Preservation Of Patterns and Input-Output Privacy

Shaofeng Bu, Laks V.S. Lakshmanan, Raymond T. Ng and Ganesh Ramesh
University of British Columbia
Vancouver, BC, Canada V6T 1Z4
{sfbu,laks,rng,ramesh}@cs.ubc.ca

Abstract

Privacy preserving data mining so far has mainly focused on the data collector *scenario where individuals supply their personal data to an untrusted collector in exchange for value. In this scenario, random perturbation has proved to be very successful. An equally compelling, but overlooked scenario, is that of a* data custodian*, which either owns the data or is explicitly entrusted with ensuring privacy of individual data. In this scenario, we show that it is possible to minimize disclosure while guaranteeing* no outcome change*. We conduct our investigation in the context of building a decision tree and propose transformations that preserve the exact decision tree. We show with a detailed set of experiments that they provide substantial protection to both* input data privacy *and* mining output privacy*.*

1 Introduction

One of the key motivations for privacy preserving data mining is the mining-as-a-service model, in which there are at least two different scenarios. In [2], Agarwal and Srikant consider the *data collector* scenario where individual data owners selectively submit their data to a data collector, who may not be trusted, in exchange for the value coming from mining the data collection. Many existing studies focus on this scenario.

In this paper, we consider a different, yet equally prevalent, scenario of a *data custodian* which either owns the data (e.g., a company owns its data) or is explicitly given the trust and responsibility of protecting the privacy of the individuals. For example, a medical research group (i.e., the custodian) obtains consent forms from patients to participate in a biomarker study. Because the data custodian has little expertise in data mining, it considers hiring a company to help mine the data. It does not implicitly trust the company and wants to guard against possible privacy breaches. To do so, the data custodian needs to transform its data. To determine the appropriate transformation, there are two critical considerations: *minimizing disclosure* versus *minimizing outcome change* (i.e., the deviation between the outcome obtained by mining the original data and the outcome of mining the transformed data). Among the transformations studied in the literature, random perturbation is a dominant approach, i.e., transforming data values by adding random noise in a principled way [2]. The more noise is added, the more disclosure is minimized – but *the more the mining outcome is changed*. It is implicitly assumed that to minimize disclosure, it is inevitable that there be outcome change.

In this paper, we show that it is possible to minimize disclosure *while guaranteeing no outcome change*. We conduct our investigation in the context of decision tree classifiers. We propose classes of monotone and anti-monotone functions, that preserve the decision tree in an exact sense. For privacy preserving data mining, we claim there are *three equally important pillars*. The no-outcome-change guarantee is the first pillar. The second pillar is the protection of the input data – *input privacy* in [8]. The third pillar is the protection of the mining outcome – *output privacy*. Existing studies focus on input privacy, but ignore the other two pillars. We show that using (anti-)monotone transformations can satisfy all three pillars at the same time. Moreover, when (anti-) monotone transformations are used, decoding the mining outcome is straightforward. Finally, with the proposed transformations, *every* data value is transformed. In contrast, with random perturbations, there is a chance that a discrete data value is not changed and the true value is revealed.

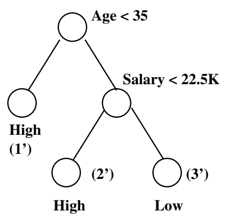
Figure 1 shows a simple example of a monotone transformation. The original data D is shown in (a). The attributes `age` and `salary` are transformed with the linear monotone functions: $\mathsf{age}' = 0.9 * \mathsf{age} + 10$, and $\mathsf{salary}' = 0.5 * \mathsf{salary}$. The transformed data D' is now given to the mining service provider (Fig-

Age	Salary	Risk
23	50K	High
17	30K	High
43	40K	High
68	50K	Low
32	70K	Low
20	20K	High

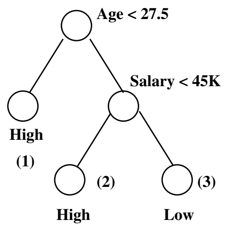
(a) Original Data D

Age	Salary	Risk
31	25K	High
25	15K	High
49	20K	High
71	25K	Low
39	35K	Low
28	10K	High

(b) Transformed Data D′



(c) Transformed Decision Tree T′



(d) Original Decision Tree T

Figure 1. A Training Data Set before and after transformation and their corresponding Decision Trees

ure 1(b)). The classifier T′ based on D′ is shown in (c). Note that D′ provides input privacy, whereas T′ provides output privacy. To decode T′ to get the real decision tree, the data custodian uses the inverse function per attribute to obtain T, shown in (d). The inverse transformations are $\mathsf{age} = (\mathsf{age}' - 10)/0.9$, and $\mathsf{salary} = \mathsf{salary}'/0.5$. Notice that this is exactly the same outcome if the decision tree algorithm were to be applied to D directly without the transformation. Also note that T′ looks realistic enough that a hacker may not even know that it is encoded.

For many situations, simply using (anti-)monotone functions may not be too effective for privacy protection. One of the key technical contributions here is the generalization from (anti-)monotone functions to *piecewise* (anti-)monotone functions. The two central ideas are to introduce *breakpoints* and to exploit *monochromatic pieces*. By going piecewise, three levels of uncertainty are added: (i) the uncertainty of the number of breakpoints, (ii) the uncertainty of the locations of the breakpoints; and (iii) the uncertainty of the function used in each piece. A key insight is that other than applying randomization to every value in the domain, it is more beneficial to apply randomizaiton to appropriate sub-domains only.

We analyze disclosure risk performance (the second and third pillars) with a comprehensive framework. First, for input privacy, we consider *domain disclosure* and *subspace association disclosure*. For output privacy, we consider *outcome disclosure*, which protects paths of a decision tree. Second, we consider different *attack models* for the hacker. And, third, we consider a hacker's *prior knowledge*, modeled as *knowledge points* in the attribute domain. Based on benchmark data sets, we provide empirical results showing the effectiveness of the proposed framework.

2 Related Studies

Extensive research has been done in statistical databases to provide statistical answers without breaching sensitive information about individuals. A commonly used technique is random perturbation [1]. The study by Agrawal and Srikant shows how a decision tree can be built on data perturbed with random noise added in a principled way [2]. However, preserving the exact decision trees is not their goal, and they show that there is a tradeoff between classification accuracy and privacy level.

Recall the distinction between the data collector and data custodian scenarios. The random perturbation approach is designed to deal with the former, but can be applied to the latter as well. The piecewise monotone framework proposed here is designed for the data custodian scenario. Under this scenario, the proposed model delivers the three pillars of privacy preservation, whereas the perturbation approach cannot.

Evfimievski et al. [5]apply the random perturbation approach to association rule mining. Rizvi and Haritsa propose a variant based on probabilistic distortion [8]. The mining outcome is changed. Output privacy is not a stated design objective.

Recently, Kargupta et al. showed that a hacker can use spectral analysis based matrix decomposition techniques to separate the random noise from the real values [7]. Huang et al. [6] use a principal component analysis based method to exploit possible correlations among attributes to reconstruct original data, demonstrating that more accurate individual data can be revealed than originally thought.

Clifton and others consider privacy preservation with vertically partitioned data [10]. The focus is on developing protocols to make sure that each site cannot gain additional information about the data via the collaboration. There are studies on various other mining tasks with privacy preservation as the main goal, including clustering [10], order-preserving comparisons [3], and data exchange [9]. For data exchange, the notion of k-anonymity is designed for input privacy. If the transformed data were mined directly, the mining outcome could be significantly affected.

3 Preliminaries

3.1 Monotone and Anti-monotone Functions

The training data set is a relation instance D with m attributes $A_1, \ldots, A_m$, and a categorical class label attribute C. Throughout this paper, we are interested in *active domains* of attributes, which contain values of attributes appearing in a given data set. We denote the active domain of an attribute A by $\delta(A)$. Often we will transform it into another active domain $\delta'(A)$.

Let A be a numeric attribute with a linear ordering $<$. Let $f : \delta(A) \to \delta'(A)$ be a function. Then f is monotone (resp., anti-monotone) if for every $x, y \in \delta(A)$, $x < y$ implies $f(x) < f(y)$ (resp., $f(x) > f(y)$). Throughout this paper, we only consider transformation f whose inverse f^{-1} is well-defined, because the data custodian needs f^{-1} to decode the mining result. Given a tuple $\langle t, c\rangle$ (c being the class label) in D, the tuple is transformed from $\langle t.A_1, \ldots, t.A_m, c\rangle$ to $\langle f_1(t.A_1), \ldots, f_m(t.A_m), c\rangle$. We use $\vec{f}$ to denote the vector of the m transformations. Let $D' = \{\langle\vec{f}(t), c\rangle | \langle t, c\rangle \in D\}$ be the data set consisting of these transformed tuples.

We refer to a tuple $\langle t.A, c\rangle$ as an A-projected tuple, i.e., it retains the A-value and the class label. Whenever there is no confusion, we refer to an A-projected tuple as simply a tuple.

3.2 Domain, Subspace Association and Pattern Disclosure

Definition 1 *Let $f : \delta(A) \to \delta'(A)$ be a transformation. A domain crack function $g : \delta'(A) \to \delta(A)$ represents the guess the hacker makes on each transformed value. For a value $v' \in \delta'(A)$ that appears in D', a guess is a* crack *if the guess falls within a radius ρ from the actual value, i.e., $|g(v') - f^{-1}(v')| \leq \rho$. The* domain disclosure risk *is the fraction of the number of cracks to the number of distinct values of A' appearing in D'.*

Input privacy [8] refers to the protection of D and is measured using various metrics [2, 5], *domain disclosure* being the popular one, as discussed above. However, for many applications, the data custodian might care more about the association between domain values rather than domain disclosure. For example, for an insurance application, the company cares more about protecting Bob of age 45 earning 50K, rather than the individual values of age or salary. We refer to this as *subspace association disclosure*.

Definition 2 *Let $S \subseteq \{A_1, \ldots, A_m\}$ be a subset of attributes of the input training data. For simplicity, let $S = \{A_1, \ldots, A_s\}$. A subspace crack function $\vec{g} : (\delta'(A_1) \times \ldots \times \delta'(A_s)) \to (\delta(A_1) \times \ldots \times \delta(A_s))$ represents the guess the hacker makes on each transformed S-tuple. A guess is a* (S-tuple) crack *if $\bigwedge_{i=1}^{s} |\vec{g}_i(v'_i) - \vec{f}_i^{-1}(v'_i)| \leq \rho_i$ (radius for A_i). The* subspace association disclosure risk *is the fraction of the number of cracks to the number of S-tuples in D'.*

Definition 3 *Let a path in the decision tree T' be of the form: $\bigwedge_{i=1}^{h} A_i \theta_i v'_i$, where θ_i is a comparison operator. A path crack function $\vec{g} : (\delta'(A_1) \times \ldots \times \delta'(A_h)) \to (\delta(A_1) \times \ldots \times \delta(A_h))$ represents the guess the hacker makes on a transformed path. A guess is a* (path) crack *if $\bigwedge_{i=1}^{h} |\vec{g}_i(v'_i) - \vec{f}_i^{-1}(v'_i)| \leq \rho_i$. The* pattern disclosure risk *is the fraction of the number of cracks to the number of paths in T'.*

In output privacy, the focus is on protecting the data mining outcome from being disclosed. For decision trees, the paths of the tree are to be protected. Most studies adopting the random perturbation paradigm only focus on input privacy. The mining outcomes are not encoded. Since the hacker only has access to the perturbed data, one may argue that in a twisted sense, the exact identity of the pattern is protected inasmuch as perturbation changes mining outcome. But then the custodian suffers the same fate: she cannot fully recover the exact pattern. In contrast, for (anti-)monotone transformations, the true classifier is revealed to the data custodian (given the no-outcome-change guarantee to be shown in Section 4), whereas the hacker has to guess the true identities of the paths of the decision tree, out of many possibilities.

3.3 Attack Models Possibly with Prior Knowledge

Sources of prior knowledge may be from published statistics (e.g., the minimum `age` being 17, the median `salary` being 35K), from samples of similar data (e.g., a rival company having data similar to D), or the top k modal classes (e.g., the mode of employee `age` being 34). Alternatively, the hacker may know that a given attribute follows a certain distribution (e.g., Zipf, Gaussian), even though the parameters may not be known exactly. These various forms of prior knowledge can be captured by the notion of *knowledge points*.

Definition 4 *Let v' be a value of attribute A in D'. Let the hacker guess that $v' \in \delta'(A)$ corresponds to $v \in \delta(A)$. We say that (v, v') is a* knowledge point *if $|v - f_A^{-1}(v')| \leq \rho$.*

Following Definition 1, the radius ρ defined above is the same as the radius used for a crack. Moreover, the hacker can use these knowledge points to form the basis of a curve fitting attack.

Definition 5 (Curve Fitting Attack) *Let* (x_1, y_1), ..., (x_m, y_m) *be* m *knowledge points. Apply a curve fitting method to fit the points into a crack function* g.

We consider three curve fitting methods: (i) a regression line that minimizes the residuals of the m knowledge points; (ii) a polyline that connects the m knowledge points; and (iii) a spline that fits the m points.

Another attack model is *sorting*. Consider the `age` attribute again. Even though the defined range of age is within $[0, 100]$, for a practical training data set with say $10,000$ tuples on employees, it is almost guaranteed that every age in the interval $[20, 65]$ occurs in D at least once. Then the hacker takes the transformed values of `age` and sorts them in increasing order. Knowing the nature of a domain such as `age`, the hacker then maps the transformed values in increasing order to consecutive values starting with the (guessed) minimum value all the way to the (guessed) maximum. In the rest of this paper, we show how to safeguard against these attack models, and simultaneously provide the no-outcome-change guarantee.

4 The No-Outcome-Change Guarantee

In this section, we show that the decision tree constructed using D is identical to the tree constructed using D' when either the gini index or entropy is used to select the split-points. These two selection criteria are the most widely used.

Definition 6 Let $\langle t_1, \ldots, t_n \rangle$ be the *ordered* sequence of A-projected tuples in D such that $t_i.A \leq t_j.A$ whenever $i < j$. Equal values are in some canonical order. The class string $\sigma_{A,D}$ for attribute A in data set D is defined as the string obtained by concatenating the class labels in the ordered sequence of tuples. Whenever the data set D is obvious, we denote the class string simply as σ_A.

For the data set D in Figure 1(a), consider A being the `age` attribute. Sorting the tuples on `age` gives the class string $\sigma_{age} = \mathsf{HHHLHL}$, where H and L denote the class label `High` and `Low` respectively. For the attribute `salary`, the class string $\sigma_{salary} = \mathsf{HHHHLL}$. Let us compare the corresponding class strings in D' shown in Figure 1(b). Observe that the class string for each attribute is unchanged.

Lemma 1 Let a monotone function be applied to transform attribute A from D to D'. Then the class string is preserved, i.e., $\sigma_{A,D'} = \sigma_{A,D}$. Similarly, for an anti-monotone transformation, $\sigma_{A,D'} = (\sigma_{A,D})^R$, where σ^R denotes the reverse of string σ.

Definition 7 Given a class string σ for attribute A of a data set, we decompose σ into multiple non-overlapping substrings $r_1, \ldots, r_m$ such that: (i) $\sigma = r_1 \circ \ldots \circ r_m$; (ii) for all $1 \leq i \leq m$, the substring r_i consists of a single class label; and (iii) for all $1 \leq i \leq (m-1)$, the substrings r_i and r_{i+1} are of different class labels. Each piece r_i is a label run.

For the class string $\sigma_{age} = \mathsf{HHHLHL}$ in Figure 1, there are four label runs $\mathsf{HHH}, \mathsf{L}, \mathsf{H}$ and L. From Lemma 1, monotone functions preserve all the label runs of each attribute. Based on the definitions of the gini index and entropy, we have the following lemma.

Lemma 2 For each attribute, the split-point that optimizes either the gini index or entropy must not occur within a label run; it must be at the end points between two successive label runs.

Consider the first label run HHH in `age` in Figure 1, corresponding to the `age` values 17, 20 and 23 in D. The lemma says that the best split point for `age` cannot occur at `age = 20`. The candidate locations of the best split point for `age` must correspond to the end points of successive label runs – in this case 23, 32 and 43. Note that the same lemma applies to the transformed data D'. From Figure 1(b), the candidate locations of the best split point for `age'` are again the end points of successive label runs – in this case 31, 39, and 49. As formalized below, the candidate locations of the best split point are identical in D and D'.

Theorem 1 The winning attribute and the split point location in D' – relative to the sequential ordering of the label runs – is identical to that in D.

The theorem follows from the fact that in D', the label runs are preserved as in D. Consequently, all the candidate split point locations are preserved as well. Note the difference between split points and split point locations. E.g., consider $\sigma_{age} = \mathsf{HHHLHL}$ in Figure 1. The split point in Figure 1(a) is `age = (23+32)/2` and is located between the first and second label runs. In D', as shown in Figure 1(b), the actual value of the split point is changed to `age' = (31+39)/2 = 35`, providing output privacy. However, the split point is still at the same location with respect to the sequential ordering of the label runs. Finally, even though the data values change from D to D', the gini index and entropy calculations remain unchanged as they are based on relative frequencies. Hence, the winning attribute and the split point locations are preserved.

Let T' be the decision tree obtained by mining the transformed database D'. Construct another decision

```
Algorithm PieceTransform
Input: a sorted sequence of A− projected tuples
1.    Call Procedure ChooseBP or ChooseMaxMP to decompose
          A into $w$ pieces $r_1, \ldots, r_w$
2.    For each piece $r_i$, call ChooseFunction to choose $f_i$
3.    For $1 \leq i \leq w$, initialize $f_i$ such that they satisfy the
          global-monotone invariant, and apply $f_i$ to $\delta_i(A)$
```

Figure 2. A Piecewise Framework

runs in D :	*H*	*H*	*H*	*H*	*L*	*L*	*L*	*L*	*H*	*H*	*H*	*H*	*H*
values in D :	1	2	15	15	27	28	29	29	29	29	42	43	44
values in D':	20	17	16	16	2	4	5	5	5	5	12	13	14
sorted values in D':	2	4	5	5	5	5	12	13	14	16	16	17	20
runs in D':	*L*	*L*	*L*	*L*	*H*	*H*	*H*	*H*	*H*	*H*	*H*	*H*	*H*

Figure 3. Failing the Global-Monotone Invariant

tree S as follows. Start from the root of T' going top-down: for every node x' in T' of the form $A\theta v'$ (where θ is a comparison operator and v' is a value in $\delta'(A)$), create a corresponding node x in S of the form $A\theta f_A^{-1}(v')$, where $f_A : \delta(A) \to \delta'(A)$ is the data transformation used for A. Our main result is the following:

Theorem 2 Let D, D', T', S be as above. Let T be the decision tree obtained by mining D directly. Then: $S = T$.

5 A Piecewise Framework

To defend against the various attack models, the central idea of the proposed solution framework is to introduce *breakpoints* to break up the domain into multiple pieces, each of which is encoded by a different transformation function. Figure 2 gives a skeleton algorithm for implementing the piecewise framework. It consists of three main phases: (i) choosing breakpoints, (ii) choosing a transformation for each piece, and (iii) setting them up to satisfy a global constraint. We elaborate on all these aspects below.

5.1 ChooseBP: Choosing Breakpoint Locations Randomly

It is the primary objective that when breakpoints are added, the no-outcome-change guarantee is preserved. To do so, let us return to the concept of label runs introduced in Definition 7. Figure 3 shows a simple example when there are three label runs: $r_1 \equiv \mathsf{HHHH}, r_2 \equiv \mathsf{LLLL}, r_3 \equiv \mathsf{HHHHH}$. Suppose a breakpoint is introduced exactly between two successive label runs to create three pieces, each of which uses its own (anti-)monotone transformation (i.e., third row of Figure 3). In the transformed data, however, the label runs are different (i.e., fifth row), and create a new

runs in D :	*H*	*H*	*H*	*H*	*L*	*L*	*L*	*L*	*H*	*H*	*H*	*H*	*H*
values in D :	1	2	15	15	27	28	29	29	29	29	42	43	44
values in D':	20	17	16	16	23	24	25	25	25	25	102	101	100
sorted values in D':	16	16	17	20	23	24	25	25	25	25	100	101	102
runs in D':	*H*	*H*	*H*	*H*	*L*	*L*	*L*	*L*	*H*	*H*	*H*	*H*	*H*

Figure 4. Satisfying the Global-Monotone Invariant

class string. The key problem for the situation shown in Figure 3 is that the three pieces, after their individual transformations, no longer follow the original ordering among themselves. Specifically, as shown in the first row of Figure 3, the three label runs are in the ordering of $r_1 \equiv \mathsf{HHHH}$, followed by $r_2 \equiv \mathsf{LLLL}$, then by $r_3 \equiv \mathsf{HHHHH}$. However, after the transformations as shown in row 4, the runs r_2 and r_3 have moved ahead of r_1, thereby causing the class string to change. The solution to this problem is to ensure that the pieces satisfy a global constraint to preserve their relative ordering after transformation.

Definition 8 *Let the original domain be broken up into w pieces $\delta_1(A), \ldots, \delta_w(A)$ with w transformation functions $f_1, f_2, ..., f_w$. This set of transformations is said to satisfy the* global-monotone *invariant iff for all $1 \leq i < j \leq w, \forall v \in \delta_i(A), \forall u \in \delta_j(A)$, it is necessary that $f_i(v) < f_j(u)$. Similarly, the set is said to satisfy the global-anti-monotone invariant if the latter inequality is changed to $f_i(v) > f_j(u)$.*

Figure 4 shows how the global-monotone invariant is satisfied. Because the largest transformed value of r_1 is 20, the invariant requires that all the transformed values in r_2 be strictly greater than 20, which is the case in Figure 4. Notice that this invariant is satisfied even though an anti-monotone function has been applied to r_1, whereas a monotone function has been applied to r_2. Similarly, all the transformed values in r_3 are greater than 25, *irrespective* of whether a monotone or an anti-monotone function is applied to r_3. In this way, as shown in row 5 of Figure 4, the label runs in D' are identical to those in D.

While both Figure 3 and 4 consider putting breakpoints right at the boundaries between label runs, this is only a simplification for illustration purposes. In fact, *any* value can be a breakpoint location, *as long as the global-monotone invariant is obeyed.* This is essentially the nature of Procedure ChooseBP shown in Figure 5. For an attribute A, ChooseBP decomposes the domain of A into w pieces for some $w > 0$, i.e., $\delta(A) = \delta_1(A) \cup \ldots \cup \delta_w(A)$ and $\delta_i(A) \cap \delta_j(A) = \emptyset$, $i \neq j$. The w breakpoints are randomly selected from the set of distinct values of A. Even though ChooseBP

```
Procedure ChooseBP
Input: $D_{A,C}$, the number $w$ of breakpoints to be returned
1.  Set CBP to be $\{t.A | t.A \in D_{A,C}\}$, the set of A-values
2.  Randomly pick $w$ values from CBP as the breakpoints
3.  Return BP /* the set of selected breakpoints */
```

Figure 5. A Skeleton for ChooseBP

is simple, its privacy protection power arises from the fact that the number w and the exact w locations are not known to the hacker. Specifically, if the cardinality of CBP is N, then there are $O(2^N)$ combinations for the hacker to ponder over.

5.2 ChooseMaxMP: Exploiting Monochromatic Pieces

Definition 9 *A value $v \in \delta(A)$ in D is monochromatic if all the tuples with v as the A attribute value agree on the label, i.e., $\nexists t_1, t_2 \in D$ such that $t_1.A = t_2.A = v$ and $t_1.C \neq t_2.C$, where C is the class label. Furthemore, if all the tuples in a piece r contain monochromatic values and the same label, then r is called a monochromatic piece.*

In Figure 4, 29 is a non-monochromatic value; every other value, including 15, is monochromatic. The piece r_1 is a monochromatic piece. The key benefit offered by a monochromatic piece is that there is no requirement to apply either a monotone or an anti-monotone function. In Figure 4, the values in r_1: 1, 2, 15 are transformed anti-monotonically to 20, 17 and 16 respectively. Note that the label runs do not change if the transformation f is *any bijective* function. For example, 1, 2 and 15 can be transformed so that $f(1) < f(15) < f(2)$ (e.g., see row 3 of Figure 7). When such a bijective function can be used, a sorting attack is blocked. Furthermore, the space of bijective functions strictly contains the space of monotone or anti-monotone functions. Thus, while ChooseBP uses randomized breakpoints to confuse the hacker, the use of a bijective function here creates an even more serious combinatorial problem for the hacker. Specifically, if N is the total number of values in all the monochromatic pieces, there are $O(N!)$ combinations for the hacker to ponder over.

To maximize the benefit of monochromatic pieces, Procedure ChooseMaxMP, shown in Figure 6, finds all the monochromatic values and grows them into monochromatic pieces of the largest size. Given a sorted sequence of A-values, this task can be achieved by a simple scan from the smallest to the largest value.

To illustrate the procedure, consider Figure 7, which has the same original label runs as in Figures 3 and 4. Let us begin from the smallest value 1. Because the value 1 is monochromatic, 1 is added to BP in line (11).

```
Procedure ChooseMaxMP
Input: $D_{A,C}$ sorted on attribute A, the desired number $w$ of
        breakpoints to be returned
1.  Set BP to be empty, flag MP to false, curLabel to null
2.  For each value $v$ starting from the smallest {
3.      If $v$ is not monochromatic {
4.          If MP is true { /* end of a monochromatic piece */
5.              Add $v$ to BP /* a new non-monochromatic piece */
6.              Set MP to false, and curLabel to null
7.          } /* else simply skip to the next value */
8.      }
9.      else { /* $v$ is monochromatic */
10.         If MP is false { /* end of a non-mono. piece */
11.             Add $v$ to BP /* a new monochromatic piece */
12.             Set MP to true and curLabel to $v$.C.
13.         } else if curLabel $\neq v$.C { /* different label */
14.             Add $v$ to BP /* a different monochromatic piece */
15.             Set curLabel to $v$.C
16.         } /* else continues the current monochromatic piece */
17.     } } /* end for-loop */
18. If the size of BP is $h$ and is less than $w$,
19.     Randomly pick $(w - h)$ breakpoints, if any, from the set of
20.         non-monochromatic values as in ChooseBP
21. Return BP
```

Figure 6. A Skeleton for ChooseMaxMP

runs in D :	H	H	H	H	L	L	L	L	H	H	H	H	H
values in D :	1	2	15	15	27	28	29	29	29	29	42	43	44
values in D':	6	10	8	8	19	18	27	27	27	27	35	31	33
pieces obtained:		r_1			r_2			r_3				r_4	
sorted values in D':	6	8	8	10	18	19	27	27	27	27	31	33	35
runs in D':	H	H	H	H	L	L	L	L	H	H	H	H	H

Figure 7. Maximizing Monochromatic Pieces

The subsequent monochromatic values 2 and 15 are skipped according to line (16) effectively growing r_1. The next value is 27. While it is monochromatic, the label is different. Thus, a new monochromatic piece begins in line (13) and 27 is added to BP. The next value 28 remains in the same monochromatic piece containing 27. The next value 29 is non-monochromatic. This marks the end of the previous monochromatic piece and starts a non-monochromatic piece in line (4); 29 is added to BP. To complete the example, the next and last breakpoint is 42. In sum, ChooseMaxMP creates 4 pieces: $r_1 \equiv (1,H)(2,H)(15,H)(15,H)$, $r_2 \equiv (27,L)(28,L)$, $r_3 \equiv (29,L)(29,L)(29,H)(29,H)$ and $r_4 \equiv (42,H)(43,H)(44,H)$.

Because r_1, r_2 and r_4 are monochromatic, any bijective function can be used. Notice that the way that breakpoints are selected by ChooseMaxMP maximizes the number of values for which a bijective function can be applied. Row 5 in Figure 7 shows the transformed values. Notice that the global-monotone invariant is maintained between successive pieces.

Notice that there is no uncertainty in the positions of the monochromatic pieces as their sizes are maximized by ChooseMaxMP. In other words, the hacker knows exactly what the monochromatic and non-monochromatic pieces are in D'. However, knowing the pieces in D' does not help the hacker at all in cracking the corresponding values in D. For example, in Figure 7, knowing rows 5 and 6 do not help the hacker in cracking row 2.

In Figure 7, three important cases have not been illustrated. First, if the number of monochromatic pieces is less than w, ChooseMaxMP still operates by essentially selecting the remaining breakpoints randomly from among the non-monochromatic values. Experimental results later will show that even in this rare case, the proposed framework offers adequate domain disclosure protection. And whenever there are monochromatic pieces, the framework exploits them to give enhanced protection.

Second, if there are not enough non-monochromatic values to choose from, ChooseMaxMP returns all the non-monochromatic values, together with the starting values of the monochromatic pieces. For real data, this is a very good situation for the data custodian as it implies that the total number of values in the monochromatic pieces dominates.

Finally, in our simple example, the shortest piece is of length 2. We include short pieces in our example for simplicity. In practice, ChooseMaxMP may impose a minimum width threshold (e.g., width ≥ 5).

5.3 Selecting Transformations Randomly

After breakpoints are selected, the next step is to choose a transformation for each piece from a family of functions. Let $\mathcal{F}_{\mathtt{bi}}$ denote a set of bijective functions including non-monotone ones, e.g., any permutation functions. This set $\mathcal{F}_{\mathtt{bi}}$ is applicable to monochromatic pieces. For non-monochromatic pieces, we are restricted to the family of (anti-)monotone functions, denoted by $\mathcal{F}_{\mathtt{mono}}$. This set can contain polynomials of degree $\geq k$, logarithmic functions, parabolic functions and so on. Note that $\mathcal{F}_{\mathtt{mono}}$ *is closed under composition.* That is to say, for any monotone functions $f, g \in \mathcal{F}_{\mathtt{mono}}$, their composition $f \circ g$ is also monotone and is in $\mathcal{F}_{\mathtt{mono}}$. As in the case for a monochromatic piece, a randomization step is used to select the transformation for the non-monochromatic piece.

5.4 Defense Against Sorting Attacks

Next we turn to sorting attacks. As specified in Procedure ChooseFunction, if a piece r is monochromatic, f_r is chosen from $\mathcal{F}_{\mathtt{bi}}$, and a sorting attack is blocked. What about a non-monochromatic piece? Note that a non-monochromatic piece r with a monotone transformation can already have its *own inherent* protection against a sorting attack – provided that there are enough "discontinuities" within r. A value $v \in \delta_r(A)$ is a *discontinuity* if there is no tuple t in r with $t.A = v$, where $\delta_r(A)$ denotes the dynamic range $[min_A, max_A]$ of A in r, and min_A, max_A denote the least and greatest A-value occurring in r. To be more precise, if $\delta_r(A)$ contains some discontinuity values, then the attacker can only crack a value $v' \in \delta'_r(A)$ to within a range, say $\mathcal{R}_g = [v_1, v_2] \subseteq \delta_r(A)$. Recall from Definition 1 that a guess is a crack if $|g(v') - f^{-1}(v')| \leq \rho$. Let $\mathcal{R}_\rho = [v-\rho, v+\rho]$, where $v = f^{-1}(v')$. Then the probability that a value v' is cracked under sorting attack can be redefined as: $P(|(g(v') - f^{-1}(v')| \leq \rho) = \frac{|\mathcal{R}_g \cap \mathcal{R}_\rho|}{|\mathcal{R}_g|}$.

Let us consider the entire sorted sequence in D' in Figure 7 (i.e., row 5). Consider $v' = 27$. There are 5 values ranked ahead of 27 and 3 values ranked after 27. Thus, given that the original domain is [1,44], $g(v')$ can range from $\mathcal{R}_g$= [6,41]. As shown, the original value is $f^{-1}(v') = 29$. Let us say that the width of a crack is 2, giving $\mathcal{R}_\rho = [27, 31]$. Thus, the probability that v' is cracked is 5/36. From the above formula, it is clear that the larger the number of discontinuities, the wider the range $\mathcal{R}_g$ and the smaller the crack probability.

In sum, the data custodian can follow the "recipe" below to determine if an attribute A is safe for disclosure. If A has many monochromatic pieces, or if the non-monochromatic pieces contain many discontinuities, then A is safe with low crack percentage against a sorting attack. The only situation that is unsafe is when A has few monochromatic values and *simultaneously* few discontinuities. However, the data custodian must decide whether domain disclosure risk for A is a primary concern or not. As discussed before, perhaps the more important issue for the custodian is the association of the A-values with the values of other attributes, not just the A-values themselves. Empirical results in Section 6 will examine this perspective.

A final point concerns the amount of information the data custodian needs to keep in order to decode the decision tree T' to get T. While this is discussed in [4], it suffices to say that the information required is rather minimal (i.e., the locations of breakpoints and the transformations used).

6 Empirical Evaluation

6.1 Experimental Setup

Data Sets: We conducted our experimental evaluation on many benchmark data sets, including the forest covertype, census income and WDBC data sets from the UC Irvine collection. The results reported below are based on the forest covertype data set. These results are representative of those that we do not have space to show.

The forest covertype data set consists of 581,012 tuples and 10 numeric attributes (and other non-numeric attributes). The table in Figure 8 gives the key statistics of the 10 attributes. There are attributes with

attr.	dynamic range width	# of distinct values	# of mono. piece	avg Length of mono. piece	total % of mono. values
#1	2000	1978	9	163	74.2%
#2	361	361	0	0	0.0%
#3	67	67	1	15	22.4%
#4	1398	551	22	10	40.0%
#5	775	700	14	24	48.0%
#6	7118	5785	202	18	62.9%
#7	255	207	2	41	39.6%
#8	255	185	8	6	25.9%
#9	255	255	3	8	9.4%
#10	7174	5827	229	17	66.8%

Figure 8. Statistics of Attributes

many discontinuities; the number of discontinuities is the difference between the second column and the third column of the table. But attributes 2, 3 and 9 have no discontinuity. There are also attributes with several monochromatic pieces. The last column gives the percentage of values that are contained in the monochromatic pieces. For attribute 1, the percentage is the highest at 74%. But for attribute 2, the percentage is 0% as there is no monochromatic piece. This attribute represents the worst case, as it also does not contain any discontinuity.

Transformations: $\mathcal{F}_{\mathtt{bi}}$ consists of transformations for monochromatic pieces. We used a random permutation function. $\mathcal{F}_{\mathtt{mono}}$ is for non-monochromatic pieces. We used linear and higher order polynomials, log, and $\log^{\frac{1}{2}}$ (denoted sqrt(log)).

Prior Knowledge: Curve-fitting attacks – linear regression, spline, polyline – require some number of knowledge points (denoted as KP below) to work with. We simulated the **ignorant hacker** who has no prior knowledge. Clearly, curve fitting can also be adversely affected by knowledge points that the hacker thought were accurate but that turn out to be way off. We call a knowledge point (ν, ν') to be *good* if it satisfies Definition 4; but call it *bad* if $|\nu - f_A^{-1}(\nu')| > 5\rho$. The width ρ for a KP varied from 1%, 2% and 5% of the width of the dynamic range. Given the definition of ρ's, we feel that each good KP represents a considerable amount of prior knowledge. Thus, a hacker with 2 or 4 good KPs is referred to as a **knowledgeable** and an **expert** hacker respectively. The locations of KPs are selected randomly. Furthermore, given the randomization involved, each disclosure risk figure reported below is based on the median of 500 random trials. The experiments were implemented in a MATLAB environment. We ran the experiments on an Intel Pentium PC with 3GHz CPU and 2GB RAM.

6.2 Domain Disclosure Risk

6.2.1 Impact of Breakpoints, Monochromatic Pieces and KPs

Figure 9 shows the domain disclosure risks (i.e., the y-axis) for all 10 attributes of the forest covertype data set using polyline as the curve fitting model. For each attribute, the first bar gives the baseline when no breakpoint is used but with an expert hacker (i.e., 4 KPs). The second and third bar correspond to the cases when ChooseBP and ChooseMaxMP are used, with an expert hacker. Note that to make the comparisons between ChooseBP and ChooseMaxMP fair, ChooseBP uses the same number of breakpoints as ChooseMaxMP, which is determined by the number of monochromatic pieces for the attribute. (The minimum number of breakpoints in each case is set to $w = 20$.) The last bar corresponds to the case when ChooseMaxMP is used with a knowledgeable hacker. For each attribute, the runtime of ChooseMaxMP is within 1-2 seconds.

The difference between the first two bars indicates the reduction of crack percentage with breakpoints. For instance, for attribute 1, the crack percentage drops from over 65% to 30%. Reduction is achieved for every attribute. Even for the "worst case" attribute 2 as shown in Figure 8, breakpoints manage to keep the crack percentage below 25%. This shows that the use of breakpoints alone is effective, *even when there is no monochromatic piece.*

The difference between the second and third bar in Figure 9 is due entirely to the monochromatic pieces. Depending on the percentage of monochromatic values, the reduction can be very significant. For the first attribute, the crack percentage drops from 30% to below 10%. This shows the effectiveness of ChooseMaxMP in exploiting monochromatic pieces, if present.

As a reference point, when random perturbations are used, there is a chance that a discrete data value is not changed and the true value is revealed. For example, in [8], many situations examined leave a significant percentage (e.g., 30%) of values unchanged, even with no prior knowledge used. As shown below, against an ignorant hacker with no prior knowledge, the crack percentage using our framework is consistently below 5%. Furthermore, the studies in [7, 6] show that when spectral analysis techniques are used, the crack percentage can be significantly increased.

The third bar of each attribute in Figure 9 corresponds to an expert hacker (i.e., 4 KPs). For all 10 attributes, the crack percentage is significantly lower when the hacker has less domain knowledge. The fourth bar of each attribute shows the crack percentage when the hacker is only a knowledgeable hacker (i.e., 2 KPs). In all cases, the crack percentage falls below 15%. And for an ignorant hacker, the crack percentage falls below 5%. Finally, the crack percentage is sensitive to the presence of even a single bad KP. For example, for attribute 10, the crack percentage with 4 good KPs drops from about 20% to around 10% with

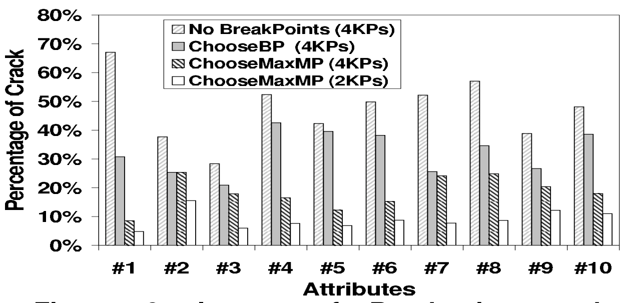


Figure 9. Impact of Breakpoints and Monochromatic Pieces on Domain Disclosure

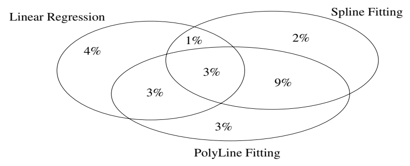


Figure 10. The Venn Diagram of Cracks: the Combination Attack

the introduction of a single bad KP.

6.2.2 A combination of Attacks

So far all the results presented here are based on the polyline curve fitting model and the sqrt(log) transformation function. The table below shows the corresponding values for alternative situations. All the figures are based on attribute 10 of the forest covertype data set with ChooseMaxMP and an expert hacker. It is clear that results shown above represents a "worst case" analysis.

	polynomial	log	sqrt(log)
Linear regression	10.39%	11.53%	10.85%
Spline attack	14.51 %	14.8 %	15.28%
Polyline attack	15.55 %	18.05 %	18.03%

A natural extension to the various attacks analyzed so far is to consider when the hacker applies all of these attack models. The central question here is whether the different attack models crack similar or different items. Figure 10 shows a Venn diagram of the cracks for linear regression, spline fitting and polyline fitting on attribute 10 using sqrt(log) as the transformation. For instance, 3% of the domains are cracked solely by polyline, whereas an additional 9% are cracked by polyline and spline but not by linear regression.

The question to ask here is: what the disclosure risk for this combination attack is. One simple answer is to add up all the percentages, which gives rise to 25%. However, for most situations, this is an over-estimation of risk. To illustrate, suppose that a specific item $\mathtt{a}$ is identified to be $\mathtt{a}$ by polyline, $\mathtt{b}$ by spline and $\mathtt{c}$ by linear regression. While it is true that *one* of the three attacks correctly reveals the identity of item $\mathtt{a}$,

attr.	# of discon-tinuities	total % of mono. values	worst case crack % by sorting
#1	22	74.2%	26%
#2	0	0.0%	100%
#3	0	22.4%	78%
#4	847	40.0%	4%
#5	75	48.0%	22%
#6	1333	62.9%	8%
#7	48	39.6%	13%
#8	70	25.9%	11%
#9	0	9.4%	90%
#10	1347	66.8%	7%

Figure 11. Sorting Attack: Worst Case

the hacker does not know *which* one. Thus, he cannot distinguish whether $\mathtt{a}$ is really $\mathtt{a}, \mathtt{b}$ or $\mathtt{c}$. One way to calculate the percentage of cracks is to use expected values. Assuming the hacker trusts the three crack models equally, this gives an expected crack percentage of 12.5%. Another way is to count only those items as cracks if two or more methods agree on them. From the Venn diagram shown in Figure 10, this gives a total combined crack percentage of 16%.

6.2.3 Sorting Attack: the Worst Case Analysis

Finally, we measure the risk of a sorting attack on all 10 attributes. The table in Figure 11 gives the worst case results because we assume that the hacker has the knowledge of the true minimum and maximum values of the dynamic range. If these two pieces of prior knowledge is not known, the crack percentage is significantly lower. The second column of the table in Figure 11 gives the number of discontinuities within the dynamic range (cf: Figure 8). The third column is directly copied from the last column of Figure 8. As expected, the three attributes 2, 3 and 9 are the most vulnerable with no discontinuity, and a small percentage of monochromatic values (hardly surprising given that these attribute domains are small and the data set contains over 500,000 rows). However, as long as an attribute has a fair number of discontinuities or monochromatic values, the attribute is safe against a sorting attack even in the worst case.

6.3 Subspace Association Risk

Next we turn to subspace association disclosure risk. For the 10 attributes of the forest covertype data set, there are over 1,000 subspaces containing two or more attributes. Figure 12 shows the results of a few selected subspaces. All subspaces shown contain either 2 or 3 attributes. For the ones selected, they are divided into two categories.

The first category are attributes for which the curve fitting attack is more serious than the sorting attack. Specifically, we consider the set of attributes $\{4, 7, 10\}$ and their subspaces. The first three bars in Figure 12 show their individual domain disclosure risks from Figure 9 with an expert hacker. The next four bars show

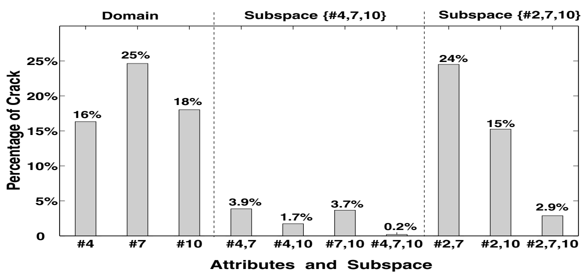


Figure 12. Subspace Association Disclosure Risk

the subspace association risks for all the combinations. For example, the individual risks of attribute 4 and 7 are 16% and 25% respectively. The association disclosure risk of attributes 4 and 7 together, however, drops significantly to 4%. And the association disclosure risk among attributes 4, 7 and 10 drops to 0.2%. In general, as the subspace becomes larger, the disclosure risk drops significantly.

The second category are attributes for which the sorting attack is more serious. For a worst case discussion, we return to attribute 2. Even though attribute 2 is 100% cracked by sorting in the worst case, note that the association disclosure risks of attribute 2 with other attributes can be acceptable (i.e., last three bars of Figure 12). Thus, for the data custodian to decide whether attribute 2 can be safely released, the data custodian needs to first determine which is the primary concern: attribute 2 alone versus the association of attribute 2 with other attributes. For many situations, it is the association disclosure that is more critical.

It is also interesting to compare the association disclosure risk with and without attribute 2 in Figure 12. While the domain disclosure risk of attribute 10 alone is 18%, the association disclosure risk of attributes 2 and 10 together is 15%. This is due to how values in attribute 2 are associated with values in attribute 10. This skew leads to the following observation: $\mathsf{risk}(A, B) < \mathsf{risk}(A) * \mathsf{risk}(B)$. This is good news for the data custodian if the primary concern is subspace association risk, rather than domain risk.

6.4 Output Privacy: Pattern Disclosure Risk

Finally, to measure pattern disclosure risk, we applied the C4.5 decision tree algorithm on the 10 attributes of the forest covertype data set. The decision tree constructed contains 1707 paths from the root. The maximum length of these paths is 40. The following table shows the frequency of path lengths and the corresponding cracks.

Length of Decision Paths	1	2	3	4	5	6	> 6
# of Paths	2	4	5	9	15	24	1648
# of Cracks	1	0	0	0	0	0	0

Among all the 1707 paths from the root, there is only 1 path of length 2 that is cracked. In fact, this result is based on an insider hacker (i.e., 8 KPs) and a 5% width. For a less powerful hacker, or a smaller radius, *all* the paths are protected showing impeccable output privacy protection.

7 Future Work

In ongoing work, we study how to generalize the piecewise framework from decision trees to SVM and other kernel methods. The difference is that the dividing planes can have arbitrary orientations. In a forthcoming paper, we show how the no-outcome-change guarantee and the other two pillars of privacy can be supported for SVM.

References

[1] N.R. Adam and J. C. Wortman. Security-control methods for statistical databases. *ACM Computing Surveys*, 21, 4, pp515–556, 1989.

[2] R. Agrawal and R. Srikant. Privacy preserving data mining. *Proc. 2000 SIGMOD*, pp. 439–450.

[3] R. Agrawal et al. Order Preserving encryption for numeric data. *Proc. 2004 SIGMOD*, pp. 563–574.

[4] S. Bu et al. Preservation Of Patterns and Input-Output Privacy. Full version of this submission: ftp://ftp.cs.ubc.ca/local/laks/papers/ decisionTree-Icde2007ExtendedVersion.pdf.

[5] A. Evfimievski, J. Gehrke and R. Srikant. Limiting privacy breaches in privacy preserving data mining. *Proc. 2003 PODS*, pp. 211–222.

[6] Z. Huang, W. Du and B. Chen. Deriving private information from randomized data. *Proc. 2005 SIGMOD*, pp. 37–48.

[7] H. Kargupta et al. On the privacy preserving properties of random data perturbation techniques. *Proc. 2003 ICDM*, pp. 99–106.

[8] S. Rizvi and J. Haritsa. Privacy-preserving association rule mining. *Proc. 2002 VLDB*, pp. 682–693.

[9] L. Sweeney. K-anonymity: a model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10, 5, p.557-570, 2002.

[10] J. Vaidya and C. Clifton. Privacy-Preserving K-Means Clustering over Vertically Partitioned Data. *Proc. 2003 SIGKDD*, pp. 206–215.