Class Noise Handling for Effective Cost-Sensitive Learning by Cost-Guided Iterative Classification Filtering

Xingquan Zhu and Xindong Wu

Abstract—Recent research in machine learning, data mining, and related areas has produced a wide variety of algorithms for cost-sensitive (*CS*) classification, where instead of maximizing the classification accuracy, minimizing the misclassification cost becomes the objective. These methods often assume that their input is quality data without conflict or erroneous values, or the noise impact is trivial, which is seldom the case in real-world environments. In this paper, we propose a Cost-guided Iterative Classification Filter (CICF) to identify noise for effective *CS* learning. Instead of putting equal weights on handling noise in all classes in existing efforts, *CICF* puts more emphasis on expensive classes, which makes it attractive in dealing with data sets with a large cost-ratio. Experimental results and comparative studies indicate that the existence of noise may seriously corrupt the performance of the underlying *CS* learners and by adopting the proposed *CICF* algorithm, we can significantly reduce the misclassification cost of a *CS* classifier in noisy environments.

Index Terms—Data mining, classification, cost-sensitive learning, noise handling.

1 Introduction

Inductive learning usually aims at forming a generalized description of a given set of data, so that further unseen instances can be classified with a minimal error rate. A common assumption is that errors result in the same amount of cost, which is seldom the case in reality. For example, in medical diagnosis, the errors committed in diagnosing a patient as healthy when he/she actually has a life-threatening disease is considered to be far more serious (hence, a higher cost) than the opposite type of error—diagnosing a patient as ill when he/she is in fact healthy. The same problem exists in financial institutions and other database marketing, where the cost of mailing to a nonrespondent is very small, but the cost of not mailing to someone who would respond is a more significant profit loss.

Recently, a body of work has attempted to address this issue, with techniques known as cost-sensitive learning [1], [2], [3], where the "cost" could be interpreted as misclassification cost, training cost, test cost, or others. Among all different types of costs, the misclassification cost is the most popular one. In general, the misclassification cost is described by a cost matrix C, with $C(i, j)$ indicating the cost of predicting that an example belongs to class i when in fact it belongs to class j. With this type of cost, the objective of a *CS* learner is to form a generalization such that the average cost on previously unobserved instances is minimized. Obviously, this minimal cost is determined by two most important factors: 1) the inductive bias of the underlying *CS* learner and 2) the quality of the training data. Existing research endeavors have made significant progress in exploring efficient *CS* learning algorithms [1], [2], [3], [4], [5], with assumptions that the input data are noise-free or noise in the data sets is not significant. However, real-world data are rarely perfect and can often suffer from corruptions that may impact interpretations of the data, models created from the data, and decisions made on the data. As a result, many research efforts have focused on noise identification and data cleansing, but none of them was originally designed for *CS* learning, with its objective of minimizing the cost instead of minimizing the number of errors. In this paper, we will propose a cost-guided noise handling approach for effective *CS* learning from noisy data sources.

- *X. Zhu is with the Department of Computer Science and Engineering, Florida Atlantic University, 777 Glades Road, Boca Raton, FL 33431. E-mail: xzhu@fau.edu.*
- *X. Wu is with the Department of Computer Science, University of Vermont, 33 Colchester Ave./ Votey 377, Burlington, VT 05401. E-mail: xwu@cs.uvm.edu.*

Manuscript received 29 Mar. 2004; revised 10 Feb. 2005; accepted 21 Mar. 2006; published online 18 Aug. 2006.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-0088-0304.

2 Cost-Guided Iterative Classification Filter

Among all existing noise handling efforts, the Classification Filter (*CF*) [6] is one of the most successful algorithms. It simplifies noise elimination as a filtering operation where noise is characterized as instances that are incorrectly classified by a set of pretrained noise classifiers, as shown in Fig. 1.

An inherent disadvantage of *CF* is that noise classifiers (H_y) are organized in parallel so they cannot benefit each other in noise identification. However, if we can organize the noise classifiers in such a way that the current classifier could take advantage of the previous results and, in addition, when conducting noise handling, each noise classifier also takes the costs into consideration, we may expect better results in identifying noise for *CS* learning. Motivated by the above observations, we propose a Cost-Guided Iterative Classification Filter (*CICF*) for noise identification.

As shown in Fig. 2, *CICF* first partitions the original noisy data set E' into n subsets (as *CF* does). Given the first iteration $It = 1$, *CICF* first aggregates $n-1$ subsets (excluding subset E_1), and trains the first noise classifier H_1. The examples in E_1, which are incorrectly classified by H_1, are forwarded to a noise subset (*NoiseSet*), and all other instances are forwarded to a good instance subset (*GoodSet*). The above steps are about the same as in *CF*. But, the difference is that after H_1 has identified noisy instances from E_1, *CICF* will introduce a cost-guided rejection sampling [9] (see Section 2.2) to remain all good instances in E_1 and randomly remove identified noise in E_1, by putting a heavier penalty to the noise in cheaper classes. So, the removed noisy instances will not take part in the following noise identification steps and, meanwhile, the noise classifier can still put an emphasis on the expensive classes. It is obvious that such a procedure will change the data distribution in E_1 through noisy instance removals. After the first round, *CICF* conducts the second iteration by aggregating another $n-1$ subsets (and excluding subset E_2), identifies noise from E_2 and conducts a cost-guided rejection sampling on E_2 as well. *CICF* repeats the above procedures until n iterations are accomplished, and provides the user with a cleansed data set E'.

2.1 Iterative Classification Filter (*ICF*)

For any current iteration It in *CICF*, the noise identification results from the former iterations i, $i = 1, 2, \ldots, It-1$, can possibly contribute to noise handling in E_{It}. Such a procedure is demonstrated in steps (12) to (21) in Fig. 2, where $Class(I_k)$ returns the class label of I_k, and $Random()$ produces a random value within [0, 1]. Under such a sequential framework, the most intuitive solution is to remove all identified noisy instances from E_{It}, $It = 1, 2, \ldots, n$, so in the following steps, the identified noise is not used to train the noise classifiers. We call it an *Iterative Classification Filter* (*ICF*) [17]. In Fig. 2, if we set $SaplChance$ to 0, the algorithm works exactly as *ICF*, and if we set $SaplChance$ to 1, the algorithm becomes *CF*.

ICF iteratively removes identified noisy examples in each round, it works similar to boosting [10] but in a counterdirection. Taking Adaboosting as an example, the data distribution in the current round D_{It} crucially depends on the classification results from the previous iterations, as denoted by (1), where Z_{It} is a

Procedure: Classification_Filter()
Input: E'(noisy dataset); **Output:** E''(cleansed dataset).
Parameter: n (# of subsets, typically 10)
(1) Form n disjoint equal-size subset E_i, where $\cup_i E_i = E'$.
(2) $NoiseSet \leftarrow \varnothing$
(3) **For** i=1, …, n
(4) Form $E_y \leftarrow E' \setminus E_i$
(5) Induce H_y based on examples in E_y.
(6) **For** every $I_k \in E_i$
(7) **If** H_y incorrectly classify I_k
(8) $NoiseSet \leftarrow NoiseSet \cup \{I_k\}$
(9) Return ($E'' \leftarrow E' \setminus NoiseSet$)

Fig. 1. Classification filter.

normalization constant (chosen so that D_{It+1} will be a distribution) and y_{Ik} is the class label of instance I_k. In (1), the priority was given to incorrectly classified examples because boosting tries to boost from uncertain examples and weaker learners, and a default assumption is that training instances are all correctly labeled [11]. Whereas for *ICF*, the objective is to boost from good examples, and the assumption is that incorrectly classified instances are essentially noise prone. In comparison with boosting, *ICF* can be denoted by (2)

$$D_{It+1}(I_k) = \frac{D_{It}(I_k)}{Z_{It}} \times \begin{cases} \frac{1-Err_{D_{It}}(H_{It})}{Err_{D_{It}}(H_{It})} & if\ H_{It}(I_k) = y_{I_k} \\ 1 & Otherwise, \end{cases} \quad (1)$$

$$D_{It+1}(I_k) = \frac{D_{It}(I_k)}{Z_{It}} \times \begin{cases} 1 & if\ H_{It}(I_k) = y_{I_k} \\ 0 & Otherwise. \end{cases} \quad (2)$$

2.2 Cost-Guided Rejection Sampling

Although *ICF* progressively reduces the noise level in each round to train the classifiers, experimental results in Section 3 will indicate that the disadvantage of *ICF* is that it boosts from good instances any falsely identified noisy instance are removed immediately and will not be reused in the following procedures. This disadvantage of *ICF* becomes especially severe when the accuracy of the noise classifier is low and the cost-ratio in the data set is relatively large because expensive classes obviously should not bear the same level of information loss as cheap ones. *CF*, on the other extreme, keeps all identified noisy instances while training the noise classifiers, which eventually results in low noise identification accuracies. Accordingly, instead of either removing or keeping identified noisy instances, we adopt a sampling mechanism by taking costs into consideration.

When conducting CS learning, some algorithms [5], [12] have used sampling mechanisms to modify the distribution of training examples with more emphasis (a higher density) on expensive classes, so the classifier learned from the modified data set becomes cost-sensitive. This motivates us to adopt similar ideas for noise handling, where at each iteration *It*, the identified noisy instances are randomly sampled and removed from the data set, therefore two objectives could be achieved through this procedure: 1) putting a focus on the expensive classes, so the classifier could have more investigation on them and 2) reducing the noise level in the succeeding iterations, so a more accurate noise identification classifier can be trained. Although well motivated, sampling instances in a correct and general manner is more challenging than it may seem. The most challenging part is to guarantee that instances are indeed independently sampled and with a probability proportional to their cost. General sampling procedures like sampling with/without replacement, however, cannot provide this

Procedure: Cost_Guided_Iterative_Classification_Filter()
Input: E'(noisy dataset); C (cost matrix). **Output:** E''(cleansed dataset).
Parameter: n (# of iterations, typically 10)
(1) Form n disjoint equal-size subset E_i, where $\cup_i E_i = E'$.
(2) $NoiseSet \leftarrow \varnothing$; $BaseSet \leftarrow \varnothing$; $GoodSet \leftarrow \varnothing$;
(3) **For** (It=1; $It \le n$; It ++)
(4) $BaseSet \leftarrow \cup_i E_i,\ i \neq It$
(5) Induce H_{It} based on examples in B.
(6) **For** each instance I_k in E_{It}
(7) **If** H_{It} correctly classifies I_k
(8) $GoodSet \leftarrow GoodSet \cup \{I_k\}$ // Good instance subset
(9) **Else**
(10) $NoiseSet \leftarrow NoiseSet \cup \{I_k\}$ // Noise subset
(11) **For** each instance I_k in E_{It}
(12) **If** H_{It} correctly classifies I_k // Keep good instances
(13) Remain I_k in E_{It}
(14) **Else** // invoke a rejection sampling for noisy instances
(15) $OgCls = Class(I_k)$
(16) $PrdCls = H_{It}(I_k)$
(17) $SaplChance = \frac{C(PrdCls, OrgCls)}{\sum_{i,j \in \{PrdCls, OrgCls\}} C(i,j)}$
(18) **If** $Random() < SaplChance$
(19) Remain I_k in E_{It} // Keep noisy instances
(20) **Else**
(21) Remove I_k from E_{It} // Reject noisy instances
(22) Return ($E'' \leftarrow GoodSet$)
Cost-guided rejection sampling

Fig. 2. Cost-guided iterative classification filter (CICF).

warranty [5]. For sampling-with-replacement, each instance (x, y, c) is drawn according to the distribution

$$p(x, y, c) = \frac{c}{\sum_{(x,y,c) \in \tilde{E}_{It}} c},$$

but there is no guarantee that instances drawn independently from $\tilde{E}_{It}$, which is the subset containing identified noise in the current round It. Sampling without replacement is not an option either because it draws an instance from the distribution

$$p(x, y, c) = \frac{c}{\sum_{(x,y,c) \in \tilde{E}_{It}} c},$$

but draws next instance from $\tilde{E}_{It} - \{x, y, c\}$ and, therefore, instances are drawn from a smaller and smaller set according to the weights of the remaining examples.

Rejection Sampling [9] from statistics provides a solution to resolve our problem. This method draws independent samples from a probability distribution $P(x) = P^*(x)/Z_P$. We may not know the normalizing constant Z_P, but we assume that we can evaluate $P^*(x)$ at any position x we choose. It does not matter here if the function $P(x)$ gives probabilities for a discrete x or describes a probability density function over a continuous x. A normal rejection sampling procedure consists of the following two steps:

- Draw a sample X from $P(x)$ and a sample U from another distribution $Q(x)$ from which we can easily make an independent selection. A uniform distribution on [0, 1] is often taken as $Q(x)$.
- Accept the sample X if $U \le P(x)$, and reject X otherwise.

2.3 Cost-Guided Iterative Classification Filter (CICF)

Based on the above analysis, we modify *ICF* and propose a Cost-guided Iterative Classification Filter. Given the current iteration It, $It = 1, 2, \ldots, n$, assuming $\tilde{E}_{It}$ denotes the subset containing all identified noisy examples from E_{It}, the original distribution in $\tilde{E}_{It}$ is D_{It}, and the transformed distribution (after the sampling) is $\hat{D}_{It}$, we conduct rejection sampling by drawing examples from $\tilde{E}_{It}$, and then keeping (or accepting) the sample with a probability proportional to $\hat{D}_{It}/D_{It}$. Here, we

TABLE 1
Benchmark Data Set Characteristics

Dataset	*# of Classes*	*Class Ratio*	*Dataset*	*# of Classes*	*Class Ratio*
Adult	2	0.76:0.24	Auto-mpg	3	0.63:0.18
Credit	2	0.56:0.44	Car	4	0.70:0.04
Krvskp	2	0.52:0.48	Glass	6	0.36:0.04
Monks-3	2	0.53:0.47	KDD99	5	$0.79{:}10^{-4}$
Mushroom	2	0.52:0.48	LED24	10	0.12:0.09
Sick	2	0.94:0.06	Lympho	4	0.55:0.01
Tictactoe	2	0.65:0.35	Nursery	5	$0.33{:}10^{-4}$
Vote	2	0.61:0.39	Soybean	19	0.13:0.01
WDBC	2	0.63:0.37	Splice	3	0.52:0.24
Wisc. Ca.	2	0.66:0.34	Wine	3	0.40:0.27

have $\hat{D}_{It}/D_{It} \propto C$, so we keep an example (I_k) with probability $SaplChance$, which is defined by (3):

$$SapleChance = C(PrdCls, OrgCls) / \sum_{i,j \in \{PrdCls, OrgCls\}} C(i,j), \quad (3)$$

where $OrgCls$ and $PrdCls$ indicate the original and the predicted class of I_k, respectively. The higher the misclassification cost of I_k, the more likely I_k is going to be kept, as shown in steps (15) to (21) in Fig. 2. With such a procedure, the identified noise in E_{It} is independently sampled (with regard to the misclassification cost of the instance) to form a modified data set E_{It} (where some identified noisy instances are removed). Accordingly, the original distribution in E_{It} was modified, with a preference on good and expensive instances. Hopefully, classifiers learned from the modified data set will have a better performance to handle noise from expensive classes.

The merit of such a rejection sampling-based noise handling mechanism is twofold: 1) rejection sampling independently draws instances to form a new distribution, and it is inherently superior to other sampling mechanisms like sampling with/without replacement, because instances are not independently drawn from the latter approaches and 2) instead of either keeping (like in *CS*) or removing (like in *ICF*) all identified noisy instances, *CICF* randomly keeps identified noise (with regard to the cost of each instance), so the noise classifier could be learned from conditionally reduced noisy environments with an emphasis on expensive instances. In comparison with boosting and *ICF*, *CICF* can be summarized as follows:

$$D_{It+1}(I_k) = \frac{D_{It}(I_k)}{Z_{It}} \times \begin{cases} 1 & \text{if } H_{It}(I_k) = y_{I_k} \\ \frac{C(PrdCls, OrgCls)}{\sum_{i,j \in \{PrdCls, OrgCls\}} C(i,j)} & Otherwise. \end{cases} \quad (4)$$

3 Experimental Evaluations

3.1 Experiment Settings

The majority of our experiments use C5.0 [13]. We evaluate different algorithms on 20 benchmark data sets [14], [15], as summarized in Table 1, where *class ratio* indicates the ratio between the most common and the least common classes. We will mainly analyze the results on several representative data sets. The summarized results from all benchmark data sets are reported in Fig. 4 and Fig. 5.

For most data sets we used, they don not actually contain much noise (at least we do not know which instances are noisy), so we implement *Total Random Corruption (TRC)* to introduce noise and evaluate the algorithms. With *TRC*, when the users specify an intended corruption level $x \cdot 100$ percent, we will randomly introduce noise to all classes. That is, an instance with its label i has a $x \cdot 100$ percent chance to be mislabeled as another random class (excluding class i). For two-class data sets, *TRC* is actually the pairwise corruption model that has been popularly used before [6], [7], [8].

To assign misclassification cost matrix values, $C(i,j), i \neq j$, we adopt a Proportional Cost (*PC*) mechanism with costs chosen at follows: For any two classes i and j ($i \neq j$), we first check their class distributions π_i and π_j. If $\pi_i \geq \pi_j$, which means that class j is relatively rarer than i, then $C(j,i) = 100$ and $C(i,j)$ equals to $100\ r$; otherwise, $C(i,j) = 100$ and $C(j,i) = 100 \cdot r$, where r is the cost-ratio, as defined by (5). In our experiments, we set $r \in [1, 10]$, which means higher costs for rarer classes. This is consistent with the reality, because assigning a large cost to the common class simply does not make sense, and if we indeed do so, the *CS* learner will tend to ignore all rare classes (because they are less in quantity and cheap in cost). In our experiments, the cost-ratio r for two-class problems are prespecified, such as $r = 2$, 5, and 10 and, for multiclass problems, the r values are randomly generated for each pair of classes. The cost matrix is generated at the beginning of each trail of 10 time cross-validation. We use average costs to evaluate the improvement of the proposed efforts in enhancing *CS* learning.

$$r = C(i,j)/C(j,i). \quad (5)$$

3.2 Cost Improvement through Noise Cleansing

To assess different methods in improving the *CS* learner performances, we perform experiments at different noise levels and report the results in Table 2, where CS_{wt} represents the average cost of a *CS* classifier trained from a noisy data set (E') without any noise cleansing. CS_{CF}, CS_{ICF}, and CS_{CICF} denote the average costs of the CS classifier learned from the cleansed data set (E'') which adopts *CF*, *ICF*, and *CICF*, respectively. In Table 2, the text in bold indicates the lowest cost value.

When evaluating results from two-class data sets (in Table 2), it is not hard for us to conclude that a data set with noise cleansing (either *CF*, *ICF*, or *CICF*), likely improves the *CS* learner considerably, where the improvement can be as significant as 10 times better (e.g., *ICIF* on the Sick data set with 40 percent noise). This shows the effectiveness and importance of noise handling for *CS* learning.

For small cost-ratio values, the improvements could be found from almost all data sets in Table 2, regardless of noise levels, where noise handling almost always leads to better performances. However, when the cost-ratio becomes larger, e.g., $r = 5$ or more, the improvement turns to be less significant. In this situation, applying noise handling on some data sets (at some noise levels) may raise the average costs of trained *CS* classifiers. For data sets with large cost-ratio values, it is actually hard to achieve a significant improvement through noise cleansing (due to the bias of *CS* learners). Under such circumstances, a less accurate noise cleansing mechanism will decrease the system performance rather than enhancing it.

Further investigations on *CF*, *ICF*, and *CICF* reveal that *ICF* appears to be very unstable. A possible reason is that *ICF* prohibits

TABLE 2
Cost Improvements from Two-Class Data Sets (TRC, $n = 10$)

Dataset	Noise (%)	r=2				r=5				r=10			
		CS_{wt}	CS_{CF}	CS_{ICF}	CS_{CICF}	CS_{wt}	CS_{CF}	CS_{ICF}	CS_{CICF}	CS_{wt}	CS_{CF}	CS_{ICF}	CS_{CICF}
Credit	10	21.88	20.66	20.79	**19.42**	34.73	32.47	36.26	**32.16**	52.67	**51.15**	52.65	51.49
	20	29.16	21.84	23.27	**20.86**	46.54	**32.15**	41.97	32.22	55.48	51.44	55.01	**48.42**
	30	34.74	29.33	30.79	**28.07**	54.70	**35.95**	42.56	36.89	55.51	51.62	56.37	**51.33**
	40	46.05	44.58	46.42	**43.21**	55.52	55.26	58.16	**51.87**	55.48	64.08	67.38	**53.65**
Monks-3	10	4.25	2.38	**2.28**	2.30	14.66	4.15	4.11	**2.96**	37.39	6.49	4.92	**3.89**
	20	10.89	**3.54**	4.17	**3.54**	43.88	6.08	4.61	**4.33**	51.90	13.68	13.06	**12.02**
	30	26.88	**19.68**	20.93	22.42	52.77	25.09	28.11	**23.58**	52.75	41.76	50.20	**36.28**
	40	49.01	**41.02**	48.03	43.45	**52.72**	74.68	80.16	56.70	**52.82**	97.99	98.92	80.57
Sick	10	3.57	**3.21**	3.34	3.26	14.71	6.57	6.55	**6.14**	68.14	10.93	11.31	**10.34**
	20	4.26	**3.41**	3.54	3.53	87.75	6.25	6.42	**6.12**	92.43	11.64	13.72	**11.03**
	30	10.84	**4.52**	5.1	4.53	93.68	8.02	7.69	**7.64**	93.77	14.24	18.32	**13.02**
	40	92.44	9.91	10.55	**9.37**	93.83	21.64	22.66	**20.03**	93.87	36.91	39.52	**32.17**
Vote	10	8.93	**6.49**	7.41	6.82	17.81	7.49	10.42	**7.07**	34.67	**12.02**	14.84	**12.02**
	20	12.73	7.16	6.75	**6.47**	41.04	10.18	10.96	**8.16**	58.31	11.70	13.14	**10.08**
	30	27.24	9.41	**8.76**	10.13	59.37	15.69	16.63	**14.25**	61.36	24.74	20.38	**18.95**
	40	46.67	27.78	**26.36**	28.08	61.51	44.73	44.71	**37.71**	61.41	46.87	58.11	**37.79**
WDBC	10	15.34	**11.39**	11.64	11.45	28.11	19.24	20.60	**18.06**	53.16	34.55	34.45	**32.37**
	20	20.66	13.44	**12.99**	13.49	50.08	24.01	24.92	**22.45**	59.48	38.45	38.35	**34.27**
	30	34.60	18.65	19.64	**17.45**	59.68	32.33	34.51	**31.70**	62.15	52.78	58.37	**47.19**
	40	60.87	37.29	**36.14**	**36.14**	**62.30**	80.43	74.68	65.44	**62.79**	116.35	124.89	101.41

TABLE 3
Experimental Comparisons with Bagging and Boosting (*TRC*, $B = 10$, $n = 10$)

Dataset	Noise (%)	r=2				r=5				r=10			
		CS_{wt}	$CS_{Bagging}$	CS_{Bsting}	CS_{CICF}	CS_{wt}	$CS_{Bagging}$	CS_{Bsting}	CS_{CICF}	CS_{wt}	$CS_{Bagging}$	CS_{Bsting}	CS_{CICF}
Credit	10	20.83	28.82	52.03	**18.77**	34.25	33.30	98.38	**31.94**	52.33	47.91	108.96	**51.03**
	20	29.58	36.47	56.17	**20.42**	45.88	42.25	95.12	**32.47**	55.52	55.52	105.18	**49.02**
	30	34.12	48.42	63.26	**27.12**	55.13	48.08	106.2	**34.13**	55.52	55.52	79.22	**50.09**
	40	47.75	59.91	64.99	**43.39**	55.50	64.04	89.61	**50.55**	55.43	55.43	85.41	**53.34**
Sick	10	3.38	10.17	27.24	**3.22**	13.92	22.26	40.23	**6.19**	65.56	35.72	48.85	**9.12**
	20	4.82	17.59	50.43	**3.39**	88.01	71.81	70.41	**6.43**	93.35	81.64	78.42	**11.98**
	30	11.34	32.48	76.21	**4.69**	93.48	86.17	86.83	**7.71**	93.81	92.08	93.61	**13.79**
	40	91.66	83.30	85.02	**9.04**	93.88	93.14	92.61	**18.58**	93.88	93.71	93.82	**33.02**
WDBC	10	17.25	21.11	33.94	**12.48**	28.76	35.15	69.64	**17.69**	52.59	39.12	104.58	**30.77**
	20	22.57	31.52	48.96	**13.09**	49.98	41.83	93.39	**22.12**	60.59	53.48	112.82	**35.63**
	30	33.78	44.54	58.53	**17.73**	59.36	49.04	81.51	**31.77**	62.71	64.20	94.75	**47.36**
	40	59.87	58.96	62.56	**36.01**	62.67	62.11	80.02	**64.46**	**62.83**	62.86	62.79	102.86

noise identified in each round from being reused at later stages, which may result in a certain amount of bias and make the noise classifier too conservative. This disadvantage becomes more apparent for data sets with a large cost-ratio because removing valuable information from expensive classes can significantly impact on the *CS* learner. In conclusion, removing all noisy instances identified in each round is an inferior solution to training noise classifiers for *CS* learning.

By integrating cost-guided rejection sampling and iterative noise handling, *CICF* receives very attractive results. For small cost ratio values, *CICF* is comparable to *CF*. When the cost ratio becomes higher, e.g., $r = 5$ or more, *CICF* dominates and outperforms *CF* most of times. This demonstrates that a noise handling mechanism which takes the cost into consideration is promising for *CS* learning, especially when some classes are much more expensive than others.

3.3 Experimental Comparisons with Bagging and Boosting

To compare the performances of bagging, boosting, and *CICF*, we implement another set of experiments: Given a noise corrupted data set E' with $||E'||$ instances, we train a *CS* classifier from E', with its misclassification cost denoted by CS_{wt}. Meanwhile, we construct a total number of B bags with each bag containing $||E'||$ instances randomly sampled from E' (with replacement). We built one *CS* classifier from each bag, and their voting result is denoted by $CS_{Bagging}$. To build a cost-sensitive boosting classifier, we follow the algorithm design of Adaboosting and randomly sample $||E'||$ instances from E' (with replacement) to generate bootstrap samples, where the sampling rate for each instance is controlled by the fact whether this instance can be correctly classified or not [10]. We build one *CS* classifier from the bootstrap samples, and repeat the same procedure for B times or until the classification error rate is no less than 0.5. The *CS* boosting classifier, CS_{Bsting}, is the voting of all classifiers built from the bootstrap samples. We report the results in Table 3, where the bold text and the shading rectangle boxes indicate the methods with the least and the second least cost, respectively (because of cross-validation and random procedures, the results of CS_{wt} and *CICF* in Table 3 and Table 2 are slightly different but statistically equivalent). We set $B = 10$ in our experiments.

In Table 3, the merit of *CICF* in comparison with bagging and boosting is quite clear. Boosting has the worst performance most of the time, and its misclassification costs are significantly higher than the others. This does not surprise us because noisy examples tend to be misclassified, receive a higher sampling rate, and they appear more often in the bootstrap samples. As shown in Fig. 3a, although noise levels for bagging are almost the same for all bags, for boosting, the noise level in the second round raises dramatically, and keeps increasing in the succeeding rounds.

The results in Table 3 show that bagging is likely inferior to the classifier built from the original noisy data sets (CS_{wt}), especially when the cost ratio (r) is small, say $r = 2$. When the cost ratio r becomes larger, bagging is comparable to CS_{wt} or even slightly better, e.g., when $r = 10$. This is surprising, as bagging independently generates bootstrap examples without considering the results of other rounds, and it should not suffer from the same problem as boosting does. In addition, existing research often [16]

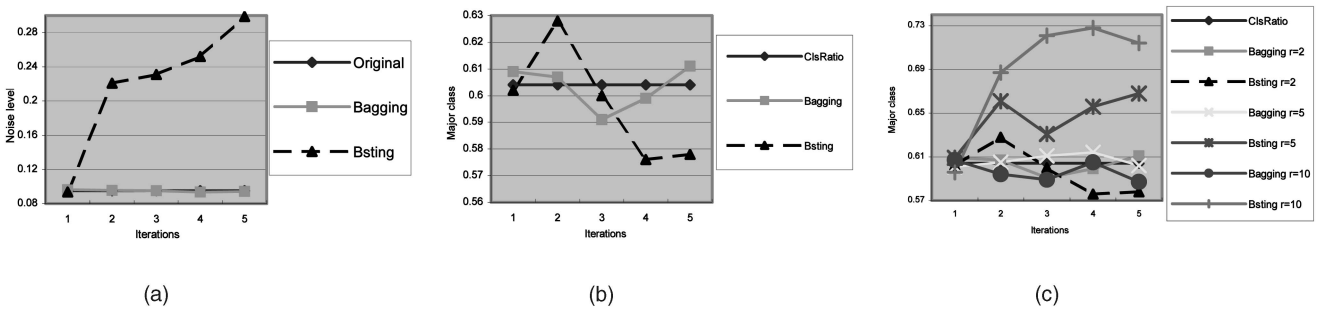

Fig. 3. Experimental comparisons with noise levels and class distributions in different rounds of bootstrap examples (*WDBC* data set, *TRC* noise corruption, noise level $x = 0.1$, $n = 10$). (a) Noise level in bootstrap samples. (b) Class distribution in bootstrap samples ($r = 2$). (c) Class distribution in bootstrap samples.

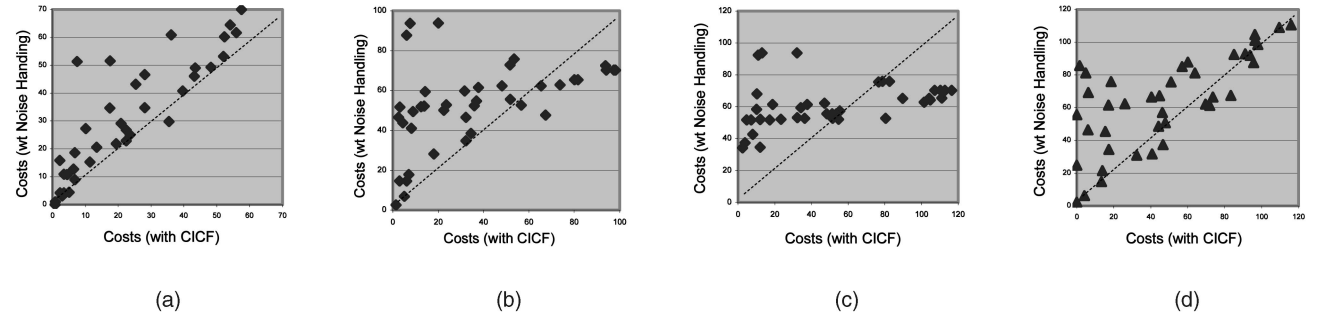

Fig. 4. Experimental summaries of cost improvements between *CICF* and no noise handling mechanism (20 data sets). (a) Two class ($r = 2$). (b) Two class ($r = 5$). (c) Two class ($r = 10$). (d) Multicalss (random r).

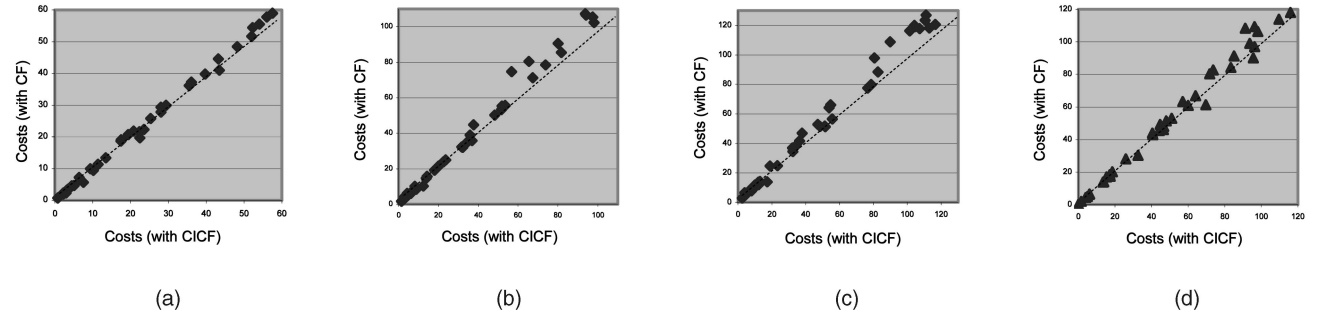

Fig. 5. Experimental comparisons of cost improvemet between *CF* and *CICF* from all 20 benchmark data sets. (a) Two class ($r = 2$). (b) Two class ($r = 5$). (c) Two class ($r = 10$). (d) Multicalss (random r).

concludes that bagging is robust and likely outperforms a single learner most of the time. To explore the reason behind this surprise, we further investigate the class distributions of bootstrap instances in each round, and report the results in Fig. 3b and Fig. 3c, where *ClsRatio* means the class ratio of the major class in E', "Bagging" and "Bsting" means the class ratio of the major class in different rounds of bootstrap examples for bagging and boosting, respectively (and we only report the results of first five rounds).

As shown in Fig. 3b, for each bootstrap bag built from bagging and boosting, its class ratio has been changed and is different from the class ratio of the original data set (E'). For non-CS learning, although the introduced class distribution bias does impact on the underlying learner, the influence is likely limited and tends to be ignorable after the final voting. For CS classification, however, the bias normally brings a heavy impact and often results in deteriorated CS learners and may eventually corrupt the bagging results. In Fig. 3c, the larger the cost ratio, the more severe the bias introduced to boosting is, and this observation alone also partially explains the ineffectiveness of boosting in noisy environments. Although the cost ratio has nothing to do with the underlying class distribution of bagging, it is understandable that with a certain amount of bias, the smaller the class ratio, the more the class distribution bias may impact on the learner, and as a result, the worse the performance of bagging is. This conclusion can be further verified in Table 3 by comparing the results of different r values.

3.4 Experimental Result Summarization

In Fig. 4 and Fig. 5, we summarize the results from all benchmark data sets and provide two sets of comparisons: 1) the cost between *CICF* and the original noisy data set (Fig. 4) and 2) the cost between *CF* and *CICF* (Fig. 5). In Fig. 4, the x-axis and y-axis represent the cost of the *CS* classifier trained from the noisy data set with and without adopting *CICF*, respectively. Each point corresponds to a data set evaluated at one noise level and one type of cost matrix (so each figure has 40 points). Points above the $y = x$ line are those receiving a better performance by adopting *CICF*. In Fig. 5, the x-axis and y-axis represent the cost of the *CS* classifier trained from the data set cleansed by *CICF* and *CF*, respectively.

In Fig. 4a, when the cost-ratio is small, adopting *CICF* can improve the performance of the *CS* classifier from almost all data sets. However, when r increases from 2 to 5, 10 points are actually below $y = x$. When r becomes 10, this number raises to 14, which means that about 35 percent (14/40) points receive negative impacts. The results in Fig. 4d show that, on average, the performance from multiclass data sets are less attractive than

two-class data sets. Among all 40 points, there are 13 points below the $y = x$ line, which is 32.5 percent. Further analysis indicates that, on average, the noise identification precision for multiclass data sets is less accurate, in comparison with two-class data sets. As a result, noise cleansing likely removes valuable information and negatively impacts on the *CS* learner. In addition, we randomly assign cost-ratio values r to multiclass data sets, which may result in relatively large cost-ratio values.

The comparisons between *CICF* and *CF* in Fig. 5 lead to a conclusion that the higher the cost-ratio, the more obvious *CICF* dominates and outperforms *CF*. For small r values, e.g., $r = 2$, *CICF* and *CF* are comparable. However, when the cost-ratio becomes higher, *CICF* tends to outperform *CF*. In these situations, the improvement from *CICF* (in comparison with *CF*) becomes significant. The above observation supports our motivation in designing *CICF*. When handling noise for *CS* learning, one should pay more attention to expensive classes and prevent noise classifiers from misidentifying noise from expensive classes. The higher the cost-ratio, the more necessary we need to do so. With cost-guided rejection sampling, *CICF* inherently attains this goal through unique procedures: 1) iteratively reducing the overall noise level by removing more suspicious instances in cheap classes and 2) putting more efforts on investigating noise in expensive classes.

4 Conclusions

In this paper, we have proposed a Cost-guided Iterative Classification Filter (*CICF*) to identify and remove noise for effective *CS* learning. Two novel features make *CICF* distinct from existing approaches. First, it provides a general framework which seamlessly integrates the misclassification cost of the instance for noise handling, with expensive classes receiving more attention. It can be easily demonstrated that this framework is general enough to accommodate any existing noise handling efforts to make them cost-sensitive in handling noise. Second, *CICF* iteratively involves noise identification results in earlier iterations to train the noise classifier in the current round. As a result, the system can progressively improve the accuracy of noise identification and eventually contribute to *CS* learning. Experimental results and comparative studies have demonstrated that one can significantly reduce the average cost of a *CS* classifier by applying *CICF* to the underlying noisy data sources.

Acknowledgments

This research was partially supported by US NSF under grant number CCF-0514819. A preliminary version of this paper was published in the *Proceedings of International Conference Data Mining 2004* [17].

References

[1] M. Tan, "Cost-Sensitive Learning of Classification Knowledge and Its Applications in Robotics," *Machine Learning,* vol. 13, pp. 7-33, 1993.
[2] P. Turney, "Cost-Sensitive Classification: Empirical Evaluation of a Hybrid Genetic Decision Tree Induction Algorithm," *J. AI Research,* vol. 2, pp. 369-409, 1995.
[3] M. Pazzani, C. Merz, P. Murphy, K. Ali, T. Hume, and C. Brunk, "Reducing Misclassification Costs," *Proc. 11th Int'l Conf. Machine Learning,* 1994.
[4] P. Domingos, "MetaCost: A General Method for Making Classifiers Cost Sensitive," *Proc. Fifth Int'l Conf. Knowledge Discovery and Data Mining,* 1999.
[5] B. Zadrozny, J. Langford, and N. Abe, "Cost-Sensitive Learning by Cost-Proportionate Example Weighting," *Proc. Third Int'l Conf. Data Mining,* 2003.
[6] C. Brodley and M. Friedl, "Identifying Mislabeled Training Data," *J. AI Research,* vol. 11, pp. 131-167, 1999.
[7] X. Zhu, X. Wu, and Q. Chen, "Eliminating Class Noise in Large Data Sets," *Proc. 20th Int'l Conf. Machine Learning,* 2003.
[8] X. Zhu, X. Wu, and Q. Chen, "Bridging Local and Global Data Cleansing: Identifying Class Noise in Large, Distributed Data Data sets," *Data Mining and Knowledge Discovery,* vol. 12, no. 3, 2006.
[9] J. Von Neumann, "Various Techniques Used in Connection with Random Digits," Nat'l Bureau of Standards Applied Math. Series 12, pp. 36-38, 1951.
[10] Y. Freund and R. Schapire, "Experiments with a New Boosting Algorithm," *Proc. 13th Int'l Conf. Machine Learning,* 1996.
[11] A. Krieger, C. Long, and A. Wyner, "Boosting Noisy Data," *Proc. 18th Int'l Conf. Machine Learning,* pp. 274-281, 2001.
[12] P. Chan and S. Stolfo, "Toward Scalable Learning with Non-Uniform Class and Cost Distributions," *Proc. Int'l Conf. Knowledge Discovery and Data Mining,* 1998.
[13] J. Quinlan http://rulequest.com/see5-info.html, 1997.
[14] C. Blake and C. Merz UCI Data Repository, 1998.
[15] S. Hettich and S. Bay The UCI KDD Archive, 1999.
[16] L. Breiman, "Bagging Predictors," *Machine Learning,* vol. 24, no. 2, pp. 123-140, 1996.
[17] X. Zhu and X. Wu, "Cost-Guided Class Noise Handling for Effective Cost-Sensitive Learning," *Proc. Fourth Int'l Conf. Data Mining,* 2004.