Khoshgoftaar TM, Rebours P. Improving software quality prediction by noise filtering techniques. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 22(3): 387~396 May 2007

Improving Software Quality Prediction by Noise Filtering Techniques

Taghi M. Khoshgoftaar and Pierre Rebours

Empirical Software Engineering Laboratory, Department of Computer Science and Engineering, Florida Atlantic University Boca Raton, FL, U.S.A.

E-mail: taghi@cse.fau.edu; prebours@fau.edu

Received March 15, 2006; revised March 6, 2007.

Abstract Accuracy of machine learners is affected by quality of the data the learners are induced on. In this paper, quality of the training dataset is improved by removing instances detected as noisy by the Partitioning Filter. The fit dataset is first split into subsets, and different base learners are induced on each of these splits. The predictions are combined in such a way that an instance is identified as noisy if it is misclassified by a certain number of base learners. Two versions of the Partitioning Filter are used: Multiple-Partitioning Filter and Iterative-Partitioning Filter. The number of instances removed by the filters is tuned by the voting scheme of the filter and the number of iterations. The primary aim of this study is to compare the predictive performances of the final models built on the filtered and the un-filtered training datasets. A case study of software measurement data of a high assurance software project is performed. It is shown that predictive performances of models built on the filtered fit datasets and evaluated on a noisy test dataset are generally better than those built on the noisy (un-filtered) fit dataset. However, predictive performance based on certain aggressive filters is affected by presence of noise in the evaluation dataset.

Keywords noise filtering, data quality, software quality classification, expected cost of misclassification, voting expert

1 Introduction

The need for reliable and high quality products leads software managers to use software quality classification models which allow them to direct improvement efforts to software modules with higher risk. Such models are designed to identify, prior to deployment, software modules that are likely to be fault-prone (*fp*) during operations[1]. Hence, a cost-effective utilization of resources can be implemented for software testing, inspection, and quality enhancement of these modules. The models are often based on *inductive learning algorithms*[2] which generalize the concepts learnt from a set of training instances (i.e., software modules in our case) and apply these concepts to the new instances. Different data mining algorithms have been used to build software quality models, such as case-based reasoning[3], and logistic regression[4].

The predictive accuracy of such a machine learning algorithm mostly depends on the inductive bias of the learner and quality of the data on which the learner is induced[5]. Noise can affect dependent variables (i.e., class noise: fault-prone or not fault-prone in our case) as well as independent variables (i.e., attribute noise). Poor-quality data leads to a training dataset with inconsistencies in terms of expected syntax, semantics, or values. Hence, inducing learners on noisy training data can produce inaccurate predictions because decisions are based on incorrect information[6,7]. Therefore, using appropriate noise handling procedures should be used before undertaking any data mining task.

The problem of effectively dealing with noise can be approached primarily in three different ways. The practitioner can use noise-tolerant (i.e., robust) algorithms, filter out noisy instances from the training dataset, or try to correct such noisy instances (i.e., polishing). A *robust learning algorithm* is employed in such a way that the classifier built will not be overfitted to the likely noisy instances[2]. *Filtering* is a rather direct attempt to improve the quality of input data by removing potentially noisy instances[8]. Hence, noise will not influence the final hypotheses constructed[9]. *Polishing* corrects one or more features of the instances suspected of being noisy[10]. Typically, *final learners*[11] are built after processing the training dataset with a given noise handling scheme. This paper focuses on the filtering approach to handle noisy datasets.

Filtering noisy instances has been an important issue in the machine learning community. Gamberger *et al.*[9] propose a simple noise filtering scheme, called the Classification Filter. For a given dataset, predictions of the learner are obtained preferably by cross-validation[2]. The learner is called the *base learner* of the filtering scheme. The misclassified instances are then identified as noisy and removed from the training dataset. However, using only one base learner may be risky, because the given classifier may not have the appropriate bias to learn the concepts of the given domain problem[11].

Brodley and Friedl[11] investigate the efficiency of the Ensemble Filter, a filtering scheme which unifies the predictions of m different base learners. By using three or five base classifiers, they find that filtering an artificially noisy training dataset allows maintaining a good performance accuracy of the final learners on the test dataset. In our recent work[1], we investigate a very large Ensemble Filter, i.e., consisting of twenty-five different classifiers. Using a large number of base learners allows the analyst the freedom of choosing from several filtering levels (i.e., level of conservativeness).

Regular Paper

Zhu *et al.*[5] argue that the Classification Filter and the Ensemble Filter are sometimes inadequate with large and/or distributed datasets. In order to overcome the data size limitation of the Classification and Ensemble Filters, Zhu *et al.* introduce a scheme that partitions the training dataset into equal subsets before filtering. This partitioning approach also defines the concept of *good rules* to distinguish exceptions from noise. Our recent study[12] investigates two modifications of the partitioning approach. The training dataset is first split into n subsets, and m base learners are induced on each of these subsets. Subsequently, an instance is identified as noisy if it is misclassified by a certain number of base learners. Two specialized filters are implemented based on the partitioning scheme. The Multiple-Partitioning Filter combines several classifiers induced on each split. The Iterative-Partitioning Filter uses only one base learner, but performs multiple iterations. The amount of noise removed is varied by tuning either the filtering level or the number of iterations[5,12].

All the above filtering techniques are built on the same basic assumption: if a large number of models misclassify a software module, then the module is likely to contain noise. More specifically, the software measurements and quality data of such a program module do not adhere to the underlying characteristics of the software system. Since the data collection process is usually similar for both the training (fit) and evaluation (test) datasets, it is likely that the test dataset has noise. A key issue is to determine whether filtering the fit dataset improves prediction on the evaluation dataset. This study analyzes the predictive performances of different learners induced on both filtered and un-filtered training datasets.

The training dataset is processed by the Multiple-Partitioning Filter and the Iterative-Partitioning Filter at different levels of conservativeness and/or at different numbers of iterations. Since most noise elimination methods involve supervised learning schemes, we can remove noisy data from the training dataset in which the dependent variables (i.e., *fp* or *nfp*) are available. However, we cannot control the quality of the test dataset since software quality data is not available. In this study, we demonstrate that the predictive performances (i.e., performances on the test dataset) of the final models built on the fit datasets processed by our filters, under certain conditions, are better than those built on the noisy (i.e., un-filtered) fit dataset.

More specifically, we demonstrate the effectiveness of some of the filtering methods. A conservative filter (i.e., that removes fewer instances) tends to improve the prediction accuracy of the learners. In contrast, an aggressive filter (i.e., that removes more instances) tends not to improve prediction accuracy. The improvement in prediction may not always be significant. Improving the data collection process of the given project may reduce the level of noise in the measurement data.

The rest of the paper is organized as follows. In Section 2, we summarize the implementation of the two filters, namely the Multiple-Partitioning Filter and the Iterative-Partitioning Filter. We also describe the modeling methodology involved in our empirical investigation. In Section 3, empirical studies on a high assurance software project analyze the performances of final learners induced on the filtered and un-filtered training datasets and applied on a potentially noisy test dataset. Finally, Section 4 concludes our work.

2 Methodology

2.1 Partitioning Filters

The *Multiple-Partitioning Filter* combines m different base classifiers induced on each of the n splits of the training dataset. If an instance is part of the training subset of a base learner, the classification predicted by the learner is based on cross-validation[2]. The Multiple-Partitioning Filter without the cross-validation constraint identifies an instance as noisy if it is misclassified by at least λ models, where λ is the filtering level[1,12]. The Multiple-Partitioning Filter with the cross-validation constraint removes an instance if it is misclassified by at least λ models and mislabeled by all m models induced on the training subset that contains that instance. Generally, a classifier often has a higher prediction accuracy for instances in the training set[5]. λ can range from $floor(n \times m/2)+1$ (i.e., majority scheme) to $m \times n$ (i.e., consensus scheme). The higher the filtering level, the more conservative the filter.

The *Iterative-Partitioning Filter* is our second specialization of the Partitioning Filter. The training dataset is first partitioned into n subsets, and a model is built on each of these subsets. In this study, two voting schemes are used: consensus and majority schemes. When the Iterative-Partitioning Filter uses the majority scheme, an instance is labeled as noise if it is misclassified by more than 50% of the models. The Iterative-Partitioning Filter with the consensus scheme removes an instance if the prediction of all n models are different from the actual class label of the instance. Additionally, in order to identify an instance as noisy, the instance should be misclassified at least by the model which was induced on the subset containing that instance. Noisy instances are eliminated in multiple iterations until the stopping criterion (discussed later) is reached[5]. By varying the required number of filtering iterations, the level of conservativeness of the filter can be varied.

2.2 Model-Selection Strategy

Our empirical study is a two-group software quality classification problem. Software modules are typically labeled as either fault-prone (*fp*) or not fault-prone (*nfp*). Hence, two types of misclassification errors can occur: *Type I error* (or false positive) and *Type II error* (or false negative). The Type I and Type II errors are

generally inversely proportional. Moreover, the cost of misclassifying an *fp* module is invariably greater than the cost of misclassifying an *nfp* module. Hence, software quality engineers often recommend selecting a classification model that has a preferred balance between the two error rates[3].

The system in this paper is the same high assurance system that was studied in our recent works[1,12], therefore, we use the same model selection strategy whenever a final classification model needs to be selected. More specifically, we select a preferred balance of equality between the Type I and Type II errors, with Type II being as low as possible. Having both Type I and Type II low ensures detection of significantly large number of *fp* modules, and at the same time, keeps the number of *nfp* modules predicted to be *fp* (i.e., ineffective testing and inspection) low. The above mentioned model-selection strategy is followed when building the base learners of the filters.

2.3 Performance Evaluation

Comparing the performance of different classification methods based on the two misclassification rates (Type I and Type II) can be a difficult task, especially when the performance of the base classifiers is being evaluated across a range of datasets (with different levels of noise, in our case). In practice, since there is likely to be a vast disparity between the prior probabilities of the two classes and among the costs of the two error types, the Expected Cost of Misclassification (*ECM*) is more appropriate as a practical measure for comparison[13]:

$$ECM = C_{\mathrm{I}} Pr\,(fp \mid nfp)\pi_{nfp} + C_{\mathrm{II}} Pr(nfp \mid fp)\pi_{fp} \quad (1)$$

where C_{I} and C_{II} are costs of Type I and Type II misclassification errors, respectively; π_{fp} and π_{nfp} are prior probabilities of *fp* modules and *nfp* modules; $Pr(fp \mid nfp)$ is the probability that an *nfp* module would be misclassified as *fp*; and $Pr(nfp \mid fp)$ is the probability that an *fp* module would be misclassified as *nfp*.

It is difficult to quantify the actual costs of misclassification at the time of modeling and analysis. Hence, the Normalized Expected Cost of Misclassification (*NECM*) is defined:

$$NECM = \frac{ECM}{C_{\mathrm{I}}} = Pr(fp \mid nfp)\pi_{nfp} + \frac{C_{\mathrm{II}}}{C_{\mathrm{I}}} Pr(nfp \mid fp)\pi_{fp}. \quad (2)$$

NECM facilitates the use of cost ratio $C_{\mathrm{II}}/C_{\mathrm{I}}$, which can be more readily estimated using software engineering heuristics for a given application[13].

2.4 Two-Factor Full Factorial Design Without Replications

A two-factor full factorial design without replications[14] is used to analyze the performance of different learning algorithms induced on the filtered and un-filtered training datasets. The filtered datasets are obtained by applying the Multiple-Partitioning Filter at different filtering levels, or by applying the Iterative-Partitioning Filter at different numbers of iterations. One of the goals of such an analysis is to observe if the filtering levels (λ) or the different numbers of iterations are significantly different from their respective counterparts.

A full factorial design with two factors, A and B respectively having a and b levels, requires ab experiments[14]. In our study, factor A is the filtering level or the number of iterations, while factor B represents the different final learners. The *NECM* value, at a given cost ratio ($C_{\mathrm{II}}/C_{\mathrm{I}}$), is used as the response variable. The design model is defined by the following equation[14]:

$$y_{ij} = \mu + \alpha_j + \beta_i + e_{ij} \quad (3)$$

where y_{ij} is the observation in the experiment with factor A at level j and factor B at level i. μ is the mean response, α_j is the effect of factor A at level j, β_i is the effect of factor B at level i, and e_{ij} is the error value. An ANOVA model is computed for each of the four filtering techniques, i.e., the Multiple-Partitioning Filter with and without the cross-validation constraint and the Iterative-Partitioning Filter with the consensus and majority voting schemes.

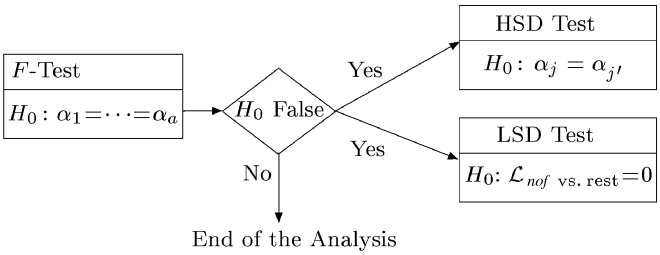

Fig.1. Two-factor full factorial analysis.

Subsequent to an analysis of variance, if it is determined that a significant treatment effect is present among the a levels of factor A, there are $\binom{a}{2}$ possible combinations of pairwise comparisons that can be analyzed[15]. Tukey's honest significant difference (HSD) method is used to reject the equality of a pair of means α_j and $\alpha_{j'}$, where $j \neq j'$[16]. We also want to know if, for a given filter, there is a significant difference between the predictive performances obtained by inducing learners on the noise-free training datasets (i.e., processed at different filtering levels or different numbers of iterations) and the un-filtered fit dataset. This contrast is denoted as $\mathcal{L}_{nof\,\mathrm{vs.\,rest}}$, where *nof* is the abbreviation for *no filter* (see Subsection 3.6). Since the contrast to be studied is selected in advance of any analysis of data, the Fisher's least significance difference method (LSD) is used[16]. Fig.1 summarizes how the statistical tests are organized.

3 Empirical Evaluation

3.1 System Description

The software metrics and quality data used in our study are from a NASA software project written in C, referred to as JM1. The data is available through the Metrics Data Program (MDP) at NASA, and includes software measurement data and associated error (fault) data collected at the function/subroutine/method level. The dataset consists of 10 883 software modules, of which 2105 modules have errors (ranging from 1 to 26) while the remaining 8778 modules are error-free, i.e., have no software faults. In this case study, a module is considered not fault-prone (*nfp*) if it has no fault, and fault-prone (*fp*) otherwise. Note that we interchangeably use the terms errors, defects, and faults in this study[17].

Each module in the JM1 project is characterized by 21 software measurements[17]: 3 McCabe metrics (`Cyclomatic_Complexity`, `Essential_Complexity`, and `Design_Complexity`); 8 derived Halstead metrics (`Halstead_Length`, `Halstead_Volume`, `Halstead_Level`, `Halstead_Difficulty`, `Halstead_Content`, `Halstead_Effort`, `Halstead_Error_Est`, and `Halstead_Prog_Time`); 5 metrics of Line Count (`Loc_Code_And_Comment`, `Loc_Total`, `Loc_Comment`, `Loc_Blank`, `Loc_Executable`); 4 basic Halstead metrics (`Unique_Operators`, `Unique_Operands`, `Total_Operators`, `Total_Operands`); and one metric for Branch Count. The types and numbers of software metrics made available are determined by the NASA Metrics Data Program. Other types of software metrics, including software process measurements are not available for analysis. The quality of the modules is described by their *Error Rate*, i.e., number of defects in the module, and *Defect*, whether or not the module has any defects. The latter is used as the class label. For additional details regarding software measurements and software quality metrics, the reader is referred to [17].

Upon removing inconsistent modules (those with identical software measurements but with different class labels) and those with missing values, the dataset is reduced from 10 883 to 8850 modules. We denote this reduced dataset as *JM1-8850*, which now had 1687 modules (19%) with one or more defects and 7163 modules (81%) with no defects. We only used the 13 base metrics in our analysis. The 8 derived Halstead metrics are not used during modeling — the metrics are derived from the 4 basic Halstead metrics. Classifiers are built using the 13 software metrics as independent variables and the module-class as the dependent variable, i.e., *fp* or *nfp*.

It is important to note that the software measurements used for the software system are primarily governed by their availability, internal workings of the project, and data collection tools. The use of the specific software metrics in the case study does not advocate their effectiveness — a different software project may collect and consider a different set of software measurements for analysis[3,17].

3.2 Empirical Design

Fig.2 illustrates the overall design of our experiments. The following steps are executed for each filtering technique at different filtering levels (for the Multiple-Partitioning Filter) or at different filtering iterations (for the Iterative-Partitioning Filter):

1) The domain dataset (E = JM1-8850 is randomly split into two equal parts of 4425 modules each. One part will uniquely act as the fit dataset (E_{fit}), i.e., JM1-4425-Fit. The other one will be used as the test dataset (E_{test}), i.e., JM1-4425-Test.

2) Instances in E_{fit} labeled as *noisy* by the filter are removed.

3) The final classifiers are built on the filtered fit dataset (E'_{fit}) by following our model selection strategy discussed earlier. Using more than one classifier is more insightful since the inductive bias introduces some variability into the final performances.

4) The final models are then applied to the test dataset (E_{test}). Due to the unavailability of the multiple releases of the software system studied in this paper, E_{test} is extracted from the initial release in which noise may exist. Therefore, predictive performance is assessed on a dataset that is potentially very noisy, and on which no noise handling technique has been applied. This reflects a real-world scenario where the class labels of instances in the test dataset are unknown. Ideally, in the case of software quality modeling, the test dataset (E_{test}) is made up of program modules being recorded in the currently under-development release of the software product.

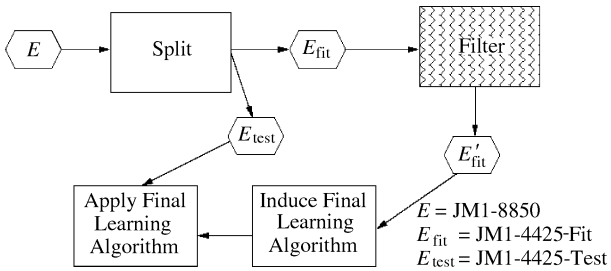

Fig.2. General procedure for building final learners.

The performances of the final learners are evaluated by using *NECM* with cost ratio ($C_{\text{II}}/C_{\text{I}}$) values of 10, 20, 30, and 50. This range of cost ratio values is considered practical for a high assurance software system, such as JM1. Similar to our recent works[1,12], this study examines the effects of noise handling techniques on a real-world dataset, in which no noise has been artificially injected. However, future work will investigate the two filtering techniques with simulated noise.

3.3 Creation of the Filters

3.3.1 Multiple-Partitioning Filter

The dataset is initially split randomly into five equal subsets ($n = 5$). Having five splits is considered appropriate, because the base learners will have enough data to learn the underlying concepts of the JM1 software measurement data. Five base algorithms ($m = 5$) are selected: J48[18], OneR[19], LWLStump[20], JRip[21], and IBk[22]. By selecting different data mining families (i.e., rule-based, tree-based, or instance-based classifiers), the bias of the learners complement one another.

The filtering level (λ) could vary from 25 ($m \times n = 5 \times 5$) to 13 — similar to an Ensemble Filter with 25 distinct classifiers[1]. We focus our investigation on four filtering levels: 23 (an instance is identified as noisy if it is misclassified by over 92% of the classifiers); 20 (over 80% of the classifiers); 17 (over 68% of the classifiers); and 13 (majority scheme). Consensus filtering (i.e., $\lambda = 25$) is not performed for the Multiple-Partitioning Filter because it is too stringent in identifying potential noise.

Two types of filters are defined: with and without the cross-validation constraint. In our study, a ten-fold cross-validation approach is used. *mpfcv*-λ refers to the Multiple-Partitioning Filter with the cross-validation constraint at filtering level λ. On the other hand, *mpf*-λ symbolizes the Multiple-Partitioning Filter without the cross-validation constraint at filtering level λ.

3.3.2 Iterative-Partitioning Filter

The Iterative-Partitioning Filter uses only one base classifier ($m = 1$). The dataset is split randomly into five equal parts ($n = 5$), and one base classifier is built on each of the five splits at each iteration. In our case study, the J48 (C4.5-based decision tree) learner is selected as the base classifier since it generally produces good results on a large variety of datasets[18]. Besides, J48 is a robust algorithm, making it tolerant to noisy data[9]. The iterative process ends once the stopping criterion, as defined in [5] is reached. More specifically, for three consecutive iterations, if the number of identified noisy examples in each iteration is less than 1% of the size of the original training dataset (i.e., $4425 \times 0.01 = 45$ instances), the iterative process stops.

Two different filtering schemes are used. The Iterative-Partitioning Filter which combines the predictions of the base models using the majority scheme is abbreviated as *ipfmaj*. Along the same lines, the Iterative-Partitioning Filter with the consensus scheme is denoted as *ipfcons*. An instance is identified as noisy by *ipfmaj* if it is misclassified by three (out of five) models, whereas *ipfcons* filters instances which are misclassified by all five models. In order to reach the stopping criterion, five iterations were required with the majority scheme for this specific dataset. On the other hand, eight iterations were necessary to satisfy the criterion with the consensus scheme.

The level of conservativeness of the Iterative-Partitioning Filter depends on the number of iterations and on the filtering scheme (i.e., majority or consensus scheme). The first five iterations as well as the last one, which satisfies the stopping criterion, are selected. Hence, this paper considers the Iterative-Partitioning Filter with majority voting (*ipfmaj*) at iterations 1, 2, 3, 4, and 5 and the Iterative-Partitioning Filter with consensus voting (*ipfcons*) at iterations 1, 2, 3, 4, 5, and 8. The *ipfcons* filter only removes fifty more instances at the last iteration than it does at its 5th iteration. Selecting the 6th and 7th iterations did not have a significant impact on removing potential noise compared to the 5th and last iterations.

The *ipfcons-i* and *ipfmaj-i* notations refer respectively to the Iterative-Partitioning Filter with consensus and majority voting schemes at their i-th iterations. For example, *ipfcons*-8 symbolizes the Iterative-Partitioning Filter with consensus scheme at its 8th (and last) iteration, and *ipfmaj*-2 indicates the Iterative-Partitioning Filter with majority scheme at its 2nd iteration.

The number of instances detected as noisy by the different filtering schemes at different filtering levels or at different iterations are listed in Table 1. For each filter, the amount and the proportions of *nfp* and *fp* instances that are potentially noisy are also provided. We observe that the Iterative-Partitioning Filter with majority scheme is the most aggressive filter. Moreover, the use of the cross-validation constraint with the Multiple-Partitioning Filter limits the potential number of instances the filter can remove.

3.4 Performance on JM1-4425

In our study, we select four different final classifiers: J48[18], OneR[19], JRip[21], and IBk[22]. The selected final learners are the base learners of the Multiple-Partitioning Filter. LWLStump[20] was not selected as a final learner because using four different final classifiers was considered sufficient enough to give an overall good estimate of the performance. In addition, the runtime of LWLStump is relatively very large as compared to the others.

The quality-of-fit of the final classifiers built on the JM1-4425-Fit filtered by the Multiple-Partitioning Filter and the Iterative-Partitioning Filter at different levels of conservativeness are summarized in Table 2. The table shows the respective *NECM* values averaged among the four learners. Similarly, the quality-of-test is presented in Table 3. The critical ranges of the costs of misclassification, obtained by a t-test[14] at significance level 0.10, are also presented. The first row of the tables (i.e., *nof*) provides the performance of the final models built on the fit dataset before filtering (E_{fit}).

Table 1. Number and Proportion of Instances Removed by the Filters

Filters	*nfp* Modules		*fp* Modules		Total
	Count	Proportion (%)	Count	Proportion (%)	Count
mpf-23	495	84.76	89	15.24	584
mpf-20	779	82.87	161	17.13	940
mpf-17	914	79.20	240	20.80	1154
mpf-13	1074	77.32	315	22.68	1389
mpfcv-23	419	84.99	74	15.01	493
mpfcv-20	577	84.85	103	15.15	680
mpfcv-17	614	83.54	121	16.46	735
mpfcv-13	632	82.83	131	17.17	763
ipfcons-1	425	80.34	104	19.66	529
ipfcons-2	584	79.67	149	20.33	733
ipfcons-3	756	80.94	178	19.06	934
ipfcons-4	835	80.91	197	19.09	1032
ipfcons-5	897	81.18	208	18.82	1105
ipfcons-8	928	79.66	237	20.34	1165
ipfmaj-1	884	78.58	241	21.42	1125
ipfmaj-2	1019	76.85	307	23.15	1326
ipfmaj-3	1046	76.52	321	23.48	1367
ipfmaj-4	1052	76.23	328	23.77	1380
ipfmaj-5	1054	76.10	331	23.90	1385

Table 2. Expected Costs of Misclassification (*NECM*) on the *Fit* Dataset

Filters	$C_{\mathrm{II}}/C_{\mathrm{I}} = 10$	$C_{\mathrm{II}}/C_{\mathrm{I}} = 20$	$C_{\mathrm{II}}/C_{\mathrm{I}} = 30$	$C_{\mathrm{II}}/C_{\mathrm{I}} = 50$
nof	0.95 ± 0.05	1.62 ± 0.10	2.29 ± 0.16	3.63 ± 0.26
mpf-23	0.68 ± 0.02	1.16 ± 0.03	1.64 ± 0.05	2.61 ± 0.07
mpf-20	0.46 ± 0.02	0.78 ± 0.03	1.11 ± 0.04	1.76 ± 0.06
mpf-17	0.25 ± 0.01	0.43 ± 0.01	0.60 ± 0.02	0.95 ± 0.03
mpf-13	0.07 ± 0.01	0.11 ± 0.02	0.15 ± 0.03	0.24 ± 0.05
mpfcv-23	0.72 ± 0.02	1.23 ± 0.03	1.74 ± 0.05	2.76 ± 0.08
mpfcv-20	0.64 ± 0.02	1.09 ± 0.03	1.54 ± 0.04	2.44 ± 0.06
mpfcv-17	0.60 ± 0.02	1.02 ± 0.03	1.45 ± 0.04	2.29 ± 0.06
mpfcv-13	0.60 ± 0.02	1.02 ± 0.03	1.44 ± 0.04	2.28 ± 0.07
ipfcons-1	0.66 ± 0.03	1.12 ± 0.04	1.58 ± 0.06	2.50 ± 0.10
ipfcons-2	0.55 ± 0.01	0.94 ± 0.02	1.32 ± 0.03	2.09 ± 0.04
ipfcons-3	0.45 ± 0.02	0.76 ± 0.04	1.07 ± 0.06	1.70 ± 0.10
ipfcons-4	0.39 ± 0.03	0.66 ± 0.04	0.94 ± 0.06	1.49 ± 0.10
ipfcons-5	0.34 ± 0.04	0.57 ± 0.06	0.81 ± 0.08	1.28 ± 0.13
ipfcons-8	0.27 ± 0.04	0.45 ± 0.07	0.64 ± 0.10	1.01 ± 0.15
ipfmaj-1	0.36 ± 0.01	0.61 ± 0.02	0.86 ± 0.03	1.35 ± 0.05
ipfmaj-2	0.17 ± 0.02	0.28 ± 0.04	0.40 ± 0.05	0.63 ± 0.08
ipfmaj-3	0.13 ± 0.03	0.22 ± 0.04	0.31 ± 0.06	0.49 ± 0.09
ipfmaj-4	0.11 ± 0.02	0.18 ± 0.03	0.25 ± 0.05	0.39 ± 0.07
ipfmaj-5	0.10 ± 0.03	0.16 ± 0.05	0.22 ± 0.07	0.35 ± 0.11

Table 3. Expected Costs of Misclassification (*NECM*) on the *Test* Dataset

Filters	$C_{\mathrm{II}}/C_{\mathrm{I}} = 10$	$C_{\mathrm{II}}/C_{\mathrm{I}} = 20$	$C_{\mathrm{II}}/C_{\mathrm{I}} = 30$	$C_{\mathrm{II}}/C_{\mathrm{I}} = 50$
nof	0.96 ± 0.08	1.66 ± 0.18	2.36 ± 0.29	3.76 ± 0.50
mpf-23	0.89 ± 0.06	1.47 ± 0.11	2.06 ± 0.16	3.24 ± 0.27
mpf-20	0.93 ± 0.04	1.57 ± 0.09	2.22 ± 0.14	3.52 ± 0.23
mpf-17	0.93 ± 0.05	1.59 ± 0.11	2.25 ± 0.18	3.57 ± 0.31
mpf-13	0.95 ± 0.02	1.65 ± 0.04	2.35 ± 0.06	3.74 ± 0.10
mpfcv-23	0.91 ± 0.09	1.54 ± 0.22	2.17 ± 0.35	3.44 ± 0.60
mpfcv-20	0.91 ± 0.02	1.53 ± 0.05	2.15 ± 0.09	3.40 ± 0.15
mpfcv-17	0.92 ± 0.05	1.56 ± 0.11	2.20 ± 0.16	3.48 ± 0.27
mpfcv-13	0.91 ± 0.09	1.55 ± 0.22	2.20 ± 0.35	3.48 ± 0.62
ipfcons-1	0.90 ± 0.06	1.53 ± 0.12	2.16 ± 0.19	3.41 ± 0.32
ipfcons-2	0.93 ± 0.05	1.60 ± 0.10	2.27 ± 0.15	3.61 ± 0.25
ipfcons-3	0.90 ± 0.03	1.50 ± 0.04	2.11 ± 0.06	3.32 ± 0.10
ipfcons-4	0.91 ± 0.02	1.54 ± 0.05	2.17 ± 0.08	3.42 ± 0.13
ipfcons-5	0.88 ± 0.03	1.47 ± 0.08	2.06 ± 0.13	3.24 ± 0.24
ipfcons-8	0.94 ± 0.05	1.60 ± 0.12	2.27 ± 0.19	3.61 ± 0.32
ipfmaj-1	0.93 ± 0.04	1.59 ± 0.11	2.25 ± 0.18	3.57 ± 0.32
ipfmaj-2	0.97 ± 0.06	1.71 ± 0.13	2.45 ± 0.20	3.92 ± 0.35
ipfmaj-3	0.97 ± 0.02	1.70 ± 0.04	2.42 ± 0.06	3.88 ± 0.11
ipfmaj-4	0.98 ± 0.03	1.73 ± 0.06	2.48 ± 0.10	3.97 ± 0.17
ipfmaj-5	1.00 ± 0.02	1.76 ± 0.06	2.53 ± 0.10	4.06 ± 0.17

The quality-of-fit results indicate that the expected cost of misclassification improves (decreases) as the filtering level or the number of iterations goes from the most conservative ($\lambda = 23$ or first iteration) to the least conservative ($\lambda = 13$ or last iteration) approach. An opposite trend is observed for the test dataset, i.e., *NECM* increases as the filtering goes from most conservative to least conservative. However, the quality-of-test of the learners induced on the non-filtered fit dataset is lower than the quality-of-test of the learners induced on the filtered fit datasets.

The predictive performance generally improves when the fit dataset is filtered, except for the *ipfmaj* filter. An aggressive filter does not guaranty a better predictive accuracy compared to a relatively conservative filter of the same family. For example, the learners induced on the fit dataset filtered by *ipfcons*-1 perform better, on average, than those filtered by *ipfcons*-8. For the Multiple-Partitioning Filter without the cross-validation constraint, it is observed that the more conservative is the filter, the better is the predictive performance.

3.5 ANOVA Models

A two-factor full factorial design[14] is employed to investigate whether the four final classification techniques and the datasets processed (or not) by the different filters yield significantly different *NECM* values with respect to one another. *NECM* is used as the response variable with cost ratio ($C_{\mathrm{II}}/C_{\mathrm{I}}$) values of 10, 20, 30, and 50.

Factor A consists of five levels ($\lambda = \{13, 17, 20, 23\}$ and *nof*) for *mpf* and *mpfcv*. On the other hand, factor A has six levels for *ipfmaj* (1st to 5th iteration and *nof*) and seven levels for *ipfcons* (1st to 5th iteration, 8th iteration, and *nof*). Factor B represents the four different final learners ($b = 4$). An ANOVA model is built for each filter and on both the fit and test datasets at different cost ratios. Due to paper-size consideration, only the ANOVA models for cost ratio 20 are presented as an example. The ANOVA results on the fit and test datasets are presented in Tables 4 and 5, respectively. In the tables SS, DF, and MS refer to the sum of squares, the degrees of freedom, and the mean squares, respectively. The F-value is selected at 90% confidence level (i.e., the significance level is 0.10).

Examining the ANOVA models based on the quality-of-fit of the final classifiers for all cost ratios, indicated that the *NECM* values across the datasets processed either at different filtering levels or at different numbers of iterations are significantly different. For example, in Table 4 this is indicated by p-values less than 0.0001. The results statistically confirm the intuitive assumption that the quality-of-fit would improve as more and more software modules likely to be noisy are eliminated from the fit dataset.

The performances on the test dataset are similar (at 0.10 significance) across the different levels of factor A for both *mpfcv* and *ipfmaj* (Table 5). However, there is a significant difference across the filtering levels and the number of iterations for *mpf* and *ipfcons*, respectively. The computed F-values are, however, far lower than those computed for the performance on the fit datasets. These conclusions are true at any cost ratio. Depending

Table 4. ANOVA Models on the *Fit* Dataset at $C_{\mathrm{II}}/C_{\mathrm{I}} = 20$

Filter	Component	SS	Variation (%)	DF	MS	F-Computed	F-Table	p-value
	Levels	5.65	99.49	4	1.41	786.80	2.48	0.0000
mpf	Learners	0.01	0.13	3	0.00	1.40	2.61	0.2904
	Errors	0.02	0.38	12	0.00			
	Levels	1.01	96.96	4	0.25	196.65	2.48	0.0000
mpfcv	Learners	0.02	1.56	3	0.01	4.21	2.61	0.0298
	Errors	0.02	1.48	12	0.00			
	Iterations	3.77	98.55	6	0.63	398.78	2.13	0.0000
ipfcons	Learners	0.03	0.71	3	0.01	5.71	2.42	0.0063
	Errors	0.03	0.74	18	0.00			
	Iterations	6.41	99.39	5	1.28	567.13	2.27	0.0000
ipfmaj	Learners	0.01	0.08	3	0.00	0.79	2.49	0.5179
	Errors	0.03	0.53	15	0.00			

Table 5. ANOVA Models on the *Test* Dataset at $C_{\mathrm{II}}/C_{\mathrm{I}} = 20$

Filter	Component	SS	Variation (%)	DF	MS	F-Computed	F-Table	p-value
	Levels	0.09	38.09	4	0.02	3.02	2.48	0.0613
mpf	Learners	0.06	24.10	3	0.02	2.55	2.61	0.1048
	Errors	0.09	37.82	12	0.01			
	Levels	0.05	12.92	4	0.01	0.60	2.48	0.6700
mpfcv	Learners	0.08	22.43	3	0.03	1.39	2.61	0.2940
	Errors	0.23	64.65	12	0.02			
	Iterations	0.11	37.68	6	0.02	2.66	2.13	0.0504
ipfcons	Learners	0.06	19.76	3	0.02	2.78	2.42	0.0705
	Errors	0.12	42.57	18	0.01			
	Iterations	0.07	31.98	5	0.01	1.99	2.27	0.1388
ipfmaj	Learners	0.04	19.77	3	0.01	2.05	2.49	0.1502
	Errors	0.11	48.25	15	0.01			

Table 6. Evaluation of the Contrasts $\hat{\mathcal{L}}_{nof\text{ vs. rest}}$ on the *Test* Dataset

Filter	$C_{\rm II}/C_{\rm I}=10$	$C_{\rm II}/C_{\rm I}=20$	$C_{\rm II}/C_{\rm I}=30$	$C_{\rm II}/C_{\rm I}=50$
mpf	0.0422	0.0914	0.1405	0.2388
mpfcv	0.0536	0.1171	0.1807	0.3078
ipfcons	0.0555	0.1215	0.1875	0.3195
ipfmaj	−0.0061	−0.0352	−0.0644	−0.1227

on the filtering scheme used, the results indicate that improving the quality of the fit dataset by using appropriate filtering schemes may significantly improve the performance accuracy of the final learners on the test dataset.

The generally low p-value related to factor B indicates that the bias of the four final learners has some impact on the performance accuracy for both the fit and the test datasets. This study, however, does not focus on the relative performance of the selected learners. However, in a recent study[1], we performed such an analysis with twenty-five final learners built on the JM1-8850 dataset processed by the Ensemble Filter.

3.6 Filtering Versus No Filtering

For a given filter, it is relevant to evaluate the contrast between the predictive performance of the learners induced on the filtered fit datasets and on the un-filtered fit dataset. This contrast and its evaluation are denoted as $\mathcal{L}_{nof\text{ vs. rest}}$ and $\hat{\mathcal{L}}_{nof\text{ vs. rest}}$, respectively. For example, the contrast related to *mpf* at cost ratio 10 is evaluated using Table 3[16]:

$$\begin{aligned}&\bar{y}_{.nof}-\frac{1}{4}(\bar{y}_{.\lambda=13}+\bar{y}_{.\lambda=17}+\bar{y}_{.\lambda=20}+\bar{y}_{.\lambda=23})\\&=0.964-\frac{1}{4}(0.951+0.927+0.925+0.886)\\&=0.042\end{aligned}\tag{4}$$

where $\bar{y}_{.nof}$ is the average expected costs of misclassification of the four final learners induced on $E_{\rm fit}$ and applied to the test dataset; $\bar{y}_{.\lambda}$ is the *NECM* average of the final classifiers induced on the training dataset filtered by *mpf* at filtering level λ and applied to the test dataset. The evaluation of the contrast for the four filtering schemes is presented in Table 6. The positive values, except for the *ipfmaj* filter, indicate that the performance with noise filtering improves compared to the un-filtering approach.

In order to assess which contrasts are significant, the p-values are computed using the LSD statistical test[16] (Table 7). Since there is no significant treatment effect present in factor A for both *mpfcv* and *ipfmaj*, there is no difference (at 0.10 significance) between the filtering and un-filtering approaches for these two noise handling techniques. However, the contrast is significant for *mpf* at significance level 0.10 for cost ratios 10, 20, and 30 — for cost ratio 50, the significance is at 0.11. The same conclusions can be applied for *ipfcons* at significance level 0.03 at any cost ratio that we considered. A similar contrast was considered for the performance related to the quality-of-fit. As expected, it was found that the contrast is significant for all the filtering schemes and at any cost ratio (p-values below 0.0001).

Table 7. Significance of the Contrasts $\mathcal{L}_{nof\text{ vs. rest}}$ on the *Test* Dataset

Filter	$C_{\rm II}/C_{\rm I}=10$	$C_{\rm II}/C_{\rm I}=20$	$C_{\rm II}/C_{\rm I}=30$	$C_{\rm II}/C_{\rm I}=50$
mpf	0.0479	0.0805	0.0931	0.1035
mpfcv	0.1040	0.1570	0.1738	0.1868
ipfcons	0.0047	0.0131	0.0170	0.0205
ipfmaj	0.7467	0.4604	0.4039	0.3677

Removing noise from the fit dataset using the Multiple-Partitioning Filter without the cross-validation constraint (*mpf*) and the Iterative-Partitioning Filter with consensus scheme (*ipfcons*) improves the predictive accuracy of the final learners on the test dataset for most cost ratios. Additionally, we observe that the performance improves for *mpfcv*, but it is not significant at 10%. In contrast, there is no performance improvement for *ipfmaj*. Overall, we have shown that filtering the fit dataset improves the prediction accuracy on the test dataset for some noise handling techniques, namely *mpf* and *ipfcons*.

These results can be attributed to the filtering efficiency. An efficient filter removes noisy instances while retaining good quality data[12]. Only the *mpf* and *ipfcons* filters have some significant effects on the performance on the test dataset. It may indicate that filtering the fit dataset with these two filters removes noisy instances in a way that allows the final models to induce rules which describe concepts of the given domain problem more accurately[2]. *mpf* and *ipfcons* may be relatively efficient filters compared to their counterparts, *mpfcv* and *ipfmaj*.

One may argue that the predictive performance on the test dataset does not reflect the true accuracy of the final learner because the test dataset (i.e., JM1-4425-Test) is as noisy as the original training dataset (i.e., JM1-4425-Fit). It may be possible that an instance appears to be correctly classified despite the fact that it is noisy. More specifically, suppose that an instance in the test dataset is labeled *fp*, and also suppose that it was mislabeled (i.e., class noise) because of an error during the data collection process. Hence, the given instance is actually *nfp*. If the final learner classifies it as *fp*, the instance is not considered misclassified — but, in reality it is misclassified. Even if the final learner generates a hypothesis covering noisy examples by overfitting the noisy data, the hypothesis is unlikely to cover noisy data points in the test dataset. If the noise was systematic in both the fit and the test datasets, it would

be expected that training on the erroneous training set would yield good performance on the test set which is also erroneous[2].

It is also worth noting that filtering the fit dataset has less impact on the quality-of-test than on the quality-of-fit. This result leads to a very important practical conclusion: *filtering the fit dataset is not sufficient enough to ensure good predictions on the test dataset.* In other words, the practitioner should not always expect improvement in the quality-of-test results.

3.7 Multiple Pairwise Comparisons

Since there is generally a significant treatment effect among the levels of factor A for both *mpf* and *ipfcons* on the test dataset (Fig.1), Tables 8 and 9 present multiple pairwise comparisons among these levels using Tukey's test[16]. Comparisons are only shown for *mpf* and *ipfcons*. The levels of factor A (i.e., the filtering level) are ordered from the least to the most accurate predictive performances. The results are presented for cost ratios 10, 20, 30, and 50, at significance level 0.10.

Table 8. Multiple Pairwise Comparison Results for *mpf*

$C_{II}/C_I = 10$		$C_{II}/C_I = 20$		$C_{II}/C_I = 30$		$C_{II}/C_I = 50$		Levels
	X		X		X		X	*nof*
	X		X		X		X	13
Y	X	Y	X	Y	X	Y	X	17
Y	X	Y	X	Y	X	Y	X	20
Y		Y		Y		Y		23

Table 9. Multiple Pairwise Comparison Results for *ipfcons*

$C_{II}/C_I = 10$		$C_{II}/C_I = 20$		$C_{II}/C_I = 30$		$C_{II}/C_I = 50$		Iterations
	X		X		X		X	*nof*
Y	X	Y	X	Y	X	Y	X	8
Y	X	Y	X	Y	X	Y	X	2
Y	X	Y	X	Y	X	Y	X	4
Y	X	Y	X	Y	X	Y	X	1
Y		Y	X	Y	X	Y	X	3
Y		Y		Y		Y		5

Table 8 indicates that levels *nof*, 13, 17, and 20 belong to cluster X at any cost ratio. In other words, these levels are similar to each other, at significance level 0.10. On the other hand, levels 17, 20, and 23 belong to cluster Y. Since levels 17 and 20 also belong to X, level 23 is significantly different from levels 13 and *nof*. Since $\bar{y}_{.\lambda=23} < \bar{y}_{.nof}$ at any cost ratio when the filter of interest is *mpf* (Table 3), the predictive performance on the test dataset does significantly improve compared to the not filtering approach (*nof*).

Table 9 clusters the levels of factor A for the test dataset filtered by the Iterative-Partitioning Filter with consensus scheme. Similarly, combining the results in Tables 3 and 9 indicates that filtering the fit dataset using the Iterative-Partitioning Filter with consensus scheme at its 5th iteration (*ipfcons*-5) significantly improves the predictive performance compared to the not filtering approach (*nof*), at any cost ratio. Table 9 also indicates that there is a significant difference in performance between the not filtering approach and *ipfcons*-3 at cost ratio 10.

Similar to the discussion in Subsection 3.6, these results may provide an indication on the relative efficiency of the two filters. Filtering noise with *mpf*-23 and *ipfcons*-5 may remove instances in such a way that the final learners have enough instances to learn the concepts of the domain problem. Selecting the most aggressive filter may not produce the best performance on the test dataset. This may be due to the removal of clean instances from the training dataset when an aggressive filtering is used. In other words, relying on robust final learners with a manageable level of noise may be recommended instead of using a noise-free but small training dataset. Hence, it is critical for the practitioner to find the right balance between aggressive and conservative noise filtering schemes.

4 Conclusions

This paper studies the performance of final learners induced on potentially noise-free training datasets and evaluated on a noisy test dataset. The quality of the training dataset is improved by removing instances detected as noisy by the Multiple-Partitioning Filter or the Iterative-Partitioning Filter. The amount of noise removed is varied by tuning either the filtering level or the number of iterations.

It was found that processing the fit dataset with the Multiple-Partitioning Filter without the cross-validation constraint (*mpf*) or the Iterative-Partitioning Filter with consensus scheme (*ipfcons*) improves the performance of the final learners on the test dataset for most cost ratios that we considered. On the other hand, the quality-of-test does not significantly improve when the fit dataset is filtered by the Multiple-Partitioning Filter with the cross-validation constraint (*mpfcv*) and the Iterative-Partitioning Filter with majority voting (*ipfmaj*).

The improvement in performance on the test dataset remains less significant than that on the fit dataset. However, even with a noise free training dataset, the learner may still perform poorly on the noisy test dataset. In order to reduce the presence of noise in the datasets, resources should be dedicated to ensure error-free data collection for the given project.

The performance of the final learners on the test dataset allows us to make some assumptions about the relative efficiencies among the different levels of conservativeness of a given filter. For the software measurement dataset studied, it is concluded that, *mpf*-23 and *ipfcons*-5 are relatively efficient filters. By injecting artificial noise, future work can confirm the efficiency of the Multiple-Partitioning Filter without the cross-validation constraint and the Iterative-Partitioning Filter with consensus scheme.

References

[1] Taghi M Khoshgoftaar, Shi Zhong, Vedang Joshi. Noise elimination with ensemble-classifier filtering for software quality estimation. *Intelligent Data Analysis*, 2005, 9(1): 3∼27.

[2] Witten I H, Frank E. Data Mining, Practical Machine Learning Tools and Techniques. 2nd Edition, Morgan Kaufmann, 2005.

[3] Khoshgoftaar T M, Seliya N. Analogy-based practical classification rules for software quality estimation. *Empirical Software Engineering Journal*, December 2003, 8(4): 325∼350.

[4] Khoshgoftaar T M, Allen E B. Logistic regression modeling of software quality. *International Journal of Reliability, Quality, and Safety Engineering*, 1999, 6(4): 303∼317.

[5] Zhu X, Wu X, Chen Q. Eliminating class noise in large datasets. In *Proc. the 20th Int. Conf. Machine Learning*, Washington DC, August 2003, pp.920∼927.

[6] Owen D B. Data Quality Control: Theory and Pragmatics. New York: Marcel Dekker, NY, 1990.

[7] Wang R Y, Storey V C, Firth C P. A framework for analysis of data quality research. *IEEE Trans. Knowledge and Data Engineering*, August 1995, 7(4): 623∼639.

[8] Teng C M. A comparison of noise handling techniques. In *Proc. the Int. Florida Artificial Intelligence Research Symposium*, 2001, pp.269∼273.

[9] Gamberger D, Lavrač N, Džeroski S. Noise elimination in inductive concept learning: A case study in medical diagnosis. In *Algorithmic Learning Theory: Proc. the 7th Int. Workshop*, Sydney, Australia, *LNCS* 1160, Springer-Verlag, October, 1996, pp.199∼212.

[10] Teng C M. Evaluating noise correction. In *Lecture Notes in Artificial Intelligence: Proc. the 6th Pacific Rim Int. Conf. Artificial Intelligence*, Melbourne, Australia, Springer-Verlag, 2000, pp.188∼198.

[11] Brodley C E, Friedl M A. Identifying mislabeled training data. *Journal of Artificial Intelligence Research*, 1999, 11: 131∼167.

[12] Rebours P. Partitioning filter approach to noise elimination: An empirical study in software quality classification [Thesis]. Florida Atlantic University, Boca Raton, FL, April 2004, Advised by Khoshgoftaar T M.

[13] Khoshgoftarr T M, Allen E B. A practical classification rule for software quality models. *IEEE Trans. Reliability*, June 2000, 49(2): 209∼216.

[14] Jain R. The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling. John Wiley & Sons, 1991.

[15] Berenson M L, Levine D M, Goldstein M. Intermediate Statistical Methods and Applications: A Computer Package Approach. Englewood Cliffs: Prentice Hall, NJ, 1983.

[16] Christensen R. Analysis of Variance, Design and Regression. Applied Statistical Methods. 1st Edition, Chapman & Hall, 1996.

[17] Fenton N E, Pfleeger S L. Software Metrics: A Rigorous and Practical Approach. 2nd Edition, Boston: PWS Publishing, MA, 1997.

[18] Quinlan J R. C4.5: Programs for Machine Learning. San Mateo: Morgan Kaufmann, CA, 1993.

[19] Holte R C. Very simple classification rules perform well on most commonly used datasets. *Machine Learning*, 1993, 11: 63∼91.

[20] Atkeson C G, Moore A W, Schaal S. Locally weighted learning. *Artificial Intelligence Review*, 1997, 11(1/5): 11∼73.

[21] Cohen W W. Fast effective rule induction. In *Proc. the 12th Int. Conf. Machine Learning*, Priedities A, Russell S (eds.), Tahoe City: Morgan Kaufmann, CA, July 1995, pp.115∼123.

[22] Kolodner J. Case-Based Reasoning. San Mateo, CA: Morgan Kaufmann, 1993.

Taghi M. Khoshgoftaar is a professor of the Department of Computer Science and Engineering, Florida Atlantic University and the director of the Data Mining and Machine Learning and Empirical Software Engineering Laboratories. His research interests are in software engineering, software metrics, software reliability and quality engineering, computational intelligence, computer performance evaluation, data mining, machine learning, and statistical modeling. He has published more than 300 refereed papers in these areas. He is a member of the IEEE, IEEE Computer Society, and IEEE Reliability Society. He was the program chair and general chair of the IEEE International Conference on Tools with Artificial Intelligence in 2004 and 2005 respectively. Khoshgoftaar received an M.S. degree in applied mathematics from MIT, an M.S. degree in computer science from North Carolina State University, and a Ph.D. degree in statistics from Virginia Tech.

Pierre Rebours received the M.S. degree in computer engineering from Florida Atlantic University, Boca Raton, FL, USA, in April 2004. His research interests include quality of data and data mining.