Knowl Inf Syst (2007) 11(2): 171–190
DOI 10.1007/s10115-006-0022-x

Knowledge and Information Systems

REGULAR PAPER

Jason D. Van Hulse · Taghi M. Khoshgoftaar · Haiying Huang

The pairwise attribute noise detection algorithm

Received: 26 April 2005 / Revised: 19 September 2005 / Accepted: 14 January 2006 /
Published online: 8 April 2006

Abstract Analyzing the quality of data prior to constructing data mining models is emerging as an important issue. Algorithms for identifying noise in a given data set can provide a good measure of data quality. Considerable attention has been devoted to detecting class noise or labeling errors. In contrast, limited research work has been devoted to detecting instances with attribute noise, in part due to the difficulty of the problem. We present a novel approach for detecting instances with attribute noise and demonstrate its usefulness with case studies using two different real-world software measurement data sets. Our approach, called Pairwise Attribute Noise Detection Algorithm (PANDA), is compared with a nearest neighbor, distance-based outlier detection technique (denoted DM) investigated in related literature. Since what constitutes noise is domain specific, our case studies uses a software engineering expert to inspect the instances identified by the two approaches to determine whether they actually contain noise. It is shown that PANDA provides better noise detection performance than the DM algorithm.

Keywords Data quality · Noise detection · Data cleaning · PANDA

1 Introduction

Data quality is a critical issue whenever data mining techniques are applied to real-world data sets. Organizations have realized the value of information and now consider data as an asset of the company [2]. Important decisions are made every day based on the data stored within a company's databases, from understanding the behaviors of profitable customers to determining the likelihood of future loss on new business contracts. However, the decisions made are only as good as the

J. D. Van Hulse · T. M. Khoshgoftaar (✉) · H. Huang
Empirical Software Engineering Laboratory, Department of Computer Science and Engineering, Florida Atlantic University, Boca Raton, FL 33431, USA
E-mail: taghi@cse.fau.edu

data upon which they are based. High-quality data is defined as data that is fit for use by data consumers [26]. Characteristics of high-quality data belong to four categories: Intrinsic, Accessibility, Contextual, and Representational.

The most common type of problem occurs in the Intrinsic category, which relates to accuracy and objectivity of the data source. Quality issues in the Accessibility category are related to the ability to retrieve the necessary data in a timely and efficient manner. Contextual problems include missing, incomplete, or improperly defined data. The Representational category includes issues related to interpretability, ease of understanding, and the concise and consistent representation of data. Data quality issues are often multifaceted and complex, and it is crucial for information management departments to build applications that support the goal of achieving high-quality data within an organization. Errors in data can often be found when multiple data sources are merged [10, 11]. Records processed within different systems may have different data formats or representations [7]. When such databases are merged, two records referring to the same entity may not match, resulting in duplicate entries or missing data. An important area of data quality research involves detecting duplicate or near duplicate records in a database.

Data mining techniques are often used to detect interesting relationships within very large databases that would be difficult for humans to discover otherwise. The knowledge derived from these algorithms is based on the data within the database being analyzed, and low-quality data can obscure important patterns that may exist. In a classification problem, for example, it has been shown that the presence of errors in the training data set does indeed lower the predictive accuracy of a learner on test data [13, 15, 29, 30]. In the machine learning context, noise can dramatically slow the convergence of a learning algorithm, or even cause it to converge to a suboptimal set of hypotheses [5].

While a description of the complete taxonomy of data noise is an open research issue, there are generally two types of noise in a given data set [3]: class noise and attribute noise. Class noise or labeling errors occur when an instance belongs to the incorrect class. Class noise can be attributed to several causes, including subjectivity during the labeling process, data entry errors, or the absence of some representative attributes. In contrast, attribute noise reflects erroneous values for one or more attributes (independent variables) of the data set. The complexity of detecting class noise is relatively lower than that of detecting attribute noise. Subsequently, very limited attention has been given to attribute noise. Our study focuses on detecting instances with attribute noise.

Various methods have been investigated for coping with data quality issues. Generally, there are three types of such methods: robust learners, data polishing, and noise filtering. Learners are considered robust if they are less likely to be influenced by the presence of noise in data. A good example of a robust learner is the C4.5 decision tree algorithm [22]. The C4.5 learner employs pruning strategies to remove statistically insignificant portions of the tree to form the final model. However, if the noise level is relatively high, even a robust learner may have poor performance. Many learners are sensitive to data noise and require data preprocessing to address the problem. Noise filtering identifies noisy instances which can be eliminated from the training data [3, 9, 17, 30]. Finally, data polishing corrects the noisy instances prior to training a learner [27]. Though considerably tedious, such an approach is viable when the data set is small.

We present a novel noise ranking and filtering algorithm for detecting instances with noise in one or more attributes. Abbreviated as PANDA for *Pairwise Attribute Noise Detection Algorithm*, the proposed approach can be used with any number of (numeric or binary) attributes, including the class attribute (dependent variable). PANDA is evaluated in case studies using two different software measurement data sets, and is compared with a nearest neighbor, distance-based outlier detection technique (abbreviated DM) presented in related literature [23].

In addition to presenting a novel algorithm for detecting instances with attribute noise, this study is unique because it employs a software engineering expert[1] to validate the instances identified by the two techniques. To our knowledge, this is the first study to present an algorithm for detecting instances with attribute noise without using the class label.

The definition of a noisy instance is rather domain specific, and a given noise detection algorithm may not realize domain-specific exceptions and other allowable instances. An instance identified as noisy by a given technique may not actually be noise for the given domain. Therefore, it is important to note the distinctions between outliers, noise, and exceptions. An observation is called an outlier if it appears to be inconsistent with the remainder of a set of data, or which deviates so much from other observations so as to arouse suspicions that it was generated by a different mechanism [25]. In [20], outliers are defined as ‘objects deviating from the major distribution of the data set’.

In our study, outliers are more specifically defined as instances with attribute values that reside on the extremes of the general distributions observed for the given attributes in the data set. In contrast, instances may be considered noisy when the values of one or more attributes of an instance are corrupted or incorrect relative to the values of the other attributes. However, an instance without errors may appear noisy when one or more of its attributes does not follow the general distributions observed for the given attributes. These exceptions therefore often appear as noisy instances. Such instances are very difficult to detect without the input of a domain expert. In our study, the software engineering expert plays an important role in categorizing the instances detected by PANDA as either exceptions or instances with attribute noise.

The remainder of this paper is organized as follows. Section 2 discusses some related work in the noise detection domain. Section 3 describes the noise detection algorithm proposed in this paper. Section 4 describes the software measurement data sets used in the case studies, while Sect. 5 discusses the obtained results. We conclude in Sect. 6 and highlight some future research directions.

2 Related work

As mentioned earlier, noise can occur in either the dependent variable (class noise) or independent variables (attribute noise). Related literature on noise detection has primarily concentrated on the class noise problem [30]. Various techniques

[1] The expert is not involved in the modeling process and only analyzes the results of PANDA from a domain-knowledge point of view.

have been developed to identify instances with class noise with a high degree of accuracy [16]. In contrast, very few methods are available for detecting instances with attribute noise, largely due to the high complexity of the problem. In this section, some related works on noise detection are briefly discussed including association rules, noise filtering, and data polishing. A discussion on some outlier detection techniques is presented in Sect. 3.2, including the DM algorithm [23] which is considered in our comparative study.

Ordinal association rules are introduced and defined in [20] for the detection of data errors. Data errors are identified when ordering relations between attributes prevalent in the remainder of the data set do not hold for the observation in question. For example, an observation with date of birth greater than date of death would be declared as noise. This is a useful tool, but is rather limited in scope.

The majority of the algorithms for noise detection utilize classifiers for the identification or correction of noisy instances, and suffer from the limitation that they cannot be applied to a data set without the class variable. The assumption usually made is that an observation that is difficult to classify correctly is likely to be noise. Of course, it may also be the case that the instance is part of a target concept that is difficult to learn. For the most part, these algorithms do not treat any observations that are predicted correctly. It is not necessarily true, however, that these observations do not contain noise.

In [3], multiple classifiers are aggregated to create an ensemble filter. An instance is tagged as noise if a given number of classifiers are unable to correctly classify the instance. A majority filter is defined as a system which categorizes an instance as noise if out of the five classifiers built, three or more fail to correctly classify that instance. In a recent study, we investigated a very large ensemble filter (25 classifiers) and partitioning filters for noise filtering of software measurement data [14, 17].

In [8, 9], inductive learning algorithms are used to filter noisy instances. A target theory is built using a rule induction algorithm such as $CN2$ [4] to correctly classify all instances in the data set. Instances that maximize the reduction in the Minimum Description Length (MDL) cost of encoding the target theory are removed, until the cost cannot be reduced any further.

Data polishing is introduced as a two-step procedure to both detect and correct noise [27]. Given a data set which contains N independent variables and 1 dependent variable, a total of N modified classifiers are constructed, where each independent variable is switched in turn with the dependent variable. For instances that are misclassified by the base classifier T, the number of attribute misclassifications over the N modified classifiers are recorded. In the polishing phase, using the predictions for each attribute from the N classifiers, the predicted attribute values are inserted into the instance. If the base classifier T is able to correctly predict the class of this modified instance, then the polishing procedure is complete and a noisy instance has been cleansed. The complexity of this algorithm is high and increases significantly as the number of independent variables increases.

It is argued that data polishing is best used when the independent variables are highly correlated to the class [28]. Yang et al. illustrate an algorithm that shows better results for independent variables that cannot be predicted by using the class and the other independent variables (so-called predictive-but-unpredictable attributes). A set of rules predicting the class variable are learned from the data. If an instance fails to satisfy at least one rule, it is identified as noise. The instance can then be corrected by finding a rule that minimizes the number of changes necessary to make the instance consistent with the rule set.

The data polishing algorithms investigated in [27, 28] require all independent variables to be non-numeric. In addition, all of these techniques operate in a supervised learning environment, disqualifying their use when the class label is unavailable. In contrast, the noise detection algorithm proposed in this paper can be used with or without the class variable. If the class variable is available, it can be treated as another attribute. The advantages of the proposed technique include:

1. Erroneous observations can be examined prior to collecting data for the dependent variable in situations where the class label data collection process is expensive. Collecting data for the class label can be a very difficult and tedious task in some domains.
2. It can be applied to the test data set (with unknown class labels) as well as the training data set. This approach may be useful in real-world applications as a data quality control filter before information is run through the final classifiers. Observations that do not conform to the general structure of the data set can be presented to the domain-specific practitioner for inspection and validation before being processed through the classifier. This process is independent of training the classifier.

In several of the works discussed previously, performance of the noise detection technique was evaluated by injecting simulated noise into the data sets, which were primarily obtained from the UCI Repository of machine learning databases [21]. There are two important problems with this evaluation approach. First, it is unknown whether the underlying data set into which simulated noise is injected is noise-free or clean. Injecting additional noise into a data set that may already contain noise makes the analysis of the effectiveness of the algorithms difficult.

Second, the injected noise may not represent the types of noise present in a real-world data set of the given domain. Obtaining a completely clean data set prior to noise injection is a difficult task and the injection of noise into such a data set to simulate real-world conditions is an open research issue. In our case studies, we consider real-world data sets which inherently contain noisy instances (i.e., we do not inject noise into the data set). A domain expert is needed to inspect and identify instances that are actually noise. The noise detection techniques can then be evaluated with respect to their performance in detecting those instances.

3 Methodology

The proposed noise detection algorithm, PANDA, seeks to identify those instances with a large deviation from normal given the values of a pair of attributes. When a set of instances have similar values for one attribute, large deviations from normal for the second attribute may be considered suspicious. The larger the deviation, the stronger the evidence that the instance is noisy. One has to be careful in interpreting the aforementioned assumption because the variance and distribution of the two attributes may be different. In our study, we address this issue by standardizing all the attributes in the given data set. The output of PANDA is a list of instances ordered from most noisy to least noisy. Each instance is assigned an output score (Noise Factor), which is used to rank the instance relative to the other instances in the data set. After obtaining a noise ranking, some of the instances may be discarded from the data set, which would result in a cleaner data set with which to perform additional analysis. Removing noisy instances, however, is not required and the user can apply any type of domain-specific treatment using the noise ranking produced by PANDA.

Each ordered pair of attributes is considered. After the first attribute is partitioned, 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 relative to the instances with the same partitioned value for the first attribute have a higher contribution to the output score. Based on these calculations, the value of each attribute is replaced by the standardized value. This process is repeated for each combination of ordered attribute pairs, and the results are aggregated to an output score representing the amount of noise present in the instance.

It should be noted that the partitioning process may introduce additional noise into the data set, which should be considered during the respective analysis. As discussed in the case studies, we partition the attributes using several bin sizes and select the median ranking. Selecting the median ranking among multiple bin sizes helps minimize the possibility of noise being introduced due to the discretization process.

The details of PANDA are presented in Sect. 3.1. In our study, we compare the noise detection performance of PANDA with a nearest neighbor distance-based outlier detection algorithm (DM) [23]. The details of the DM algorithm are presented in Sect. 3.2.

3.1 Noise detection algorithm

The detailed procedure for PANDA is shown in Fig. 1. The parameters s_{ikj} are initialized to 0 on line 1. The binning algorithm implemented in SAS [24] is used to partition each attribute x_{*j} in the data set (line 3). The partitioned attribute $\hat{x}_{*j}$ is integer-valued, $\hat{x}_{*j} \in \{0, \ldots, L-1\}$. L is determined by the user and is the number of partitions of attribute $\hat{x}_{*j}$. SAS uses an equal frequency binning algorithm, where, in the absence of tied values for the attribute, each partition will have the same number of instances. An exception is made for instances with the same value for the attribute at the boundary of the partition. As partitions are required to be disjoint, all instances with the same attribute value are placed into the same partition. While other binning procedures exist, we chose the SAS binning algorithm for its simplicity. PANDA can be implemented with other partitioning algorithms as well, and an investigation related to the same will be considered in our future work.

After partitioning observations into the given number of disjoint bins, the mean and standard deviation of the non-partitioned attributes x_{*k}, $k \neq j$, relative to each bin $\hat{x}_{*j} = 0, \ldots, L-1$ is calculated (line 6). The standardized value for attribute value x_{ik} relative to the partitioned attribute value for instance i, $\hat{x}_{ij}$, is calculated in line 8. The standardized value is calculated by subtracting the mean value of attribute x_{*k} relative to the partitioned attribute value $\hat{x}_{ij}$ from the attribute value for instance i, x_{ik}. The absolute value of this quantity (as the sign of the difference

```
Procedure: Calculate Noise Factor(string FUNC, dataset X)
input: Dataset $X = [x_{ij}]_{n \times m}$ with $n$ observations and $m$ attributes, where $x_{ij}$ is the value of
the $j^{th}$ attribute for the $i^{th}$ observation. $x_{*j}$ denotes the $j^{th}$ attribute.
output: Noise Factor $S_i$ for instance $i$, $1 \leq i \leq n$.
 1. $s_{ikj} = 0 \;\; \forall\, i = 1, \ldots, n$ and $j, k = 1, \ldots, m$ //initialize values
 2. for $j = 1$ to $m$
 3.   Transform attribute $x_{*j}$ into the partitioned attribute $\hat{x}_{*j} \in \{0, \ldots, L-1\}$ where $L = \#$
      of bins
 4.   for $k = 1$ to $m$
 5.     if $k \neq j$
 6.       calculate $Mean(x_{*k} \mid \hat{x}_{*j} = l)$ and $Std(x_{*k} \mid \hat{x}_{*j} = l)$ for $l = 0, \ldots, L-1$
 7.       for $i = 1$ to $n$
 8.         $s_{ikj} = |x_{ik} - Mean(x_{*k} \mid \hat{x}_{*j} = \hat{x}_{ij})| \;/\; Std(x_{*k} \mid \hat{x}_{*j} = \hat{x}_{ij})$
 9.         //$s_{ikj}$ is the standardized value of the attribute value $x_{ik}$ for the $i^{th}$ observation
              given the partitioned attribute value $\hat{x}_{ij}$
10.       endfor
11.     endif
12.   endfor
13. endfor
14. if FUNC='SUM'
15.     $S_i = \sum_{k=1}^{m} \sum_{j=1}^{m} s_{ikj}$
16. endif
17. else if FUNC ='MAX'
18.     $S_i = \max_{\{j=1,\ldots,m\}\{k=1,\ldots,m\}} \left\{ s_{ikj} \right\}$
19. endif
20. return $S_i$
```

Fig. 1 Pairwise attribute noise detection algorithm: main procedure

is not important) is divided by the standard deviation of attribute x_{*k} relative to the value of the partitioned attribute $\hat{x}_{*j}$ for instance i (i.e., $\hat{x}_{ij}$). This procedure is repeated for all pairs of attributes $(x_{*k}, \hat{x}_{*j})$, $k \neq j$. Finally, the SUM (line 15) or MAX (lines 18) measures for each observation are calculated.

The instances with relatively large values of MAX or SUM may be considered as potential noise because an observation with a large standardized attribute value indicates a deviation from normal for that attribute. Based on the noise ranking calculated by PANDA, noisy instances may be removed from the data set, although this process is not required. In our case studies, we instead present the noise ranking produced by PANDA to a domain expert for evaluation. In *direct* elimination (Fig. 3), all the desired observations will be removed from the output data set after a single iteration of PANDA. In *iterative* elimination (Fig. 2), the user can specify the number of iterations to run the algorithm, eliminating a portion of the total noise in each pass. Iterative elimination has the potential advantage that noise detected prior iterations will not affect the ranking that results from the current iteration, although our experiments demonstrated similar results for both methods.

The run-time complexity of our technique can be analyzed as follows. Suppose m is the number of attributes in the data set and n is the number of instances. Computing the mean and standard deviation for each pair of attributes in the data set and standardizing each attribute value has complexity $O(m^2 n)$. In many real-world data sets, the number of attributes is much smaller than the number of instances. Additionally, as PANDA is part of the preprocessing tasks of a data mining project, the calculations can be performed offline in a batch environment. Therefore, we can conclude that the running time of PANDA is not a significant concern, especially for small- and medium-sized data sets.

```
Procedure: Iter(int $numIterations$, int $numErrorsPerIter$, string FUNC)
input: desired number of iterations, number of errors per iteration, desired function
output: Dataset $X'$ with noisy instances eliminated
1. $X' \leftarrow X$
2. do $k = 1$ to $numIterations$
3.    call Calculate Noise Factor(FUNC, $X'$)
4.    Create the set of instances {errors} by taking the $N$ instances with the largest value for
      $S_i$, where $N = numErrorsPerIter$. In particular, $|\{errors\}| = numErrorsPerIter$.
5.    $X' \leftarrow X' \setminus \{errors\}$
6. end
7. return $X'$
```

Fig. 2 PANDA—iterative elimination

```
Procedure: Direct(int $numErrors$, string FUNC)
input: Dataset $X$, desired number of errors, desired function
output: Dataset $X'$ with noisy instances eliminated
1. call Calculate Noise Factor(FUNC, $X$)
2. Create the set of instances {errors} by taking the $N$ instances with the largest value for $S_i$,
   where $N = numErrors$. $|\{errors\}| = numErrors$.
3. $X' \leftarrow X \setminus \{errors\}$
4. return $X'$
```

Fig. 3 PANDA—direct elimination

3.2 Comparing with a distance-based algorithm

Outlier detection techniques were first developed within a statistical framework. For example, given the data is sampled from an underlying normal distribution, any observation more than 3 standard deviations from the mean might be considered an outlier. A major difficulty with applying this theory to real-world data sets is that the distribution function is unknown, and is usually very difficult to determine. The work introduced in [18] provides a generalized definition of an outlier, of which the criteria mentioned earlier is a special case. An observation x in a data set D is an outlier if at least a fraction p of the objects in D are further than d away from x.

Algorithms are provided in [19] for mining these types of outliers. In [23], the authors find outliers utilizing a modified distance-based metric to cope with difficulties in determining the optimal value for d. $D^k(p)$ is defined as the distance from point p to its kth nearest neighbor. The α observations with the largest values for $D^k(p)$ for a given value of k are considered outliers. We denote this algorithm as DM. The basic algorithm implementing DM has complexity $O(n^2)$, where n is the number of instances in the data set. The authors introduce a partition-based algorithm, which obtains an order of magnitude reduction in running time given a wide range of parameters used in experiments on simulated data.

The DM algorithm is considered for comparison because like PANDA, it can be used either with or without the class variable and can handle numeric attributes. DM and PANDA also have comparable complexity. Although DM is specifically designed for outlier detection, no other noise detection algorithm (excluding class noise detection) has been proposed that can be used for this comparison. One difficulty with distance-based techniques, however, is that they rely heavily on some distance function, which looses effectiveness as the number of dimensions of the feature space increases [1].

Table 1 Software metrics for the JM1-2445 data set

Metric	Symbol
Branch	Branch count
Line count	Executable LOC
	Comments LOC
	Blank LOC
	Code and comments LOC
	Total lines of code
Basic Halstead	Total operators
	Total operands
	Unique operands
	Unique operators
McCabe's	Cyclomatic complexity
	Essential complexity
	Design complexity

4 Description of the case study data sets

4.1 JM1-2445 data set

The software measurement data set JM1-2445 was derived from a NASA software project written in C. Software metrics data was collected at the function level; hence, a program module (instance) was defined as a function or subroutine. Upon removing inconsistent instances (identical values for the software attributes but different class labels), the data set was reduced from 10,883 to 8850 observations, resulting in a data set called JM1-8850 [17].

JM1-8850 contains 21 software metrics (attributes) including McCabe Metrics, Halstead Metrics (both derived and basic), Line Count and Branch Count Metrics. See [6] for a discussion on types, characteristics, and use of software metrics. The 8 derived Halstead metrics were not used in our study, leaving 13 software metrics remaining. These 13 software product metrics are shown in Table 1, where LOC represents the lines of code in a program module.

The JM1-8850 data set also contains a dependent variable, Class, with values nfp (not fault prone) and fp (fault prone). An instance was labeled fp if it contained at least one fault and nfp otherwise. The JM1-2445 data set used in our case study was constructed from JM1-8850 using clustering techniques and expert input.

The instances in JM1-8850 were partitioned into clusters using an unsupervised clustering technique (k-means) as part of a related study [29]. Descriptive statistics such as the mean and standard deviation were computed for both the entire data set and each cluster. The software engineering expert then labeled (nfp and fp) those clusters (and consequently, instances within the clusters) for which he was completely confident. The expert-assigned labels of instances in those clusters were then matched with their actual labels, and instances that did not have matching labels were removed from the clusters. These instances were inspected by the expert, and it was verified that their class labels did not match the class labels the instances should have had based on their attribute values. As a result, 2445 instances with no class noise remained in the clusters labeled by the expert.

Table 2 Software metrics for the CCCS data set

Symbol	Abbreviation
Logical operators	logop
Total lines of code	tloc
Executable LOC	eloc
Unique operands	uopan
Total operands	topan
Unique operators	uoper
Total operators	toper
Cyclomatic complexity	ccomp

We denote this data set as JM1-2445, which consists of 2210 nfp instances and 235 fp instances. JM1-2445 is a real-world data set that was preprocessed (using clustering and expert input) to remove any instances with class noise. Instances with naturally occurring attribute noise, however, remain in the JM1-2445 data set.

4.2 CCCS data set

The CCCS data set is a large military command, control, and communications system written in Ada [12]. CCCS contains 282 instances, where each instance is an Ada package consisting of one or more procedures. CCCS contains 8 independent variables or attributes along with an additional attribute *nfaults* indicating the number of faults attributed to the module during the system integration and test phases and during the first year of deployment. The software metrics in the CCCS data set are listed in Table 2. One hundred and forty-six of the 282 modules in the CCCS data set contain no faults.

5 Empirical results

5.1 Case Study 1: JM1-2445 data set

As mentioned earlier, the outlier detection algorithm (DM) derived in [23] was compared to PANDA. $D^k(p)$ is defined as the distance from point p to its kth nearest neighbor. The instances with the largest values for $D^k(p)$ for a given value of k are considered outliers. In DM, the number of nearest neighbors, k, is a parameter that is set by the user. In this case study, all variables or attributes were standardized, and distance was calculated using the L^2 (Euclidean) norm. The DM algorithm was executed 10 times, with a different value for k from 1 to 10 at each iteration. The final ranking was determined as the median ranking of the 10 iterations to minimize any potential bias when a single value for k is used.

In order to remove any anomalies that might occur due to binning (such as introducing additional noise or using a lucky/unlucky binning size), PANDA was executed 11 times using the SUM method, with 5, 10, 15, 20, 25, 30, 40, 50, 60, 75, and 100 bins. Similar to the procedure adopted for the DM algorithm, the final ranking by PANDA was determined as the median ranking over the 11 iterations (i.e., 11 different number of bins). The instances in JM1-2445 were then rank ordered independently by both algorithms from most noisy to least noisy. The instance ordering was then presented to an expert with more than 15 years of expertise in the software engineering domain. The domain expert manually inspected the rankings produced by DM and PANDA. It should be noted that the expert only analyzes the results obtained from PANDA and DM and does not influence the obtained results.

The results of the expert inspections for the top 250 instances (ranked by both algorithms) are presented in Table 3. These 250 instances are approximately 10% of the instances in the JM1-2445 data set and represent the mostly likely data noise. Based on the software engineering expert, 10% of the instances was enough to determine the noise detection ability of the two algorithms. For another data set, a different percentage of instances may represent the most likely noise.

The expert inspected the top 250 instances of JM1-2445 ranked by the two algorithms and identified four types of instances: *outlier*, *noise*, *exception*, and *typical*. PANDA does not differentiate between these different types of instances but simply ranks them from most to least noisy. During post-analysis, the domain expert is the one who categorizes the top 250 instances into the four groups for the sole purpose of evaluating the effectiveness of PANDA and DM.

An *outlier* is defined as an instance having attributes with values that reside within the tail of the general distributions of those attributes, and is a rare occurrence in a data set. An instance is considered as *noise* when one or more of its attributes have incorrect or corrupted values. An *exception* is an instance having attributes with values that do not follow the general distributions of those attributes but can occur (albeit rarely) with a given data set. Without input from a domain expert, detecting exceptions is very difficult because they are likely to be flagged as noise. A *typical* or *normal* instance is one with attribute values that occur within the general distributions of the given data set. It should be noted that from a noise detection point of view, there is no inspection benefit in presenting typical instances to a domain expert for review.

The number of instances classified into one of the four instance types as determined by the expert after manual inspection are shown in Table 3. The results are broken down into interval ranges for each algorithm. The first group represents the 25 highest ranking instances for both PANDA and DM, respectively. For this group, the expert determined that there were 13 errors, 7 exceptions, and 5 outliers detected by PANDA. The DM algorithm detected 16 errors, 7 outliers, 1 exception, and 1 typical instance among the 25 most suspicious instances. Also included are the cumulative results for observations 1–125 and 126–250 for both algorithms.

The top 5% (rank 1–125) suspicious instances identified by PANDA included 108 errors, 10 exceptions, 7 outliers, and no typical instances, while DM detected 101 errors, 5 exceptions, 16 outliers, and 3 typical instances. For the top 5% of instances, the DM algorithm detects a few normal instances, while PANDA does not detect any normal instances. If the top 10% suspicious instances identified by both algorithms are compared, PANDA's detection included 220 errors, 11 exceptions, 12 outliers, and 7 typical instances, while DM's detection included 173 errors, 10 exceptions, 17 outliers, and 50 typical instances. We observe that the

Table 3 Expert evaluation of the 10% most suspicious instances, JM1-2445 data set

Instance category	1–25		26–50		51–75		76–125		**1–125**		126–150		151–175		176–200		201–250		**126–250**	
	PANDA	DM	PANDA	DM	PANDA	DM	PANDA	DM	PANDA	DM	PANDA	DM	PANDA	DM	PANDA	DM	PANDA	DM	PANDA	DM
Noise	13	16	21	17	25	23	49	45	108	101	23	14	23	15	23	11	43	32	112	72
Outliers	5	7	2	7	0	2	0	0	7	16	0	0	2	0	2	0	1	1	5	1
Exceptions	7	1	2	1	0	0	1	3	10	5	1	3	0	0	0	0	0	2	1	5
Typical	0	1	0	0	0	0	0	2	0	3	1	8	0	10	0	14	6	15	7	47

DM algorithm misclassifies many typical instances as noisy when compared to PANDA.

The expert also inspected some of the instances that were ranked at the other extreme by both PANDA and DM, i.e., the least suspicious instances in the data set. It would be expected that these instances are not of practical interest and would generally be labeled by the expert as typical. When using either PANDA or DM, the instances ranked as least suspicious were in all cases typical and of no interest to the expert.

Tables 4 and 5 display the number of instances from each class (fp and nfp) that fell into each interval range. From a software engineering point of view, a fp instance is likely to have relatively large values for its attributes, i.e., software metrics. In contrast, a nfp instance is likely to have relatively lower values for its attributes. The intuitive assumption is that the more complex a software program, the more likely it is fp.

The first table (Table 4) shows the distribution for the top 125 instances, while the second shows the distribution for the remainder of the top 250 instances. Overall, 124 of the 125 instances that DM ranked in the top 125 most suspicious instances belonged to the fp group, and only 1 belonged to the nfp group. In contrast, for PANDA only 74 of the 125 were fp, while 51 were nfp. For instances ranked by DM between 126 and 250, the vast majority (105 out of 125) of them are fp.

This result is not surprising, as DM is designed to capture instances that are far away from the main distribution of the data set. The fp group has observations which have large values for all of the software attributes, and hence are more likely to look like outliers. PANDA (while favoring the fp group for the top 5%) is much more balanced with respect to the class variable. Comparatively speaking, PANDA provides a more favorable noise detection result, since it is able to identify noisy instances in both classes without heavily weighing one over the other. The DM algorithm is more likely to detect normal fp instances.

The cumulative distribution of fp instances detected by the two algorithms for different rank intervals is shown in Table 6. This table shows the distribution for up to the top 2000 instances ranked by the two algorithms. Recall the JM1-

Table 4 Class of instances ranked 1–125 by PANDA or DM, JM1-2445 data set

Instance class	1–25		26–50		51–75		76–125		**1–125**	
	PANDA	DM	PANDA	DM	PANDA	DM	PANDA	DM	PANDA	DM
fp	24	25	16	25	11	25	23	49	74	124
nfp	1	0	9	0	14	0	27	1	51	1

Table 5 Class of instances ranked 126–250 by PANDA or DM, JM1-2445 data set

Instance class	126–150		151–200		201–250		**125–250**	
	PANDA	DM	PANDA	DM	PANDA	DM	PANDA	DM
fp	13	25	16	50	18	30	47	105
nfp	12	0	34	0	32	20	78	20

Table 6 Cumulative distribution of fp instances, JM1-2445 data set

Rank	PANDA	DM
25	24	25
50	40	50
75	51	75
100	66	100
125	74	124
150	87	149
200	103	199
250	121	229
300	129	234
400	151	235
500	165	235
1000	208	235
1500	227	235
2000	233	235
Total fp	235	235

Table 7 Commonality between PANDA and DM, JM1-2445 data set

Rank	Overlap (%)
25	44.00
50	40.00
75	44.00
100	44.00
125	46.40
150	47.00
200	47.00
250	54.00
300	62.00
400	72.00
500	72.00
1000	76.00

2445 data set consists of 2210 nfp and 235 fp instances. The DM algorithm identifies all the fp instances within the top 400 instances. In contrast, among the top 400 instances ranked by PANDA, only 151 are fp. Even among the top 2000 instances ranked by PANDA, there are two fp instances still remaining. These results further demonstrate that PANDA provides a more favorable noise detection result than DM does.

Table 7 illustrates the number of common instances between PANDA and DM at various levels, starting from the top 25 instances to the top 1000 instances. Among the top 25 suspicious instances identified by both algorithms, 40% are in common. Similarly, among the top 125 suspicious instances identified by both algorithms, 46.4% are in common. For the top 1000 suspicious instances, representing 40.9% of the data set, the overlap is still only 76%. This data confirms that there is a significant difference between the rankings produced by these two algorithms.

Considering the expert-based analysis of results obtained by PANDA and DM, we conclude that PANDA is detecting more noise and fewer outliers and typical instances than DM. Another interesting result of this analysis is that PANDA detects exceptions, i.e., those rare instances that lie outside the general distributions of the data set. Even if they do not contain errors, such instances are often of interest to the expert because they may represent potentially new and interesting cases. PANDA is also more balanced in regards to the class of the detected instances. DM was unable to detect many of the nfp instances that were noise, since the distance measure is dominated by large values, and fp instances generally have relatively larger attribute values when compared to the nfp instances.

5.2 Case Study 2: CCCS data set

This case study examines the noise detection capability of both PANDA and DM on another real-world data set called CCCS. In contrast to data set JM1-2445, which was considered in Sect. 5.1, CCCS did not undergo any data preprocessing prior to the application of PANDA. Preprocessing on JM1-2445 was done to eliminate class noise from the data set, and JM1-2445 is a relatively cleaner data set than the one from which it was derived, JM1-8850. More specifically, JM1-2445 does not contain any instances with class noise, but it does contain instances with attribute noise. CCCS, on the other hand, is a completely unaltered and unprocessed real-world data set, which contains real-world noise in both the dependent and independent variables.

PANDA was executed with 5, 7, 9, 12, 15, 17, and 20 bins with the final noise rank equal to the median rank over these 7 iterations. Multiple bin sizes were used to reduce the possibility of introducing anomalies that might be caused by the partitioning process into the data set. Compared to JM1-2445, different bin sizes were used for CCCS, since the size of the data sets are drastically different (282 instances in CCCS and 2445 instances in JM1-2445). The same parameters used for DM in Case Study 1 were also used in this case study. Specifically, DM was executed 10 times with a different value for k from 1 to 10 at each iteration. The final ranking produced by DM was the median rank over these 10 iterations.

Table 8 displays the results of the expert evaluation of the 30 most suspicious instances from the CCCS data set (approximately 10% of the data set) as identified by PANDA or DM. PANDA identifies many noisy instances as well as a few exceptions and outliers, while DM detects a larger number of outliers (12 vs. 5 for PANDA) but fewer noisy instances (18 vs. 21 for PANDA).

Table 8 Expert evaluation of the 10% most suspicious instances, CCCS data set

Instance category	1–10		11–20		21–30		**1–30**	
	PANDA	DM	PANDA	DM	PANDA	DM	PANDA	DM
Noise	6	6	7	4	8	8	21	18
Outliers	2	4	2	6	1	2	5	12
Exceptions	2	0	1	0	1	0	4	0
Typical	0	0	0	0	0	0	0	0

Table 9 Class of instances ranked 1–30 by PANDA or DM, CCCS data set

Instance category	1–10		11–20		21–30		**1–30**	
	PANDA	DM	PANDA	DM	PANDA	DM	PANDA	DM
fp	4	8	5	10	1	8	10	26
nfp	6	2	5	0	9	2	20	4

Table 10 Cumulative distribution of fp instances, CCCS data set

Rank	PANDA	DM
20	9	18
40	13	32
60	18	39
80	25	45
100	27	53
125	32	55
150	37	55
175	40	55
200	45	55
225	47	55
250	51	55
282	55	55
Total fp	55	55

CCCS contains an attribute ($nfaults$) indicating the number of faults attributed to the module during the system integration and test phases and during the first year of deployment. To obtain a data set with a binary class variable indicating whether each module should be considered fault prone (fp) or not fault prone (nfp), a threshold α can be set on the number of faults. Any module with more than α faults is considered fp, while the remaining modules are considered nfp. When calculating the instance rankings created by both PANDA and DM, the original attribute $nfaults$ was used. For comparing the class distribution of the two techniques in Tables 9 and 10, we use the binary class variable with $\alpha = 3$. Any module with more than three faults was considered fp. This particular threshold was chosen based on previous empirical studies using the CCCS data set [12].

Table 9 displays the class distribution for the 30 most suspicious instances as identified by either PANDA or DM. PANDA detects 10 fp instances and 20 nfp instances, while DM detects 26 fp instances and 4 nfp instances. Table 10 displays the cumulative distribution of fp instances for PANDA and DM. DM has ranked all 55 fp instances in the 125 most suspicious instances, while even at rank 250, PANDA has detected only 51 of the 55 fp instances. PANDA has a much more favorable class distribution related to the detected instances when compared to DM.

Table 11 displays the commonality between PANDA and DM on the CCCS data set. The overlap among the 20 most suspicious instances identified by PANDA and DM is 40%, while among the 100 most suspicious instances (which

Table 11 Commonality between PANDA and DM, CCCS data set

Rank	Overlap (%)
20	40.00
40	32.50
60	45.00
80	58.75
100	63.00
150	70.00
200	79.50

is approximately 35% of the data set) the overlap is 63%. This data confirms that PANDA and DM are detecting different instances.

6 Conclusion

This paper focuses on the problem of data quality in measurement data. Decisions made by data mining models are invariably affected by the quality of the underlying data. If the quality of the data is poor, a classification model trained using that data is likely to yield incorrect estimations. While the problem of detecting instances with class noise has seen increased attention, very limited focus has been placed on detecting instances having attribute noise.

We have introduced a new noise detection algorithm called PANDA, which is designed to detect instances that contain attribute noise. The output of PANDA is a list of instances ordered from the most noisy to least noisy. PANDA is attractive because it can be used for noise detection in the absence of the dependent variable data, i.e., class labels. Furthermore, to our knowledge PANDA is the first algorithm to address the problem of detecting instances with attribute noise without the need of class label data.

The proposed approach is investigated in two separate case studies. The first case study considers software measurement data from a NASA project which is preprocessed to eliminate instances that contain class noise. The second case study uses another software measurement data set which did not undergo any data preprocessing before the application of our technique. Note that many additional experiments with different data sets were conducted and similar empirical conclusions were obtained; however, their presentation is omitted due to space limitations. The proposed algorithm is automated in a software application and can be applied to data sets of different sizes and from other domains. We compare the performance of PANDA with an outlier detection algorithm. A software engineering expert inspected the instances detected by both procedures to determine whether they are noise, exceptions, or outliers. The algorithms were then evaluated for effectiveness in detecting the noisy instances.

PANDA is able to identify more noisy instances and fewer outliers and typical instances than the outlier detection procedure DM. Additionally, PANDA provides more balanced noise detection among the fp and nfp classes, while the instances identified by the DM algorithm were largely fp. It should be noted that from a software engineering point of view, the attribute values for the fp instances are generally larger than the corresponding attribute values for the nfp instances, which is why DM is more likely to detect fp instances.

In contrast to our expert-based approach to noise detection, related literatures have often injected randomly generated noise in order to evaluate a given noise detection technique. However, the data set into which noise is injected is often not verified to be noise-free. In the first case study, we utilize a data set without class noise which does, however, contain attribute noise. The second case study uses a data set that received no special treatment before the application of PANDA. The instances in these data sets contain the types of noise that are encountered in real-world data. A software engineering expert is used to evaluate the effectiveness of our algorithm on real-world data sets that contain real-world (as opposed to injected) noise.

Future work will continue to explore other algorithms for attribute noise detection that can be used with data sets of various domains. An investigation of PANDA by including the class attribute in its analysis can be compared with other supervised noise detection approaches. Whether the class label is treated like the other attributes or as a special (or weighted) attribute needs investigation. Another important issue that needs further investigation is to determine the attributes within an instance that caused the instance to be classified as noisy. As with any empirical study, additional case studies will further validate the effectiveness of the proposed noise detection approach.

Acknowledgements We thank Naeem Seliya for his suggestions and assistance with patient modifications and editorial reviews of the paper. We thank Dr. Shi Zhong for reviewing the manuscript. We also thank the anonymous reviewers and guest editors of the special issue on Mining Low-Quality Data for their helpful comments and thoughtful reviews.

References

1. Aggarwal C, Yu P (2001) Outlier detection for high dimensional data. In: Proceedings of ACM SIGMOD conference on management of data, ACM Press, Dallas, TX
2. Bobrowski M, Marre M, Yankelevich D. A software engineering view of data quality. Available at www.citeseer.ist.psu.edu/277636.html
3. Brodley CE, Friedl MA (1999) Identifying mislabeled training data. J Artif Intell Res 11:131–167
4. Clark P, Niblett T (1991) Rule induction with CN2: some recent improvements. In: Proceedings of the 5th European working session on learning, pp 151–163
5. Dunagan JD (2002). A geometic theory of outliers and perturbation. Ph.D. Dissertation. Available at http://research.microsoft.com/~jdunagan/thesis.pdf
6. Fenton NE, Pfleeger SL (1997) Software metrics: a rigorous and practical approach, 2nd edn. PWS Publishing Company: ITP, Boston, MA
7. Galhardas H, Florescu D, Shasha D, Simon E (2000) An extensible framework for data cleaning. In: Proceedings of 18th international conference on data engineering, IEEE Computer Society, San Jose, CA
8. Gamberger D, Lavrac N, Dzeroski S (1999) Noise elimination in inductive concept learning: a case study in medical diagnosis. In: Proceedings of the 7th international workshop on algorithmic learning theory, Springer, Berlin Heidelberg Ney York, pp 199–212
9. Gamberger D, Lavrac N, Groselj C (1999) Experiments with noise filtering in a medical domain. In: Proceedings of the 16th international conference on machine learning. Morgan Kaufmann, San Mateo, California, pp 143–153
10. Hernandez MA, Stolfo SJ (1995) The merge/purge problem for large databases. In: Proceedings of ACM SIGMOD conference on management of data, ACM, pp 127–138. citeseer.ist.psu.edu/stolfo95mergepurge.html
11. Hernandez MA, Stolfo, SJ (1998) Real-world data is dirty: data cleansing and the merge/purge problem. Data Min Knowl Discov 2(1):9–37

12. Khoshgoftaar TM, Allen EB (1998) Classifcation of fault-prone software modules: prior probabilities, costs and model evaluation. Empiric Software Eng 3:275–298
13. Khoshgoftaar TM, Bullard LA, Gao K (2003) Detecting outliers using rule-based modeling for improving CBR-based software quality classification models. In: Ashley KD, Bridge DG (eds) Proceedings of the 16th international conference on case-based reasoning. LNAI, vol 1689. Springer-Verlag, Berlin Heidelberg New York, pp 216–230
14. Khoshgoftaar TM, Rebours P (2004) Generarting multiple noise elimination filters with the ensemble-partitioning filter. In: Proceedings of the IEEE international conference on information reuse and integration, IEEE Systems, Man and Cybernetics Society, Las Vegas, NV, USA, pp 369–375
15. Khoshgoftaar TM, Seliya N (2004) The necessity of assuring quality in software measurement data. In: Proceedings of 10th international software metrics symposium, IEEE Computer Society, Chicago, IL, pp 119–130
16. Khoshgoftaar TM, Seliya N, Gao K (2005) Detecting noisy instances with the rule-based classification model. Intell Data Anal 9(4):347–364
17. Khoshgoftaar TM, Zhong S, Joshi V (2005). Noise elimination with ensemble-classifier filtering for software quality estimation. Intell Data Anal 9(1):3–27
18. Knorr E, Ng R (1997) A unified notion of outliers: Properties and computation. In Proceedings of knowledge discovery and data mining. American Association for Artificial Intelligence, Newport Beach, CA, pp 219–222
19. Knorr E, Ng R (1998) Algorithms for mining distance-based outliers in large datasets. In: Proceedings of 24th international conference on very large databases, New York, NY, pp 392–403
20. Marcus A, Maletic J, Lin K-I (2001) Ordinal association rules for error identification in datasets. In: Proceedings of 10th international conference on information and knowledge management. ACM Press, Atlanta, GA, pp 589–591
21. Murphy, PM, Aha DW (1998) UCI repository of machine learning databases. University of California, Irvine, Department of Information and Computer Science. http://www.ics.uci.edu/∼mlearn/MLRepository.html
22. Quinlan JR (1993) C4.5: programs for machine learning. Morgan Kaufmann, San Mateo, California
23. Ramasway S, Rastogi R, Shim K (2000) Efficient algorithms for mining outliers from large datasets. In: Proceedings of ACM SIGMOD conference on management of data, ACM, pp 427–438
24. SAS Institute (2004) SAS/STAT user's guide. SAS Institute Inc
25. Shekhar S, Lu C, Zhang P (2002) Detecting graph-based spatial outliers. Intell Data Anal 6:451–458
26. Strong D, Lee Y, Wang R (1997) Data quality in context. Commun ACM 40(5):103–110
27. Teng CM (1999) Correcting noisy data. In: Proceedings of 6th international conference machine learning (ICML 99). Morgan Kaufmann, San Mateo, California, pp 239–248
28. Yang Y, Wu X, Zhu X (2004) Dealing with predictive-but-unpredictable attributes in noisy data sources. In: Proceedings of 8th European conference on principles and practice of knowledge discovery in databases, Pisa, Italy
29. Zhong S, Khoshgoftaar TM, Seliya N (2004) Analyzing software measurement data with clustering techniques. IEEE Intell Syst, pp 22–29
30. Zhu X, Wu X (2004) Class noise vs attribute noise: a quantitative study of their impacts. Artif Intell Rev 22(3–4):177–210

Jason Van Hulse is a Ph.D. candidate in the Department of Computer Science and Engineering at Florida Atlantic University. His research interests include data mining and knowledge discovery, machine learning, computational intelligence and statistics. He is a student member of the IEEE and IEEE Computer Society. He received the M.A. degree in mathematics from Stony Brook University in 2000, and is currently Director, Decision Science at First Data Corporation.

Taghi M. Khoshgoftaar is a professor at the Department of Computer Science and Engineering, Florida Atlantic University, and the director of the Empirical Software Engineering and Data Mining and Machine Learning 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 subjects. He has been a principal investigator and project leader in a number of projects with industry, government, and other research-sponsoring agencies. He is a member of the IEEE, the IEEE Computer Society, and IEEE Reliability Society. He served as the program chair and general chair of the IEEE International Conference on Tools with Artificial Intelligence in 2004 and 2005, respectively. Also, he has served on technical program committees of various international conferences, symposia, and workshops. He has served as North American editor of the *Software Quality Journal*, and is on the editorial boards of the journals *Empirical Software Engineering*, *Software Quality*, and *Fuzzy Systems*.

Haiying Huang received the M.S. degree in computer engineering from Florida Atlantic University, Boca Raton, Florida, USA, in 2002. She is currently a Ph.D. candidate in the Department of Computer Science and Engineering at Florida Atlantic University. Her research interests include software engineering, computational intelligence, data mining, software measurement, software reliability, and quality engineering.