Error Detection and Impact-Sensitive Instance Ranking in Noisy Datasets

Xingquan Zhu, Xindong Wu, and Ying Yang

Department of Computer Science, University of Vermont, Burlington VT 05405, USA
{xqzhu, xwu, yyang}@cs.uvm.edu

Abstract

Given a noisy dataset, how to locate erroneous instances and attributes and rank suspicious instances based on their impacts on the system performance is an interesting and important research issue. We provide in this paper an Error Detection and Impact-sensitive instance Ranking (*EDIR*) mechanism to address this problem. Given a noisy dataset D, we first train a benchmark classifier T from D. The instances, that cannot be effectively classified by T are treated as suspicious and forwarded to a subset S. For each attribute A_i, we switch A_i and the class label C to train a classifier AP_i for A_i. Given an instance I_k in S, we use AP_i and the benchmark classifier T to locate the erroneous value of each attribute A_i. To quantitatively rank instances in S, we define an impact measure based on the Information-gain Ratio (*IR*). We calculate IR_i between attribute A_i and C, and use IR_i as the impact-sensitive weight of A_i. The sum of impact-sensitive weights from all located erroneous attributes of I_k indicates its total impact value. The experimental results demonstrate the effectiveness of our strategies.

1. Introduction

The goal of inductive learning is to form generalizations from training instances such that the classification accuracy on previously unobserved instances is maximized. This maximum accuracy is usually determined by two most important factors: (1) the quality of the training data; and (2) the inductive bias of the learning algorithm. Given a specific learning algorithm, it is obvious that its classification accuracy depends vitally on the quality of the training data. Basically, the quality of a real-world dataset depends on a number of issues (Wang *et al.* 2001), but the source of the data is a crucial factor. Data entry and acquisition are inherently prone to errors. Unless an organization takes extreme measures in an effort to avoid data errors the field error rates are typically around 5% or more (Orr 1998; Maletic & Marcus 2000).

There have been many approaches for data preprocessing (Maletic & Marcus 2000; Wang *et al*. 2001) and noise handling (Bansal *et al*. 2000; Brodley & Friedl 1999; Gamberger *et al*. 1999; Kubica *et al*. 2003; Little & Rubin 1987; Liu *et al.* 2002; Teng 1999; Wu 1995; Zhu *et al*. 2003) to enhance the data quality, where the enhancement is achieved through noise elimination, missing value prediction, or noisy value correction. Basically, a general noise handling mechanism consists of three important steps: (1) noisy instance identification; (2) erroneous attribute detection; and (3) error treatment.

For errors introduced by missing attribute values, the first two steps are trivial, because the instance itself will explicitly indicate whether it contains noise or not (e.g, a "?" represents a missing attribute value). Therefore, techniques for this type of attribute noise handling mainly focus on predicting correct attribute values (which is called imputation in statistics) by using a decision tree (Shapiro 1987) or other mechanisms (Little & Rubin 1987). If an instance contains an erroneous value, how to distinguish this error becomes a challenging task, because such a noisy instance likely acts as a new training example with valuable information. Teng (1999) proposed a polishing mechanism to correct noisy attribute values by training classifiers for each attribute. However, this correcting procedure tends to introduce new noise when correcting the "suspicious" attributes. A recent research effort from Kubica & Moore (2003) employed probabilistic models to model noise and data generation, in which the trained generative models are used to detect suspicious instances in the dataset. A similar approach was adopted in Schwarm & Wolfman (2000) where a Bayesian model is adopted to identify erroneous attribute values. Unfortunately, since real-word data rarely comply with any generative model, these model-based methods still suffer from a common problem of noise correction: introducing new errors during data correction. Meanwhile, researchers from statistics have also put significant efforts to locate and correct problematic attribute values. Among various solutions from statistics, the Fellegi-Holt editing method (Fellegi & Holt 1976) is the most representative one. To identify and correct errors, this approach takes a set of edits as input, where the edits indicate the rules that attribute values should comply with. For example "age < 16" should not come with "Marital status = Married". The Fellegi-Holt method has the advantage that it determines the minimal number of fields to change (located errors) so that a record satisfies all edits in one pass through the data. This approach has been extended to many editing systems, such as SPEER at the Census Bureau (Greenberg & Petkunas 1990). Unfortunately, the most challenging problem of the Fellegi-Holt method is to find a set of good edits, which turns to be impossible in many situations.

All methods above are efficient in their own scenarios, but some important issues are still open. First of all, due to the fact that noise correction may incur more troubles, such as ignoring outliers or introducing new errors, the reliability of these "automatic" mechanisms is questionable, especially when the users are very serious with their data. On the other hand, for real-world datasets, doing data cleansing "by hand" is completely out of the question given the amount of human labor and time involved. Therefore, the contradiction between them raises a new research issue: how to rank instances' impacts, so that given a certain amount of expenses (e.g, processing time), the data manager can maximize the system performance by putting priority on instances with higher impacts.

In this paper, we provide an error detection and impact-sensitive instance ranking system to address this problem. Our experimental results on real-world datasets will demonstrate the effectiveness of our approach: with datasets from the UCI repository (Blake & Merz 1998), at any noise level (even 50%), our system shows significant effectiveness in locating erroneous attributes and ranking suspicious instances.

2. Proposed Algorithms

Our *EDIR* system consists of two major steps: Error Detection and Impact-sensitive Ranking. The system flowchart is depicted in Fig. 1, and the procedures are given in Figs. 2 and 3.

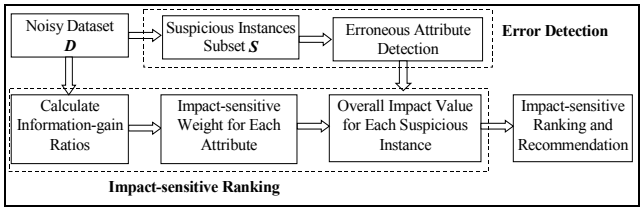


Fig. 1. The flowchart of *EDIR* system

Procedure: ***EDIR*** ()
Input: Dataset D; **Output:** Ranked suspicious instances.
Parameter: K, # of maximal changes for one instance

(1). $S \leftarrow \phi$; $EA \leftarrow \phi$; $IR \leftarrow \phi$
(2). Train benchmark classifier T from D.
(3). For each instance I_k in D
(4). If {!*CorrectClassify*(I_k, T) or !*Cover*(I_k, T)}
(5). $S \leftarrow S \cup \{I_k\}$
(6). For each attribute A_i
(7). Calculate Information-gain Ratio (IR_i) between A_i and C
(8). $IR \leftarrow IR \cup \{IR_i\}$
(9). Switch A_i and C to learn AP_i and rule set AR_i
(10). Evaluate accuracy of each rule in AR_i on D.
(11). For each instance I_k in S
(12). $EA_k \leftarrow \phi$; $A\tilde{V}^k \leftarrow \phi$; $CV^k \leftarrow \phi$
(13). For each attribute A_i of I_k
(14). Calculate $A\tilde{V}_i^k$ and confidence CV_i^k from AP_i
(15). $A\tilde{V}^k \leftarrow A\tilde{V}^k \cup \{A\tilde{V}_i^k\}$; $CV^k \leftarrow CV^k \cup \{CV_i^k\}$
(16). For (L=1; L<=K; L++)
(17). If {*ErrorDetection*(I_k, L, $A\tilde{V}^k$, CV^k, T, &EA_k)}
(18). $EA_k \leftarrow$ Located L erroneous attributes
(19). $EA \leftarrow EA \cup \{EA_k\}$
(20). Break
(21). ImpactSensitiveRanking (S, IR, EA) // see Section 2.2

Fig. 2. Error Detection and Impact-sensitive Ranking

Procedure: ***ErrorDetection*** (I_k, L, $A\tilde{V}^k$, CV^k, T, &EA_k)

(1). $EA_k \leftarrow \phi$; $\Omega[] \leftarrow \phi$; $ChgNum \leftarrow 0$; $Iteration \leftarrow 0$; $CV[] \leftarrow \phi$
(2). Do { $AV_{i_1}^k \leftarrow A\tilde{V}_{i_1}^k; .., AV_{i_L}^k \leftarrow A\tilde{V}_{i_L}^k$; $A_{i_1}, .. A_{i_L} \in \Re$;
$A_{i_1}, ... A_{i_L} \notin \Omega[]$; $A_{i_1} \neq .. \neq A_{i_L}$; $A_{i_j} \in \forall \Re$
(3). If {*CorrectClassify* (I_k, T)}
(4). $\Omega[ChgNum] \leftarrow \{A_{i_1}, ..., A_{i_L}\}$
(5). $CV[ChgNum] = CV_{i_1}^k + ... + CV_{i_L}^k$
(6). $ChgNum$++
(7). Restore old values of $A_{i_1}, ... A_{i_L}$; $Iteration$++;
(8). } while { $Iteraction \leq \binom{N}{L}$ }
(9). If {$ChgNum = 0$} Return (0)
(10). Else
(11). $EA_k \leftarrow \Omega[l]$; l=arg{max($CV[j]$), j=1, 2.., $ChgNum$}
(12). Return (1)

Fig. 3. Erroneous attribute detection

2.1 Error Detection

2.1.1 Suspicious Instance Subset Construction

Our first step of error detection is to construct a subset S to separate suspicious instances from the dataset D. We first train a benchmark classifier T from D, and then use T to evaluate each instance in D. The subset S is constructed by using the following criteria, as shown in steps (3) to (5) of Fig. 2.

1. If an instance I_k in D cannot be correctly classified by T, we forward it to S.
2. If an instance I_k in D does not match any rule in T, we forward it to S too.

The proposed mechanism relies on the benchmark classifier (which is also imperfect) trained from the noisy dataset to explore noisy instances. Our experimental analysis has suggested that although the benchmark classifier is not perfect (actually we may never learn a perfect classifier), it can still be relatively reliable to detect some noisy instances.

2.1.2 Erroneous Attribute Detection

Given an instance I_k in S, assume I_k contains N attributes $A_1, A_2, .., A_N$ and one class label C. Further assume each attribute A_i has V_i possible values. We denote the aggregation of all attributes by $\Re$. To locate erroneous attributes from I_k (where an erroneous attribute means the attribute has an incorrect value), we adopt an Attribute Prediction (*AP*) mechanism, as shown in Figs. 2 and 3.

Basically, Attribute Prediction uses all other attributes $A_1, .. A_j, ... A_N$ ($j \neq i$) and the class label C to train a classifier, AP_i, for A_i (using instances in D). Given an instance I_k in S, we use AP_i to predict the value of attribute A_i. Assume I_k's current value of A_i is AV_i^k, and the predicted value from AP_i is $A\tilde{V}_i^k$. If AV_i^k and $A\tilde{V}_i^k$ are different, it implies that A_i of I_k may possibly contain an incorrect value. Then we use the benchmark classifier T (which is relatively reliable in evaluating noisy instances) to determine whether the predicted value from AP_i makes more sense: If we change AV_i^k to $A\tilde{V}_i^k$, and I_k can be correctly classified by T, it will indicate that the change results in a better classification. We therefore conclude that attribute A_i contains an erroneous value. However, if the change still makes I_k incorrectly classified by T, we will leave A_i unchanged and try to explore errors from other attributes. If the prediction from each single attribute does not conclude any error, we will start to revise multiple attribute values at the same time. For example, change the values of two attributes A_i (from AV_i^k to $A\tilde{V}_i^k$) and A_j (from AV_j^k to $A\tilde{V}_j^k$) at the same time, and then evaluate whether the multiple changes make sense to T. We can iteratively execute the same procedure until a change makes I_k correctly classified by T. However, allowing too many changes in one instance may actually have the error detection algorithm make more mistakes. Currently, the *EDIR* system allows to change up to K ($K \leq 3$) attributes simultaneously to locate errors. If all the changes above still make I_k incorrectly resolved by T, we will leave I_k unprocessed.

With the proposed error detection algorithm, one important issue should be resolved in advance: which attribute to select if multiple attributes are found to contain errors.

Our solution in solving this problem is to maximize the prediction confidence while locating the erroneous attributes, as shown in Fig. 3. When learning AP_i for each attribute A_i, we use a classification rule algorithm, e.g., C4.5rules, to learn an Attribute

prediction Rule set (AR_i). Assuming the number of rules in AR_i is M_i, for each rule AR_i^r, r=1,..M_i, in AR_i, we evaluate its accuracy (AR_i^r) on dataset D. This accuracy will indicate the confidence that AR_i^r classifies the instance. In our system, we use C4.5rules (Quinlan 1993) to learn the rule set AR_i, so the accuracy value has been provided with each learned rule.

When adopting the rule set AR_i to predict the value of A_i, we use the first hit mechanism (Quinlan 1993), which means we rank the rules in AR_i in advance, and classify instance I_k by its first coved rule in AR_i. Meanwhile, we also use the accuracy of the selected rule as the confidence (AC_i^k) of AP_i in predicting A_i of I_k.

Given an instance I_k, assume the predicted values for each attribute are $A\widetilde{V}_1^k$, .., $A\widetilde{V}_N^k$ respectively, with the confidences for each of them denoted by AC_1^k,..,AC_N^k. We first set L to 1 to locate an erroneous attribute by using Eq. (1). If this procedure does not find any erroneous attribute, we increase the value of L by 1 and repeat the same procedure, until we find erroneous attributes or L reaches the maximal allowable changes for one instance (K).

$$EA_k = \{A_{i_1},..,A_{i_L}\};\ \arg\{\max_{i_1,..,i_L}\{\sum_{l}^{L} AC_{i_l}^k\};\ CorrectClassify(A\widetilde{V}_{i_1}^k,..,A\widetilde{V}_{i_L}^k,T)=1\}_{\{A_{i_1},..A_{i_j},..A_{i_L}\}\in\Re;\ \ i_1\neq i_j\neq i_L;\ \ A_j\in\forall\Re} \quad (1)$$

2.1.3 Validity Analysis

Our algorithm above switches each attribute A_i and class C to learn an AP_i classifier, then uses AP_i to locate erroneous attributes. There are two possible concerns with the algorithm: (1) switching A_i and C to learn classifier AP_i does not make much sense to many learning algorithms, because attribute A_i may have very little correlation with other attributes (or not at all); and (2) when the prediction accuracy from AP_i is relative low (e.g., less than 50%), does the algorithm still work? Our experimental results in Section 3 will indicate that even the prediction accuracy from all AP_i classifiers are relatively low, the proposed algorithm can still provide good results.

As we can see from Fig. 3, the prediction from each AP_i classifier just provides a guide for the benchmark classifier T to evaluate whether a change makes the classification better or not. The prediction from AP_i won't be adopted unless T agrees that the value predicted by AP_i will make the instance correctly classified. In other words, the proposed mechanism relies more on T than on any AP_i. Even if the prediction accuracy from AP_i is 100%, we won't take its prediction unless it gets the support from T. Therefore, a low prediction accuracy from AP_i does not have much influence with the proposed algorithm. However, we obviously prefer a high prediction accuracy from each AP_i. Then the question comes to how good AP_i could be with a normal dataset? Actually, the performance of AP_i is inherently determined by correlations among attributes. It is obvious that if all attributes are independent (or conditionally independent given the class C), the accuracy of AP_i could be very low, because no attribute could be used to predict A_i. However, it has often been pointed out that this assumption is a gross over-simplification in reality, and the truth is that the correlations among attributes extensively exist (Freitas 2001; Shapiro 1987). Instead of taking the assumption of conditional independence, we take the benefits of interactions among attributes, as well as between the attributes and class. Just as we can predict the class C by using the existing attribute values, we can turn the process around and use the class and some attributes to predict the value of another attribute. Therefore, the average accuracy of AP_i classifiers from a normal dataset usually maintains a reasonable level, as shown in Tab. 1.

2.2 Impact-sensitive Ranking

To rank suspicious instances by their impacts on the system performance, we define an impact measure based on the Information-gain Ratio (IR). We first calculate IR between each attribute A_i and class C, and take this value as the impact-sensitive weight (IW) for A_i. The impact value for each suspicious instance I_k, $Impact(I_k)$, is then defined by Eq. (2), which is the sum of the impact-sensitive weights of all located erroneous attributes in I_k,

$$Impact(I_k)=\sum_{i=1}^{N} IW(A_i,I_k);\quad IW(A_i,I_k)=\begin{cases} IR_i; & If\ A_i\ contains\ error \\ 0 & otherwise \end{cases} \quad (2)$$

The Information-gain Ratio (IR) is one of the most popular correlation measures used in data mining. It was developed from information gain (IG) which was initially used to evaluate the mutual information between an attribute and the class (Hunt *et al.* 1966). The recent development from Quinlan (1986) has extended IG to IR to remove the bias caused by the number of attribute values. Since then, IR has become very popular in constructing decision trees or exploring correlations. Due to the space limit of the paper, we omit the technique on calculating IR. Interested readers may refer to Quinlan (1986; 1993) for details.

With Eq. (2), instances in S can be ranked by their $Impact(I_k)$ values. Given a dataset with N attributes, the number of different impact values of the whole dataset D is determined by Eq. (3), if we allow the maximal number of changes for one instance to be K. For example, a dataset D with N=6 and K=3 will have $NC(D)$ equal to 41. It seems that $NC(D)$ is not large enough to distinguish each suspicious instance, if the number of suspicious instance in D is larger than 41 (which is likely a normal case). However, in reality, we usually work on a bunch of suspicious instances rather than a single one to enhance the data quality. Therefore it may not be necessary to distinguish the quality of each instance, but to assign suspicious instances into various quality levels. Given the above example, we can separate its suspicious instances into 41 quality levels. From this point of view, it's obvious that this number is large enough to evaluate the quality of instances in S.

$$NC\ (D) = \sum_{l=1}^{K} \binom{N}{l} \quad (3)$$

3. Experimental Evaluations

3.1 Experiment Settings

The majority of our experiments use C4.5, a program for inducing decision trees (Quinlan 1993). To construct the benchmark classifier T and AP_i classifiers, C4.5rules (Quinlan 1993) is adopted in our system. We have evaluated our algorithms extensively on datasets collected from the UCI data repository (Blake & Merz 1998). Due to the size restrictions, we will mainly report the results on two representative datasets: Monks-3 and Car, because these two datasets have relatively low attribute prediction accuracies, as shown in Tab. 1. If our algorithms achieve good results on these two datasets, they can possibly have good performances on most real-world datasets. In Tab. 1, AT_i represents the average prediction accuracy for attribute A_i (from AP_i). For Soybean, Krvskp and Mushroom datasets, we only show the results of the first six attributes. We also report summarized results from these three datasets in Tab. 5.

For most of the datasets that we used, they don't actually contain much noise, so we use manual mechanisms to add attribute

noise, where error values are introduced into each attribute with a level *x·100*%, and the error corruption for each attribute is independent. To corrupt each attribute A_i with a noise level *x·100*%, the value of A_i is assigned a random value approximately *x·100*% of the time, with each alternative value being approximately equally likely to be selected. With this scheme, the actual percentage of noise is always lower than the theoretical noise level, as sometimes the random assignment would pick the original value. Note that, however, even if we exclude the original value from the random assignment, the extent of the effect of noise is still not uniform across all components. Rather, it is dependent on the number of possible values in the attribute. As the noise is evenly distributed among all values, this would have a smaller effect on attributes with a larger number of possible values than those attributes that have only two possible values (Teng 1999). In all figures and tables below, we only show the noise corruption level *x·100*%, but not the actual noise level in each dataset.

Tab. 1. The average prediction accuracy for each attribute

DataSet	AT_1(%)	AT_2(%)	AT_3(%)	AT_4(%)	AT_5(%)	AT_6(%)
Monks-3	45.02	57.08	54.31	37.52	45.82	49.55
Car	36.48	35.47	27.68	49.57	41.24	56.5
Soybean	55.38	95.66	94.9	86.35	96.58	54.67
Krvskp	97.74	98.63	99.17	95.59	85.92	78.98
Mushroom	46.28	55.77	44.76	100	81.25	99.79

3.2 Suspicious Subset Construction

To evaluate the performance of the suspicious subset (*S*) construction, we need to assess the noise level in *S*. Intuitively, the noise level in *S* should be higher than the noise level in *D*, otherwise the existence of *S* becomes useless. Also, the size of *S* should be reasonable (not too large or too small), because a subset with only several instances has a very limited contribution in locating erroneous attributes, even if all instances in *S* are noise. To this end, we provide the following measures: (1) *S/D*, which is the ratio between the sizes of *S* and *D*; (2) I_S and I_D, which are the instance-based noise levels in *S* and *D*, as defined by Eq. (4); and (3) A_i_S and A_i_D, which represent the noise levels for attribute A_i in *S* and *D*, as defined by Eq. (5).

$$I\_X = \#\text{ erroneous instances in } X \,/\, \#\text{ instances in } X \quad (4)$$

$$A_i\_X = \#\text{ instances in } X \text{ with error in } A_i \,/\, \#\text{ instances in } X \quad (5)$$

We have evaluated our suspicious subset construction at different noise levels and provided the results in Tab. 2. Basically, the results indicate that in terms of *S/D*, the proposed algorithm in Section 2.1 has constructed a suspicious subset with a reasonable size. When the noise level increases from 10% to 50%, the size of *S* also proportionately increases (from 14.17% to 41.07% for the Car dataset), as we have anticipated. If we take a look at the third column with instance-based noise, we can find that the noise level in *S* is always higher than the noise level in *D*. For the Car dataset, when the noise corruption level is 10%, the noise level in *S* is about 30% higher than the noise level in *D*. With Monks-3, the noise level in *S* is significantly higher than *D*. It indicates that with the proposed approach, we can construct a reasonable size of the subset concentrating on noisy instances for further investigation.

The above observations conclude the effectiveness of the proposed approach, but it's still not clear whether *S* equally captures noise from all attributes or more focuses on some of them. We therefore evaluate the noise level on each attribute, which is shown from columns 5 to 10 in Tab. 2. As we can see, most of the time, the attribute noise level in *S* (A_i_S) is higher than the noise level in *D* (A_i_D), which means the algorithm has a good performance on most attributes. However, the algorithm also shows a significant difference in capturing noise from different attributes. For example, the noise level on attributes 2 and 5 in Monks-3 (in *S*) is much higher than the attribute noise level in the original dataset *D*. The reason is that these attributes have significant contributions in constructing the classifier, hence the benchmark classifier *T* is more sensitive to errors in these attributes. This is actually helpful for us to locate erroneous attributes: if noise in some attributes has less impact with the system performance, we can simply ignore them or put less effort on them.

3.3 Erroneous Attribute Detection

To evaluate the performance of our erroneous attribute detection algorithm, we define the following three measures: Error detection Recall (*ER*), Error detection Precision (*EP*), and Error detection Recall for each Attribute (ERA_i). Their definitions are given in Eq. (6), where n_i, p_i and d_i represent the number of actual errors, the number of correctly located errors and the number of located errors in A_i respectively.

$$ER = \sum_{i=1}^{N} ERA_i,\ ERA_i = \frac{p_i}{n_i}; \qquad EP = \sum_{i=1}^{N} EPA_i,\ EPA_i = \frac{p_i}{d_i} \quad (6)$$

We provide the results in Tab. 3, which are evaluated at three noise levels (from 10% to 50%). As we can see from the third column (*EP*) of Tab. 3, the overall error detection precision is pretty attractive, even with the datasets (Monks-3 and Car) that have low attribute prediction accuracies. On average, the precision is maintained at 70%, which means most located erroneous attributes actually contain errors. We have also provided the experimental results from other three datasets in Tab. 5. All these results prove the reliability of the proposed error detection algorithm.

Tab. 2. Suspicious subset construction results

Dataset	Noise Level	S/D	Instance Noise		AT_1 Noise		AT_2 Noise		AT_3 Noise		AT_4 Noise		AT_5 Noise		AT_6 Noise	
			I_D	I_S	A_1_D	A_1_S	A_2_D	A_2_S	A_3_D	A_3_S	A_4_D	A_4_S	A_5_D	A_5_S	A_6_D	A_6_S
Monks-3	10%	6.29	32.41	97.09	6.04	6.71	6.45	44.85	4.58	5.21	6.43	9.96	7.49	54.92	5.62	5.34
	30%	17.14	71.06	94.78	20.22	20.95	20.33	51.46	14.94	15.83	20.75	22.29	21.63	49.2	14.73	16.02
	50%	33.42	89.45	95.68	32.61	33.18	33.52	54.36	25.3	25.74	33.07	33.72	38.11	53.25	24.8	25.27
Car	10%	14.17	36.3	64.14	7.51	14.25	7.73	14.15	7.86	8.65	6.87	16.85	6.72	9.75	6.66	18.6
	30%	33.71	76.32	85.97	22.71	26.78	22.75	26.35	22.52	22.69	19.95	28.47	20.56	21.21	20.02	29.78
	50%	41.07	92.06	96.61	36.82	39.45	36.42	38.67	37.65	37.27	33.29	40.71	32.56	34.64	33.52	42.79

Obviously, having a high precision is only a partial advantage of the algorithm. A system that predicts only one error is likely useless, even if its precision is 100%. We therefore evaluate the error detection recall (*ER*), as shown on columns 4 to 10 in Tab. 3. On average, the algorithm can locate about 10% of errors. Meanwhile, for different attributes, the ERA_i values vary significantly. For example, when the noise level is 30%, the ERA_1 for Monks-3 is about 5.4% which is much less than the value (about 24%) of attribute 2. If we go further to analyze the correlations between attributes and the class, we can find that the proposed algorithm actually has a good performance in locating errors for some important attributes. For example, among all attributes in Monks-3, attribute 2 has the highest correlation with class *C*, and its recall value (ERA_2) also turns out to be the highest.

Just looking at the *ER* values may make us worry that the algorithm has missed too much noise (90% of errors). We'd like to remind the readers that from the data correction point of view, having a higher precision is usually more important, because an algorithm should avoid introducing more errors when locating or correcting existing errors. Actually, the 10% located errors likely bring more troubles than others (because they obviously cannot be handled by the benchmark classifier *T*). Our experimental results in the next subsection will indicate that correcting this part of the errors will improve the system performance significantly.

Tab. 3. Erroneous attribute detection results

Data Set	Noise Level	*EP*	*ER*	ERA_1	ERA_2	ERA_3	ERA_4	ERA_5	ERA_6
Monks-3	10%	84.61	7.18	0.68	24.08	0.94	1.07	15.22	1.08
	30%	77.08	11.36	5.41	24.01	4.34	6.09	20.9	7.45
	50%	80.58	10.54	5.61	22.21	6.35	6.17	16.58	6.57
Car	10%	65.84	11.35	7.31	6.91	3.43	21.91	4.85	23.69
	30%	71.43	11.17	8.25	7.58	4.56	21.5	6.18	18.97
	50%	73.29	8.55	6.05	6.2	5.78	14.11	5.54	13.62

3.4 Impact-sensitive Instance Ranking

We evaluate the performance of the proposed impact-sensitive instance ranking mechanism in Section 2.2 from three aspects: training accuracy, test accuracy and the size of the constructed decision tree. Obviously, it's hard to evaluate the ranking quality instance by instance, because one instance likely does not impact too much with the system performance. We therefore separate the ranked instances into three tiers, each tier consisting of 30% of instances from the top to the bottom of the ranking. Intuitively, correcting instances in the first tier will produce a bigger improvement than correcting instances in any of the other two tiers, because instances in the first tier have more negative impacts (with relatively larger impact values), so does the second tier to the third tier. Accordingly, we manually correct the instances in each tier (because we know which instance was corrupted, the manual correction has a 100% accuracy), and compare the system performances on the corrected dataset and the original dataset.

We evaluate the system performance at three noise levels and provide results in Tab. 4, where "Org" means the performance (training, test accuracy and tree size) from the original dataset (*D*), and "Fst", "Snd" and "Thd" indicate the performance of only correcting instances in the first, second and third tier respectively.

From Tab. 4, we can find that at any noise level, correcting instances in the first tier always results in a better performance than correcting instances in any of the other two tiers, so does the second tier to the third tier. For example, with the Car dataset at 30% noise level, correcting instances at the first tier will achieve 0.7% and 2.3% more improvements with the test accuracy than correcting instances in the second and third tiers. In terms of the decision tree size, the improvement is even more significant. It proves that our system provides an effective way to automatically locate errors, and rank them by their negative impacts (danger levels). So we can put more emphasis on instances in the top tier than those in the bottom tier.

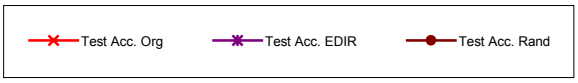
When comparing the performances from each tier with the performance from the original dataset, we find that correcting recommended instances has a significant improvement, especially when the noise level goes higher. For example, with the Car dataset at 30% noise level, correcting instances at any tier will contribute a 2.1% (or more) improvement with the test accuracy. This also proves the effectiveness of our *EDIR* system in enhancing the data quality, even in high noise-level environments.

When using *EDIR* as a whole system for noise detection and data correction, we'd like to know its performance in comparison with other approaches, such as random sampling. We therefore perform the following experiments. Given a dataset *D*, we use *EDIR* to recommend α% of instances in *D* for correction. For comparison, we also randomly sample α% of instances in *D* for correction. We compare the test accuracies from these two mechanisms (because the comparison of the test accuracies is likely more objective), and report the results in Fig. 4 (b) to Fig. 4 (g), where "*Org*", "*EDIR*" and "*Rand*" represent the performances from the original dataset, the corrected dataset by *EDIR* and the random sampling mechanism respectively. We have evaluated the results by setting α 100% to four levels: 5%, 10%, 15% and 20%. In Tab. 5, we provide summarized results from other three datasets by setting α 100% to 5%. The results from Fig. 4 indicate that when the value of α increases, the performances of *EDIR* and *Rand* both get better. This does not surprise us, because when recommending more and more instances for correction, we actually lower the overall noise level in the dataset, therefore better performances could be achieved. However, the interesting point is that when comparing "*EDIR"* and "Rand", we can find that *EDIR* always has a better performance than *Rand*.

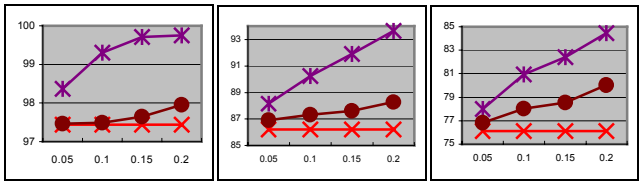
Tab. 4. Impact-sensitive instance ranking results

DataSet	Noise Level	Training Accuracy (%)				Test Accuracy (%)				Tree Size			
		Org	Fst	Snd	Thd	Org	Fst	Snd	Thd	Org	Fnt	Sed	Thd
Monks-3	10%	94.31	96.27	95.9	95.48	97.45	98.94	98.52	98.13	29.2	26.1	27.8	28.6
	30%	86.44	90.23	89.98	89.07	88.06	92.17	91.58	91.37	76.1	60.8	63.3	66.1
	50%	81.73	86.37	85.84	84.96	76.09	82.64	82.01	81.09	112.8	93.9	101.5	107.6
Car	10%	90.25	92.73	92.2	91.86	87.77	90.44	89.62	89.04	250.2	215.8	225.6	232.5
	30%	83.77	86.9	86.32	86.14	76.77	80.98	80.21	78.69	394.1	363.3	375.2	391.3
	50%	80.76	83.51	82.8	82.09	69.37	74.59	73.18	72.66	463.7	415.2	429.4	437

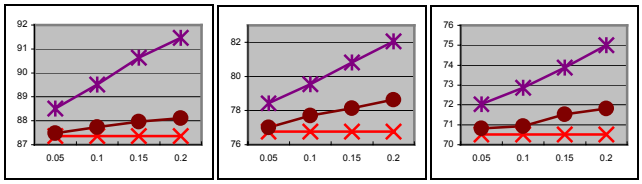
Actually, most of the time, the results from *EDIR* by recommending 5% of instances are still better than using *Rand* to recommend 20% of instances. For example, when *EDIR* recommends 5% of instances for correction, the test accuracy for the Car dataset at 10% noise level is 88.51%, which is still better than the results (88.11%) of using *Rand* to recommend 20% of instances. The same conclusion can be drawn from most other datasets. It indicates that *EDIR* has a significantly good performance in locating and recommending suspicious instances to enhance the data quality.



(a) Meaning of each curve in Fig. 4 (b) to Fig. 4 (g)



(b) Monks-3 (10%) (c) Monks-3 (30%) (d) Monks-3 (50%)



(e) Car (10%) (f) Car (30%) (g) Car (50%)

Fig. 4. Experimental comparisons of *EDIR* and random sampling approaches from three noise levels: 10%, 30% and 50%

In Figs. 4 (b) to (g), the *x*-axis denotes the percentage of recommended data (α) and the *y*-axis represents the test accuracy.

Tab. 5. Experimental summary from other three datasets (α=0.05)

Data Set	Noise Level	*EP*	*ER*	Test Accuracy			Tree Size		
				Org	*EDIR*	*Rand*	*Org*	*EDIR*	*Rand*
Soybean	10%	88.51	10.4	86.23	86.99	86.23	424.7	407.2	418.9
	30%	87.23	12.66	76.18	77.14	76.25	496.8	485.3	497.1
	50%	84.17	9.42	57.44	61.64	58.09	532.1	526	532.8
Krvskp	10%	76.57	10.56	95.22	96.78	95.89	500.2	379.5	466.4
	30%	72.33	8.69	80.23	85.94	82.66	986.8	887.1	964
	50%	71.08	6.83	67.65	79.41	70.78	1096	998.3	1072
Mushroom	10%	90.28	9.79	99.89	100	99.97	545	322	529
	30%	84.33	7.48	99.21	99.97	99.32	1644	1126	1620
	50%	81.49	7.04	98.32	99.82	99.11	3060	2383	2935

4. Conclusions

In this paper, we have presented an *EDIR* system, which automatically locates erroneous instances and attributes and ranks suspicious instances according to their impact values. The experimental results have demonstrated the effectiveness of our proposed algorithms for error detection and impact-sensitive ranking. By adopting the proposed *EDIR* system, correcting the instances with higher ranks always results in a better performance than correcting those with lower ranks. The novel features that distinguish our work from existing approaches are threefold: (1) we provided an error detection algorithm for both instances and attributes; (2) we explored a new research topic on impact-sensitive instance ranking, which can be very useful in guiding the data manager to enhance the data quality with minimal expenses; and (3) by combining error detection and impact-sensitive ranking, we have constructed an effective data recommendation system. It's more efficient than the manual approach and more reliable than automatic correction algorithms.

Acknowledgement

This research has been supported by the U.S. Army Research Laboratory and the U.S. Army Research Office under grant number DAAD19-02-1-0178.

References

Bansal, N., Chawla, S., & Gupta, A., (2000), Error correction in noisy datasets using graph mincuts, *Technical Report, CMU.*

Blake, C.L. & Merz, C.J. (1998), *UCI Repository of machine learning databases.*

Brodley, C.E. & Friedl, M.A. (1999), Identifying mislabeled training data, *J. of Artificial Intelligence Research*, 11: 131-167.

Greenberg, B. & Petkunas, T. (1990), SPEER: structured programs for economic editing and referrals, *American Statistical Asso., Proc. of the 1990 Section on Survey Research Methods*.

Fellegi, I. P. & D. Holt, (1976), A systematic approach to automatic edit and imputation, *Journal of the American Statistical Association*, vol.71, pp.17-35.

Freitas, A., (2001), Understanding the crucial role of attribute interaction in data mining, *AI Review*, 16(3):177-199.

Gamberger, D., Lavrac, N., & Groselj C. (1999), Experiments with noise filtering in a medical domain, *Proc. of 16th ICML*, CA.

Hunt, E. B., Martin, J. Stone, P. (1966), *Experiments in Induction*. Academic Press, New York.

Kubica, J. & Moore A., (2003), Probabilistic noise identification and data cleaning, *Proc. of ICDM*, FL, USA.

Little, R.J.A. & Rubin, D.B. (1987), *Statistical analysis with missing data*, Wiley, New York.

Liu, X., Cheng, G., & Wu, J. (2002), Analyzing outliers cautiously, *IEEE Trans. on TKDE*, 14:432-437.

Maletic J. & Marcus A. (2000), Data cleansing: Beyond integrity analysis, *Proc. of Information Quality*, pp. 200–209.

Orr, K., (1998), Data quality and systems theory, *CACM,* 41 (2):66-71.

Quinlan, J.R. (1986). Induction of decision trees. *Machine Learning*, 1(1): 81-106.

Quinlan, J.R. (1993). *C4.5: programs for machine learning*, Morgan Kaufmann, San Mateo, CA.

Schwarm S. & Wolfman S., (2000), Cleaning data with Bayesian methods, *Technical Report, University of Washington*.

Shapiro A. (1987), *Structured induction in expert systems*, Addison-wesley.

Teng M. T., (1999), Correcting noisy data, *Proc. of 16th ICML*.

Wang R., Ziad M., & Lee Yang, (2001), *Data quality*, Kluwer.

Wu, X. (1995), *Knowledge acquisition from database*, Ablex Pulishing Corp., USA.

Zhu, X., Wu, X. & Chen Q. (2003). Eliminating class noise in large datasets, *Proc. of 20th ICML*, Washington D.C., USA.