k-Anonymous Decision Tree Induction

Arik Friedman, Assaf Schuster, and Ran Wolff

Technion – Israel Institute of Technology
Computer Science Dept.
{arikf,assaf,ranw}@cs.technion.ac.il

Abstract. In this paper we explore an approach to privacy preserving data mining, which relies on the k-anonymity model. The k-anonymity model guarantees that no private information in a table can be linked to a group of less than k individuals. We suggest extended definitions of k-anonymity, that allow to determine the k-anonymity of a data mining model. Using the definitions we present decision tree induction algorithms that are guaranteed to maintain k-anonymity of the learning examples. Experiments show that embedding anonymization within the decision tree induction provides better accuracy than anonymizing the data first, and inducing the tree later.

1 Introduction

In recent years, the effectiveness of data mining tools in revealing the knowledge locked within huge databases has raised the concern about its impact on the privacy of individuals. Two main approaches to privacy preserving data mining were suggested: The *data transformation approach* – e.g., [2] – tries to modify the data so as to hide the sensitive part of it while retaining interesting pattern. The *cryptographic approach* – e.g., [5, 11, 18] – uses cryptographic tools to prevent information leakage during the computation of the data mining model. The latter approach only applies to distributed data mining and does not prevent leakage due to the model itself (see, [8]).

k-Anonymity is a definition for privacy which was conceived in the context of databases and which has come a long way in the public arena. Roughly speaking, k-anonymity provides a "blend in the crowd" approach to privacy. It assumes that the owner of a data table can separate the columns into *public* ones (*quasi-identifiers*) and *private* ones. Public columns may appear in external tables, and thus be available to an attacker. private columns contains data which is not available in external tables and needs to be protected. The guarantee provided by k-anonymity is that an attacker would not be able to link private information to groups of less than k individuals. This is enforced by making certain that every combination of public attribute values appears in at least k rows of the released table, or in no row at all. k-Anonymity is today accepted by both legislators and corporations, and considered as providing the kind of privacy required by legislation (i.e., by HIPAA [17]).

Table anonymization is NP-Hard [13]. Thus, heuristic efficient anonymization of tables is the concern of most work in the area [1, 3, 6, 7, 10, 16]. Specific care is given to preserving as much of the original data as possible. Interestingly, some of this work deals with preserving data which would be useful should the table be data mined following its release [7, 3, 6]. Data mining is envisioned as the main target application of released data.

This paper takes a direct approach for the combination of k-anonymity and data mining. instead of asking how data can be anonymized such that it is useful for data mining, we ask how can data be mined such that the resulting model is guaranteed to provide k-anonymity. We specifically discuss this question in the context of decision tree induction. Hence, we describe a decision tree induction algorithm whose output is guaranteed not to compromise k-anonymity of the data from which it was induced.

Our approach is superior to existing methods (i.e., [3, 6, 7]) which guarantee k-anonymity of a data mining model by building it from a k-anonymized table. This is for some reasons: Firstly, for sake of efficiency, these methods choose to generalize one attribute at a time, and when they do, generalize all of the tuples at once. This kind of anonymization was termed *single-dimension recoding* [9]. Secondly, these methods are driven by heuristic cost metrics (e.g., classification metric – in essence the classification error over the entire data) which are not optimal for data mining. consequently, we show that a decision tree induced using our method is usually more accurate than that induced by existing methods. Needless to say, both decision trees provide the same level of anonymity.

This paper makes the following contributions:

- It suggests extended definitions of k-anonymity, which allow to determine the k-anonymity of a data mining model with respect to the learning examples.
- It demonstrates how the definitions of k-anonymity can be applied to determine the anonymity of a decision tree.
- It presents a decision tree induction algorithm which guarantees k-anonymous output and which performs better than existing methods in terms of accuracy on standard benchmarks.

Additionally, the extension of k-anonymity, and of the related l-diversity, model to data mining provides a new data transformation method for privacy preserving data mining. Unlike previous data transformation methods, this new extension has a clear model of the attacker and of the privacy guaranteed. Furthermore, this method is backed by privacy related legislation.

The organization of this paper is as follows: Section 2 outlines the extended definitions of k-anonymity of data mining models. Section **??** demonstrates how the definitions can be incorporated within decision tree induction algorithms to guarantee k-anonymous output. Section 4 compares this new algorithm experimentally to previous work. Conclusions are drown in Section 5.

2 Extending k-Anonymity to Models

We start by extending the definition of k-anonymity beyond the release of tables. Just as in the original k-anonymity model, we assume that the data owner can determine which attributes are known to a potential attacker and can be used to de-identify individuals, and which attributes are private knowledge. Additionally, without loss of generality, we assume that each tuple in the database pertains to a different individual.

Definition 1. *[A Private Database] A private database is a collection of tuples from a domain* $D = A \times B = A_1 \times ... \times A_k \times B_1 \times ... \times B_\ell$. $A_1, \ldots, A_k$ *are public attributes (a.k.a. quasi-identifiers) and* $B_1, \ldots, B_\ell$ *are private attributes.*

We denote $A = A_1 \times \ldots \times A_k$ the *public subdomain* of D. For every tuple $x \in D$, the projection of x into A, x_A, is the tuple in A that has the same assignment to each public attribute as x. The projection of a table T onto A is denoted $T_A = \{x_A : x \in T\}$.

Definition 2. *[A Model] A model* M *is a function from a domain* D *to an arbitrary output domain* O.

Every model induces an equivalence relation on D, i.e., $\forall x, y \in D$, $x \equiv y \Leftrightarrow M(x) = M(y)$. Respectively, the model partitions D into equivalence classes such that $[x] = \{y \in D : y \equiv x\}$.

In our terminology, a decision tree is a function that assigns bins to tuples in D. According to this, every bin within every leaf constitutes an equivalence class. Two learning examples which fit into the same bin cannot be distinguished from one another using the tree, even if they do not agree on all attribute values.

The users of the model are intended to be the data owner or the client. Both know all the attribute values of the client's tuple, and therefore, given the tuple they can compute the function of the model. We now define the way the model is perceived by an attacker, who only knows the values of public attributes in some external table, but can nevertheless try to use the model.

Definition 3. *[A Public Identifiable Database] A public identifiable database* $T_{ID} = \{(id_x, x_A) : x \in T\}$ *is a projection of a private database* T *to the public subdomain* A*, such that every tuple of* T_A *is associated with the identity of the individual to whom the original tuple in* T *has pertained.*

Given a tuple $(id_x, x_A) \in T_{ID}$, the private attribute values of x are unknown to the attacker, and depending on those values there could be a number of possible results for $M(x)$. Consequently, there may be several equivalence classes $[x]$ to which an attacker can associate x_A. We call this set of equivalence classes the *span* of x_A.

Definition 4. *[Span] Given a model* M*, the span of a tuple* $a \in A$ *is the set of equivalence classes induced by* M*, which contain tuples* $x \in D$ *whose projection into* A *is* a*. Formally,* $S_M(a) = \{[x] : x \in D \wedge x_A = a\}$*. When* M *is evident from the context, we will use the notation* $S(a)$.

Just like a model induces an equivalence relation on D, the spans induce an equivalence relation on A, i.e., $\forall x, y \in A$, $x \equiv_S y \Leftrightarrow S(x) = S(y)$. Respectively, the spans partition A into equivalence classes such that $[S(x)] = \{y \in A : y \equiv_S x\}$. Given a public identifiable database T_{ID}, we will use $[S(x)]_{T_{ID}} = \{(id_y, y_A) \in T_{ID} : y_A \in [S(x)]\}$ to denote tuples from T_{ID} associated with the equivalence class $[S(x)]$.

It is important to note here that tuples from D may belong to the same equivalence class according to the model, while their projections on A may have different spans. The decision tree in Figure 1 demonstrates this. The decision tree was formed using the data in Table 1. The *Marital Status* attribute is public, while the *Sports Car* and *Loan Risk* attributes are private. The tuples of *Anna* and *Ben* belong to the same bin, because of the *Sports Car* and *Loan Risk* attributes. The model ignores the value of the *Marital Status* attribute for these tuples. On the other hand, an attacker, which has no access to the *Sports Car* attribute values, is forced to consider routing *Anna*'s tuple to the leaf $l_{Unmarried}$, and routing *Ben*'s tuple to the leaf $l_{Married}$, in addition to routing each of them to the leaf l_{No}. Therefore, the decision tree implies just two spans: $\{l_{Married/good}, l_{Married/bad}, l_{no/good}, l_{no/bad}\}$ for *John*, *Ben*, and *Laura*, and $\{l_{Unmarried/good}, l_{Unmarried/bad}, l_{no/good}, l_{no/bad}\}$ for *Lisa*, *Robert*, and *Anna*.

Name	Marital Status	*Sports* Car	*Loan Risk*
Lisa	Unmarried	*Yes*	*good*
John	Married	*Yes*	*good*
Ben	Married	*No*	*bad*
Laura	Married	*No*	*bad*
Robert	Unmarried	*Yes*	*bad*
Anna	Unmarried	*No*	*bad*

Table 1. Mortgage company data

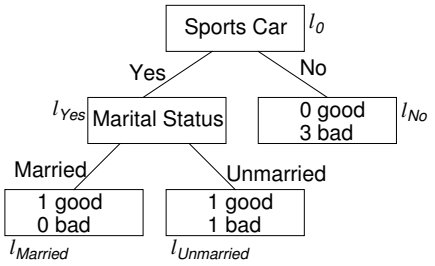

Fig. 1. A k-anonymous decision tree

The structure of the model, and potentially additional information about the equivalence classes, depending on the specific data mining model in use, provide knowledge about the distribution of private attribute values within any equivalence class. However, tuples that have the same span are indistinguishable for the attacker with respect to the model. Hence the attacker can link private information only to groups of tuples with the same span.

Definition 5. *[Linking attack using a model] A linking attack using a model M and a public identifiable database T_{ID} is performed by dividing the individuals described in T_{ID} to the different spans implied by M, i.e., by computing $S(x_A)$ for each individual $(id_x, x_A) \in T_{ID}$. Individuals that share the same value $S(x_A)$ are grouped together. Those individuals are now linked to any private information disclosed on this span.*

For instance, an attacker who knows the identity, gender and marital status of each of the mortgage company's clients in Table **??** can learn, by applying

the model, that *Donna*, *Emily* and *Fiona* will be classified by means of *leaf 3* [*Female, Married*]. This leaf constitutes the span of the relevant tuples. It contains two equivalence classes: one, with a population of 3, of individuals who are identified as *bad* loan risks, and another, with a population of 0, of individuals who are identified as *good* loan risks. Therefore the attacker can link *Donna*, *Emily* and *Fiona* to 3 *bad* loan risk classifications. This example stresses the difference between *anonymity* and *inference* of private data. Anonymity guarantees "blending in the crowd", and depends only on the size of a group of identifiable individuals, regardless of inferred private attribute values. Hence, so long as the k constraint is 3 or less, this information alone does not constitute a k-anonymity breach.

Definition 6. *[k-anonymous model] A model M is k-anonymous with respect to a public identifiable table T_{ID} if a linking attack on the tuples in T_{ID} using the model M will not succeed in linking private data to fewer than k individuals. In other words, a model M is k-anonymous with respect to T_{ID} if, for every $(id_x, x_A) \in T_{ID}$, $|[S(x_A)]_{T_A}| \geq k$.*

3 Inducing k-Anonymous Decision Trees

This section presents an algorithm which only induces k-anonymous decision trees. The algorithm is based on the well known ID3 algorithm (Quinlan, [14]) and on its extension to real valued attributes (C4.5, [15]). ID3 applies greedy hill-climbing to construct a decision tree. Starting from a root that holds the entire learning set, it chooses the attribute that maximizes the information gain, and splits the current node into several new nodes. The learning set is then devided among the new nodes according to the value each tuple takes on the chosen attribute, and the algorithm is applied recursively on the new nodes.

The k-anonymity preserving equivalent of ID3, kADET (Algorithm 1), Uses the same hill-climbing approach, with two changes: One, when considering all possible splits of a node, kADET eliminates those splits which would lead to a tree which breaches k-anonymity. Two, the algorithm is not recursive. Instead, at every moment, all of the potential node-attibute pairs are considered in a single priority queue and the best k-anonymity retaining one is picked. This is because but k-anonymity is defined in terms of public spans, which may include private spans associated with several decision tree nodes. A decision regarding one decision tree node may, thus, influence other nodes.

3.1 kADET Algorithm

The input for kADET is a private database T, the identification of the public attributes P, of the private ones Q, and of the class attribute C, and the anonymity argument k. First, kADET computes the initial set of equivalence classes (bins) and spans: a single span containing all of the bins and all of the tuples if the class is private, and as many spans as bins, with each span containing a single bin and

its tuple population in case the class is public. If one of the spans contains less than k tuples from T kADET returns *null* and terminates. Otherwise, kADET creates the initial queue of possible splits, each of which contains the root node and one of P or Q. The queue is ordered according to the gain from each split. kADET then enters the algorithm's main loop.

The main loop of kADET has the following steps: First, the most gainful candidate split $(node, attribute, gain)$ is popped out of the queue. If the node regarded in this split was already splitted, the candidate is purged. Otherwise, kADET tests whether spliting the node according to the suggested attribute would breach k-annonymity. If it would then, again, this candidate is purged. However, if the attribute can be generalized, then a new candidate is inserted to the queue with the generalized attribute this time. Last, if k-anonymity is not breached, the node is splitted.

To split a node several actions are performed: First, every bin of the parent node is splitted between the decendents nodes, and the spans containing the original bins are updated with the new list of bins. The decendent nodes are initialized with the same list of spans their parent had, and are added to the nodes lists of those spans. In case the splitting attribute is private, no further action is required, because the attacker is not able to distinguish the new bins from one another. However, if the splitting attribute is public, than any span associated with the original node can be devided by the attacker into as many spans as there are values to that attribute. Therefore, each of the updated spans has to be splitted into new spans, which contains bins from one new decendent node, and all of the bins associated with other nodes pointed to by the original span. The population of the original span has to likewise be devided according to the value the tuples take on the splitting attributes. Every new node points to (is associated with) all of the spans which contain its bins. Nodes whose bins are contained in the new spans, and which are not decendents of the original node, are associated with all of the new spans. This is, as explained in Section 2, because tuples that are routed to them may end up in any of the new nodes, if the private values are varied.

Figure 3 demonstrates an execution of a k-anonymous-ID3 algorithm, using the data in Table 1 as input. *Marital Status* is a public attribute; *Sports Car* and *Loan risk* are private attributes. The result of the execution is the decision tree in Figure 1.

3.2 Correctness and Overhead Analysis

The key to proving correctness of the algorithm is showing that the population of each model span, as computed by the algorithm, is the same one defined in Section 2: the set of tuples which, varying private attribute values, may end in the same set of bins. The proof is omitted here due to lack of space.

The computational overhead incurred by the algorithm, respective to that of ID3, stems from the need to compute and track all of the different spans. At worst, the number of those spans can reach the size of the public domain A. To see how, consider a tree in which all of the top nodes split on private attributes,

Algorithm 1 k-Anonymous Decision – kADET Tree

Input: T – private dataset, P – public attributes, Q – private attributes, C – the class attribute, k – anonymity parameter
Data Structures: See figure 2
procedure MAIN
    Create *root* node in $Tree$. Create one bin for each value of C and divide T among the bins. Set the Class variable to the value with the largest bin.
    if $C \in Q$ **then**
        Create one span for every value of C such that each span contains the index of the corresponding bin, and its population is the tuples of that bin. For each span set $Nodes$ to $\{root\}$ and set $root.Spans$ to the list of all spans.
    else
        Create a single span. Set $span.Bins$ to the list of all bins, the population of the span to T, $Nodes$ to $\{root\}$ and $root.Spans$ to the single span.
    if $\exists span \in SpanList$ such that $|span.population| < k$ **then return** nil
    for $att \in P \cup Q$ **do** add $(root, att, Gain\,[root, att])$ to the Queue.
    while $QUEUE \neq \emptyset$ **do**
        Let $(n, a, gain) = \arg\max_{gain} \{Queue\}$
        if $n.sons \neq \emptyset$ **then** Continue
        if $Breach\,(n, a, k)$ **then**
            if a has generalization a' **then** insert $(root, a', gain\,[root, a'])$ to $QUEUE$
            Continue
        $Split\,(n, a)$

procedure BREACH(*node*, *att*)
    if $att \in Q$ **then return** true
    for $v \in att.values$ **do**
        for $span \in node.Spans$ **do**
            if $|\{t \in span.population : t[att] = v\}| < k$ **then return** false
    return true

procedure SPLIT(*node*, *att*)
    for $v \in att.values$ **do**
        Let $node.sons\,[v]$ be a new decendent node
        Let $node.sons\,[v]\,.Bins\,[b]$ be a new bin, which refines $node.Bins\,[b]$ such that $node.sons\,[v]\,.Bins\,[b]\,.tuples \leftarrow \{t \in node.Bins\,[b]\,.tuples : t[att] = v\}$
        Let $node.sons\,[v]\,.Spans \leftarrow node.Spans$
    for $span \in node.Spans$ **do** replace each bin of the original node with its refinements, computed above
    if $att \in P$ **then**
        for $v \in att.values$ and $span \in node.Spans$ **do**
            Remove *span* from every node $n \in span.Nodes$
            Create a new span s such that $s.Nodes$ contains $span.Nodes \setminus \{node.sons\,[u] : u \neq v\}$, $s.Bins$ contains $span.Bins \setminus \{bin \in node.sons\,[u]\,.Bins : u \neq v\}$, and $s.population$ contains $\{t \in span.population : t[att] = v\}$
            Add s to $node.sons\,[v]\,.spans$
            Add s to every node $n \in span.Nodes \setminus node.sons$

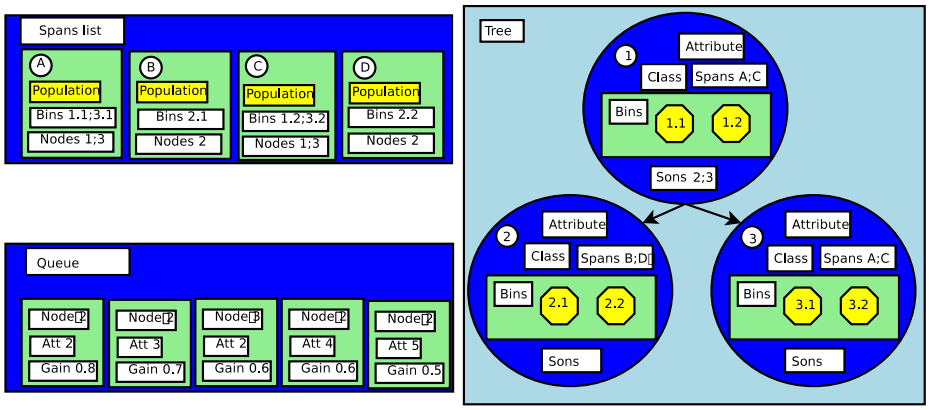

Fig. 2. kADET uses three data structures: The tree, a list of spans, and a priority queue for candidate splits.

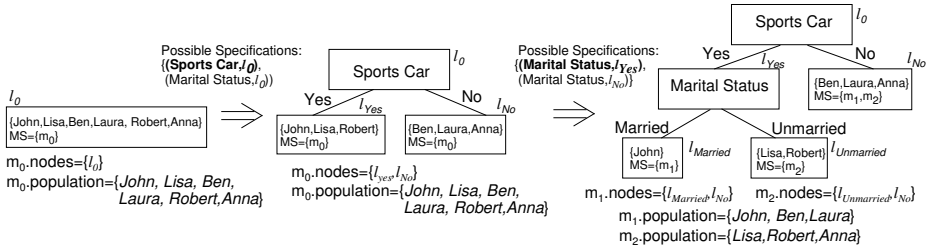

Fig. 3. Inducing a 3-anonymous decision tree

until the number of leaves is equal to the number of public attributes; then, every leaf is splitted according to a *different* public attribute. The number of model spans in the tree is equal to the size of A.

While this overhead is indeed high, the source of the overhead is not inefficiency of the algorithm. Rather it the need to validate that every model span is populated by more than k tuples. This same overhead would incur to any anonymization algorithm that would compute this same scheme. Therefore, it is inherent to the problem. In practice, the number of spans will be much smaller. For example, when only the class attribute is private, the number of spans is the number of leaves.

3.3 From ID3 to C4.5

C4.5 was introduced by Quinlan 3.1 in order to extend and improve ID3. It implements better attribute scoring metrics (gain ratio instead of gain), error-based

pruning, continuous attribute quantization, and treatment of missing values. All these extentions, beside the change of the scoring function – which has no effect of privacy – require careful analysis when likewise extending kADET.

Pruning C4.5 uses error-based pruning in two ways: Subtrees can be discarded if they increase the error on a test-set; and nodes, including their subtrees, can be placed under one of their branches. Using the first method is safe – undoing a split unifies equivalence classes, and may unify spans, meaning the population of a span can only increase. The second method, however, may cause a k-anonymous tree to become non-k-anonymous. This is because a span's population may shrink if a public attribute is added to a path in the tree. In our implementation we therefore avoided this technique.

Continuous attributes In the C4.5 algorithm, continuous attributes are handled by creating binary splits. The algorithm considers all the possible split points, and chooses the one with the best information gain. Implemented the same approach, adding the constraint that a split point should not create a k-anonymization breach.

Missing values Missing values extend the k-anonymity model in ways which do not yet have an agreed upon model. It is not clear, for instance, whether a value that is missing in the learning examples would be missing in the data available to the attacker. We leave the extension of the k-anonymity model to missing values for further research.

4 Evaluation

To conduct our experiments we implemented the algorithms using the Weka package [19]. We use as a benchmark the Adults database from the UC Irvine Machine Learning Repository [4], which contains census data, and has become a commonly used benchmark for k-anonymity. The data set has 6 continuous attributes and 8 categorial attributes. The class attribute is income level, with two possible values, $\leq 50K$ or $> 50K$. After records with missing values have been removed, there are 30,162 records for training and 15,060 records for testing. For the categorial attributes we use the same generalization hierarchies described in [6]. For the ID3 experiments we dropped the continuous attributes, because of ID3 limitations. The experiment was performed on a 3.0GHz Pentium IV processor with 512MB memory.

The anonymized decision trees algorithms use the training data to induce an anonymous decision tree. Then the test data (in a non-anonymized form) is classified using the anonymized tree. For all values of k the decision tree induction took less than 4 seconds for ID3, and less than 10 seconds for C4.5.

4.1 Accuracy vs. Anonymity Tradeoffs

Our first goal is to assess the tradeoff between classification accuracy and the privacy constraint.

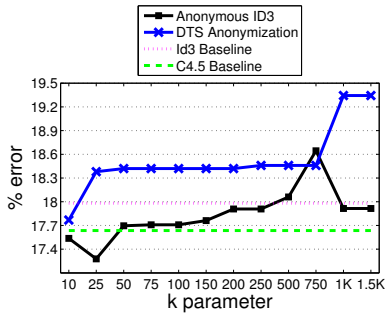

Fig. 4. Classification error vs. k parameter for ID3

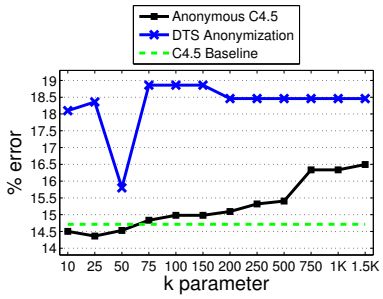

Fig. 5. Classification error vs. k parameter for C4.5

Figure 4 shows the classification error of the anonymous ID3 for various k parameters, compared to the classification error achieved with ID3 and C4.5. In spite of the anonymity constraint, the classifier maintains good accuracy. At $k = 750$ there is a local optimum when the root node is split using the *Relationship* attribute. At $k = 1000$ this attribute is discarded because of anonymity breach, and instead the *Marital Status* attribute is chosen, yielding better classification.

We compared our results with those obtained using the top-down specialization (TDS) algorithm presented in [6], which is aimed at producing anonymized data that is useful for classification problems. The algorithm starts with the top most generalization level, and iteratively chooses attributes to specialize based on a metric that measures the information gain for each unit of anonymity loss. The same generalization scheme is applied on all the tuples. We note that the TDS uses both training and test data to choose a generalization. This may provide different generalization results, though not necessarily better or worse than those obtained when generalizing the training data alone. TDS results also appear in Figure 4. In contrast to the TDS algorithm, our algorithm can apply different generalizations on different groups of tuples, and it achieves an average reduction of 0.6% in classification error with respect to TDS.

Figure 5 shows a similar comparison using all 14 attributes of the Adult dataset, based on the anonymous C4.5 algorithm. The large size of the quasi-identifier impacts the accuracy attained with the TDS generalization, and our algorithm achieves an average reduction of 3% in classification error with respect to TDS.

4.2 Privacy Risks and ℓ-Diversity

k-anonymity makes no restriction regarding private attribute values. As a consequence, it is possible that a k-anonymous model would allow the attacker a complete inference of these values. In this section, our goal is to assess how many individuals are prone to immediate inference attacks and show how such inference can be thwarted.

We look at the number of individuals (learning examples) for whom an attacker may infer the class attribute value with full certainty. This is the number of tuples associated with spans for which all the tuples share the same class. Figure 6 shows the percentage of tuples exposed to such inference, as a function of the parameter k. For the anonymous Id3, completely avoiding this kind of inference requires high values of k, and even in those cases the attacker may still be able to infer attribute values with high probability. The inference problem is less acute in the case of the anonymous C4.5, because of pruning. The number of exposed tuples drops to zero at $k = 75$, and is very low even for smaller values of k.

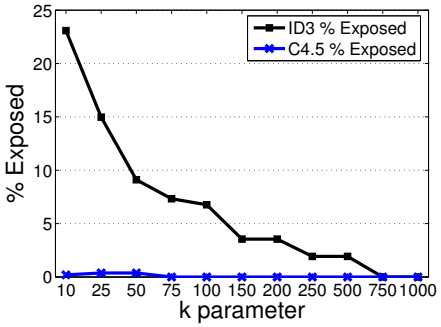

Fig. 6. % Exposed tuples

The ℓ-diversity model [12] suggests solving the inference problem by requiring a certain level of diversity in class values for every group of identifiable tuples. For example, *entropy ℓ-diversity* is maintained when the entropy of the class values for every such group exceeds a threshold value $log(\ell)$.

We altered our algorithms by replacing the $MaintainsAnonymity$ function with one that checks the entropy ℓ-diversity constraint, ruling out splits in the tree that violate this constraint. Note that the parameters for k-anonymity and ℓ-diversity are not comparable. In particular, as there are only two class values, the best we can hope for is entropy 2-diversity. This is achieved when there is equal chance for each class value, in which case we loss classification ability. However, for $\ell < 2$, entropy ℓ-diversity allows us to limit the attacker's confidence in inference attacks. For example, to deny the attacker the ability to infer a class value with confidence$> 85\%$, we should keep the entropy higher than $-0.85 \times \log 0.85 - 0.15 \times \log 0.15 = 0.61$. This amounts to applying entropy ℓ-diversity with $\ell = 1.526$ ($\log 1.526 = 0.61$).

Following this discussion, Figures 7 and 8 display the tradeoff between the confidence limit and the accuracy of the induced decision trees. So long as the confidence threshold is high enough, it is possible to induce decision trees without a significant accuracy penalty. The lowest achievable confidence level is 75.1%,

as it pertains to the class distribution in the root node. In the case of ID3, every split of the root node results in a node with confidence> 85%. Therefore, a confidence limit of 85% or lower prohibits the induction of a useful decision tree. The additional attributes available to the C4.5 algorithm, allow to stretch the boundary to a lower confidence threshold.

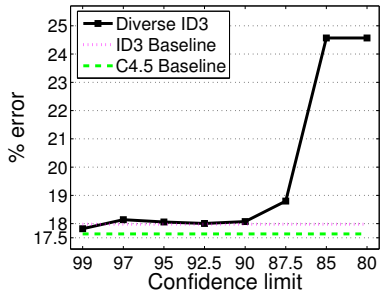

Fig. 7. Confidence level vs. error rate for ID3, 9 Attributes

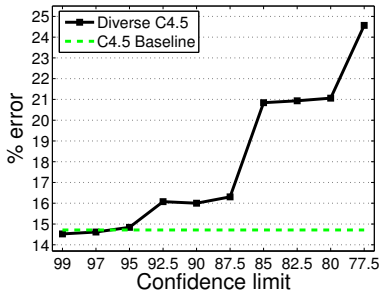

Fig. 8. Confidence level vs. error rate for C4.5, 15 Attributes

5 Conclusions

In this paper we showed how decision tree induction algorithms can be modified to guarantee that their output maintains k-anonymity. Using our definitions, it is possible to introduce similar constraints in other data mining algorithms as well. Another interesting use of this method is promoted by the ability to construct a table which is equivalent to a data mining model. Such a table would maintain k-anonymity, while preserving the data patterns evident in the data mining model. This suggests that data mining algorithms can be used as templates for pattern-preserving anonymization schemes.

References

1. Gagan Aggarwal, Tomás Feder, Krishnaram Kenthapadi, Rajeev Motwani, Rina Panigrahy, Dilys Thomas, and An Zhu. Approximation algorithms for k-anonymity. In *Journal of Privacy Technology (JOPT)*, 2005.
2. Rakesh Agrawal and Ramakrishnan Srikant. Privacy-preserving data mining. In *Proc. of the ACM SIGMOD Conference on Management of Data*, pages 439–450. ACM Press, May 2000.
3. Roberto J. Bayardo Jr. and Rakesh Agrawal. Data privacy through optimal k-anonymization. In *ICDE*, pages 217–228, 2005.
4. C.L. Blake D.J. Newman, S. Hettich and C.J. Merz. UCI repository of machine learning databases, 1998.

5. Wenliang Du and Zhijun Zhan. Building decision tree classifier on private data. In *Proc. of CRPITS'14*, pages 1–8, Darlinghurst, Australia, December 2002. Australian Computer Society, Inc.
6. Benjamin C. M. Fung, Ke Wang, and Philip S. Yu. Top-down specialization for information and privacy preservation. In *Proc. of ICDE'05*, Tokyo, Japan, April 2005.
7. Vijay S. Iyengar. Transforming data to satisfy privacy constraints. In *Proc. of ACM SIGKDD'02*, pages 279–288, 2002.
8. Murat Kantarcioğlu, Jiashun Jin, and Chris Clifton. When do data mining results violate privacy? In *Proc. of ACM SIGKDD'04*, pages 599–604, New York, NY, USA, 2004. ACM Press.
9. Kristen LeFevre, David J. DeWitt, and Raghu Ramakrishnan. Incognito: Efficient full-domain k-anonymity. In *Proc. of SIGMOD'05*, pages 49–60, New York, NY, USA, 2005. ACM Press.
10. Kristen LeFevre, David J. DeWitt, and Raghu Ramakrishnan. Mondrian multidimensional k-anonymity. In *Proc. of ICDE*, April 2006.
11. Yehuda Lindell and Benny Pinkas. Privacy preserving data mining. In *Proc. of CRYPTO'04*, pages 36–54. Springer-Verlag, 2000.
12. Ashwin Machanavajjhala, Johannes Gehrke, Daniel Kifer, and Muthuramakrishnan Venkitasubramaniam. ℓ-diversity: Privacy beyond k-anonymity. In *Proc. of ICDE*, April 2006.
13. Adam Meyerson and Ryan Williams. General k-anonymization is hard. In *Proc. of PODS'04*, 2003.
14. J. Ross Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.
15. J. Ross Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.
16. Latanya Sweeney. Achieving k-anonymity privacy protection using generalization and suppression. *International Journal on Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(5):571–588, 2002.
17. US Dept. of HHS. Standards for privacy of individually identifiable health information; final rule, August 2002.
18. Jaideep Vaidya and Chris Clifton. Privacy-preserving decision trees over vertically partitioned data. In *DBSec*, pages 139–152, 2005.
19. Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, San Francisco, 2nd edition, 2005.