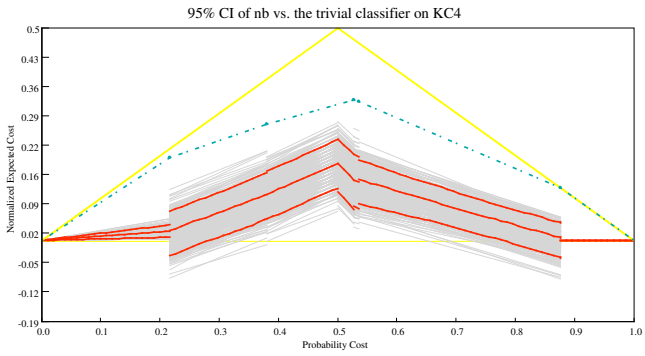

Figure 6. The 95% cost curve CI comparing nb and trivial classifiers on KC4.

4 Experiments and Results

Having determined risk levels for all the projects in the study, we want to understand the impact of cost curve modeling in evaluating fault prediction models. More precisely, we want to know how well the models perform when misclassification cost is taken into account. Can cost modeling be used to argue for or against software quality modeling in the context of a specific project?

We developed cost curves describing the performance of fault prediction models from the five classifiers on each of the 16 data sets listed in Table 1. Figure 7 shows all the 16 cost curve diagrams. To increase readability of these diagrams, we limit the displayed range of the probabil-

ity cost (x-axis) to include only the region of interest inferred from the project's risk level: low - $(100, 5)$, medium -$(5, \frac{1}{5})$, or high - $(\frac{1}{5}, \frac{1}{100})$. The lower and upper bound of each region are marked with thin solid vertical lines. Most cost curve diagrams also show the operational point where $\mu = C(+|-) = C(-|+) = 1$.

A quick look at the diagrams in Figure 7 reveals two general trends: (1) when PC(+) is close to 0 or close to 1, the performance of fault prediction models becomes very similar to trivial classifiers; (2) the most significant performance benefit of fault prediction modeling is usually achieved in the middle region of the cost curve, surrounding $PC(+) = 0.5$. This observations imply that fault prediction modeling has limited impact in cases of extremely risk averse or cost averse projects. On the extreme risk averse side, we would like the model to identify all the faulty modules. On the extreme cost averse side, we do not want to misclassify any fault free module as fault prone. These goals are, in fact, achieved by the two trivial classifiers which assign all the modules in a single class! In binary classification problems, near perfect module assignment to one class results in high misclassification rate in the other class. High misclassification cost differential between the two classes, therefore, leads to trivial solutions.

Table 2 depicts each project's risk level, the corresponding probability cost range of interest and the classification algorithm that achieves the best performance inside the probability cost range. The table also shows the probability cost range in which the best model is significantly better (using the 95% confidence interval) than trivial classification, the size of statistically significant interval with respect to the probability cost range of interest and the corresponding range of misclassification cost ratio μ in which the model outperforms trivial classification. The last column offers the project - specific misclassification cost interval in which software quality models offer benefits. If the project's actual misclassification cost ratio is not within this interval, using models to select fault prone modules would not justify the effort.

4.1 High Risk Projects

The data sets described as $high$ risk were derived from mission and safety critical development projects. Figures 7(a)-(g) depict the cost curves of high risk projects. Although we noted that highly imbalanced misclassification cost may favor trivial classification, the high risk data sets we analyzed all indicate that fault prediction models outperform trivial classifiers. However, statistical significance does not necessarily span over the entire range of the misclassification cost ratio $(\frac{1}{5}, \frac{1}{100})$. For example, for JM1 in Figure 7(a), the model generated using random forest algorithm performs significantly better than the trivial classifier in the probability cost region $(0.545, 0.72)$, which corresponds to the misclassification cost ratio range $\mu = (\frac{1}{5}, \frac{1}{10})$[1]. Similar observations can be made for projects PC1 and PC3: when $PC(+) < 0.80 (\mu \approx \frac{1}{50})$ or $PC(+) < 0.82$ $(\mu \approx \frac{1}{40})$, respectively, best models perform better than the trivial classifier. The same is not true for higher misclassification cost ratios, $\mu > \frac{1}{50}$ or $\mu > \frac{1}{40}$, respectively. In PC2 data set, the statistically significant performance difference between the best classifier and the trivial classifier lays in the region of the highest misclassification cost ratio, $\mu = (\frac{1}{40}, \frac{1}{100})$. With the remaining three high risk projects, MC1, PC4 and PC5, the best model outperforms the trivial classifier in the entire range of misclassification cost ratio. It is also interesting to observe that in all high risk projects, models developed using random forest algorithm outperform other machine learning algorithms used in our experiments.

4.2 Medium Risk Projects

The cost curves of medium risk projects CM1, MW1, KC1, and KC3, where $\mu = (5, \frac{1}{5})$, are shown in Figures 7(h)-(k). Interestingly, fault prediction models do not outperform trivial classification throughout the range of interest. Cost curve diagrams indicate that "medium risk" resides in the low range of the probability cost, i.e., $PC(+) < 0.5$. This is not a region of the most beneficial performance impact. Bagging, boosting and random forest are the classification algorithms which seem to perform the best for medium risk projects.

4.3 Low Risk Projects

The cost curves of low risk projects MC2, ar3, ar4 and ar5 are depicted in Figures 7(l)-(o) while KC4 is shown in Figure 4. In these projects, trivial classification is as good for selecting fault prone modules as any of the models we experimented with. In other words, if the goal of software quality modeling is to avoid wasting tight V&V budgets on verification of fault-free modules, the best way to achieve this is by not classifying any module as fault prone.

4.4 Discussion

The utility of fault prediction models varies with the misclassification cost: (1) in low risk projects, building software quality models may not be justified; (2) in $medium$ risk projects the utility of a model is typically limited to a subset of misclassification cost ratio range; (3) the most benefit is drawn by $high$ risk projects.

Explaining this observation is not difficult. In most projects the proportion of fault prone modules is small, causing a severe class size imbalance between faulty and fault-free modules. For example, in MC1, only 0.64% of the modules are fault prone and 99.36% are fault free. In

[1] $PC(+) = 0.72$ corresponds to $\mu = \frac{93}{1000}$, approximately $\frac{1}{10}$.

Table 2. Probability cost ranges.

Data set	Risk	Probability cost range of interest	Best classifier	Significant performance range and $\% \frac{significantRange}{interestedRange}$		Significant μ range
JM1	high	(0.545, 0.960)	rf	(0.545, 0.72)	42.17%	$(\frac{1}{5}, \frac{1}{10})$
MC1	high	(0.031, 0.391)	rf	entire	100%	$(\frac{1}{5}, \frac{1}{100})$
PC2	high	(0.021, 0.297)	rf	(0.15, 0.297)	53.26%	$(\frac{1}{40}, \frac{1}{100})$
PC5	high	(0.134, 0.756)	rf	entire	100%	$(\frac{1}{5}, \frac{1}{100})$
PC1	high	(0.261, 0.876)	rf	(0.261, 0.80)	87.64%	$(\frac{1}{5}, \frac{1}{50})$
PC3	high	(0.368, 0.921)	rf,bst	(0.368, 0.82)	81.74%	$(\frac{1}{5}, \frac{1}{40})$
PC4	high	(0.411, 0.933)	rf	entire	100%	$(\frac{1}{5}, \frac{1}{100})$
CM1	med.	(0.037, 0.489)	rf	(0.11, 0.489)	83.85%	$(1.5, \frac{1}{5})$
MW1	med.	(0.014, 0.264)	bag,bst	(0.13, 0.264)	53.6%	$(\frac{1}{2}, \frac{1}{5})$
KC1	med.	(0.031, 0.447)	rf	(0.14, 0.447)	73.8%	$(1, \frac{1}{5})$
KC3	med.	(0.013, 0.252)	bag	(0.18, 0.252)	30.13%	$(\frac{1}{3.3}, \frac{1}{5})$
KC4	low	(0.009, 0.156)	nb	(0.10, 0.156)	38.10%	$(8, 5)$
MC2	low	(0.005, 0.087)	nb	no	0	
ar3	low	(0.001, 0.028)	bst	no	0	
ar4	low	(0.002, 0.044)	bst	no	0	
ar5	low	(0.003, 0.054)	bst	no	0	

our study, KC4 has a large percentage of faulty modules, 48%, but it is also the smallest project (only 125 modules) and the only one that uses a scripting language (Perl). In MC1, identifying 61 faulty amongst $9,644$ modules is a difficult task for any classification algorithm. What makes the effort worthy is the high cost premium for fault detection. If the premium is not there, given what we know about this data set, no fault prediction model justifies the investment.

Looking at the shape of cost curves, the most significant benefits for software assurance are achieved in the mid range of the probability cost axis, surrounding $PC(+) = 0.5$. Due to class size imbalance, the classification performance for neutral misclassification cost ratio is at the value of $PC(+)$ equal to the (low) percentage of faulty modules in the data set. For this simple reason, most economical opportunity to apply fault prediction modeling lays to the right of the neutral misclassification cost point. This is the region of interest for $high$ and some of the $medium$ risk projects.

When we planned this study, our original goal was to compare the performance of different classification algorithms for projects in different risk regions. We learned that random forest consistently outperforms other classifiers in the high risk region. But as the study evolved, it became obvious that we need to understand whether fault prediction models are any better than trivial classifiers within specific misclassification cost ranges. For this purpose, we applied a rigorous statistical significance analysis procedure. This analysis indicates that although NaiveBayes, boosting and bagging outperform other algorithms for low and medium risk projects, their classification results are in many cases not significantly better than trivial classification. We can relate this observation to a steady trend in fault prediction modeling literature recommending model evaluation with lift charts, sometimes called Alberg diagrams [16, 17]. Lift is a measurement of the effectiveness of a classifier to detect faulty modules. It calculates the ratio of correctly identified faulty modules with and without the predictive model. Lift chart is especially useful when the project has resources to apply verification activities to a limited number of modules. Lift chart evaluation can be used in concert with cost curves.

We want to add two caveats. Currently, we assume that the misclassification cost of each software module is the same. For NASA MDP projects this is not an unreasonable assumption. The process at NASA calls for the application of Independent Verification and Validation only to software configuration items which justify the expense either due to risk and criticality or due to some project specific requirements. If misclassification cost of modules varies significantly within the project, a uniform misclassification cost ratio in cost curve analysis may not be sufficiently flexible. Cost effectiveness measure described by Arisholm et. al. [2], for example, can supplement cost curve model evaluation because it can account for the nonuniform cost of module-level V&V. The last caveat is the fact that in this study we knew the exact proportion of faulty modules during fault prediction. In actual deployments this is not known, causing some uncertainty in the calculation of probability cost function. From this perspective, having wider evaluation ranges of interest along the probability cost axis of cost curve diagrams is desirable.

5 Conclusion

Do software quality models offer tangible benefits in software development? The answer in the literature appears to be a resounding yes. The traditional V&V return-

on-investment wisdom states that catching faults before deployment reduces the overall development cost. This is undoubtedly true. But, software applications are different, some provide user's with convenience at low cost, others address mission critical system needs. Evaluation of fault prediction models must differentiate the business context in which projects emerge. The problem with traditional performance evaluation measures of binary classification problems, such as PD, PF, $precision$, $accuracy$, etc., is that they are highly interrelated. Choosing the "best" model becomes a difficult multidimensional problem.

Cost curve is a complementary model evaluation technique. If offers a different viewpoint by integrating the cost of module misclassification into performance evaluation. We are aware that determining misclassification cost ratio is not an exact science. This problem arguably deserves additional consideration by the research community. The software quality professionals we discussed cost curve methodology with did not appear to have difficulty assessing approximate cost of misclassifying fault free modules as fault prone or the other way around. Importantly, misclassification cost assessment is project-specific. Factors that influence the cost are the size of the V&V budgets (i.e., how many modules can be analyzed for faults), schedule pressures, software configuration item's criticality, failure risks and severity, and various nonfunctional requirements including reliability, availability, etc. However, it remains to be investigated whether the specific misclassification cost ratio ranges we used to characterize high, medium and low risk projects are the most appropriate generalizations.

Based on our initial experience with cost curves, we strongly recommend their inclusion in the evaluation of software quality models. Results of our experiments indicate that high risk projects draw the most significant benefit from fault prediction models. We are further intrigued by the observation that some modeling algorithms are consistently better for high risk projects, while others have advantages for medium or low risk projects. This result indicates that future research may need to consider fault prediction modeling methodologies tailored for the type of software under development.

References

[1] Metric data program. NASA Independent Verification and Validation facility, Available from `http://MDP.ivv.nasa.gov`.

[2] E. Arisholm, L. C. Briand, M. Fuglerud. Data Mining Techniques for Building Fault-proneness Models in Telecom Java Software. 18^{th} *Int'l IEEE Symp. Software Reliability Engineering*, Trollhattan, Sweeden, pp. 215–224, Nov. 2007.

[3] B. Boehm and P. Papaccio. Understanding and controlling software costs. *IEEE Transactions on Software Engineering*, 14(10):1462–1477, 1988.

[4] G. Boetticher, T. Menzies, and T. Ostrand. Promise repository of empirical software engineering data, 2007. available from `http://promisedata.org/`.

[5] L. Breiman. Random forests. *Machine Learning*, 45:5–32, 2001.

[6] J. Dabney. Return on investment for IV&V. NASA funded study. Available from `http://sarpresults.ivv.nasa.gov/ViewResearch/24.jsp`.

[7] C. Drummond and R. C. Holte. Cost curves: An improved method for visualizing classifier performance. *Machine Learning*, 65(1):95–130, 2006.

[8] S. Elbaum, A. Malishevsky, and G. Rothermel. Incorporating varying test costs and fault severities into test case prioritization. *International Conference on Software Engineering*, 329–338, May 2001.

[9] Y. Jiang, B. Cukic, and T. Menzies. Fault prediction using early lifecycle data. *The* 18^{th} *IEEE Int'l Symp Software Reliability Engineering*, Trolhattan, Sweden, pp. 237–246, Nov. 2007.

[10] Y. Jiang, B. Cukic, and T. Menzies. Can data transformation help in the detection of fault-prone modules. In *International Workshop on Defects in Large Software Systems (DEFECTS'08)*, July 2008.

[11] Y. Jiang, B. Cukic, and T. Menzies. Comparing design and code metrics for software quality prediction. In *Proc.* 3^{rd} *Int'l Workshop on Predictor Models in Software Engineering*, May 2008.

[12] Y. Jiang, Y. Ma, and B. Cukic. Techniques for evaluating fault prediction models. *Emprical Softw. Engg*, in print, 2008.

[13] T. M. Khoshgoftaar and E. B. Allen. Classification of fault-prone software modules: Prior probabilities,costs, and model evaluation. *Empirical Softw. Engg.*, 3(3):275–298, 1998.

[14] T. M. Khoshgoftaar, E. B. Allen, W. D. Jones, and J. P. Hudepohl. Cost-benefit analysis of software quality models. *Software Quality Control*, 9(1):9–30, 2001.

[15] T. Menzies, J. Greenwald, and A. Frank. Data mining static code attributes to learn defect predictors. *IEEE Transactions on Software Engineering*, 33(1):2–13, January 2007.

[16] T. Ostrand, E. Weyuker, R. Bell. Predicting the location and number of faults in large software systems. *IEEE Transactions on Software Engineering*, 31(4): 340-355, 2005.

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

[18] I. H. Witten, E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 2005.

[19] H. Zhang, X. Zhang. Comments on "data mining static code attributes to learn defect predictors". *IEEE Transactions on Software Engineering*, 33(9):635–637, Sept. 2007.

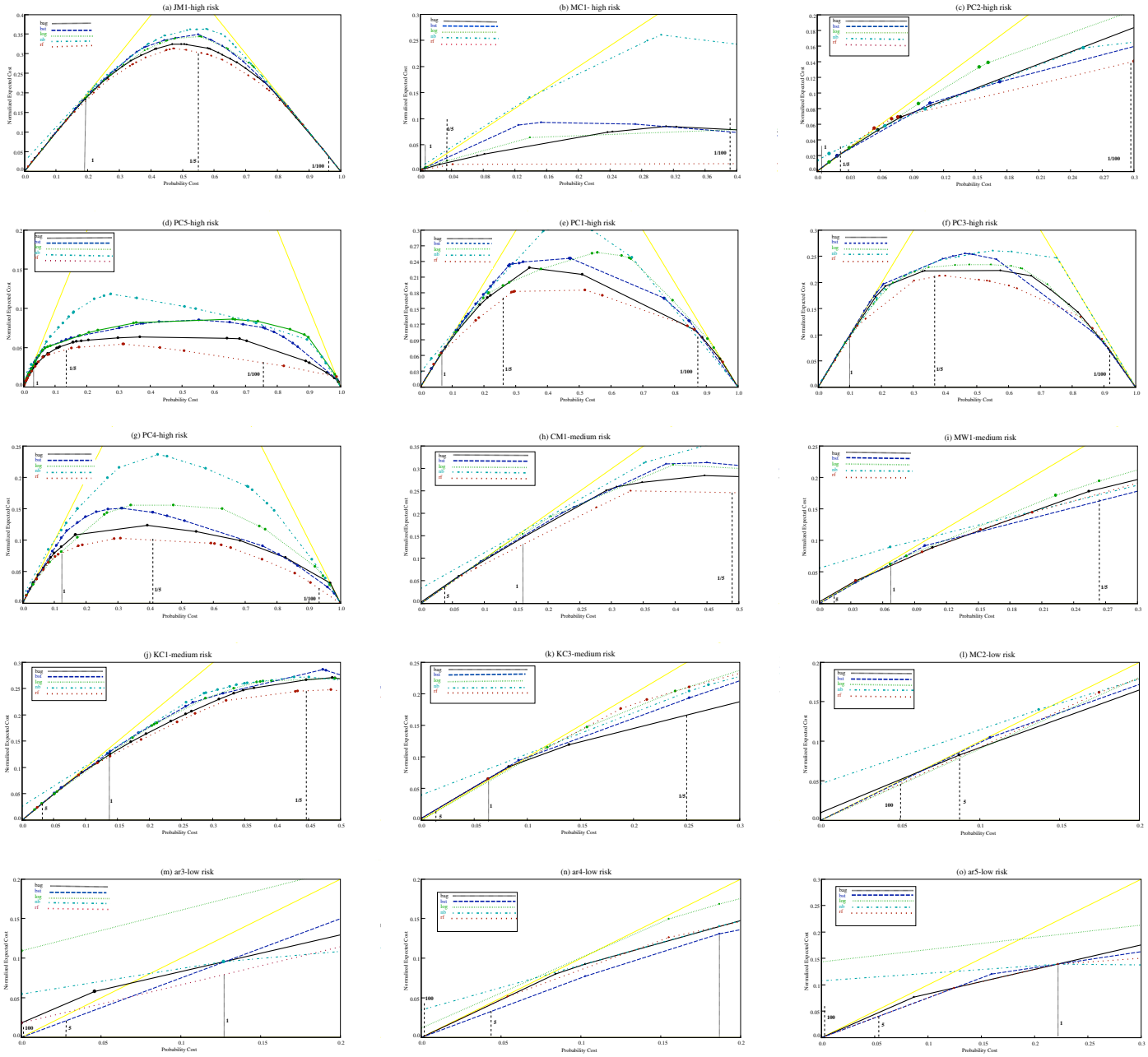

Figure 7. Cost curves.