SCIENCE CHINA

December 2012 Vol. 55 No. 12: 1–18
doi:

Misleading Classification

JIANG He[1,2*], XUAN JiFeng[1], REN ZhiLei[1], WU YouXi[3] & WU XinDong[2]

[1]*School of Software, Dalian University of Technology, Dalian* 116621*, China,*
[2]*Computer Science Department, University of Vermont, Burlington, Vermont* 05403*, USA,*
[3]*School of Computer Science and Software, Hebei University of Technology, Tianjin* 300130*, China*

Received August xx, 20xx; accepted xx xx, 20xx

Abstract In this paper, we investigate a new problem–misleading classification in which each test instance is associated with an original class and a misleading class. Its goal for the data owner is how to form the training set out of candidate instances, such that the data miner will be misled to classify those test instances to their misleading classes rather than original classes. We discuss two cases of misleading classification. For the case that the classification algorithm is unknown to the data owner, a KNN based Ranking Algorithm (KRA) is proposed to rank all candidate instances based on the similarities between candidate instances and test instances. For the case that the classification algorithm is known, we propose a Greedy Ranking Algorithm (GRA) which evaluates each candidate instance by building up a classifier to predict the test set. In addition, we also show how to accelerate GRA in an incremental way when Naive Bayes is employed as the classification algorithm. Experiments on 16 UCI data sets indicated that the ranked candidate instances by KRA can achieve promising leaking and misleading rates. When the classification algorithm is known, GRA can dramatically outperform KRA in terms of leaking and misleading rates though more running time is required.

Keywords misleading classification, naive Bayes, K-nearest neighbor

Citation Jiang H, Xuan J F, Ren Z L. et al. Misleading classification. Sci China Inf Sci,

1 Introduction

In the traditional research of data mining, a data owner may conduct the classification task on the data set by himself or some trusted data miners. However, in many applications, a rival may act as the data miner who collects the data set and conducts the classification task on it. For example, company A may collect the marketing report from company B's website and conduct the classification task on it to figure out the sale plan of B. For the data owner, a passive way to prevent the rival from doing so is to introduce some erroneous information into his data set. However, this passive way doesn't always work (e.g., it's illegal for a public company to announce a wrong marketing report on the website).

In this paper, we investigate a new model named *misleading classification* which can tackle this challenge in a more proactive way. The idea of misleading classification for the data owner is to distribute a part of the data set on purpose instead of the whole data set, such that the classifier built on the distributed part (i.e., the training set) will classify those test instances to some misleading classes rather than their original classes. For brevity, the whole data set is also called the *candidate set* hereafter and every instance in the data set is called a *candidate instance*. Since it is chosen from the data owner's

*Corresponding author (email: hejiang@ieee.org)

candidate set, every instance in the training set can be verified. Therefore, these "true" instances can mislead the rival to a wrong way as the data owner desires.

Many applications related to misleading classification can be found in business competition, patent protection, combat intelligence, etc.

- *Business Competition* In business, one company usually needs to distribute a lot of business reports on many publicly accessible sources. Meanwhile, it's also essential to prevent the rival from mining the hidden commercial knowledge, i.e. new market plan, sale trend, new product design, etc.

- *Patent Protection* In patent protection, the patent owner should provide the agency (e.g. [1]) with some publicly accessible descriptions about his new patent. The patent owner can actively protect his innovation by misleading the rivals to some wrong research directions if they conduct the classification task on those descriptions.

- *Combat Intelligence* It's very common in wars to mislead the enemy by transferring some wrong combat intelligence information. To achieve such goals, those experts usually leak some misleading information accompanied with a few pieces of true (or verifiable) information. A famous example is the "Operation Mincemeat" [2] during World War II, a successful deception plan to convince German that the Allies planned to invade Greece and Sardinia rather than Sicily.

In misleading classification, the data owner need not only mislead the rival but also protect those true hidden knowledge. Therefore, two measures are adopted in misleading classification, i.e., the leaking and misleading rates, which are defined as the fractions of test instances classified to their original and misleading classes respectively. Under these measures, misleading classification aims to maximize the misleading rate while minimizing the leaking rate.

All algorithms in this paper are designed to rank those candidate instances such that the top ranked instances are more apt to achieve higher misleading and lower leaking rates. So the data owner can determine by himself which part of instances should be chosen for distribution. In this paper, we discuss two cases of misleading classification with respect to whether or not the data owner knows the classification algorithm used by the rival.

For the first case that the classification algorithm is unknown to the data owner, the KNN based Ranking Algorithm (KRA) is proposed. Every candidate instance is associated with a score which represents its contribution to misleading classification. The score will be adjusted according to the correlation between the candidate instance and all test instances. When this candidate instance is among the K nearest neighbors of a test instance, its score will be incremented (decremented) if its class equals to the misleading (original) class of the test instance. All candidate instances will be eventually ranked by their scores in descending order.

For the case that the classification algorithm is known to the data owner, we propose the Greedy Ranking Algorithm (GRA) which ranks those candidate instances, one by one. GRA consists of two phases. After the ranked list is set to empty in the initialization phase, a series of iterations will be conducted in the greedy phase. For each iteration, the ranked list will be updated greedily with such a candidate instance that the classifier built on it can achieve the maximum gap between the misleading rate and the leaking rate. In addition, we show how to effectively accelerate GRA in an incremental way when Naive Bayes is employed as the classification algorithm.

Experiments on 16 UCI data sets demonstrated that those ranked candidate instances generated by KRA can achieve promising leaking and misleading rates when compared with randomly ranked ones. When the classification algorithm is known, GRA can dramatically outperform KRA in terms of both leaking and misleading rates though more running time is needed. In addition, the accelerated version of GRA is much faster than GRA, e.g., the accelerated one is about 15 times faster than GRA on the data set *glass*.

2 Related Work

There are some similar problems in the literature, including inverse classification [3], adversarial classification [4], feature selection [5,6,7], and privacy preserving classification [8,9,10].

Aggarwal C C, Chen C, and Han J[3] proposed the problem of inverse classification, which investigates how to set some missing values in a test instance such that the test instance will be classified to some predefined class. In inverse classification, the training set are given a priori and the classification algorithm is uncertain. Misleading classification is different from inverse classification in that no value of a test instance will be changed in misleading classification. In addition, misleading classification needs to select some candidate instances to form the training set.

Dalvi N, Domingos P, et al. [4] investigated the problem of adversarial classification, which can be viewed as a game between the classifier and the adversary. They aims to design an automatic way to adjust the classifier to the adversary's evolving manipulations. Misleading classification differs from adversarial classification in that it focuses on how to form the training set rather than to dynamically adjust the classifier.

In classification, the goal of feature selection (e.g. [5,6,7]) is to remove most irrelevant and redundant features from the training and test sets, such that the performance of the classifier can be improved in terms of speed, interpretability, and generalization capability. Misleading classification doesn't delete or update any feature but select a fraction of those candidate instances to form the training set.

In privacy preserving classification[8], a common technology is to randomly update the data in certain ways (e.g. [9,10]) so as to disguise some sensitive information while preserving the critical data property for building the classifier. It is different from misleading classification in several aspects. Firstly, the data owner usually hopes the data miner to find some hidden knowledge in privacy preserving classification, while the data owner aims to mislead the data miner (the rival) in misleading classification to some desired directions. Secondly, in misleading classification, the data owner will select a part of those candidate instances rather than change them.

3 Notations

In misleading classification, there are two roles, i.e., the data owner and the rival.

- *Data Owner* The data owner has all instances at hand, including candidate instances and test instances. The data owner attempts to construct the training set from the candidate set so as to mislead the rival.

- *Rival* In misleading classification, the rival acts as a data miner to build a classifier upon the training set for predicting those test instances.

Given a tuple (C, T), where C is the *candidate set* consisting of N_C instances $x_1, x_2, \ldots, x_{N_C}$, T is the *test set* of N_T instances $y_1, y_2, \ldots, y_{N_T}$. Each instance is represented by a vector of M features $(F_1, F_2, \ldots, F_M)$, where each feature F_i $(1 \leqslant i \leqslant M)$ may take N_{F_i} values $f_{i1}, f_{i2}, \ldots, f_{iN_{F_i}}$. Let N_F be the sum of N_{F_i} $(1 \leqslant i \leqslant M)$, i.e., $N_F = \sum_{i=1}^{M} N_{F_i}$. Every instance x_i in C is assigned to a class $c_{x_i} \in L$, where L is the class set of N_L classes $c_1, c_2, \ldots, c_{N_L}$. Every instance y_i in T is associated with two classes, i.e., the *original class* (denoted by c_{y_i}) and the *misleading class* (denoted by $\hat{c}_{y_i}$), where $c_{y_i}, \hat{c}_{y_i} \in L$ and $c_{y_i} \neq \hat{c}_{y_i}$. In this paper, we assume that those test instances with original classes follow the same distribution as those ones in the candidate set C.

Let C_s be a subset of C. Let A be the classification algorithm used by the rival. The *leaking rate* $\lambda_l(C_s, A)$ is defined as the extent to which the rival finds the original classes of instances in T, i.e., $\lambda_l(C_s, A) = N_A/N_T$, where N_A is the number of test instances being predicted by A to their original classes. Similarly, the *misleading rate* $\lambda_m(C_s, A)$ is defined by $\lambda_m(C_s, A) = \hat{N}_A/N_T$, where $\hat{N}_A$ is the number of test instances classified to misleading classes. Obviously, we have that $\lambda_l(C_s, A) + \lambda_m(C_s, A) =$

1 when there are only two classes in L (i.e., $N_L = 2$), and $\lambda_l(C_s, A) + \lambda_m(C_s, A) \leqslant 1$ when more than two classes are contained in L.

Under the above definitions, the goal of misleading classification for the data owner is how to mislead the rival to predict those instances in T to their misleading classes rather than original ones. More formally, misleading classification aims to select a subset $C_{s*} \subseteq C$ such that the misleading rate $\lambda_m(C_{s*}, A)$ is maximized and the leaking rate $\lambda_l(C_{s*}, A)$ is minimized.

In this paper, we consider the following two cases.

- *Classification Algorithm Unknown:* For this case, the data owner is assumed to know nothing about the classification algorithm used by the rival. For some applications, the rival may design some new classification algorithms which are unknown for the data owner. In addition, the data owner may even face a new rival who never appears before.

- *Classification Algorithm Known:* In this case, the data owner is assumed to know the classification algorithm used by the rival. In many applications (e.g., patent protection, combat intelligence), the data owner may also collect data from the rival and conduct the classification task. Under such circumstances, the data owner turns to be a "data miner" against the rival. Therefore, the data owner may forecast which kind of classification algorithm will be conducted by the rival.

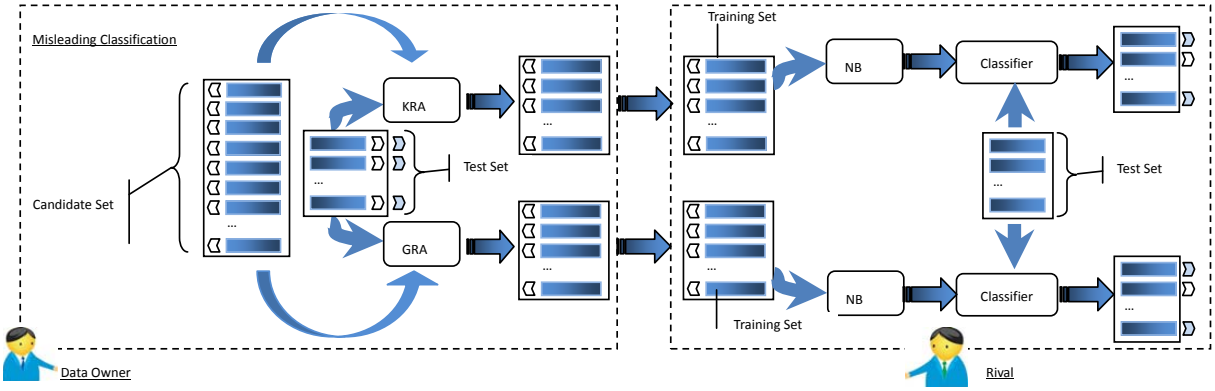


Figure 1: Scenario of Misleading Classification.

To present the difference of misleading classification and classification, we illustrate the scenario of misleading classification in Figure 1. In Figure 1, there are two roles, namely the data owner and the rival. The data owner possesses all the candidate instances and the test instances (test instances will be used by the rival), whereas the rival will collect all the distributed instances as the training set and predict the class labels for the test instances. The data owner wishes that every test instance will be predicted by the rival as its misleading class, rather than its original class. When the data owner doesn't know the classification algorithm used by the rival, KRA is proposed to choose some candidate instances out and distribute publicly. In such a case, the rival will collect the distributed candidate instances and form the training set to set up a classifier and predict all the class labels for test instances (see the right part of Figure 1). In this paper, we assume that the rival uses Naive Bayes as the classification algorithm. When the data owner knows the classification algorithm (i.e., Naive Bayes in this paper), GRA is proposed to choose some candidate instances by the data owner and distribute publicly. Then, the rival will collect the distributed candidate instances and form the training set to set up a classifier and predict all the class labels for test instances.

4 Classification Algorithm Unknown

A candidate set C consisting of N_C instances has totally $\binom{N_C}{0} + \binom{N_C}{1} + \ldots + \binom{N_C}{N_C} = 2^{N_C}$ subsets. Obviously, the total number of subsets are so large that it's intractable in practice to evaluate every

subset. For example, there will be about $1.07 * 10^{301}$ subsets even when $N_C = 1000$, a quite small number of instances in real applications.

Table 1: Pseudocode of KRA

Algorithm: KNN based Ranking Algorithm
Input: candidate set C, test set T, K
Output: ranked list R

Begin
1 let $R = \emptyset$
2 for every instance $x_i \in C$ do $score(x_i) = 0$
3 for every instance $y_i \in T$ do
  3.1 let $C_N = \emptyset$
  3.2 add y_i's K nearest neighbors in C to C_N
  3.3 for every instance $x_j \in C_N$ do
    3.3.1 if $c_{x_j} = \hat{c}_{y_i}$ then $score(x_j) = score(x_j) + 1$
    3.3.2 if $c_{x_j} = c_{y_i}$ then $score(x_j) = score(x_j) - 1$
4 sort all candidate instances by scores in descending order and add the result to R
5 return R
End

On the other hand, those two objective functions (i.e., the leaking and misleading rates) may conflict with each other in practice. For example, given two subsets $C_{s1}, C_{s2} \subseteq C$, it's hard to determine which one is better than the other, if both the leaking and misleading rates of C_{s1} are lower than those of C_{s2} respectively. Therefore, all new algorithms in this paper are designed to rank all candidate instances such that the top ranked instances are more likely to achieve a higher misleading rate and lower leaking rate than those following ones. By such a way, the data owner could determine by himself which fraction of candidate instances should be chosen out for distribution.

For the case that the data owner cannot forecast which classification algorithm will be adopted, we propose an algorithm named KNN based Ranking Algorithm (KRA). The intuition of KRA is to use the similarity between test instances and candidate instances to guide the ranking process. In this paper, we use the K-nearest neighbor algorithm (KNN) [11] to calculate the similarities between instances.

As shown in Table 1, every candidate instance is associated with a score, which is initialized to be 0 (see Step 2 in KRA). Then, we check every test instance y_i as follows (see Step 3 in KRA). Firstly, KNN is employed to find the K nearest neighbors of y_i in the candidate set. Secondly, we adjust every neighbor's score according to the correlation between its class and y_i's classes. If its class is identical to y_i's misleading class, then the score will be incremented, since it implies this neighbor may contribute to misleading the rival to the desired (misleading) class. On the other hand, the score will be penalized by decrementing 1 if the neighbor's class is the same as y_i's original class. By this penalty, this neighbor will be less likely to be chosen out for constructing the training set. Under the above discussion, it's easy to see that a higher score indicates more contribution to misleading classification, while a lower score implies more risk to leak the original classes of those test instances. After all test instances have been processed, a sorting procedure will be performed to rank all candidate instances. The instance with the highest score will be ranked top 1, the one with the lowest score will be ranked as the last 1. All the ranked instances will be returned in a list as the output of KRA.

The running time of KRA can be estimated as follows. Step 1 and 5 can be conducted in O(1) running time, respectively. Step 2 takes $O(N_C)$ running time. The main part of Step 3 could be completed in $O(N_C N_T)$ time. When a quick sort method [12] is used, Step 4 can be executed in $O(N_C \log N_C)$ time. Therefore, the total running time of KRA is $O(N_C N_T + N_C \log N_C)$.

5 Classification Algorithm Known

5.1 Greedy Ranking Algorithm

In this section, we discuss the case that the classification algorithm A can be forecast by the data owner. Since the classification algorithm is available, we can apply it directly to rank those candidate instances. Activated by this idea, a new algorithm named Greedy Ranking Algorithm (GRA) is presented in this section (see Table 2).

In contrast to KRA, GRA constructs the ranked list, one instance by one instance. GRA consists of two phases. In the initialization phase, the ranked list R will be set to be empty. In the greedy phase, the "best" instance from the candidate set will be chosen out and added to the ranked list. The way to determine the "best" instance is done as follows. We simply evaluate every instance in the candidate set by combining it together with the current ranked list to build up a new classifier. Therefore, the "best" instance is the one resulting in the "best" classifier. Obviously, a classifier could be evaluated by its misleading and leaking rates on the test set. Since those two rates may conflict with each other, in GRA we introduce a new measure–the gap between the misleading rate and the leaking rate. A positive value of the gap implies that the classifier could mislead the rival more than what is leaked. After the "best" one among all instances in the candidate set is singled out, it will be removed from the candidate set and added to the ranked list. The process of seeking for the "best" instance will be repeated until the candidate set is empty (i.e., all candidate instances have been ranked).

Obviously, the main running time of GRA is spent on the greedy phase. In the inner loop of the greedy phase, a new classifier will be built for every instance in the candidate set. In addition, this newly built classifier will be employed to classify all the instances in the test set. Therefore, the greedy phase of GRA will totally build $N_C + (N_C - 1) + \ldots + 1 = N_C(N_C + 1)/2$ classifiers and classify the test set for $N_C(N_C + 1)/2$ times. However, it should be noted that the overall running time of GRA could be dramatically reduced since the greedy phase could be implemented in an incremental way. In the remaining part of this section, we shall present an accelerated version GRA when Naive Bayes is applied as the classification algorithm.

Table 2: Pseudocode of GRA

Algorithm: Greedy Ranking Algorithm
Input: candidate set C, test set T, classification algorithm A
Output: ranked list R

Begin
// initialization phase
1 let $R = \emptyset$, let $j = 0$

// greedy phase
2 while $C \neq \emptyset$ do
  2.1 let $max_{gap} = -\infty$
  2.2 for every instance $x_i \in C$ do
    2.2.1 let $C_s = R \cup \{x_i\}$, build a classifier $A_{x_i}$with A on C_s
    2.2.2 classify all instances in the test set T to compute the misleading rate $\lambda_m(C_s, A)$, the leaking rate $\lambda_l(C_s, A)$
    2.2.3 let $gap = \lambda_m(C_s, A) - \lambda_l(C_s, A)$
    2.2.4 if $gap > max_{gap}$ then
      $max_{gap} = gap$
      $best = x_i$
  2.3 $R[j] = best$
  2.4 $j = j + 1$
  2.5 $C = C \backslash best$
3 return R
End

5.2 Naive Bayes

Naive Bayes (NB) method is a well known classification algorithm based on the Bayes theorem with independence assumption that the presence of a feature of a class is unrelated to the presence of any other feature. Given an instance $y = (f_1, f_2, \ldots, f_M)$, its class will be predicted by NB as

$$\text{argmax}_{c_i}\, p(c = c_i) \prod_{j=1}^{M} p(F_j = f_j | c = c_i) \qquad (1)$$

In practice, those class probabilities $p(c = c_i)$ $(1 \leqslant i \leqslant N_L)$ and conditional probabilities $p(F_j = f_j | c = c_i)$ $(1 \leqslant j \leqslant M)$ will be estimated by the relative frequencies from the training set. Given a training set Z consisting of N_Z instances $z_1, z_2, \ldots, z_{N_Z}$, a vector V and a matrix W are employed in this paper to record its statistical information. Each entry $V[c_i]$ represents the number of instances in Z taking the class c_i. Every entry $W[c_i][F_j][f_j]$ represents the number of instances in Z taking c_i as the class and f_j as the value of feature F_j. Therefore, the probability $p(c = c_i)$ could be approximated by $p(c = c_i) = V[c_i]/N_Z$. Similarly, the probability $p(F_j = f_j | c = c_i)$ could be estimated by $p(F_j = f_j | c = c_i) = W[c_i][F_j][f_j]/V[c_i]$.

Obviously, the running time of NB can be estimated as follows. Firstly, it costs $O(N_L)$ and $O(N_L \sum_{j=1}^{M} N_{F_j}) = O(N_L N_F)$ to initialize the vector V and the matrix W respectively. In addition, it takes $O(1)$ and $O(M)$ running time to update V and W for every instance in Z respectively. Therefore, the overall running time of building a NB classifier is $O(N_Z M + N_L N_F)$.

When NB is incorporated into GRA (denoted by GRA-NB) as the classification algorithm, the overall running time can be computed as follows.

Claim 1. The running time of GRA-NB is $O(N_T N_L M N_C^2 + N_C^3 M + N_C^2 N_L N_F)$.

5.3 Accelerated GRA for Naive Bayes

In this subsection, we shall show how NB could be adjusted for GRA and present the Accelerated GRA for NB (AGRA-NB).

Given an instance $y = (f_1, f_2, \ldots, f_M)$ and the training set Z, we can conduct a simple transformation on equation (1) for predicting y's class.

$$\begin{aligned}
&\text{argmax}_{c_i}\, p(c = c_i) \prod_{j=1}^{M} p(F_j = f_j | c = c_i) \\
&= \text{argmax}_{c_i}\, V[c_i]/N_Z \prod_{j=1}^{M} (W[c_i][F_j][f_j]/V[c_i]) \qquad (2) \\
&= \text{argmax}_{c_i}\, V[c_i] \prod_{j=1}^{M} (W[c_i][F_j][f_j]/V[c_i]) \\
&= \text{argmax}_{c_i} (\prod_{j=1}^{M} W[c_i][F_j][f_j])/V[c_i]^{M-1}
\end{aligned}$$

Therefore, it's independent of N_Z for NB to predict y's class. In addition to the vector V and the matrix W, we also introduce a new matrix P in AGRA-NB for the test set T, where every entry $P[y][c_i]$ is used to record the value of $(\prod_{j=1}^{M} W[c_i][F_j][f_j])/V[c_i]^{M-1}$ for instance y in T. With this new matrix, we only need to traverse $P[y][c_1], P[y][c_2], \ldots, P[y][c_{N_L}]$ once to determine the class for y.

Table 3 presents the pseudocode of AGRA-NB. The key step of AGRA-NB improving on GRA is based on the observation that only a small fraction of those related information should be updated in every iteration of the greedy phase. The greedy phase of GRA consists of nested loops. Every iteration of the outer loop will find the "best" instance in the current candidate set which can provide maximum gap between the misleading rate and the leaking rate. In the inner loop, the training set used in one iteration is only one instance different from that in the next iteration. Therefore, we can accelerate GRA by updating those data structures (V, W, and P) in an incremental way. Motivated by this idea, we employ two routines in the greedy phase of AGRA-NB, i.e., Increment and Swap. Every iteration of the outer loop of the greedy phase in AGRA-NB is conducted by calling Increment once, followed by a series of calls to Swap.

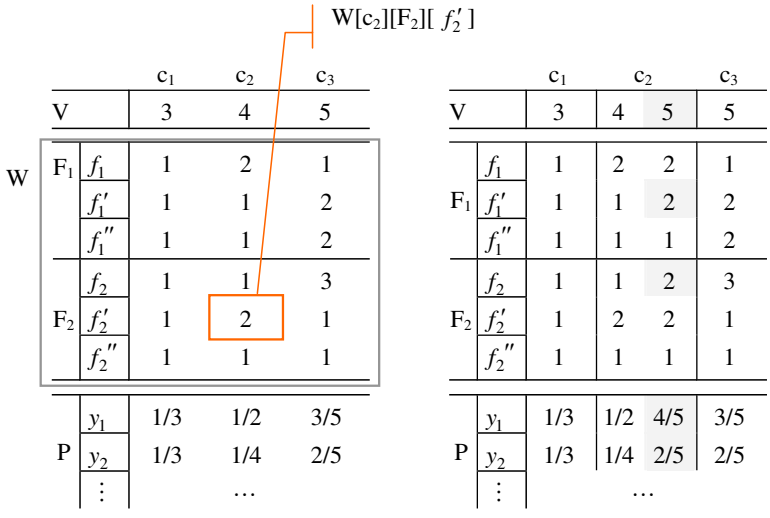
The routine Increment (see Table 4) aims to update a NB classifier when a new instance is introduced to the training set, e.g., at the beginning of every iteration of the outer loop in the greedy phase. Figure 2 presents an example of Increment routine. Figure 2(a) demonstrates the data structures before the Increment routine is called. There are already 12 instances in the current training set, including 3, 4, 5 instances with classes c_1, c_2, and c_3, respectively. The centering part of Figure 2 (a) shows the entries of

Table 3: Pseudocode of AGRA-NB

Algorithm: Accelerated GRA for Naive Bayes
Input: candidate set C, test set T
Output: ranked list R

Begin
// initialization phase
1 let $R = \emptyset$, let $j = 0$
2 let D be the list of sorted instances in C by classes
3 initialize V, M, and P

// greedy phase
4 while D is not empty do
  4.1 update V, M, P by calling Increment(D[0]);
  4.2 let $C_s = R \cup \{D[0]\}$ compute the misleading rate $\lambda_m(C_s, A)$, the leaking rate $\lambda_l(C_s, A)$ with P
  4.3 let $max_{gap} = \lambda_m(C_s, A) - \lambda_l(C_s, A)$
  4.4 for $i = 1$ to $n - 1$ do
    4.4.1 update V, M, P by calling Swap(D[i-1], D[i]);
    4.4.2 let $C_s = R \cup \{D[i]\}$, compute the misleading rate $\lambda_m(C_s, A)$, the leaking rate $\lambda_l(C_s, A)$ with P
    4.4.3 let $gap = \lambda_m(C_s, A) - \lambda_l(C_s, A)$
    4.4.4 if $gap > max_{gap}$ then
      $max_{gap} = gap$
      $best = D[i]$
  4.5 update M, P by calling Swap(D[n-1], best)
  4.6 $R[j] = best$
  4.7 $j++$
  4.8 remove *best* from D
5 return R
End



(a) V, W, and P (b) Increment

Figure 2: An example of Increment routine, where $y_1 = (f_1, f_2)$, $y_2 = (f_1', f_2'')$. In (b), a new instance (f_1', f_2) with class c_2 is added.

W. The bottom part of Figure 2(a) presents the entries of P. In Figure 2(b), a new instance (i.e., (f_1', f_2) with class c_2) is added to the training set such that the new training set size is increased by 1. Obviously, only those entries related to class c_2 should be updated (the gray part in Figure 2(b)). Therefore, the running time for Increment routine is $O(MN_T + 1 + M) = O(MN_T)$.

Table 4: Pseudocode of Increment

Routine: Increment

Input: instance $x = (f_1^*, f_2^*, \ldots, f_M^*)$ with class c

Begin

1 $V[c] + +$

2 for $i = 1$ to M do $W[c][F_i][f_i^*] + +$

3 for every test instance $y = (f_1, f_2, \ldots, f_M)$ do

$P[y][c] = (\prod_{i=1}^{M} W[c][F_i][f_i])/V[c]^{M-1}$

End

		c_1	c_2	c_3	
V		3	5	5	
F_1	f_1	2	2	1	2
	f_1'	2	1	2	1
	f_1''	1	1	2	2
F_2	f_2	2	1	3	2
	f_2'	2	2	1	2
	f_2''	1	1	1	1
P	y_1	1/3	4/5	3/5	4/5
	y_2	1/3	2/5	2/5	1/5
	⋮		…		

(a) Swapping instances with identical class labels

		c_1		c_2	c_3	
V		3	4	5	5	4
F_1	f_1	1	1	2	2	2
	f_1'	1	2	2	1	1
	f_1''	1	1	1	2	1
F_2	f_2	1	2	2	2	1
	f_2'	1	1	2	2	2
	f_2''	1	1	1	1	1
P	y_1	1/3	1/2	4/5	4/5	1/2
	y_2	1/3	1/4	2/5	1/5	1/4
	⋮			…		

(b) Swapping instances with distinct class labels

Figure 3: Examples of Swap routine, where $y_1 = (f_1, f_2)$, $y_2 = (f_1', f_2'')$. In (a), an existing instance (f_1', f_2) with class c_3 is replaced with a new one (f_1, f_2') with class c_3. In (b), an existing instance (f_1'', f_2) with class c_3 is replaced with a new one (f_1', f_2) with class c_1.

Table 5: Pseudocode of Swap

Routine: Swap

Input: instance $x = (f_1^*, f_2^*, \ldots, f_M^*)$ with class c, $x' = (f_1', f_2', \ldots, f_M')$ with class c'

Begin

1 if $c = c'$ then *// case 1: swapping instances with identical classes*

1.1 for $i = 1$ to M do

$W[c][F_i][f_i^*] - -$

$W[c][F_i][f_i'] + +$

1.2. for every test instance $y = (f_1, f_2, \ldots, f_M)$ do

$P[y][c] = (\prod_{i=1}^{M} W[c][F_i][f_i])/V[c]^{M-1}$

2 else *// case 2: swapping instances with distinct classes*

2.1 $V[c] - -$

2.2 for $i = 1$ to M do $W[c][F_i][f_i^*] - -$

2.3 for every test instance $y = (f_1, f_2, \ldots, f_M)$ do

$P[y][c] = (\prod_{i=1}^{M} W[c][F_i][f_i])/V[c]^{M-1}$

2.4 $V[c'] + +$

2.5 for $i = 1$ to M do $W[c'][F_i][f_i'] + +$

2.6 for every test instance $y = (f_1, f_2, \ldots, f_M)$ do

$P[y][c'] = (\prod_{i=1}^{M} W[c'][F_i][f_i])/V[c']^{M-1}$

End

Once an Increment routine is conducted in an iteration of the outer loop in the greedy phase, we only need to perform Swap operations, each of which will replace an existing instance in the training set with

a new one. There are two cases in the Swap operation.

Case 1: Swapping instances with identical classes

In this case, an instance will be replaced by another instance with the same class. For example in Figure 3(a), instance (f_1', f_2) with class c_3 is replaced by (f_1, f_2') with class c_3. Among all entries of the matrix W, only those ones related to these two instances should be updated. For the matrix P, those entries indexed by c_3 are to be recalculated. We use the gray part to show those updated entries in Figure 3(a).

Case 2: Swapping instances with distinct classes

In contrast to Case 1, an instance in Case 2 will be swapped with another instance with distinct class. In Figure 3(b), instance (f_1'', f_2) with class c_3 is replaced by (f_1', f_2) with class c_1. In this example, both entry $V[c_3]$ and $V[c_1]$ should be modified. In addition, there are totally $2M$ entries in the matrix W will be updated. Similarly, those entries indexed by either c_1 or c_3 in P will be recomputed.

Since it needs $O(M)$ running time to recompute an entry in P, the running time for Case 1 is $O(2M + N_T M)$. Similarly, it takes $O(2M + 2 + 2N_T M)$ running time to conduct Case 2. It's easy to see that Case 2 costs nearly twice as much running time as Case 1. To reduce the running time, we conduct a sorting on all candidate instances ahead of the greedy phase, such that the instances with class c_1 are put together, followed by the instances with class c_2, and so on. Therefore, Case 2 will be met at most $N_L - 1$ times in an iteration of the outer loop in the greed phase. The running time of Swap routine will be $max(O(2M + N_T M), O(2M + 2 + 2N_T M)) = O(N_T M)$.

We have the following claim for the running time of AGRA-NB.

Claim 2. The running time of AGRA-NB is $O(N_C^2 M N_T + N_C^2 N_T N_L + N_T N_L M + M N_F)$.

6 Experiments

6.1 Experiment Setup

All experiments are conducted on a Pentium(R) IV CPU 3.4G PC with 1G memory running Windows XP. We implemented those new algorithms in Java compiled by Netbeans IDE 6.8 using Weka 3.6.2 library[13]. Since there's no data set for misleading classification available, 16 data sets from the UCI repository [14] are used for our experiments, including *yeast*, *vehicle*, *glass*, *chess*, etc. For those numeric attributes, we simply discretized them to 10 equal intervals with Weka. The baseline parameters are set as follows. Each data set is partitioned into 10 folds, where each fold will be used for the test set and the other 9 folds will be used as the candidate set. Therefore, every data set will be used 10 times for each new algorithm and the average result over those 10 runs are given. Since there's no misleading classes available for those test sets, we set the misleading class of every test instance by simply shuffling its original class. Table 6 summarizes the shuffling ways for some representative UCI data sets, where org. represents the original class, and mis. represents the misleading one. For example, the org. of data set *vehicle* takes the classes *opel, saab, bus, van*, while the mis. takes *saab, bus, van, opel*. It indicates that the original class *opel* is mapped to *saab* as the misleading class, *saab* is mapped to *bus*, *bus* is mapped to *van*, and *van* is mapped to *opel*. In such a way, a test instance of *vehicle* will be associated with *van* as the misleading class if its original class is *bus*.

In the following part of this section, we assume that the rival will use NB as the classification algorithm. In Subsection 6.2 and 6.3, we run KRA and GRA algorithms respectively and then NB is employed by the rival to evaluate the quality of training sets generated by these new algorithms. Due to the space limit, we mainly present the results on some representative data sets. More experimental results can be achieved on our website, www.oscar-lab.org/eng/publication.htm.

6.2 Evaluation of KRA

In this subsection, we shall evaluate KRA on UCI data sets. Due to the space limit, we only report the results for some representative data sets. Since KRA employs KNN to calculate the similarities between

Table 6: Original and misleading classes of some UCI data sets.

Data set	Class	Class Values
yeast	org.	*CYT, NUC, MIT, ME3, ME2, ME1, EXC, VAC, POX, ERL*
	mis.	*NUC, MIT, ME3, ME2, ME1, EXC, VAC, POX, ERL, CYT*
vehicle	org.	*opel, saab, bus, van*
	mis.	*saab, bus, van, opel*
glass	org.	*'build wind float', 'build wind non-float', 'vehic wind float', 'vehic wind non-float', containers, tableware, headlamps*
	mis.	*'build wind non-float', 'vehic wind float', 'vehic wind non-float', containers, tableware, headlamps, 'build wind float'*
chess	org.	*won, nowin*
	mis.	*nowin, won*

instances, we also investigate the effect of parameter K on KRA in this subsection. For every ranked list generated by those KRA algorithms taking different K values, the top $4\%, 8\%, \ldots, 100\%$ candidate instances are selected out by the rival as the training sets respectively to build classifiers with NB. Then the rival uses every classifier to predict its test set for calculating the leaking and misleading rates. In such a way, Figure 4 reports the curves of the leaking and misleading rates against the fraction of ranked candidate instances used for the training set.

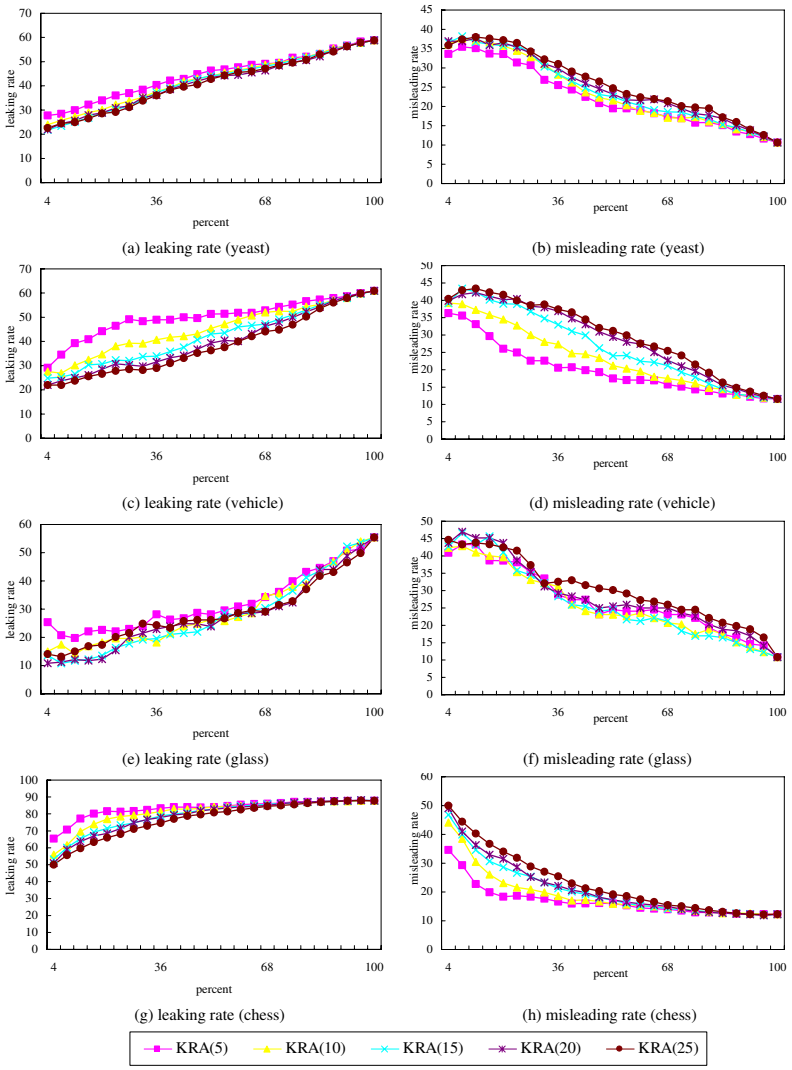


Figure 4: Performance of KRA taking different K values.

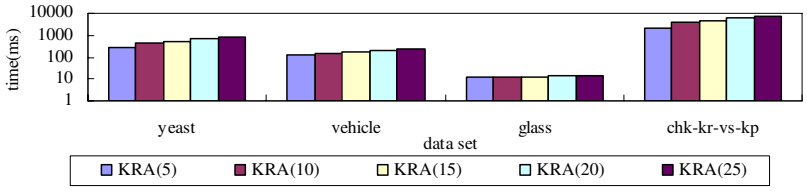


Figure 5: Running time (ms) of KRA with different K values.

As shown in Figure 4, the leaking rates gradually increase along with the growth of the fraction of ranked candidate instances used. On the other hand, the misleading rates follow a decreasing trend against the fraction of ranked candidate instances used. It indicates that KRA could surely contribute to misleading classification that the front part of ranked candidate instances are more likely to mislead the rival than the back part. For example, we can see that the misleading rate of KRA with $K = 25$ on *yeast* drops from 35% to about 25% when the fraction of ranked candidate instances rises from 4% to 50%.

We can also find that performance (i.e., the leaking and misleading rates) of KRA will slightly improve along with the increase of K value. For those representative data sets, the leaking rate will be usually lower when a larger K is used in KRA. In contrast, the misleading rate will generally grows when a larger K is used. To further investigate the effect of K value in KRA, larger K values are also used, i.e., 30, 35, 40, 45, 50. The experimental results of KRA taking larger K values are presentd in Table 7. Due to the space limit, we only report the leaking rates and misleading rates when 20%, 40%, 60%, 80%, and 100% ranked candidate instances are used as the training sets. As shown in Table 7, when K values larger than 25 are taken, the performance of KRA will not change much on *yeast*, *vehicle*, and *glass*. For *chess*, the performance of KRA may slight improve when K value grows. It implies that the impact of K may be dependent on the data set.

It can be concluded for every data set that all curves of those KRA algorithms taking different K values will eventually converge to the same pixel when 100% candidate instances are employed as the training set. The reason for this phenomenon is that the classification algorithm NB (used by the rival) is independent of the order of instances in the training set. Once the whole candidate set is taken as the training set, NB will always produce the same results for the test set regardless of the order of candidate instances.

In addition, we also report the running time of KRA taking different parameter K in Figure 5. Obviously, it takes more time to run KRA when a larger K is taken. The reason is that the main part of running time in KRA is spent on KNN for calculating the similarities between candidate instances and test instances. In general, the running time of KNN will grows when K increases.

6.3 Evaluation of GRA

In this subsection, we evaluate the performance of GRA algorithms, including AGRA-NB and GRA-NB. Firstly, we report the running time (ms) of AGRA-NB and GRA-NB on UCI data sets in Figure 6. It should be noted that a logarithm coordinate is adopted for presenting the running time. For comparison, Figure 6 also presents the running time of KRA taking $K = 25$. In addition, we also implement a random algorithm (denoted by *rand* in this paper) which randomly shuffle those candidate instances to form the ranked list. As shown in Figure 6, *rand* takes the least running time among all 4 algorithms. In general, KRA(25) costs far less running time than either AGRA-NB or GRA-NB. At the mean time, AGRA-NB is dramatically faster than GRA-NB on all 4 data sets. For example, the running time of AGRA-NB on data set *glass* is 304ms while that of GRA-NB is 4678ms. On the data set *glass*, AGRA-NB is nearly 15 times faster than GRA-NB.

To evaluate the leaking and misleading rates achieved by GRA algorithms, we assume that the rival uses NB as the classification algorithm and builds a series of classifiers on the training sets consisting of the top ranked $4\%, 8\%, \ldots, 100\%$ candidate instances respectively by GRA algorithms. Then, the rival applies those classifiers to their corresponding test sets. Figure 7 reports the curves of the leaking

Table 7: Effect of K Values in KRA

data set	algorithm	leaking					misleading				
		20	40	60	80	100	20	40	60	80	100
yeast	KRA(25)	28.57	38.35	45.56	50.74	58.83	37.19	29.03	22.36	19.46	10.64
	KRA(30)	27.96	37.94	45.62	51.62	58.83	36.45	29.23	23.31	18.99	10.64
	KRA(35)	27.16	37.47	44.82	50.61	58.83	36.79	29.91	23.17	20.07	10.64
	KRA(40)	27.42	38.95	44.55	50.88	58.83	36.25	27.55	23.24	20.68	10.64
	KRA(45)	26.68	38.14	45.02	50.74	58.83	36.86	29.37	24.18	20.95	10.64
	KRA(50)	25.20	36.53	44.61	50.74	58.83	37.66	30.58	24.32	21.76	10.64
vehicle	KRA(25)	26.62	30.97	39.96	50.26	60.88	41.60	36.54	27.54	19.14	11.59
	KRA(30)	25.19	30.03	39.97	49.44	60.88	40.54	37.00	29.54	20.32	11.59
	KRA(35)	25.55	28.02	37.83	48.96	60.88	42.07	40.54	31.67	21.15	11.59
	KRA(40)	23.77	28.97	37.01	47.42	60.88	44.08	40.08	33.21	23.04	11.59
	KRA(45)	24.37	29.09	37.25	48.13	60.88	43.96	40.44	33.69	22.68	11.59
	KRA(50)	25.07	29.08	37.95	48.71	60.88	43.02	41.03	33.81	23.40	11.59
glass	KRA(25)	17.34	23.35	28.61	37.03	55.43	42.42	33.01	27.29	22.14	10.84
	KRA(30)	19.22	23.85	29.48	36.56	55.43	41.02	32.94	29.20	23.10	10.84
	KRA(35)	16.02	25.28	30.41	36.99	55.43	46.60	33.40	29.20	23.57	10.84
	KRA(40)	18.77	26.19	29.05	37.42	55.43	42.36	32.03	30.11	23.14	10.84
	KRA(45)	15.48	23.90	28.12	37.45	55.43	43.35	32.94	29.65	24.52	10.84
	KRA(50)	16.43	24.81	30.02	39.85	55.43	40.95	31.04	29.16	24.03	10.84
chess	KRA(25)	65.99	77.06	82.54	86.33	87.77	34.01	22.94	17.46	13.67	12.23
	KRA(30)	64.17	76.28	83.60	87.02	87.77	35.83	23.72	16.40	12.98	12.23
	KRA(35)	63.14	75.62	83.04	86.89	87.77	36.86	24.38	16.96	13.11	12.23
	KRA(40)	61.89	75.25	82.98	87.36	87.77	38.11	24.75	17.02	12.64	12.23
	KRA(45)	60.76	73.56	82.70	87.02	87.77	39.24	26.44	17.30	12.98	12.23
	KRA(50)	59.20	72.37	82.32	87.33	87.77	40.80	27.63	17.68	12.67	12.23

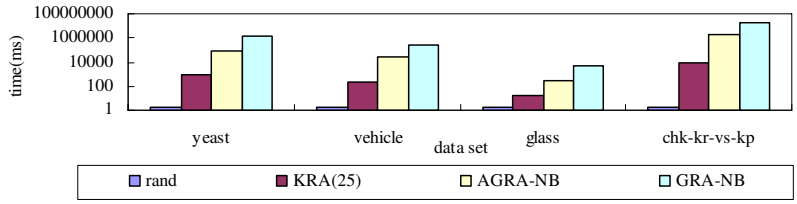


Figure 6: Running time (ms) of *rand*, KRA(25), AGRA-NB, and GRA-NB.

and misleading rates against the fraction of ranked candidate instances used for the training set. Since AGRA-NB will produce the same ranked candidate instances as GRA-NB, only the results of AGRA-NB are presented in Figure 7.KRA(25) and *rand* are run as well on those UCI data sets for comparison. To show how well the classification algorithm will fit for those test sets, the classifiers are also built on those test instances with misleading classes and employed to predict the test sets themselves (the leaking and misleading curves of such classifiers are marked as self).

Obviously, the leaking rate curves of *rand*, KRA(25), and AGRA-NB follow similar trends that those leaking rates increase along with the growth of the fraction of ranked list used for the training set. The leaking rate curve of *rand* shows a flat shape after a short sharp increase, while the curves of both KRA(25) and AGRA-NB exhibit a much slower growth. On the other hand, the most parts of those misleading rate curves of *rand*, KRA(25), and AGRA-NB gradually drop against the growth of the fraction of ranked list used. For both the leaking rate and the misleading rate, all curves of *rand*, KRA(25), and AGRA-NB for every data set converge to the same pixel when the total candidate set is used as the training set.

In contrast to *rand*, KRA(25) achieves far lower leaking rates and higher misleading rates on those representative data sets, especially when a small fraction of ranked candidate instances are used as the training sets. For example in Figure 7(c), the leaking rate of KRA(25) is around 25% while that of *rand* is about 60%, when the top ranked 12% candidate instances are selected out as the training set. Among those three algorithms, our AGRA-NB is the best one which usually returns the lowest leaking rates and the highest misleading rates. In addition, the leaking and misleading curves of AGRA-NB are obviously the closest to those of self. For *yeast*, *vehicle*, and *glass*, AGRA-NB can even produce lower leaking rates than self when the top ranked 4%-12% candidate instances are chosen as the training sets! It's a promising result with respect to the fact that AGRA-NB ranks those candidate instances with original classes rather than misleading classes.

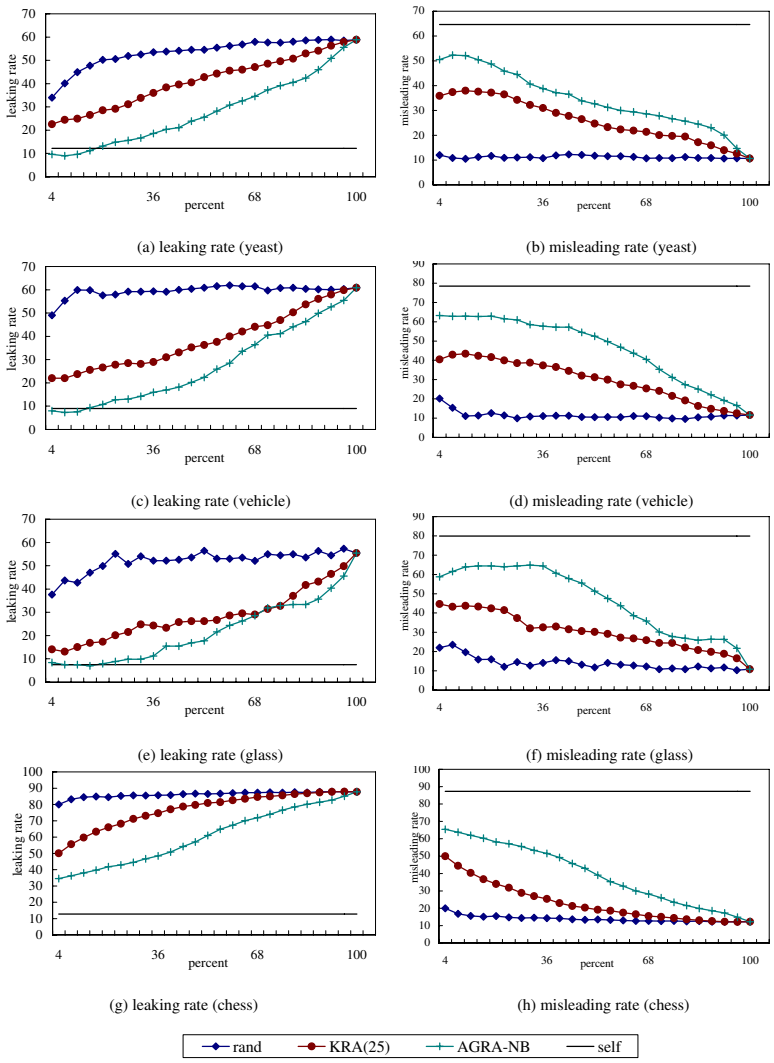


Figure 7: Performance of *rand*, KRA(25), AGRA-NB, and self.

In addition to those representative data sets, we summarize the experimental results for the other 12 UCI data sets in Table 8. Due to the space limit, we only reports the leaking and misleading rates when the top ranked 20%, 40%, 60%, 80%, and 100% candidate instances are selected as the training sets. Obviously, AGRA-NB takes more running time than KRA(25). Among those 3 algorithms, *rand* costs the least running time and produces the worst leaking as well as misleading rates. In general, AGRA-NB can dramatically outperform KRA(25) in terms of both the leaking and misleading rates. For example, the leaking rate generated by AGRA-NB is 12.79% while the leaking rate by KRA(25) is 48.26% for *ecoli*, when the top 20% candidate instances from those ranked lists are used as the training sets. Mean while, the misleading rates are 59.87% and 11.59% by AGRA-NB and KRA(25) respectively. It should be also noted that AGRA-NB achieves lower leaking rates (higher misleading rates) than self leaking rates (misleading rates) on *cmc*, *tic-tac-toe*, and *blood-transfusion*. These experiments indicate that a good algorithm (e.g. AGRA-NB) can actually mislead the rival in a desired way.

7 Conclusion and Future Work

In this paper, we investigate the model named misleading classification with numerous applications arising in business competition, patent protection, and combat intelligence. The goal of misleading classification

Table 8: Summary of Algorithms Performance

data set	self leaking	self misleading	algorithm	time (ms)	leaking 20	leaking 40	leaking 60	leaking 80	leaking 100	misleading 20	misleading 40	misleading 60	misleading 80	misleading 100
balance -scale	6.40	88	*rand.*	1	78.41	85.61	88.01	90.41	91.20	11.36	7.35	6.39	5.43	3.53
			KRA(25)	217	20.16	43.85	58.89	76.33	91.20	49.44	35.67	28.63	14.38	3.53
			AGRA-NA	3542	14.58	25.60	45.46	68.95	91.20	68.63	54.41	35.19	16.80	3.53
car	7.41	87.62	*rand.*	3	82.12	83.57	84.03	85.76	85.82	4.11	3.53	3.65	3.18	3.53
			KRA(25)	1708	36.46	78.65	81.31	83.62	85.82	42.82	14.87	12.27	7.64	3.53
			AGRA-NB	85098	27.43	48.38	66.49	84.49	85.82	46.24	28.76	18.92	6.13	3.53
cmc	18.53	62.25	*rand.*	4	49.83	49.76	50.04	49.7	50.04	25.39	24.71	24.98	25.39	24.98
			KRA(25)	732	20.17	30.15	36.93	42.09	50.04	54.99	43.65	37.27	32.45	24.98
			AGRA-NB	73528	12.15	18.46	25.94	35.17	50.04	62.73	55.54	46.02	37.95	24.98
ecoli	1.78	88.13	*rand.*	1	72.62	76.21	78.91	79.80	81.60	1.50	0.89	0.60	0.60	0.60
			KRA(25)	37	48.26	58.35	69.38	78.04	81.60	11.59	2.99	0.90	0.60	0.60
			AGRA-NB	956	12.79	24.74	41.69	69.16	81.60	59.87	39.32	17.61	3.28	0.60
ionosphere	2.57	97.43	*rand.*	1	87.44	91.14	91.14	90.86	90.57	12.56	8.86	8.86	9.14	9.43
			KRA(25)	84	55.28	72.29	78.87	86.00	90.57	44.72	27.71	21.13	14.00	9.43
			AGRA-NB	2945	23.67	47.04	68.06	81.45	90.57	76.33	52.96	31.94	18.55	9.43
iris	0.67	98.67	*rand.*	1	90.67	91.33	94.00	95.33	94.67	5.33	4.00	2.00	1.33	2.00
			KRA(25)	6	64.67	85.33	85.33	92.00	94.67	27.33	12.67	10.00	4.00	2.00
			AGRA-NB	67	24.00	40.00	63.33	85.33	94.67	58.67	44.00	26.00	10.00	2.00
lymph	0.71	99.29	*rand.*	1	74.48	79.10	81.86	79.76	83.14	14.10	10.81	9.43	10.14	7.43
			KRA(25)	12	33.67	45.24	64.86	77.05	83.14	56.10	44.57	25.67	12.86	7.43
			AGRA-NB	154	21.67	27.62	50.00	66.95	83.14	61.48	58.76	40.52	22.24	7.43
tic-tac-toe	22.34	77.66	*rand.*	1	69.73	70.87	69.31	69.93	69.20	30.27	29.13	30.69	30.07	30.80
			KRA(25)	365	30.16	37.05	51.25	64.09	69.20	69.84	62.95	48.75	35.91	30.80
			AGRA-NB	17706	16.28	29.97	42.07	56.68	69.20	83.72	70.03	57.93	43.32	30.80
wine	0	100	*rand.*	1	93.27	95.49	96.63	97.19	97.19	3.37	2.25	1.11	1.11	1.11
			KRA(25)	9	52.84	69.15	85.98	90.46	97.19	35.92	21.90	8.43	4.51	1.11
			AGRA-NB	221	30.75	45.36	63.99	83.73	97.19	61.41	42.19	28.73	12.35	1.11
zoo	0	100	*rand.*	1	80.18	92.09	92.09	94.09	94.09	7.00	2.00	1.00	1.00	1.00
			KRA(25)	3	74.18	81.18	84.09	93.09	94.09	10.00	4.00	4.00	2.00	1.00
			AGRA-NB	45	13.82	29.64	48.36	65.36	94.09	55.45	37.73	27.00	22.00	1.00
blood -transfusion	18.18	81.82	*rand.*	1	75.12	75.40	74.46	75.13	76.60	24.88	24.60	25.54	24.87	23.40
			KRA(25)	68	62.96	64.70	68.31	71.78	76.60	37.04	35.30	31.69	28.22	23.40
			AGRA-NB	5217	16.85	27.56	48.40	64.84	76.60	83.15	72.44	51.60	35.16	23.40
credit-g	15	85	*rand.*	1	73.50	74.70	74.30	75.70	76.00	26.50	25.30	25.70	24.30	24.00
			KRA(25)	409	30.00	44.00	60.20	72.40	76.00	70.00	56.00	39.80	27.60	24.00
			AGRA-NB	33860	17.40	22.40	43.80	63.20	76.00	82.60	77.60	56.20	36.80	24.00

for the data owner is to construct the training set from a set of candidate instances, such that the rival will be misled to classify those test instances to their misleading classes rather than original classes. We discuss two cases of misleading classification. For the first case that the classification algorithm is unknown by the data owner, we present the algorithm KRA, which ranks all candidate instances based on the similarities between instances. For the case that the data owner knows the classification algorithm, we propose the algorithm GRA which greedily to rank those candidate instances by building up a series of classifiers. In addition, we also illustrate how to accelerate GRA when NB is used as the classification algorithm.

For the future work, there are several directions to investigate.

- *Impact of K in KRA* In the experimental section, we find that the value of K may influence the performance of KRA. For some data sets, KRA may achieve relatively stable performance when K is set to 25. However, the performance of KRA continuously improves when K grows for the data set *chess*. It's an interesting direction to investigate the relationship between the distribution of data sets and the value of K.

- *GRA with Other Classification Algorithms* Obviously, it's easy to incorporate other existing classification algorithms into GRA, including decision trees, decision rules, etc. A key issue is how to accelerate GRA like what AGRA-NB does. Following the same idea of AGRA-NB, GRA can be improved by building classifiers in an incremental way.

- *Misleading Classification on New Data Types* Similar as traditional classifications task, there are also a lot of misleading classification work to be done on those forthcoming data types, including high dimensional data, stream data, etc.

- *Misleading Data Mining* In addition to classification, we can also generalize the conception of misleading classification to other tasks in data mining, such as clustering, regression, etc. For example, we can investigate how to mislead those clustering algorithm (e.g. [15,16]) in a desired way.

Acknowledgements

This work is partially supported by the US National Science Foundation (NSF) under Grant CCF-0905337, the Natural Science Foundation of China under Grant 61175062 and 61033012, and the Fundamental Research Funds for the Central Universities under Grant No.DUT12JR02.

References

1 *http://patft.uspto.gov/*
2 *http://kottke.org/10/05/operation-mincemeat/*
3 Aggarwal C C, Chen C, Han J W. *On the Inverse Classification Problem and its Applications*. Proc. of 22nd International Conference on Data Engineering (ICDE), 2006.
4 Dalvi N, Domingos P, Mausam, Sanghai S, Verma D. *Adversarial Classification*. Proc. of 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), 2004.
5 Peng H C, Long F, Ding C, *Feature Selection based on Mutual Information: Criteria of Max-dependency, Max-relevance, and Min-redundancy*, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 27, No. 8, pp. 1226-1238, 2005.
6 Molina L C, Belanche L, Nebot A. *Feature Selection Algorithms: A Survey and Experimental Evaluation*. Proc. of 2nd IEEE International Conference on Data Mining (ICDM'02), 2002.
7 Bi W, Shi Y, Lan Z. *Transferred Feature Selection*. Proc. of 9th IEEE International Conference on Data Mining Workshops, pp.416-421, 2009
8 Agrawal R, Srikant R. *Privacy-preserving Data Mining*. Proc. of the 2000 ACM SIGMOD Conference on Management of Data, Dallas, Tx, 2000.
9 Chen K, Liu L. *Privacy Preserving Data Classification with Rotation Perturbation*. Proc. of 5th IEEE International Conference on Data Mining (ICDM'05), 2005.
10 Vaidya J, Kantarcioglu M, Clifton C. *Privacy-perserving Naive Bayes Classification*. The VLDB Journal, vol. 17, issue 4, pp. 879-898, 2008.
11 Cover T M, Hart P E. *Nearest Neighbor Pattern Classification*. IEEE Transactions on Information Theory, vol. 13, issue 1, pp. 21-27, 1967.
12 Hoare C A R. *Quicksort*, Computer Journal, vol.5, issue 1, pp. 10-15, 1962.
13 *http://www.cs.waikato.ac.nz/ml/weka/*
14 *http://archive.ics.uci.edu/ml/*
15 Jiang H, Ren Z L, Xuan J F, Wu X D. *Extracting Elite Pairwise Constraints for Clustering*. Neurocomputing, vol. 99, issue 1, pp. 124-133, 2013.
16 Zhang T, Ramakrishnan R, Livny M. *BIRCH: An Efficient Data Clustering Method for Very Large Databases*. Proc. of ACM SIGMOD Conference, Montreal, Canada, pp. 103-114, 1996.