Software Quality Analysis of Unlabeled Program Modules With Semisupervised Clustering

Naeem Seliya, *Member, IEEE*, and Taghi M. Khoshgoftaar, *Member, IEEE*

***Abstract*—Software quality assurance is a vital component of software project development. A software quality estimation model is trained using software measurement and defect (software quality) data of a previously developed release or similar project. Such an approach assumes that the development organization has experience with systems similar to the current project and that defect data are available for all modules in the training data. In software engineering practice, however, various practical issues limit the availability of defect data for modules in the training data. In addition, the organization may not have experience developing a similar system. In such cases, the task of software quality estimation or labeling modules as fault prone or not fault prone falls on the expert. We propose a semisupervised clustering scheme for software quality analysis of program modules with no defect data or quality-based class labels. It is a constraint-based semisupervised clustering scheme that uses k-means as the underlying clustering algorithm. Software measurement data sets obtained from multiple National Aeronautics and Space Administration software projects are used in our empirical investigation. The proposed technique is shown to aid the expert in making better estimations as compared to predictions made when the expert labels the clusters formed by an unsupervised learning algorithm. In addition, the software quality knowledge learnt during the semisupervised process provided good generalization performance for multiple test data sets. An analysis of program modules that remain unlabeled subsequent to our semisupervised clustering scheme provided useful insight into the characteristics of their software attributes.**

***Index Terms*—Constraint-based clustering, semisupervised learning, software measurements, software quality, unlabeled data.**

I. Introduction

SOFTWARE quality models are useful tools toward achieving the objectives of a software quality assurance initiative. A software quality model can be used to identify program modules that are likely to be defective [11], [25]. Subsequently, the limited resources allocated for software quality inspection and improvement can be targeted toward only those program modules, achieving a cost-effective resource utilization [13]. A software quality estimation model allows the software development team to track and detect potential software defects relatively early on during development, which is critical to many high-assurance systems.

Manuscript received April 29, 2005; revised November 30, 2006. This paper was recommended by Guest Editor H. Pham.

N. Seliya is with the Department of Computer and Information Science, University of Michigan–Dearborn, Dearborn, MI 48128 USA (e-mail: nseliya@umich.edu).

T. M. Khoshgoftaar is with the Department of Computer Science and Engineering, Florida Atlantic University, Boca Raton, FL 33431 USA (e-mail: taghi@cse.fau.edu).

Digital Object Identifier 10.1109/TSMCA.2006.889473

A software quality model is typically trained using software measurement and defect (quality) data of a previously developed release or similar project [10], [12]. The trained model is then applied to modules of the current project to estimate their quality. Such a supervised learning approach assumes that the development organization has experience with systems similar to the current project and that defect data are available for all program modules in the training data.

In software development practice, however, various practical issues limit the availability of defect data for modules in the training data. For example, an organization may have not recorded or collected software defect data from previous releases or similar projects. In addition, since the organization may not have experience developing a similar system, the use of software measurement and defect data of previous projects for modeling purposes is inappropriate. In current times, where globalization of technology has gained momentum, distributed software development is not uncommon. Under such conditions, software defect data may not be collected by all development sites depending on the organizational structure and resources of individual sites.

The absence of defect data or quality-based class labels from the training data prevents following the commonly used supervised learning approach to software quality modeling. Consequently, the task of software quality estimation or labeling program modules as fault prone (fp) or not fault prone (nfp) falls on the software engineering expert. The process of labeling each program module one at a time is a laborious, expensive, and time-consuming effort. We propose a semisupervised clustering scheme to aid the expert in the labeling process.

The proposed scheme is based on constraint-based clustering using k-means as the underlying algorithm. During the k-means clustering process, the constraint maintains membership of modules (instances) to clusters that are already labeled as either fp or nfp. The proposed approach is semisupervised because the clustering process is aided with a set of labeled program modules. This set of modules is obtained after a software engineering expert labels certain modules based on confidence in his prediction as discussed further in Section VI. To our knowledge, this is the first study to investigate semisupervised clustering for the software quality modeling, analysis, and estimation problem.

Clustering is an appropriate choice for software quality analysis in the absence of fault-proneness labels or defect data [32]. Given software measurements (attributes) of the unlabeled program modules, clustering algorithms group the modules according to similarity of their software attributes. The underlying assumption is that program modules with similar attributes will have similar quality characteristics. Hence, fp modules will have similar software measurements and will likely group together as clusters. Similarly, nfp modules will also group together as clusters. Upon completing the clustering process, the software engineering expert can label clusters as fp or nfp based on characteristics of the data within each cluster.

The expert who labels the unlabeled program modules clearly benefits from applying a clustering algorithm. The expert can inspect and label each cluster (and the modules within it) as one entity instead of inspecting and labeling each module one at a time. This reduces the tedious task of labeling and expedites the software quality analysis initiative. As we shall see shortly, in our proposed semisupervised clustering approach, the expert plays an important role by labeling clusters as fp or nfp during the iterative process.

We investigate the proposed approach using software measurement and defect data from a previously developed National Aeronautics and Space Administration (NASA) software project JM1. The end result after a run of the proposed semisupervised clustering approach is a set of labeled clusters and a set of unlabeled clusters. The labeled clusters are the result of the expert's labeling during the process, whereas the unlabeled clusters are those that the expert did not label.

For the labeled program modules, we compare classification performances of the semisupervised clustering technique and an unsupervised clustering method investigated in a related study [32]. Classification of the labeled modules by the semisupervised clustering technique is better than the unsupervised clustering method. The software quality knowledge learnt during our semisupervised clustering of unlabeled modules (JM1) was applied to classify modules in several test data sets—other NASA software projects. It was found that semisupervised clustering generally had better classification of unseen modules as compared to unsupervised clustering. With respect to classification performances on the test data sets, the semisupervised clustering approach was compared to random classification. It was observed that random classification of program modules in the test data sets was considerably inferior to the semisupervised clustering approach.

An analysis of program modules belonging to clusters that remain unlabeled after our semisupervised clustering study suggests that many of them are likely noisy instances, difficult to model, or simply do not follow the underlying trend of the given software project. This can be reflective of the quality of the data obtained for the case study projects. We believe that this observation is of interest to the software project expert and analyst since it provides useful insight into the characteristics of program modules that remain unlabeled.

The remainder of this paper continues with Section II, where a brief discussion on related work is presented. The proposed semisupervised clustering scheme is detailed in Section III, followed by Section IV, in which a discussion is presented on methods used to analyze the remaining unlabeled program modules. A description of the software systems used in our case study is presented in Section V. The modeling details of our empirical study and the results obtained are presented in Sections VI and VII, respectively. Finally, we summarize this paper in Section VIII, including suggestions for future work.

II. Related Work

Semisupervised learning implies a methodology in which both labeled and unlabeled data are used during the learning process [8], [26]. However, most semisupervised learning approaches investigate the framework in the context of training a classifier and have been successfully used for various applications such as text classification [7], [17], [18], gesture recognition [30], and content-based image retrieval [4].

Nigam *et al.* [19] investigate semisupervised text classification based on a combination of expectation–maximization (EM) and a naive Bayes classifier. A cotraining semisupervised approach involves two different supervised learning algorithms that are used to complement each other based on the assumption that each algorithm would learn a different hypothesis from the labeled data set [2], [8],[18]. Fung and Mangasarian [6] present a concave minimization approach for classifying unlabeled data based on a combination of a clustering technique and a linear support vector machine. In contrast to the above existing semisupervised learning algorithms, the proposed approach is a semisupervised "clustering" approach without any actual labeled data: the partial supervision is obtained by a domain expert. In addition, some existing techniques assume that the data follow a predetermined distribution, such as the EM algorithm.

A few studies investigate semisupervised learning in the context of clustering a given set of instances [31]. These studies have generally been done in the context of document (text) classification and categorization. Semisupervised clustering has been rarely investigated for addressing similar problems in other domains, such as software quality engineering. Pedrycz and Waletzky [21], [22] investigate semisupervised clustering using fuzzy logic-based clustering for analyzing software reusability. In contrast, this paper investigates semisupervised clustering for software quality estimation.

The labeled instances in a semisupervised clustering scheme have been used for initial seeding of the clusters [1], incorporating constraints in the clustering process [28], or providing feedback subsequent to regular clustering. The seeded schemes use the labeled data to help initialize cluster centroids prior to clustering. The constraint-based approaches keep the grouping of the labeled data fixed throughout the clustering process. The feedback-based schemes use the labeled data to adjust the clusters subsequent to running the clustering process.

In this paper, we present a constraint-based semisupervised clustering scheme (Section VI) for software quality analysis of unlabeled program modules. The scheme utilizes the labeled program modules for initial seeding of (some of) the clusters. Semisupervised clustering has some definite advantages over semisupervised classification, in that it can assist in determining new categories of data (if needed) other than the known categories. In contrast, a semisupervised classification approach requires all the categories or classes of instances in the data set to be known as prior knowledge.

In the literature, most works on software quality estimation have focused on using a supervised learning approach for building fault-proneness detection or defect prediction models [9], [23], [24]. Very limited attention has been given to addressing the problem of software quality modeling and analysis in the absence of defect data or quality-based class labels from previous project development experiences. To our knowledge, this is the first study to investigate a semisupervised clustering scheme for software quality analysis of unlabeled program modules.

Pedrycz *et al.* investigate self-organizing maps (a clustering technique) for analysis and visualization of software measurement data [20]. However, their study does not address the problem of software quality analysis and estimation using unlabeled program modules. This paper analyzes unlabeled program modules in the context of their quality and characteristics of software measurements.

In a recent study, we investigated several clustering algorithms for analyzing (unsupervised) software measurement data [32]. A software engineering expert was used to label the clusters as either fp or nfp based on domain knowledge and descriptive statistics of each cluster. One of the motivations behind this paper is to help the expert improve his software quality estimation performance. As will be shown later in the text (Section VII), we compare the expert's classification performance for both unsupervised clustering and semisupervised clustering approaches.

III. Semisupervised Clustering Approach

The basic essence of using a semisupervised approach during clustering is to aid the clustering algorithm in making better partitions or grouping of instances in the given data set. In such a setting, the clustering algorithm is aided with a small set of labeled instances, which are used for either initial seeding, establishing constraints, or providing feedback. Hence, the data in a semisupervised clustering scheme consist of a small set of labeled instances and relatively large set of unlabeled instances—the labeled–unlabeled problem in data mining and machine learning [26].

The proposed semisupervised clustering approach for software quality analysis is a constraint-based scheme that uses labeled instances (in this paper, an instance is a program module of a given software project) for initial seeding (centroids) of some clusters among the maximum allowable clusters—a user-defined quantity when using k-means as the clustering algorithm. In addition, the semisupervised clustering approach is an iterative scheme that allows the domain (software engineering) expert to label additional clusters as either nfp or fp based on his/her domain knowledge and some descriptive statistics of the clusters.

Let D be a data set of labeled (nfp or fp) and unlabeled (ul) program modules and containing the subsets L and U of labeled and unlabeled program modules, respectively. The labeled data set is further divided into subsets L_nfp and L_fp of nfp and fp modules, respectively. In this paper, since all program modules in D are unlabeled, the L_nfp and L_fp subsets are obtained through expert-based labeling of some modules in D as nfp or fp (see Section VI). The procedure used in our constraint-based semisupervised clustering approach with k-means is enumerated as follows.

Step 1) **Obtain initial numbers of nfp and fp clusters.**
- Given L_nfp, execute the algorithm presented in Section III-A to obtain the optimal number of nfp clusters among $\{1, 2, \ldots, Cin\_nfp\}$ number of clusters, where Cin_nfp is the user-defined maximum number of clusters for L_nfp. Let p denote the obtained number of nfp clusters.
- Given L_fp, execute the algorithm presented in Section III-A to obtain the optimal number of fp clusters among $\{1, 2, \ldots, Cin\_fp\}$ number of clusters, where Cin_fp is the user-defined maximum number of clusters for L_fp. Let q denote the obtained number of fp clusters.

Step 2) **Initialize centroids of clusters.** Given the maximum number of clusters $C_{\max}$ allowed during the semisupervised clustering process with k-means.
- The centroids of p clusters out of $C_{\max}$ are initialized to centroids of the clusters labeled as nfp.
- The centroids of q clusters out of $\{C_{\max} - p\}$ are initialized to centroids of the clusters labeled as fp.
- The centroids of the remaining r (i.e., $C_{\max} - p - q$) clusters are initialized to r randomly selected modules from U. We randomly select five unique sets [see Step 3)] of instances for initializing the unlabeled clusters. Thus, centroids of the $\{p + q + r\}$ clusters can be initialized using five different combinations. We selected five unique sets for the initial seeding of the unlabeled clusters to allow a good (unbiased) coverage of the unlabeled program modules. It was felt that selecting three sets would be too few to avoid a (positive or negative) bias in the initial seeding of the unlabeled clusters. On the same token, it was felt that selecting more than five sets for initializing centroids of unlabeled clusters would increase the computational complexity without any substantial benefit.
- The sets of nfp, fp, and unlabeled clusters are thus $C\_nfp = \{c\_nfp_1, c\_nfp_2, \ldots, c\_nfp_p\}$, $C\_fp = \{c\_fp_1, c\_fp_2, \ldots, c\_fp_q\}$, and $C\_ul = \{c\_ul_1, c\_ul_2, \ldots, c\_ul_r\}$, respectively.

Step 3) **Execute constraint-based clustering.**
- The k-means clustering algorithm with the Euclidean distance function is run on D using the initialized centroids for the $C_{\max}$ clusters and under the constraint that an existing membership of a program module to a labeled cluster remains unchanged. The Euclidean distance is used due to its simplicity and common use (an investigation with various distance functions is out of the scope of this paper).
- The constraint-based clustering process with k-means is repeated for each of the five centroid initializations, and the respective sum-of-squares error (SSE) values are computed.

- The clustering results associated with the median SSE value is selected for continuation to the next step. This is done to minimize the likelihood of working with a lucky/unlucky initialization of cluster centroids.

Step 4) **Expert-based labeling of clusters.**

- The software engineering expert is presented with descriptive statistics of the r unlabeled clusters and is asked to label them as either nfp or fp. The specific statistics presented for attributes of modules in each cluster depends on the expert's request and include data such as minimum, maximum, mean, standard deviation, etc.
- The expert labels only those clusters for which he/she is very confident in the label estimation.
- If the expert labels at least one of the unlabeled clusters, then go to Step 2) and repeat; otherwise, continue.

Step 5) **Stop semisupervised clustering.**

- The iterative process is stopped when the sets C_nfp, C_fp, and C_ul remain unchanged. The modules in the nfp (fp) clusters are labeled and recorded as nfp (fp), whereas those in the ul clusters are not assigned any label. In addition, the centroids of the $\{p+q\}$ labeled clusters are also recorded.

A. *Optimal Clusters in Sum-of-Squares Clustering*

The optimal numbers of clusters for the nfp and fp modules in the initial labeled data set are obtained using the C_g criterion proposed by Krzanowski and Lai [15]. Assume (notations in this section are exclusive to the rest of this paper) that p variables have been measured for each instance in the data set and that $\mathbf{W}$ is the pooled within-group covariance matrix for any given cluster of the data set. Then, the sum of squares over all clusters provides an intuitive objective function where small values indicate good partitions.

Given the maximum number of clusters allowed (user-defined) in the data set, the optimal value for C_g is computed as follows.

Step 1) Compute the difference function $DF(g)$ when the number of groups in the partition is increased from $(g-1)$ to g, i.e.,

$$DF(g) = (g-1)^{\frac{2}{p}} \widetilde{S}_{(g-1)} - g^{\frac{2}{p}} \widetilde{S}_g \qquad (1)$$

where $\widetilde{S}_g$ is the optimum value of the sum-of-squares objective function for the given g.

Step 2) Compute $C_g = |(DF(g)/DF(g-1))|$.

Step 3) Repeat Steps 1) and 2) for $g = 1, 2, 3, \ldots, g_{\max}$, where $g_{\max}$ is defined by the user as the maximum number of clusters in determining the optimal value of g.

Step 4) The optimal value of g is the value that maximizes C_g.

IV. Analysis of the Remaining Unlabeled Modules

The program modules that remain unlabeled at the end of our semisupervised clustering scheme were further investigated for an insight into their characteristics. From a software practitioner's point of view, we wanted to investigate why those modules remain unlabeled subsequent to semisupervised clustering. In addition, what are the specific characteristics of these modules that lead them to remain unlabeled in this paper?

A possible opinion can be obtained from a data mining and machine learning perspective: the unlabeled modules constitute noisy instances in the data set. They could also be considered as difficult-to-learn modules or that they belong to a class other than the nfp and fp classes. Moreover, it can also be viewed that the software measurements of such modules do not adhere to the underlying measurement and quality trends of the given software system.

We compute the commonality of the modules that remain unlabeled after semisupervised clustering with those identified by two noise detection techniques, namely ensemble filter (EF) [14] and the pairwise attribute noise detection algorithm (PANDA) [27]. Commonality of modules among two sets of instances implies an intersection of the two sets, i.e., the same modules (based on module number) exist in both sets. The EF is a technique used to detect mislabeling errors or class noise. In contrast, our recently proposed PANDA approach detects instances with noisy attributes or measurements.

In an EF [3], a given number of classifiers are trained on the given data set and form the EF. The trained classifiers then work as a collective (by voting) in filtering instances that are consistently misclassified by a given majority number of classifiers. In a recent study [14], a large EF of 25 classifiers was built using the same software measurement data set used in this paper. We compute the commonality of program modules filtered by the EF with majority voting (13 or more agree) and the unlabeled modules remaining at the end of our semisupervised clustering scheme.

The classification methods (including notations) used in the EF are enumerated in Table I. Most of these classifiers were built using cross validation, whereas models were selected based on resubstitution error rates in the case of TD, LR, LOC, GP, ANN, and RBM due to the limitations of the respective tools used. A descriptive discussion on the 25 techniques is out of the scope of this paper; however, additional details for each classifier are provided in [14]. Several of the classifiers are implemented in the Waikato Environment for Knowledge Analysis data mining and machine learning tool, which was written in Java [29].

PANDA detects those instances that have a large deviation from normal given the values of a pair of attributes [27]. When a set of instances has similar values for one attribute, large deviations from normal for the other attribute may be considered suspicious. The underlying assumption is that the larger the deviation, the greater is the likelihood of that instance being noisy. The output of PANDA is a list of instances in the data set ordered from most likely noise to least likely noise, where each instance is assigned an output score (noise likelihood).

TABLE I
SUPERVISED CLASSIFIERS IN THE EF

Symbol	Classifier
CBR	Case-Based Reasoning
TD	Treedisc Decision Tree Algorithm
LR	Logistic Regression
LOC	Lines of Code
GP	Genetic Programming
ANN	Artificial Neural Networks
LBOOST	LogitBoost Classifier
RBM	Rule-Based Modeling
BAG	Bagging Classifier
RSET	Rough Sets-Based Classifier
MCOST	MetaCost Classifier
ABOOST	AdaBoost Classifier
DTABLE	Decision Table
ADT	Alternating Decision Table
SMO	Sequential Minimal Optimization
IB1	Instance-Based (1 nearest neighbor)
IBK	Instance-Based (k nearest neighbor)
PART	Partial Decision Trees
ONER	OneR Algorithm (based on one most informative attribute)
JRIP	Repeated Incremental Pruning to Produce Error Reduction
RDR	Ripple Down Rule Algorithm
J48	C4.5 Decision Tree Algorithm
NBAYES	Naive Bayes
HPIPES	Hyperpipes Algorithm
LWLS	Locally Weighted Learning

In PANDA, ordered pairs of attributes in the given data set are considered iteratively. Subsequent to discretization of the first attribute, the conditional mean and standard deviation of the second attribute are computed relative to the discretized value of the first attribute. Large deviations from normal by the second attribute within the discretized group of the first attribute have a higher contribution to the output score. This process is iterated for all ordered pairs of attributes, and the individual results are aggregated to an output score, indicating the amount of noise present in the instance. A very useful feature of PANDA is that it can be used with any number of given attributes, including (or without) the class label.

V. SOFTWARE MEASUREMENT DATA

The software measurements and quality data used in this paper to investigate the proposed semisupervised clustering approach are those of a large NASA software project JM1. Written in C, the JM1 project is a real-time ground system that uses simulations to generate certain predictions for missions. The data were made available through the Metrics Data Program (MDP) at NASA (http://mdp.ivv.nasa.gov/) and included software measurement data and associated error (fault or defect) data collected at the function level. Hence, a program module for the system consisted of a function or method.

The fault data collected for the system represents, for a given module, faults detected during software development. The original JM1 data set consisted of 10 883 software modules, of which 2105 modules had software defects (ranging from 1 to 26), whereas the remaining 8778 modules were defect free, i.e., with no software faults. In this paper, a program module with no faults was considered nfp and fp otherwise. We note that the terms errors, defects, and faults are used interchangeably in this paper.

TABLE II
SOFTWARE MEASUREMENTS IN JM1 AND KC2 DATA

Line Count Metrics	Total_Lines_of_Code Executable_LOC Comments_LOC Blank_LOC Code_And_Comments_LOC
Halstead Metrics	Total_Operators Total_Operands Unique_Operators Unique_Operands
McCabe Metrics	Cyclomatic_Complexity Essential_Complexity Design_Complexity
Branch Count Metric	Branch_Count

The JM1 data set contained some inconsistent modules (those with identical software measurements but with different class labels) and those with missing values. Upon removing such modules, the data set was reduced from 10 883 to 8850 modules. We denote this reduced data set as JM1-8850, which consisted of 1687 modules with one or more defects and 7163 modules with no defects.

Each program module in the JM1 was characterized by 21 software measurements [5]: 13 metrics, as shown in Table II, and eight derived Halstead metrics (Halstead_Length, Halstead_Volume, Halstead_Level, Halstead_Difficulty, Halstead_Content, Halstead_Effort, Halstead_Error_Est, and Halstead_Prog_Time). We only used the 13 basic software metrics in our analysis. The eight derived Halstead metrics were not used.

The metrics for the JM1 (and other software metrics data sets) were primarily governed by their availability, the internal workings of the projects, and the data collection tools used. The type and numbers of metrics made available were solely determined by the NASA MDP. Other metrics, including software process and object-oriented metrics, were not available. The use of the specific software metrics in this paper does not advocate their effectiveness: a different project may consider a different set of software measurements for analysis [11], [13]. For additional details on software measurements, readers are referred to Fenton and Pfleeger [5].

A. Software Measurement Test Data Sets

The JM1-8850 data set is used by our semisupervised clustering approach without the quality-based class labels, i.e., nfp and fp, and acts as the training data for software quality analysis. In order to gauge the performance of the semisupervised clustering results, we use software measurement data of six other NASA projects as test data sets, namely: 1) KC1; 2) KC2; 3) KC3; 4) CM1; 5) MW1; and 6) PC1. These software measurement data sets were also obtained through the NASA MDP.

The definitions of what constituted the fp and nfp modules for these projects are the same as those of the JM1 system. In addition, a program module of these projects also consisted of a function, subroutine, or method. These six projects were characterized by the same software product metrics used for the JM1 project and were built in a similar software development organization. Evaluating a given software quality model with an independent test data set simulates the application on a similar currently underdevelopment software project. The six test data sets are described as follows:

1) The KC1 project is a single computer software configuration item (CSCI) within a large ground system and consists of 43 000 lines of code (KLOC) of C++ code. A given CSCI comprises logical groups of computer software components. The data set contains 2107 modules, of which 325 have one or more faults and 1782 have zero faults. The maximum number of faults in a module is 7.
2) The KC2 project, which is written in C++, is the science data processing unit of a storage management system used for receiving and processing ground data for missions. The data set includes only those modules that were developed by NASA software developers and not commercial-of-the-shelf software. The data set contains 520 modules, of which 106 have one or more faults and 414 have zero faults. The maximum number of faults in a software module is 13.
3) The KC3 project, which is written in 18 KLOC of Java, is a software application that collects, processes, and delivers satellite metadata. The data set contains 458 modules, of which 43 have one or more faults and 415 have zero faults. The maximum number of faults in a module is 6.
4) The CM1 software measurement data set is that of a science instrument application written in C code with approximately 20 KLOC. The data set contains 505 modules, of which 48 have one or more faults and 457 have zero faults. The maximum number of faults in a module is 5.
5) The MW1 project is a software application from a zero-gravity combustion experiment that has been completed. The MW1 data set consists of 8 KLOC of C code. The data set contains 403 modules, of which 31 have one or more faults and 372 have zero faults. The maximum number of faults in a module is 4.
6) The PC1 project is flight software from an Earth-orbiting satellite that is no longer operational. It consists of 40 KLOC of C code. The software measurement data set contains 1107 modules, of which 76 have one or more faults and 1031 have zero faults. The maximum number of faults in a module is 9.

VI. Empirical Settings

The first order in our semisupervised clustering approach is to establish the initial labeled and unlabeled data sets. In a related study [32], clustering with k-means was used to cluster the JM1-8850 modules. Some descriptive statistics for the entire data set and each cluster (mean, standard deviation, etc.) were presented to the software engineering expert. The expert then labeled (nfp and fp) those clusters for which he was completely confident, resulting in a subset of 2863 labeled modules consisting of 2393 nfp and 470 fp modules. Thus, in this paper, the L_nfp, L_fp, and U data sets consist of the 2393 nfp, 470 fp, and 5987 ul program modules, respectively.

TABLE III
Initial Number of nfp and fp Clusters

	Max = 10	Max = 20
nfp	5	5
fp	5	11

The initial numbers of nfp and fp clusters, i.e., p and q, were obtained by setting both Cin_nfp and Cin_fp to two values, i.e., 10 and 20. These values were selected after a discussion with the software engineering expert. It was decided that considering these values would be sufficient to capture the diversity of the fp and nfp groups of labeled program modules. We thus have two case studies depending on the initial settings of Cin_nfp and Cin_fp. The case study denoted as JM1_2863_10 implies that $Cin\_nfp = 10$ and $Cin\_fp = 10$, whereas the case study denoted as JM1_2863_20 implies that $Cin\_nfp = 20$ and $Cin\_nfp = 20$.

The maximum number of clusters allowed during our semisupervised clustering with k-means was set to three values, i.e., $C_{\max} = \{30, 40, 50\}$. These values were selected based on input from the domain expert and reflected a similar setting used in our previous work [32]. Investigating the semisupervised clustering scheme with three different $C_{\max}$ values provides insight into the affect of $C_{\max}$ on the obtained performance. Given the two settings of Cin_nfp and Cin_fp and the three values for $C_{\max}$, there were six experiments performed in this paper.

To aid the expert with labeling of unlabeled clusters at a given iteration during the semisupervised clustering process, the following statistics were computed (as per expert's request) for each unlabeled cluster: minimum, maximum, mean, median, standard deviation, and the 75, 80, 85, 90, and 95 percentiles. These values were computed for all 13 software attributes of modules in a given unlabeled cluster. In addition, the expert was also presented with the following statistics for JM1-8850 and the unlabeled data set at a given iteration: minimum, maximum, mean, median, standard deviation, and the 5, 10, 15, 20, 25, 30, 35, 40, 45, 55, 60, 70, 75, 80, 85, 90, and 95 percentiles. The extent to which the above descriptive statistics were used was at the disposal of the expert during the labeling task.

VII. Results and Discussion

A. Semisupervised Clustering

The initial numbers of nfp and fp clusters were determined based on the algorithm presented in Section III-A, as shown in Table III. Please note that some important notations used to present the results are summarized in Table IV for quick reference. For case study JM1_2863_10, the optimal initial number for both nfp and fp clusters is 5, implying that 10 clusters out of the given $C_{\max}$ were initially labeled as either nfp or fp. Similarly, for case study JM1_2863_20, the initial numbers of nfp and fp clusters are 5 and 11, respectively, implying that 16 out of the given $C_{\max}$ were initially labeled.

The clustering results obtained subsequent to each semisupervised clustering run are summarized in Table V, which shows the results for both case studies. The first column, i.e., $C_{\max}$, indicates the three different maximum numbers of

TABLE IV
IMPORTANT NOTATIONS OF RESULT TABLES

Symbol	Description
nfp	A not fault-prone module
fp	A fault-prone module
pred_nfp	# of modules predicted as *nfp*
pred_fp	# of modules predicted as *fp*
n_lab	# of modules labeled as *nfp* or *fp*
n_unlab	# of unlabeled modules
num_rem_cls	# of clusters that remain unlabeled
C_{max}	Maximum # of clusters allowed
num_iter	# of semi-supervised clustering iterations
Type I error	A *nfp* module misclassified as *fp*
Type II error	A *fp* module misclassified as *nfp*
Overall	The total misclassification rate
common	# of modules intersecting two populations, e.g. noisy modules by PANDA vs. *n_unlab*
proportion	% of modules intersecting two populations with respect to *n_unlab*

TABLE V
RESULTS OF EXPERT-BASED CLUSTER LABELING

Case Study JM1_2863_10						
C_{max}	*num_iter*	*nfp*	*fp*	*n_lab*	*n_unlab*	*num_rem_cls*
30	4	4145	1791	5936	2914	10
40	4	4547	1810	6357	2493	16
50	4	4320	1628	5948	2902	27
Case Study JM1_2863_20						
C_{max}	*num_iter*	*nfp*	*fp*	*n_lab*	*n_unlab*	*num_rem_cls*
30	2	4840	1628	6468	2382	7
40	2	4469	1541	6010	2840	13
50	6	3898	1755	5653	3197	18

clusters used in this paper. The column num_iter lists the number of iterations needed for the respective semisupervised clustering process. The columns n_lab and n_unlab, respectively, indicate the numbers of program modules that were labeled and remain unlabeled subsequent to a given experiment. The last column indicates the number of clusters that remained unlabeled by the expert at the end of each experiment.

The number of modules that remain unlabeled seems to increase with $C_{\max}$ for case study JM1_2863_20, and conversely, the number of modules that were labeled decreased with an increase in $C_{\max}$. This would suggest that the expert's labeling process is affected by the granularity of the groupings or clusters made for the given data set. This may be indicative that for the JM1 system, when more (a finer granularity) information about the unlabeled program modules is presented to the expert, some additional characteristics of those modules become visible to the expert and are subsequently used in his labeling process. However, the above trend was not clearly observed for case study JM1_2863_10. An increase in the number of modules that remain unlabeled (and a decrease in the number of labeled modules) was only seen when $C_{\max}$ was increased from 40 to 50 clusters. When $C_{\max} = 30$, the number of modules that remain unlabeled was the highest, i.e., 2914.

B. Classification Performance

An important aim of our semisupervised clustering scheme was to help the software engineering expert make better labeling estimations in the presence of no defect data or quality-based labels. We compare the classification performance of our semisupervised clustering scheme with the classification

TABLE VI
CLASSIFICATION PERFORMANCE OF LABELED MODULES

	Case Study JM1_2863_10			Unsupervised Clustering		
C_{max} →	30	40	50	30	40	50
n_lab	5936	6357	5948	5936	6357	5948
Type I	0.1800	0.1700	0.1546	0.2265	0.2127	0.2007
Type II	0.2051	0.2448	0.2276	0.3785	0.4121	0.4073
Overall	0.1850	0.1847	0.1686	0.2552	0.2501	0.2387
	Case Study JM1_2863_20			**Unsupervised Clustering**		
C_{max} →	30	40	50	30	40	50
n_lab	6468	6010	5653	6468	6010	5653
Type I	0.1406	0.1374	0.1842	0.1837	0.1840	0.2332
Type II	0.2804	0.2483	0.1841	0.4466	0.4245	0.3627
Overall	0.1674	0.1589	0.1842	0.2321	0.2285	0.2579

obtained by unsupervised clustering and expert-based labeling of clusters. In a recent study [32], we investigated unsupervised clustering techniques on the JM1-8850 data set. In that study, the k-means and Neural-Gas [16] clustering algorithms were used for $C_{\max} = 30$ clusters. Similar to this study, the expert was given descriptive statistics for each cluster and was asked to label them as either nfp or fp.

The classification performances of the semisupervised and unsupervised clustering schemes are compared for the modules that were labeled during the semisupervised clustering process. We use the actual class labels of the JM1 program modules to compute the misclassification error rates: Type I (an nfp module misclassified as fp), Type II (an fp module misclassified as nfp), and overall error rates. The misclassification error rates of the two techniques are summarized in Table VI.

The unsupervised clustering scheme (Table VI) reflects the results based on the Neural-Gas algorithm since it yielded better performance than k-means [32]. The row n_lab indicates the number of modules that were labeled during the semisupervised clustering process. It is clearly seen that the semisupervised clustering scheme has helped the expert in making better predictions than relying on unsupervised clustering. The Type I, Type II, and overall misclassification error rates of all six semisupervised clustering experiments are better than the corresponding error rates of unsupervised clustering. The significance in the improvement was statistically verified by using a paired t-test at 5% significance level for the overall misclassifications.

The program modules of the six NASA software measurement data sets, other than JM1 (as described in Section V), are used as test data sets to evaluate the software quality knowledge learnt through the semisupervised clustering process as compared to unsupervised clustering with Neural-Gas. The test data modules are classified based on their Euclidean distance from centroids of the final nfp and fp clusters at the end a semisupervised clustering run. A similar classification is made using centroids of the nfp and fp clusters labeled by the expert after unsupervised clustering with the Neural-Gas algorithm. The assumption is that the closer a module is to a given labeled cluster, the more likely it is of the same class label.

The classification performances obtained by unsupervised clustering for the test data sets are shown in Table VII. The classification performances obtained by the respective semisupervised clustering runs are summarized in Table VIII, which shows the misclassification error rates for the case studies

TABLE VII
TEST DATA PERFORMANCE WITH UNSUPERVISED CLUSTERING

Data	*Type I*	*Type II*	*Overall*
KC1	0.0617	0.6985	0.1599
KC2	0.0918	0.4151	0.1577
KC3	0.1229	0.5116	0.1594
CM1	0.2123	0.4792	0.2376
MW1	0.1156	0.4516	0.1414
PC1	0.1736	0.5395	0.1987

TABLE VIII
TEST DATA MISCLASSIFICATION RATES WITH SEMISUPERVISED CLUSTERING

	Case Study JM1_2863_10			Case Study JM1_2863_20		
Data	*Type I*	*Type II*	*Overall*	*Type I*	*Type II*	*Overall*
$C_{max} = 30$ Clusters						
KC1	0.0988	0.3600	0.1391	0.0847	0.3692	0.1286
KC2	0.1159	0.3113	0.1558	0.1039	0.3396	0.1519
KC3	0.1277	0.3488	0.1485	0.1253	0.3488	0.1463
CM1	0.2801	0.3542	0.2871	0.2560	0.3958	0.2693
MW1	0.1263	0.3548	0.1439	0.1102	0.3871	0.1315
PC1	0.1775	0.3289	0.1879	0.1659	0.3553	0.1789
$C_{max} = 40$ Clusters						
KC1	0.0999	0.3569	0.1395	0.0847	0.3692	0.1286
KC2	0.1184	0.3019	0.1558	0.1039	0.3396	0.1519
KC3	0.1325	0.3488	0.1528	0.1253	0.3488	0.1463
CM1	0.2845	0.3542	0.2911	0.2560	0.3958	0.2693
MW1	0.1317	0.3548	0.1489	0.1183	0.3871	0.1390
PC1	0.1814	0.3421	0.1924	0.1697	0.3553	0.1825
$C_{max} = 50$ Clusters						
KC1	0.0999	0.3508	0.1386	0.0932	0.3508	0.1329
KC2	0.1208	0.3019	0.1577	0.1329	0.3113	0.1692
KC3	0.1301	0.3488	0.1507	0.1325	0.3256	0.1507
CM1	0.2867	0.3542	0.2931	0.2648	0.3542	0.2733
MW1	0.1290	0.3548	0.1464	0.1210	0.3548	0.1390
PC1	0.1794	0.3289	0.1897	0.1697	0.3421	0.1816

TABLE IX
AVERAGE CLASSIFICATION PERFORMANCE FOR TEST DATA SETS

	Case Study JM1_2863_10			Case Study JM1_2863_20		
Data	*Type I*	*Type II*	*Overall*	*Type I*	*Type II*	*Overall*
KC1	0.0995	0.3559	0.1391	0.0875	0.3631	0.1300
KC2	0.1184	0.3050	0.1564	0.1135	0.3302	0.1577
KC3	0.1301	0.3488	0.1507	0.1277	0.3411	0.1477
CM1	0.2837	0.3542	0.2904	0.2589	0.3819	0.2706
MW1	0.1290	0.3548	0.1464	0.1165	0.3763	0.1365
PC1	0.1794	0.3333	0.1900	0.1684	0.3509	0.1810

JM1_2863_10 and JM1_2863_20. We observed that for each of the two case studies, the misclassification error rates obtained with different $C_{\max}$ values were relatively similar for a given test data set. The average misclassification error rates over the three $C_{\max}$ values are shown in Table IX.

The unsupervised clustering results generally tend to have lower Type I error rates with relatively much higher Type II error rates, except for the CM1 test data set. In contrast and as compared to the unsupervised clustering approach, while yielding relatively similar Type I (false positive) error rates, the semisupervised clustering approach yielded much better performances (lower false negative error rates) in classifying fp modules of the test data sets—with the exception of the CM1 data set. In the case of CM1, there is a larger difference between the Type I and Type II error rates with unsupervised clustering than semisupervised clustering. However, since neither false-positive nor false-negative error rates of the respective techniques are relatively similar, a conclusive comparison cannot be made.

TABLE X
CLASS DISTRIBUTIONS OF TEST DATA SETS

Data	*n*	*nfp*	*fp*	*p*	*(1 - p)*
KC1	2107	1782	325	0.1542	0.8458
KC2	520	414	106	0.2038	0.7962
KC3	458	415	43	0.0939	0.9061
CM1	505	457	48	0.0950	0.9050
MW1	403	372	31	0.0769	0.9231
PC1	1107	1031	76	0.0687	0.9313

Among the six test data sets, the classification of (unseen) fp modules improves for five test data sets when going from an unsupervised approach to a semisupervised clustering scheme. This would suggest that additional knowledge regarding software quality was learnt during the semisupervised clustering process. From a software practitioner's point of view, the semisupervised clustering performance is preferred over that of unsupervised clustering: for relatively similar numbers of nfp modules correctly detected, more fp modules are correctly detected by the semisupervised clustering technique.

We were curious to investigate the performance of random classification in comparison to classification based on knowledge gained by the semisupervised clustering process. The overhead of utilizing a given software quality modeling and analysis process is not warranted if a random classification yields similar or better software quality performance. The six (test data sets) software measurement data sets are used in this investigation. To allow a relative comparison with the semisupervised clustering results, it would be appropriate only to randomly label modules as fp and nfp in the given data set such that the respective numbers of predicted fp and nfp modules are the same as those obtained by the semisupervised clustering approach.

For the given software measurement data set, if the above-described random classification was performed several times, the expected Type I and Type II error rates would be governed by the proportions of the two classes, i.e., nfp and fp. If a data set consists of 900 nfp and 100 fp modules, a randomly selected module has a 0.10 probability of being correctly predicted as fp $(p = 10\%)$. Similarly, a module has a 0.90 probability of being correctly predicted as nfp $(1 - p = 90\%)$. Assuming that 200 modules are randomly predicted as fp and the remaining 800 modules are predicted as nfp, then the expected Type I and Type II error rates are p and $(1 - p)$, respectively: Type I $= (0.10 \times 800/800)$ and Type II $= (0.90 \times 200/200)$. Among the 200 modules predicted as fp, the expected number of correct predictions is $0.10 \times 200 = 20$, and among the 800 modules predicted as nfp, the expected number of correct predictions is $0.90 \times 800 = 720$.

The nfp and fp class distributions of the test data sets are shown in Table X. The expected random classification results are summarized in Table XI, where $pred\_nfp$ and $pred\_fp$ represent the numbers of predicted nfp and fp modules. These numbers are the same as the average numbers of nfp and fp modules predicted by the semisupervised clustering approach for case study JM1_2863_10. Obviously, the Type I and Type II

TABLE XI
RANDOM CLASSIFICATION FOR TEST DATA SETS

Data	*pred_nfp*	*pred_fp*	*Type I*	*Type II*	*Overall*
KC1	1720	387	0.1542	0.8458	0.2811
KC2	397	123	0.2038	0.7962	0.3436
KC3	376	82	0.0939	0.9061	0.2393
CM1	344	161	0.0950	0.9050	0.3527
MW1	335	68	0.0769	0.9231	0.2197
PC1	871	236	0.0687	0.9313	0.2523

TABLE XII
COMMONALITY WITH EF AND MAJORITY VOTING

Case Study JM1_2863_10			
C_{max}	*n_unlab*	*common*	*proportion*
30	2914	1263	0.4334
40	2493	1183	0.4745
50	2902	1363	0.4697

Case Study JM1_2863_20			
C_{max}	*n_unlab*	*common*	*proportion*
30	2382	1172	0.4920
40	2840	1389	0.4891
50	3197	1346	0.4210

error rates correspond to the p and $(1-p)$ values of a given data set.

When comparing the results to those of semisupervised clustering (Table IX), it is clear that random classification provided results that are not practical and useful to the software practitioner. The skewness or imbalance between the error rates leads to most modules being classified as nfp, lowering the Type I error rate but drastically increasing the Type II error rate. Such a software quality model cannot be employed in software engineering practice since it provides very limited guidance for utilizing resources allocated for software quality improvement.

C. Explorative Analysis of Remaining Unlabeled Modules

The program modules that remain unlabeled at the end of our semisupervised clustering scheme were further investigated for an insight into their characteristics. Among possible reasons as to why these modules remain unlabeled is the likelihood that they can be constituted as noisy instances in the JM1-8850 data set. As mentioned earlier, we determine the commonality of the remaining unlabeled modules with those detected as likely noise by two techniques for noise detection, i.e., EF and PANDA. We chose to compare with these two techniques because the former is geared toward detecting class noise or mislabeling errors, whereas the latter is geared toward detecting noise in the attributes.

We compare the software modules misclassified by a majority EF with the remaining unlabeled modules. Since our EF consists of 25 classifiers, a majority consists of 13 or more classifiers. We do not compare with the consensus EF (misclassified by all 25 classifiers) because it is considered a very conservative filter: very few modules are identified as noisy. It was found that 2820 program modules were misclassified by the majority EF.

The number of modules common between these 2820 modules and those that remain unlabeled after semisupervised clustering are shown in Table XII. For a given $C_{\max}$ value, the last column indicates the proportion of remaining unlabeled modules that are common with the 2820 modules flagged by the EF. This proportion amounts to an average of about 46% and 47% for case studies JM1_2863_10 and JM1_2863_20, respectively. Since almost half of the remaining unlabeled modules are common with modules that are likely noisy, we get an insight into their characteristics: many noisy modules were not labeled during our semisupervised clustering process.

TABLE XIII
COMMONALITY WITH TOP 50% OF PANDA RANKING

Case Study JM1_2863_10			
C_{max}	*n_unlab*	*common*	*proportion*
30	2914	1544	0.5299
40	2493	1378	0.5527
50	2902	1672	0.5762

Case Study JM1_2863_20			
C_{max}	*n_unlab*	*common*	*proportion*
30	2382	1333	0.5596
40	2840	1653	0.5820
50	3197	1706	0.5336

The PANDA noise detection scheme ranked the unlabeled JM1-8850 modules according to their likely noise factor. We compare the top 50% (4425) of the modules ranked by PANDA with the unlabeled modules remaining after semisupervised clustering. The comparative results are summarized in Table XIII. For case study JM1_2863_10, over 55% of the remaining unlabeled modules are in common with the top (most noisy) 4425 modules ranked by PANDA. A similar observation is made for case study JM1_2863_20. Since over half of the remaining unlabeled modules are common with the top 4425 ranked by PANDA, we once again get an insight into their characteristics: many of the remaining unlabeled modules are noisy.

Since both noise detection techniques provided valuable insight into characteristics of the remaining unlabeled modules, we determine the commonality among modules remaining unlabeled after the different semisupervised clustering runs. This would provide additional information as to "Are the same modules being filtered out by the semisupervised clustering process?" If a large number of modules are common among the different runs, then it is likely that those modules do not belong to the nfp and fp classes of modules for the JM1 system or that they are noisy instances in the JM1 software measurement data set.

The observed results are summarized in Table XIV for the two case studies. For a given case study, the table is shown as a matrix, where the diagonal indicates numbers of modules remaining unlabeled after semisupervised clustering with a given $C_{\max}$ value. For case study JM1_2863_10, there is a commonality of about 77% and 89% between $C_{\max} = 30$ and $C_{\max} = 40$ and $C_{\max} = 30$ and $C_{\max} = 50$, respectively. The respective commonality percentages are even higher for case study JM1_2863_20. The last column in the table shows the numbers of modules that are common among all three runs (for each $C_{\max}$ value) for each case study. The high commonality between the different sets of remaining unlabeled modules is of interest to the software practitioner. These program modules should be further investigated for their software measurements and possible reasons or problems as to why they remained unlabeled.

TABLE XIV
COMMONALITY AMONG REMAINING UNLABELED MODULES

Case Study JM1_2863_10				
C_{max}	30	40	50	All
30	2914	2241 (0.7690)	2584 (0.8868)	2214
40	—	2493	2330 (0.9346)	
50	—	—	2902	

Case Study JM1_2863_20				
C_{max}	30	40	50	All
30	2382	2029 (0.8518)	2148 (0.9018)	1838
40	—	2840	2562 (0.9021)	
50	—	—	3197	

D. Discussion on Proposed Algorithm

We have shown that the proposed semisupervised clustering scheme is useful for the software quality estimation problem when there is no knowledge of software fault-proneness data, i.e., predicting software quality for unlabeled program modules. The benefits have been shown in the context of real-world software systems. In this section, we analyze and provide insight into the pros and cons of the technique.

In related literature, semisupervised clustering schemes have primarily been applied for text categorization [31]: such studies generally model the text (or word) occurrence based on predetermined distributions, e.g., Gaussian. Applying a similar assumption in this paper would be inappropriate since the fault-proneness occurrence of program modules may not follow a given distribution. The software quality (software faults or fault-proneness labels) distribution among program modules often depends on the characteristic of the software system. Clustering the program modules based solely on their software metrics and without assumption on structure of the data is a more practical approach to the software quality estimation problem.

As stated earlier, the problem of software quality estimation is often aggravated by the absence of prior development experience with similar software systems. Under such conditions, the proposed algorithm is very practical since it does not depend on training a supervised classifier (as done by various semisupervised classification approaches) during the semisupervised learning and prediction process.

The proposed approach involves the domain expert at a considerable level; hence, its performance may likely be affected by the knowledge of the expert. To avoid a heavy dependence on a domain expert, the software quality analyst may employ a team of domain experts instead of one person. In addition, since the semisupervised clustering approach is carried out after the initial step of obtaining the labeled clusters, the algorithm performance may likely be affected by the initial step. To avoid any heavy bias due to this step, the initial labeled clusters are based on very high expert confidence.

VIII. Conclusion

The problem of software quality analysis and estimation is important during software development. However, building software quality models in the absence of defect data from previous experiences is a difficult task. This paper presented a novel approach to software quality analysis in the absence of defect data or quality-based class labels for program modules.

The proposed approach is a constraint-based semisupervised clustering scheme that uses a software engineering expert's domain knowledge to iteratively label clusters as either nfp or fp. The empirical study is performed using software measurement data obtained from multiple NASA software projects. It is shown that in comparison to unsupervised clustering with expert-based labeling, the semisupervised clustering scheme improves the expert's classification of unlabeled program modules. In addition, the knowledge learnt during semisupervised clustering generally yielded better classification of unseen program modules as compared to both unsupervised clustering and random classification.

An explorative analysis of program modules that remain unlabeled after the different semisupervised clustering runs indicated that about half of them were likely noisy instances in the given software measurement data set. Moreover, a high consistency (commonality) was observed among the different sets of modules that remain unlabeled. It would be interesting to learn why these modules remain unlabeled as compared to those that were labeled. Some likely areas of possible investigation may include errors during software metrics data collection; absence of certain metrics that could improve the captured software quality knowledge; problems with data collection tools; and issues inherent to specific software processes adopted during development. However, we could not investigate these areas due to nonaccess of other information on the respective software projects.

Some useful directions for future research may include the following: investigating semisupervised classification schemes to facilitate minimal amount of expert involvement; analyzing other software systems using the proposed semisupervised clustering scheme; implementing the semisupervised clustering scheme with other underlying clustering algorithms, such as self-organizing maps and Neural-Gas; and an in-depth study of the unlabeled program modules from a software engineering point of view.

Acknowledgment

The authors would like to thank the Associate Editor, H. Pham, and the anonymous reviewers for the constructive remarks and suggestions; L. Mayron and A. Srinivasan for their assistance with editorial reviews of the manuscript; S. Zhong for his suggestions and reviews of this paper; and the staff of the NASA MDP for making the software measurement data available.

References

[1] S. Basu, A. Banerjee, and R. Mooney, "Semi-supervised clustering by seeding," in *Proc. 19th Int. Conf. Mach. Learn.*, Sydney, Australia, Jul. 2002, pp. 19–26.

[2] A. Blum and T. Mitchell, "Combining labeled and unlabeled data with co-training," in *Proc. 11th Annu. ACM Conf. Comput. Learn. Theory*, P. Bartlett and Y. Mansour, Eds., Madison, WI, Jul. 1998, pp. 92–100.

[3] C. E. Brodley and M. A. Friedl, "Identifying mislabeled training data," *J. Artif. Intell. Res.*, vol. 11, pp. 131–167, 1999.

[4] A. Dong and B. Bhanu, "A new semi-supervised EM algorithm for image retrieval," in *Proc. IEEE Int. Conf. Comput. Vis. Pattern Recog.*, Madison, MI, Jun. 2003, pp. 662–667.

[5] N. E. Fenton and S. L. Pfleeger, *Software Metrics: A Rigorous and Practical Approach*, 2nd ed. Boston, MA: PWS-Kent, 1997.

[6] G. Fung and O. Mangasarian, “Semi-supervised support vector machines for unlabeled data classification,” *Optim. Methods Softw.*, vol. 15, no. 1, pp. 29–44, Apr. 2001.

[7] Z. Ghahramani and M. I. Jordan, “Supervised learning from incomplete data via an EM approach,” in *Advances in Neural Information Processing Systems*, vol. 6, J. D. Cowan, G. Tesauro, and J. Alspector, Eds. San Francisco, CA: Morgan Kaufmann, 1994, pp. 120–127.

[8] S. Goldman and Y. Zhou, “Enhancing supervised learning with unlabeled data,” in *Proc. 17th Int. Conf. Mach. Learn.*, Jun. 2000, pp. 327–334.

[9] A. R. Gray and S. G. MacDonell, “Software metrics data analysis: Exploring the relative performance of some commonly used modeling techniques,” *Empir. Softw. Eng. J.*, vol. 4, no. 4, pp. 297–316, Dec. 1999.

[10] L. Guo, B. Cukic, and H. Singh, “Predicting fault prone modules by the Dempster–Shafer belief networks,” in *Proc. 18th Int. Conf. Automated Softw. Eng.*, Montreal, QC, Canada, Oct. 2003, pp. 249–252.

[11] K. E. Imam, S. Benlarbi, N. Goel, and S. N. Rai, “Comparing case-based reasoning classifiers for predicting high-risk software components,” *J. Syst. Softw.*, vol. 55, no. 3, pp. 301–320, Jan. 2001.

[12] T. M. Khoshgoftaar and N. Seliya, “Comparative assessment of software quality classification techniques: An empirical case study,” *Empir. Softw. Eng. J.*, vol. 9, no. 3, pp. 229–257, Sep. 2004.

[13] ——, “Analogy-based practical classification rules for software quality estimation,” *Empir. Softw. Eng. J.*, vol. 8, no. 4, pp. 325–350, Dec. 2003.

[14] T. M. Khoshgoftaar, S. Zhong, and V. Joshi, “Noise elimination with ensemble-classifier filtering for software quality estimation,” *Intell. Data Anal.*, vol. 9, no. 1, pp. 3–27, 2005.

[15] W. J. Krzanowski and Y. T. Lai, “A criterion for determining the number of groups in a data set using sums-of-squares clustering,” *Biometrics*, vol. 44, no. 1, pp. 23–34, Mar. 1988.

[16] T. M. Martinez, S. G. Berkovich, and K. J. Schulten, “Neural-Gas: Network for vector quantization and its application to time-series prediction,” *IEEE Trans. Neural Netw.*, vol. 4, no. 4, pp. 558–569, Jul. 1993.

[17] A. K. McCallum and K. Nigam, “Employing EM and pool-based active learning for text classification,” in *Proc. 15th Int. Conf. Mach. Learn.*, Madison, WI, Jul. 1998, pp. 350–358.

[18] K. Nigam, A. K. McCallum, S. Thrun, and T. Mitchell, “Text classification from labeled and unlabeled documents using EM,” *Mach. Learn.*, vol. 39, no. 2/3, pp. 103–134, May/Jun. 2000.

[19] ——, “Learning to classify text from labeled and unlabeled documents,” in *Proc. 15th Conf. Amer. Assoc. Artif. Intell.*, Madison, WI, Jul. 1998, pp. 792–799.

[20] W. Pedrycz, G. Succi, M. Reformat, P. Musilek, and X. Bai, “Self organizing maps as a tool for software analysis,” in *Proc. Can. Conf. Elect. and Comput. Eng.*, Toronto, ON, Canada, May 2001, vol. 1, pp. 93–97.

[21] W. Pedrycz and J. Waletzky, “Fuzzy clustering with partial supervision,” *IEEE Trans. Syst., Man, Cybern.*, vol. 5, no. 27, pp. 787–795, Oct. 1997.

[22] ——, “Fuzzy clustering in software reusability,” *Softw. Pract. Exp.*, vol. 27, no. 3, pp. 245–270, Mar. 1997.

[23] M. Reformat, W. Pedrycz, and N. J. Pizzi, “Software quality analysis with the use of computational intelligence,” in *Proc. IEEE Int. Conf. Fuzzy Syst.*, Honolulu, HI, May 2002, vol. 2, pp. 1156–1161.

[24] P. Runeson, M. C. Ohlsson, and C. Wohlin, “A classification scheme for studies of fault-prone components,” in *Proc. 3rd Int. Conf. Product Focused Softw. Process Improvement*, 2001, vol. 2188, pp. 341–355.

[25] N. F. Schneidewind, “Investigation of logistic regression as a discriminant of software quality,” in *Proc. 7th Int. Softw. Metrics Symp.*, London, U.K., Apr. 2001, pp. 328–337.

[26] M. Seeger, “Learning with labeled and unlabeled data,” Inst. Adaptive Neural Comput., Univ. Edinburgh, Edinburgh, U.K., Feb. 2001.

[27] J. Van Hulse, T. M. Khoshgoftaar, and H. Huang, “The pairwise attribute noise detection algorithm,” *Knowl. Inf. Syst. J.*, to be published.

[28] K. Wagstaff and C. Cardie, “Clustering with instance-level constraint,” in *Proc. 17th Int. Conf. Mach. Learn.*, Jun. 2000, pp. 1103–1110.

[29] I. H. Whitten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques With JAVA Implementations*. San Francisco, CA: Morgan Kaufmann, 2000.

[30] Y. Wu and T. S. Huang, “Self-supervised learning for visual tracking and recognition of human hand,” in *Proc. 17th Nat. Conf. Artif. Intell.*, Jul. 2000, pp. 243–248.

[31] H. Zeng, X. Wang, Z. Chen, H. Lu, and W. Ma, “CBC: Clustering based text classification using minimal labeled data,” in *Proc. IEEE Int. Conf. Data Mining*, Melbourne, FL, 2003, pp. 443–450.

[32] S. Zhong, T. M. Khoshgoftaar, and N. Seliya, “Analyzing software measurement data with clustering techniques,” *IEEE Intell. Syst.*, vol. 19, no. 2, pp. 20–27, Mar./Apr. 2004.

Naeem Seliya (M’03) received the Ph.D. degree in computer engineering from Florida Atlantic University, Boca Raton, in 2005.

He is currently an Assistant Professor of computer and information science with the Department of Computer and Information Science, University of Michigan–Dearborn. His research interests include software engineering, data mining and machine learning, software measurement, software reliability and quality engineering, software architecture, computer data security, and network intrusion detection.

Dr. Seliya is a member of the Association for Computing Machinery.

Taghi M. Khoshgoftaar (M’86) is a Professor with the Department of Computer Science and Engineering, Florida Atlantic University, Boca Raton, and the Director of the Empirical Software Engineering and Data Mining and Machine Learning Laboratories. His research interests include 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 has served as North American Editor of the *Software Quality Journal* and serves on the editorial boards of the journals *Software Quality* and *Fuzzy Systems*.

Dr. Khoshgoftaar is a Member of the IEEE Computer Society and the 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. He has served on technical program committees of various international conferences, symposia, and workshops.