Privacy Preserving Data Mining

Arie Friedman
Privacy Preserving Data Mining

Research Thesis

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy

Arie Friedman

Submitted to the Senate of
the Technion — Israel Institute of Technology
Shvat 5771 Haifa January 2011
The research thesis was done under the supervision of Prof. Assaf Schuster in the Computer Science Department.

I would like to thank my advisor, Prof. Assaf Schuster, for his professional assistance and remarkable insights. I would like to express my gratitude to Dr. Ran Wolff for helping me take the first steps in this research area and for his part in this work. And finally, special thanks go to my family for providing the moral support in this challenging period, and especially to my wife Tali for her patience throughout the long weekends I spent studying.

The generous financial support of the Technion is gratefully acknowledged.
# Contents

Abstract 1

Abbreviations and Notations 3

1 Introduction 4

2 Extending $k$-Anonymity to Data Mining 10
   2.1 Introduction . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11
   2.2 $k$-Anonymity of Tables . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13
      2.2.1 The $k$-Anonymity Model: Pros and Cons . . . . . . . . . . . . . . 15
   2.3 Extending $k$-Anonymity to Models . . . . . . . . . . . . . . . . . . . . . 16
   2.4 Examples . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21
      2.4.1 $k$-Anonymity of a Decision Tree . . . . . . . . . . . . . . . . . . . . . 21
      2.4.2 Clustering . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22
      2.4.3 $k$-Anonymity of Association Rules . . . . . . . . . . . . . . . . . . . 23
   2.5 $k$-Anonymity Preserving Data Mining Algorithms . . . . . . . . . . . . . . . 25
      2.5.1 Inducing $k$-Anonymized Decision Trees . . . . . . . . . . . . . . . . . 25
      2.5.2 Inducing $k$-Anonymized Clusters . . . . . . . . . . . . . . . . . . . . 28
   2.6 Related Work . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 29
   2.7 Conclusions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 32

3 $k$-Anonymous Decision Tree Induction 34
   3.1 Introduction . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 35
   3.2 Inducing $k$-Anonymous Decision Trees . . . . . . . . . . . . . . . . . . . . . 37
      3.2.1 kADET Algorithm . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 40
      3.2.2 Generalization in Decision Tree Induction . . . . . . . . . . . . . . . . 42
      3.2.3 Correctness and Overhead Analysis . . . . . . . . . . . . . . . . . . . . 44
      3.2.4 From ID3 to C4.5 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 47
      3.2.5 Handling Missing Values . . . . . . . . . . . . . . . . . . . . . . . . . . 47
   3.3 Evaluation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 50
3.3.1 Accuracy vs. Anonymity Tradeoffs .............................................. 50
3.3.2 Model-based $k$-Anonymization .................................................. 54
3.3.3 Missing Values ........................................................................ 55
3.4 Enhancements to $k$-Anonymity ....................................................... 56
3.5 Conclusions .................................................................................. 58

4 Data Mining with Differential Privacy ............................................... 60
4.1 Introduction ................................................................................. 61
4.2 Background .................................................................................. 62
  4.2.1 Differential Privacy ................................................................. 62
  4.2.2 Noisy Counts ......................................................................... 64
  4.2.3 The Exponential Mechanism .................................................... 64
  4.2.4 PINQ .................................................................................... 66
4.3 Approaches to Differentially Private Computation ............................ 67

5 Decision Tree Induction with Differential Privacy .............................. 70
5.1 Introduction ................................................................................. 71
5.2 Revisiting Decision Tree Induction with Sub-Linear Queries ............. 71
  5.2.1 Improving the SuLQ-based Algorithm with the Partition Operator . 77
5.3 Privacy Considerations in Decision Tree Induction ......................... 77
  5.3.1 Stopping Criteria .................................................................... 77
  5.3.2 Choosing an Attribute with the Exponential Mechanism ............. 79
  5.3.3 Continuous Attributes .............................................................. 84
  5.3.4 Pruning .................................................................................. 86
5.4 Experimental Evaluation ................................................................ 87
  5.4.1 Synthetic Datasets ................................................................... 87
  5.4.2 Real Dataset ........................................................................... 90

6 Summary and Discussion .................................................................. 93

Abstract in Hebrew

Technion - Computer Science Department - Ph.D. Thesis PHD-2011-01 - 2011
# List of Figures

1.1 Example of a Pareto frontier. Given a number of learning samples, what are the privacy and accuracy tradeoffs? ................................................. 7

2.1 Mortgage company decision tree ................................................. 13
2.2 A $k$-anonymous decision tree ..................................................... 23
2.3 Clustering model ........................................................................... 24
2.4 Inducing a 3-anonymous decision tree ............................................. 27
2.5 Inducing $k$-anonymous clusters ....................................................... 29

3.1 Linking attack on a decision tree ..................................................... 36
3.2 Execution of $kADET$ ..................................................................... 41
3.3 2-Anonymous Mortgage company decision tree with generalization ...... 42
3.4 Decision tree resulting from data with missing values ......................... 49
3.5 Classification error vs. $k$ parameter for ID3 ................................... 50
3.6 Classification error vs. $k$ parameter for C4.5 .................................. 53
3.7 Handling missing values, Adult data set ......................................... 56
3.8 Handling missing values, Credit Approval data set ............................ 56
3.9 Exposed individuals in Adult data set ............................................. 57
3.10 Confidence level vs. error rate for ID3, 9 attributes ........................... 58
3.11 Confidence level vs. error rate for C4.5, 15 attributes ....................... 58

4.1 Data access with differential privacy: (a) the data miner has direct access to data; (b) the data miner can access data through a data access mechanism (DP) that enforces differential privacy; (c) the data miner works on an offline synthetic dataset, which was generated by an algorithm (DP) that preserves differential privacy. ......................................................... 67

5.1 Evaluating a threshold-based stopping criterion with noisy counts and the exponential mechanism. ......................................................... 79
5.2 Splitting a single node with a binary attribute, $B = 0.1$, $p_{\text{noise}} = 0.1$ 88
5.3 Splitting a single node with a continuous attribute, $B = 0.1$, $p_{\text{noise}} = 0.1$ . . . 88
5.4 Comparing numerical split to discretization, $B = 0.1$, $p_{\text{noise}} = 0.1$ . . . . . . . 89
5.5 Inducing a tree of depth 5 with 10 binary attributes and a binary class, $B = 1.0$, $p_{\text{leaf}} = 0.3$, $p_{\text{noise}} = 0.1$ . . . . . . . . . . . . . . . . . . . . . . . 90
5.6 Inducing a tree of depth 5 with 7 binary attributes, 3 numeric attributes and a binary class, $B = 1.0$, $p_{\text{leaf}} = 0.3$, $p_{\text{noise}} = 0.1$ . . . . . . . . . . . . . . . . . . . . . . . 90
5.7 Accuracy vs. privacy ($B$) in the Adult dataset . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 91
List of Tables

2.1 Mortgage company data ................................................. 12
2.2 Anonymized mortgage company data ................................. 12
2.3 Table anonymization .................................................. 15
2.4 Mortgage company data ................................................. 22
2.5 Individuals’ data ....................................................... 24

3.1 Mortgage company data ................................................. 42
3.2 Mortgage company data – generalization example .................. 43
3.3 Data with missing values .............................................. 49
3.4 Classification metric comparison .................................... 54

5.1 Applying the exponential mechanism to choose a split point for a continuous attribute with Max scorer, $\epsilon = 1.$ ........................................ 85
5.2 Accuracy and standard deviation of single split with binary attribute and binary class, $B = 0.1$ ........................................ 89
Abstract

In recent years, privacy preserving data mining (PPDM) has emerged as a very active research area. This field of research studies how knowledge or patterns can be extracted from large data stores while maintaining commercial or legislative privacy constraints. Quite often, these constraints pertain to individuals represented in the data stores. While data collectors strive to derive new insights that would allow them to improve customer service and increase their sales by better understanding customer needs, consumers are concerned about the vast quantities of information collected about them and how this information is put to use. Privacy preserving data mining aims to settle these conflicting interests. The question how these two contrasting goals, mining new knowledge while protecting individuals’ privacy, can be reconciled, is the focus of this research. We seek ways to improve the tradeoff between privacy and utility when mining data.

In this work we address the privacy/utility tradeoff problem by considering the privacy and algorithmic requirements simultaneously. We take data mining algorithms, and investigate how privacy considerations may influence the way the data miner accesses the data and processes them. We study this problem in the context of two privacy models that attracted a lot of attention in recent years, $k$-anonymity and differential privacy. Our analysis and experimental evaluations confirm that algorithmic decisions made with privacy considerations in mind may have a profound impact on the accuracy of the resulting data mining models. In chapter 2 we study this tradeoff in the context of $k$-anonymity, which was the starting point of this research. Since $k$-anonymity was originally defined in the context of relational tables, we begin with extending the definition of $k$-anonymity with definitions of our own, which can then be used to prove that a given data mining model is $k$-anonymous. We exemplify how our definitions can be used to validate the $k$-anonymity of several data mining algorithms, and demonstrate how the definitions can be incorporated within a data mining algorithm to guarantee $k$-anonymous output. In chapter 3 we focus on decision tree induction and demonstrate that our technique can be used to induce decision trees which are more accurate than those acquired by anonymizing the data first and inducing the decision tree later. This way, anonymization is done in a manner that interferes as little as possible with the tree induction process. In chapters 4 and 5 we study the privacy/accuracy tradeoff in the context of differential privacy. We show that when applying differential privacy, the
method chosen by the data miner to build a classifier could make the difference between an accurate classifier and a useless one, even when the same choice without privacy constraints would have no such effect. Moreover, an improved algorithm can achieve the same level of accuracy and privacy as the naive implementation but with an order of magnitude fewer learning samples.
Abbreviations and Notations

Notations for chapters 2 and 3

\[ D = A \times B \] — Data domain comprised of public attributes (A) and private attributes (B)
\[ T \subseteq D \] — A private database
\[ x_A, T_A \] — Projection of a tuple \( x \in D \) or a database \( T \subseteq D \) respectively into the public attributes \( A \)
\( M \) — A data mining model: a function from \( D \) to an arbitrary output domain
\( [x] \) — The equivalence class induced by \( M \) that contains the tuple \( x \)
\( b_i \) — A bin, indicates a target value for a decision tree model
\( p_T \) — A population function that assigns to each equivalence class in \( M \) the number of tuples from \( T \) that belong to it
\( (M, p_T) \) — A release
\( T_{ID} \) — A public identifiable database, in which every tuple in \( T_A \) is associated with the identity of the corresponding individual
\( S_M(a) \) — The span of a tuple \( a \in A \), comprised of all the equivalence classes induced by \( M \) that contain tuples whose projection on \( A \) is \( a \). \( M \) is omitted where understood from context
\( T_{ID}|S_M(a) \) — Population of a span — the set of tuples from \( T_{ID} \) whose span is \( S_M(a) \)

Notations for chapters 4 and 5

\( \mathcal{T} \) — A set of instances
\( \tau \) (\( N_T \)) — The (estimated) number of instances in a set \( \mathcal{T} \)
\( \tau_c \) (\( N_c \)) — The (estimated) number of instances in \( \mathcal{T} \) with class \( c \in C \)
\( \tau_j^A \) (\( N_j^A \)) — The (estimated) number of instances in \( \mathcal{T} \) that take the value \( j \) on attribute \( A \)
\( \tau_{jc}^A \) (\( N_{j,c}^A \)) — The (estimated) number of instances in \( \mathcal{T} \) that take the value \( j \) on attribute \( A \) and with class \( c \in C \)
\( t \) — The maximal number of values that an attribute can take
\( |C| \) — The number of class values
Chapter 1

Introduction
“You have zero privacy anyway. Get over it!”

Scott McNeally (SUN CEO, 1999)

“If you have something that you don’t want anyone to know, maybe you shouldn’t be doing it in the first place.”

Eric Schmidt (Google CEO, 2009)

“People have really gotten comfortable not only sharing more information and different kinds, but more openly and with more people. That social norm is just something that has evolved over time.”

Mark Zuckerberg (Facebook CEO, 2010)

“No matter how many times a privileged straight white male technology executive proclaims the death of privacy, Privacy Is Not Dead.”

danah boyd (Social Media Researcher, Microsoft, 2010)
The proliferation of information technologies and the internet in the past two decades has brought a wealth of individual information into the hands of commercial companies and government agencies. As hardware costs go down, organizations find it easier than ever to keep any piece of information acquired from the ongoing activities of their clients. Data owners constantly seek to make better use of the data they possess, and utilize data mining tools to extract useful knowledge and patterns from the data. In result, there is a growing concern about the ability of data owners, such as large corporations and government agencies, to abuse this knowledge and compromise the privacy of their clients – concern which has been reflected in the actions of legislative bodies (e.g., the debate about and subsequent elimination of the Total Information Awareness project in the US [16]). This concern is exacerbated by actual incidents that demonstrate how difficult it is to use and share information while protecting individuals’ privacy. One example is from August 2006 [10], when AOL published on their website a data set of 20 million web searches for research purposes. Although the data set was believed to be anonymized, New York Times journalists have shown how the released information can be used to expose the identities of the searchers and learn quite a lot about them. Another example relates to the Netflix prize contest¹ that took place between October 2006 and September 2009. Netflix has published a dataset consisting of more than 100 million movie ratings from over 480 thousands of its customers, and invited the research community to contend for improvements to its recommendation algorithm. To protect customer privacy, Netflix removed all personal information identifying individual customers and perturbed some of the movie ratings. Despite these precautions, researchers have shown that with relatively little auxiliary information anonymous customers can be re-identified [64].

Given the rising privacy concerns, the data mining community has faced a new challenge [3]. Having shown how effective its tools are in revealing the knowledge locked within huge databases, it is now required to develop methods that restrain the power of these tools to protect the privacy of individuals. The question how these two contrasting goals, mining new knowledge while protecting individuals’ privacy, can be reconciled, is the focus of this research. We seek ways to improve the tradeoff between privacy and utility when mining data.

To illustrate this problem, we present it in terms of Pareto efficiency [72]. Consider three objective functions: the accuracy of the data mining model (e.g., the expected accuracy of a resulting classifier, estimated by its performance on test samples), the size of the mined database (number of training samples), and the privacy requirement, represented by a privacy parameter. In a given situation, one or more of these factors may be fixed: a client may present a lower acceptance bound for the accuracy of a classifier, the database may contain a limited number of samples, or a regulator may pose privacy restrictions. Within the given

¹www.netflixprize.com
Figure 1.1: Example of a Pareto frontier. Given a number of learning samples, what are the privacy and accuracy tradeoffs?

constraints, we wish to improve the objective functions: achieve better accuracy with fewer learning examples and better privacy guarantees. However, these objective functions are often in conflict. For example, applying stronger privacy guarantees could reduce accuracy or require a larger dataset to maintain the same level of accuracy. Instead, we should settle for some tradeoff. With this perception in mind, we can evaluate the performance of data mining algorithms. Consider, for example, three hypothetical algorithms that produce a classifier. Assume that their performance was evaluated on datasets with 50,000 records, with the results illustrated in Figure 1.1. We can see that when the privacy settings are high, algorithm 1 obtains on average a lower error rate than the other algorithms, while algorithm 2 does better when the privacy settings are low. A Pareto improvement is a change that improves one of the objective functions without harming the others. Algorithm 3 is dominated by the other algorithms: for any setting, we can make a Pareto improvement by switching to one of the other algorithms. A given situation (a point in the graph) is Pareto efficient when no further Pareto improvements can be made. The Pareto frontier is given by all the Pareto efficient points. Our goal is to investigate algorithms that can further extend the Pareto frontier, allowing for better privacy and accuracy tradeoffs.

There are well established methods to measure the accuracy of a resulting data mining model. For example, a classifier can be measured by its performance on a test dataset, or using methods such as 10-fold cross validation. However, finding a good measure of privacy in data mining seems to be a difficult problem, and several approaches to this problem were proposed throughout the last decade.

One approach toward privacy protection in data mining was to perturb the input (the data) before it is mined [6]. Thus, it was claimed, the original data would remain secret, while the added noise would average out in the output. This approach has the benefit of simplicity. At the same time, it takes advantage of the statistical nature of data mining and directly protects the privacy of the data. The drawback of the perturbation approach
is that it lacks a formal framework for proving how much privacy is guaranteed. This lack has been exacerbated by some recent evidence that for some data, and some kinds of noise, perturbation provides no privacy at all [42, 48]. Recent models for studying the privacy attainable through perturbation [13, 18, 21, 29, 32] offer solutions to this problem in the context of statistical databases.

At the same time, a second branch of privacy preserving data mining was developed, using cryptographic techniques to prevent information leakage during the computation of the data mining model. This branch became hugely popular [24, 40, 46, 55, 78, 91, 79] for two main reasons: First, cryptography offers a well-defined model for privacy, which includes methodologies for proving and quantifying it. Second, there exists a vast toolset of cryptographic algorithms and constructs for implementing privacy-preserving data mining algorithms. However, recent work (e.g. [47, 24]) has pointed that cryptography does not protect the output of a computation. Instead, it prevents privacy leaks in the process of computation. Thus, it falls short of providing a complete answer to the problem of privacy preserving data mining.

$k$-anonymity [71, 74] – a definition for privacy that was conceived in the context of relational databases – has received a lot of attention in the past decade. Roughly speaking, $k$-anonymity provides a “blend into 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 contain data which is not available in external tables and needs to be protected. The guarantee provided by $k$-anonymity is that an attacker will 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 in the release appears in at least $k$ rows. The $k$-anonymity model of privacy was studied intensively in the context of public data releases [5, 11, 12, 37, 44, 51, 61, 71, 73, 74], when the database owner wishes to ensure that no one will be able to link information gleaned from the database to individuals from whom the data has been collected. This method was also leveraged to provide anonymity in other contexts, such as anonymous message transmission [83] and location privacy [39]. In recent years, the assumptions underlying the $k$-anonymity model have been challenged [65, 38], and the AOL and Netflix data breaches demonstrated the difficulties in ensuring anonymity.

A more recent privacy model is differential privacy [25, 27]. Basically, differential privacy requires that computations be insensitive to changes in any particular individual’s record. Once an individual is certain that his or her data will remain private, being opted in or out of the database should make little difference. For the data miner, however, all these individual records in aggregate are very valuable. Differential privacy has several advantages over prior approaches to privacy: first, it relies on a mathematical definition, making it possible to rigorously prove whether a mechanism conforms to differential privacy, and to deduce which
calculations can or cannot be made in this framework. Second, differential privacy does not make any assumptions on an adversary’s background knowledge or computational power. This independence frees data providers who share data from concerns about past or future data releases and is adequate given the abundance of personal information shared on social networks and public Web sites. Third, differential privacy maintains composability [38], meaning that differential privacy guarantees hold also when two independent differentially private data releases are combined by an adversary. This property frees the data provider from concerns about external or future data releases and their privacy implications on a current data release.

In this work we address the privacy/accuracy tradeoff problem by considering the privacy and algorithmic requirements simultaneously. We consider algorithmic data mining processes, and investigate how privacy considerations may influence the way the data miner accesses the data and processes them. Our analysis and experimental evaluations confirm that algorithmic decisions made with privacy considerations in mind may have a profound impact on the accuracy of the resulting data mining models. In chapter 2 we study this tradeoff in the context of $k$-anonymity, which was the starting point of this research. Since $k$-anonymity was originally defined in the context of relational tables, we begin with extending the definition of $k$-anonymity with definitions of our own [36], which can then be used to prove that a given data mining model is $k$-anonymous. We exemplify how our definitions can be used to validate the $k$-anonymity of several data mining algorithms, and demonstrate how the definitions can be incorporated within a data mining algorithm to guarantee $k$-anonymous output. In chapter 3 we focus on decision tree induction and demonstrate that our technique can be used to induce decision trees which are more accurate than those acquired by anonymizing the data first and inducing the decision tree later [35]. This way, anonymization is done in a manner that interferes as little as possible with the tree induction process. In chapters 4 and 5 we study the privacy/accuracy tradeoff in the context of differential privacy [34]. We show that when applying differential privacy, the method chosen by the data miner to build a classifier could make the difference between an accurate classifier and a useless one, even when the same choice without privacy constraints would have no such effect. Moreover, an improved algorithm can achieve the same level of accuracy and privacy as the naive implementation but with an order of magnitude fewer learning samples.
Chapter 2

Extending $k$-Anonymity to Data Mining
2.1 Introduction

One definition of privacy which has received a lot of attention in the past decade is that of $k$-anonymity [74, 70]. The guarantee given by $k$-anonymity is that no information can be linked to groups of less than $k$ individuals. The $k$-anonymity model of privacy was studied intensively in the context of public data releases [5, 11, 12, 37, 44, 51, 61, 71, 73], when the database owner wishes to ensure that no one will be able to link information gleaned from the database to individuals from whom the data has been collected. In the next section we provide, for completeness, the basic concepts of this approach.

We focus on the problem of guaranteeing privacy of data mining output. To be of any practical value, the definition of privacy must satisfy the needs of users of a reasonable application. Two examples of such applications are (1) a credit giver, whose clientele consists of numerous shops and small businesses, and who wants to provide them with a classifier that will distinguish credit-worthy from credit-risky clients, and (2) a medical company that wishes to publish a study identifying clusters of patients who respond differently to a course of treatment. These data owners wish to release data mining output, but still be assured that they are not giving away the identity of their clients. If it could be verified that the released output withstands limitations similar to those set by $k$-anonymity, then the credit giver could release a $k$-anonymous classifier and reliably claim that the privacy of individuals is protected. Likewise, the authors of a medical study quoting $k$-anonymous cluster centroids could be sure that they comply with HIPAA privacy standards [76], which forbid the release of individually identifiable health information.

One way to guarantee $k$-anonymity of a data mining model is to build it from a $k$-anonymized table. However, this poses two main problems: First, the performance cost of the anonymization process may be very high, especially for large and sparse databases. In fact, the cost of anonymization can exceed the cost of mining the data. Second, the process of anonymization may inadvertently delete features that are critical for the success of data mining and leave out those that are useless; thus, it would make more sense to perform data mining first and anonymization later.

To demonstrate the second problem, consider the data in Table 2.1, which describes loan risk information of a mortgage company. The $Gender$, $Married$, $Age$ and $Sports Car$ attributes contain data that is available to the public, while the $Loan Risk$ attribute contains data that is known only to the company. To get a 2-anonymous version of this table, many practical methods call for the suppression or generalization of whole columns. This approach was termed single-dimension recoding [50]. In the case of Table 2.1, the data owner would have to choose between suppressing the $Gender$ column and suppressing all the other columns.

The methods we describe in this chapter would lead to full suppression of the $Sports Car$ column as well as a partial suppression of the $Age$ and $Married$ columns. This would result
in Table 2.2. This kind of generalization was termed *multi-dimensional recoding* [50]. While more data is suppressed, the accuracy of the decision tree learned from this table (Figure 2.1) is better than that of the decision tree learned from the table without the *Gender* column. Specifically, without the *Gender* column, it is impossible to obtain a classification better than 50% *good* loan risk, 50% *bad* loan risk, for any set of tuples.

<table>
<thead>
<tr>
<th>Gender</th>
<th>Married</th>
<th>Age</th>
<th>Sports Car</th>
<th>Loan Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Male</td>
<td>*</td>
<td>Young</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>Male</td>
<td>*</td>
<td>Young</td>
<td>*</td>
<td>good</td>
</tr>
<tr>
<td>Male</td>
<td>*</td>
<td>Young</td>
<td>*</td>
<td>good</td>
</tr>
<tr>
<td>Male</td>
<td>*</td>
<td>Old</td>
<td>*</td>
<td>good</td>
</tr>
<tr>
<td>Male</td>
<td>*</td>
<td>Old</td>
<td>*</td>
<td>bad</td>
</tr>
<tr>
<td>Male</td>
<td>*</td>
<td>Old</td>
<td>*</td>
<td>bad</td>
</tr>
<tr>
<td>Female</td>
<td>No</td>
<td>*</td>
<td>*</td>
<td>good</td>
</tr>
<tr>
<td>Female</td>
<td>No</td>
<td>*</td>
<td>*</td>
<td>good</td>
</tr>
<tr>
<td>Female</td>
<td>No</td>
<td>*</td>
<td>*</td>
<td>bad</td>
</tr>
<tr>
<td>Female</td>
<td>Yes</td>
<td>*</td>
<td>*</td>
<td>bad</td>
</tr>
<tr>
<td>Female</td>
<td>Yes</td>
<td>*</td>
<td>*</td>
<td>bad</td>
</tr>
<tr>
<td>Female</td>
<td>Yes</td>
<td>*</td>
<td>*</td>
<td>bad</td>
</tr>
</tbody>
</table>

Table 2.2: Anonymized mortgage company data

In this chapter we extend the definition of $k$-anonymity with definitions of our own, which can then be used to prove that a given data mining model is $k$-anonymous. The key for these extended definitions is in identifying how external data can be used to perform a
linking attack on a released model. We exemplify how our definitions can be used to validate the \( k \)-anonymity of classification, clustering, and association rule models, and demonstrate how the definitions can be incorporated within a data mining algorithm to guarantee \( k \)-anonymous output. This method ensures the \( k \)-anonymity of the results while avoiding the problems detailed above.

This chapter is organized as follows: In Section 2.2 we reiterate and discuss Sweeney’s and Samarati’s formal definition of \( k \)-anonymity. We then proceed in Section 2.3 to extend their definition with our definitions for \( k \)-anonymous data mining models. In Section 2.4 we exemplify the use of these definitions, and we present two \( k \)-anonymous data mining algorithms in Section 2.5. Section 2.6 discusses related work. We present our conclusions in Section 2.7.

### 2.2 \( k \)-Anonymity of Tables

The \( k \)-anonymity model was first described by Sweeney and Samarati [71], and later expanded by Sweeney [74] and Samarati [70] in the context of data table releases. In this section we reiterate their definition and then proceed to analyze the merits and shortcomings of \( k \)-anonymity as a privacy model.

The \( k \)-anonymity model distinguishes three entities: individuals, whose privacy needs to be protected; the database owner, who controls a table in which each row (also referred to as record or tuple) describes exactly one individual; and the attacker. The \( k \)-anonymity model makes two major assumptions:

1. The database owner is able to separate the columns of the table into a set of quasi-identifiers, which are attributes that may appear in external tables the database owner does not control, and a set of private columns, the values of which need to be protected. We prefer to term these two sets as public attributes and private attributes, respectively.

2. The attacker has full knowledge of the public attribute values of individuals, and no
knowledge of their private data. The attacker only performs linking attacks. A linking attack is executed by taking external tables containing the identities of individuals, and some or all of the public attributes. When the public attributes of an individual match the public attributes that appear in a row of a table released by the database owner, then we say that the individual is linked to that row. Specifically the individual is linked to the private attribute values that appear in that row. A linking attack will succeed if the attacker is able to match the identity of an individual against the value of a private attribute.

As accepted in other privacy models (e.g., cryptography), it is assumed that the domain of the data (the attributes and the ranges of their values) and the algorithms used for anonymization are known to the attacker. Ignoring this assumption amounts to “security by obscurity,” which would considerably weaken the model. The assumption reflects the fact that knowledge about the nature of the domain is usually public and in any case of a different nature than specific knowledge about individuals. For instance, knowing that every person has a height between zero and three meters is different than knowing the height of a given individual.

Under the $k$-anonymity model, the database owner retains the $k$-anonymity of individuals if none of them can be linked with fewer than $k$ rows in a released table. This is achieved by making certain that in any table released by the owner there are at least $k$ rows with the same combination of values in the public attributes. In many cases, the tables that the data owners wish to publish do not adhere to the $k$-anonymity constraints. Therefore, the data owners are compelled to alter the tables to conform to $k$-anonymity. Two main methods are used to this end: generalization and suppression. In generalization, a public attribute value is replaced with a less specific but semantically consistent value. In suppression, a value is not released at all. However, it was shown that anonymizing tables such that their contents are minimally distorted is an NP-hard problem [62]. Thus, most work in the area [51, 12, 5, 11, 37, 85, 44, 73, 71] concerns heuristic efficient anonymization of tables, with specific care 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 [11, 37, 44]. Data mining is envisioned as one of the main target applications of released data.

Table 2.3 illustrates how $k$-anonymization hinders linking attacks. The joining of the original Table 2.3.A with the public census data in 2.3.C would reveal that Laura’s income is High and Ben’s is Middle. However, if the original table is 2-anonymized to that in 2.3.B, then the outcome of joining it with the census data is ambiguous.

It should be noted that the $k$-anonymity model is slightly broader than what is described here [74], especially with regard to subsequent releases of data. We chose to provide the minimal set of definitions required to extend $k$-anonymity in the next section.
### Table 2.3: Table anonymization

<table>
<thead>
<tr>
<th>Zipcode</th>
<th>Income</th>
<th>Zipcode</th>
<th>Income</th>
<th>Zipcode</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>11001</td>
<td>High</td>
<td>110XX</td>
<td>High</td>
<td>11001</td>
<td>John</td>
</tr>
<tr>
<td>11001</td>
<td>Low</td>
<td>110XX</td>
<td>Low</td>
<td>11001</td>
<td>Lisa</td>
</tr>
<tr>
<td>12033</td>
<td>Mid</td>
<td>120XX</td>
<td>Mid</td>
<td>12033</td>
<td>Ben</td>
</tr>
<tr>
<td>12045</td>
<td>High</td>
<td>120XX</td>
<td>High</td>
<td>12045</td>
<td>Laura</td>
</tr>
</tbody>
</table>

A. Original  
B. 2-Anonymized  
C. Public

### 2.2.1 The \( k \)-Anonymity Model: Pros and Cons

The limitations of the \( k \)-anonymity model stem from the two assumptions above. First, it may be very hard for the owner of a database to determine which of the attributes are or are not available in external tables. This limitation can be overcome by adopting a strict approach that assumes much of the data is public. The second limitation is much harsher. The \( k \)-anonymity model assumes a certain method of attack, while in real scenarios there is no reason why the attacker should not try other methods, such as injecting false rows (which refer to no real individuals) into the database. Of course, it can be claimed that other accepted models pose similar limitations. For instance, the well-accepted model of semi-honest attackers in cryptography also restricts the actions of the attacker.

A third limitation of the \( k \)-anonymity model published in the literature [56] is its implicit assumption that tuples with similar public attribute values will have different private attribute values. Even if the attacker knows the set of private attribute values that match a set of \( k \) individuals, the assumption remains that he does not know which value matches any individual in particular. However, it may well happen that, since there is no explicit restriction forbidding it, the value of a private attribute will be the same for an identifiable group of \( k \) individuals. In that case, the \( k \)-anonymity model would permit the attacker to discover the value of an individual’s private attribute. The ability to expose sensitive information by analyzing the private attribute values linked to each group of individuals has motivated several works [56, 54, 88, 19] that proposed modifications of the privacy definition.

Despite these limitations, \( k \)-anonymity has gained a lot of traction in the research community, and it still provides the theoretical basis for privacy related legislation [76]. This is for several important reasons: (1) The \( k \)-anonymity model defines the privacy of the output of a process and not of the process itself. This is in sharp contrast to the vast majority of privacy models that were suggested earlier, and it is in this sense of privacy that clients are usually interested. (2) It is a simple, intuitive, and well-understood model. Thus, it appeals to the non-expert who is the end client of the model. (3) Although the process of computing a \( k \)-anonymous table may be quite hard [5, 61], it is easy to validate that an outcome is
indeed $k$-anonymous. Hence, non-expert data owners are easily assured that they are using the model properly.

In recent years, the assumptions regarding separation of quasi-identifiers and mode of attack have been challenged [65, 38], and two famous cases, the AOL data breach [10] and the Netflix data breach [64] have demonstrated the difficulties in ensuring anonymity. In Chapters 4 and 5 we consider an alternative privacy definition that circumvents these assumptions.

### 2.3 Extending $k$-Anonymity to Models

We start by extending the definition of $k$-anonymity beyond the release of tables. Our definitions are accompanied by a simple example to facilitate comprehension.

Consider a mortgage company that uses a table of past borrowers’ data to build a decision tree classifier predicting whether a client would default on the loan. Wishing to attract good clients and deter bad ones, the company includes the classifier on its Web page and allows potential clients to evaluate their chances of getting a mortgage. However, it would be unacceptable if somebody could use the decision tree to find out which past clients failed to return their loans. The company assumes that all the borrowers’ attributes (age, marital status, ownership of a sports car, etc.) are available to an attacker, except for the Loan Risk attribute (good/bad loan risk), which is private.

Figure 2.1 describes a toy example of the company’s decision tree, as induced from a set of learning examples, given in Table 2.1, pertaining to 12 past clients. We now describe a table which is equivalent to this decision tree in the sense that the table is built from the tree and the tree can be reconstructed from the table. The equivalent table (Table 2.2) has a column for each attribute that is used in the decision tree and a row for each learning example. Whenever the tree does not specify a value (e.g., Marital Status for male clients), the value assigned to the row will be *.

The motivation for the definitions which follow is that if the equivalent table is $k$-anonymous, the decision tree should be considered to be “$k$-anonymous” as well. The rationale is that, because the decision tree in Figure 2.1 can be reconstructed from Table 2.2, it contains no further information. Thus, if a linking attack on the table fails, any similar attack on the decision tree would have to fail as well. This idea that a data mining model and a $k$-anonymous table are equivalent allows us to define $k$-anonymity in the context of a broad range of models. We begin our discussion by defining a private database and then defining a model of that database.

**Definition 1 (A Private Database)** A private database $T$ 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 $A = A_1 \times \cdots \times A_k$ the public subdomain of $D$. For every tuple $x \in D$, the projection of $x$ into $A$, denoted $x_A$, is the tuple in $A$ that has the same assignment to each public attribute as $x$. The projection of a table $T$ into $A$ 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., $\forall x, y \in D, x \equiv y \iff M(x) = M(y)$ . The model partitions $D$ into respective equivalence classes such that $[x] = \{y \in D : y \equiv x\}$.

In the mortgage company decision tree example, the decision tree is a function that assigns bins to tuples in $T$. Accordingly, every bin within every leaf constitutes an equivalence class. Two tuples 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. For example, although the tuples of Anthony and Brian do not share the same value for the Sports Car attribute, they both belong to the good loan risk bin of leaf 1. This is because the tree does not differentiate tuples according to the Sports Car attribute. On the other hand, while the tuples of David and Edward will both be routed to leaf 2, they belong to different bins because their loan risk classifications are different.

The model alone imposes some structure on the domain. However, when a data owner releases a model based on a database, it also provides information about how the model relates to the database. For instance, a decision tree model or a set of association rules may include the number of learning examples associated with each leaf, or the support of each rule, respectively. As we shall see, a linking attack can be carried out using the partitioning of the domain, together with the released populations of different regions.

**Definition 3 (A Release)** Given a database $T$ and a model $M$, a release $M_T$ is the pair $(M, p_T)$, where $p_T$ (for population) is a function that assigns to each equivalence class induced by $M$ the number of tuples from $T$ that belong to it, i.e., $p_T([x]) = |T \cap [x]|$.

Note that other definitions of a release, in which the kind of information provided by $p_T$ is different, are possible as well. For example, a decision tree may provide the relative frequency of a bin within a leaf, or just denote the bin that constitutes the majority class. In this work we assume the worst case, in which the exact number of learning examples in each bin is provided. The effect of different kinds of release functions on the extent of private data that can be inferred by an attacker is an open question. Nevertheless, the anonymity analysis provided herein can be applied in the same manner for all of them. In other words, different definitions of $p_T$ would reveal different private information on the same groups of tuples.

As described, the released model partitions the domain according to the values of public and private attributes. This is reasonable because the users of the model are intended to
be the database owner or the client, both of whom supposedly know the private attributes’ values. We now turn to see how the database and the release are perceived by an attacker.

**Definition 4 (A Public Identifiable Database)** A public identifiable database $T_{ID} = \{(id_x, x_A) : x \in T\}$ is a projection of a private database $T$ into 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 $T$ pertained.

Although the attacker knows only the values of public attributes, he can nevertheless try to use the release $M_T$ to expose private information of individuals represented in $T_{ID}$. Given a tuple $(id_x, x_A) \in T_{ID}$ and a release, the attacker can distinguish the equivalence classes to which the original tuple $x$ may belong. We call this set of equivalence classes the span of $x_A$.

**Definition 5 (A 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, $S_M(a) = \{[x] : x \in D \land x_A = a\}$. When $M$ is evident from the context, we will use the notation $S(a)$.

In the aforementioned mortgage company’s decision tree model, every leaf constitutes a span, because tuples can be routed to different bins within a leaf by changing their private Loan Risk attribute, but cannot be routed to other leaves unless the value of a public attribute is changed. For example, an attacker can use the public attributes Gender and Married to conclude that the tuples of Barbara and Carol both belong to leaf 4. However, although these tuples have different values for the public attributes Age and Sports Car, the attacker cannot use this knowledge to determine which tuple belongs to which bin. These tuples are indistinguishable from the attacker’s point of view, with respect to the model: both share the same span formed by leaf 4.

We will now consider the connection between the number of equivalence classes in a span and the private information that can be inferred from the span.

**Claim 1** If $S(a)$ contains more than one equivalence class, then for every two equivalence classes in the span, $[x]$ and $[y]$, there is at least one combination of attribute values that appears in $[x]$ and does not appear in $[y]$.

**Proof.** By definition, for every equivalence class $[x] \in S(a)$, there exists $x \in [x]$ such that $x_A = a$. Let $[x]$ and $[y]$ be two equivalence classes in $S(a)$, and let $x \in [x]$, $y \in [y]$ be two tuples such that $x_A = y_A = a$. Since $x$ and $y$ have the same public attribute values, the only way to distinguish between them is by their private attribute values. The equivalence classes $[x]$ and $[y]$ are disjoint; hence, the combination of public and private attribute values
Claim 2 If $S(a)$ contains exactly one equivalence class, then no combination of private attributes can be eliminated for any tuple that has the same span.

Proof. Let $x_A \in A$ be a tuple such that $S(x_A) = S(a)$. Let $y, z \in D$ be two tuples such that $y_A = z_A = x_A$. Regardless of the private attribute values of $y$ and $z$, it holds that $[y] = [z]$. Otherwise, $S(x_A)$ would contain more than a single equivalence class, in contradiction to the assumption. Therefore, $y$ and $z$ both represent equally possible combinations of private attribute values for the tuple $x_A$, regardless of the population function $p_T$.

Corollary 1 A release exposes private information on the population of a span if and only if the span contains more than one equivalence class.

We will now see exactly how a release can be exploited to infer private knowledge about individuals. Given a public identifiable database $T_{ID}$ and a model $M$, we use $S(a)_{T_{ID}} = \{(id_x, x_A) \in T_{ID} : S(x_A) = S(a)\}$ to denote the set of tuples that appear in $T_{ID}$ and whose span is $S(a)$. These are tuples from $T_{ID}$ which are indistinguishable with respect to the model $M$ – each of them is associated with the same set of equivalence classes. Knowing the values of $p_T$ for each equivalence class in $S(a)$ would allow an attacker to constrain the possible private attribute value combinations for the tuples in $S(a)_{T_{ID}}$. For example, in the mortgage company’s decision tree, the span represented by leaf 4 [Female, Unmarried] contains two equivalence classes, which differ on the private attribute Loan Risk. Tuples that belong to the good equivalence class cannot have the private attribute bad Loan Risk, and vice versa. Given tuples that belong to a span with more than one equivalence class, the populations of each can be used to constrain the possible private attribute value combinations, hence compromising the privacy of the individuals.

On the basis of this discussion we define a linking attack as follows:

Definition 6 (Linking attack using a model) A linking attack on the privacy of tuples in a table $T$ from domain $A \times B$, using a release $M_T$, is carried out by

1. Taking a public identifiable database $T_{ID}$ which contains the identities of individuals and their public attributes $A$.

2. Computing the span for each tuple in $T_{ID}$.
3. Grouping together all the tuples in $T_{ID}$ that have the same span. This results in sets of tuples, where each set is associated with one span.

4. Listing the possible private attribute value combinations for each span, according to the release $M_T$.

The tuples that are associated with a span in the third step are now linked to the private attribute value combinations possible for this span according to the fourth step.

For instance, an attacker who knows the identity, gender and marital status of each of the mortgage company’s clients in Table 2.1 can see, 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. As mentioned in Section 2.2.1, anonymity 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 7 (k-anonymous release)** A release $M_T$ is $k$-anonymous with respect to a table $T$ if a linking attack on the tuples in $T$ using the release $M_T$ will not succeed in linking private data to fewer than $k$ individuals.

**Claim 3** A release $M_T$ is $k$-anonymous with respect to a table $T$ if, for every $x \in T$, either or both of the following hold:

1. $S(x_A) = \{[x]\}$
2. $|S(x_A)_T| \geq k$

Recall that while $S(x_A)_T$ may be different for various tables $T$, the set of equivalence classes $S(x_A)$ depends only on the model $M$.

**Proof.** Assume an attacker associated an individual’s tuple $(id_x, x_A) \in T_{ID}$ with its span $S(x_A)$. We will show that if one of the conditions holds, the attacker cannot compromise the $k$-anonymity of $x$. Since this holds for all tuples in $T$, the release is proven to be $k$-anonymous.

1. $S(x_A) = \{[x]\}$. Since the equivalence class $[x]$ is the only one in $S(x_A)$, then according to Claim 2, tuples whose span is $S(x_A)$ belong to $[x]$ regardless of their private attribute values. Therefore, no private attribute value can be associated with the span, and the
attacker gains no private knowledge from the model in this case. In other words, even if the attacker manages to identify a group of less than \( k \) individuals and associate them with \( S(x_A) \), no private information will be exposed through this association.

2. \( |S(x_A)_T| \geq k \). In this case, the model and the equivalence class populations might reveal to the attacker as much as the exact values of private attributes for tuples in \( T \) that belong to equivalence classes in \( S(x_A) \). However, since \( |S(x_A)_T| \geq k \), the number of individuals (tuples) that can be associated with the span is \( k \) or greater.

Note that the first condition pertains to a case that is not mentioned in the original \( k \)-anonymity model. This condition characterizes a span that groups tuples by public attributes alone. In the context of tables it is equivalent to suppressing the private attribute values for a set of rows. Clearly there is no privacy risk in this case, even if the set contains less than \( k \) rows.

We conclude this section by stressing how the formal definitions relate to the intuitive notion of anonymity that was presented in the beginning of the section. Each equivalence class relates to a subset of tuples which adhere to the same condition on public and private attributes. In that sense, the equivalence class is equivalent to a unique combination of public and private attributes in a row that appears in the private database. Just as a private database does not necessarily adhere to \( k \)-anonymity constraints, an equivalence class may contain any number of tuples. However, the spans represent the data as perceived by an attacker whose knowledge is limited to public attributes. Tuples that share the same span have a similar projection on the public domain. \( k \) or more tuples that share the same span would result in \( k \) or more rows that have the same public attribute values in an equivalent table.

### 2.4 Examples

In this section we show how the definition of model \( k \)-anonymity given in Section 2.3 can be used to verify whether a given data mining model violates the \( k \)-anonymity of individuals whose data was used for its induction.

#### 2.4.1 \( k \)-Anonymity of a Decision Tree

Assume a mortgage company has the data shown in Table 2.4 and wishes to release the decision tree in Figure 2.2, which clients can use to see whether they are eligible for a loan. Can the company release this decision tree while retaining 3-anonymity for the data in the table?
<table>
<thead>
<tr>
<th>Name</th>
<th>Marital Status</th>
<th>Sports Car</th>
<th>Loan Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lisa</td>
<td>Unmarried</td>
<td>Yes</td>
<td>good</td>
</tr>
<tr>
<td>John</td>
<td>Married</td>
<td>Yes</td>
<td>good</td>
</tr>
<tr>
<td>Ben</td>
<td>Married</td>
<td>No</td>
<td>bad</td>
</tr>
<tr>
<td>Laura</td>
<td>Married</td>
<td>No</td>
<td>bad</td>
</tr>
<tr>
<td>Robert</td>
<td>Unmarried</td>
<td>Yes</td>
<td>bad</td>
</tr>
<tr>
<td>Anna</td>
<td>Unmarried</td>
<td>No</td>
<td>bad</td>
</tr>
</tbody>
</table>

Table 2.4: Mortgage company data

The Marital Status of each individual is common knowledge, and thus a public attribute, while the classification good/bad loan risk is private knowledge. We will consider two cases, in which the Sports Car attribute can be either public or private.

The decision tree is a function that maps points in the original domain to the leaves of the tree, and inside the leaves, to bins, according to the class value. Hence those bins constitute partitions of the domain – each bin forms an equivalence class and contains all the tuples that are routed to it.

For example, the leaf $l_{\text{Unmarried}}$ contains one good loan risk classification, and one bad loan risk classification. That is, that leaf contains two bins, distinguished by means of the Loan Risk attribute. One tuple from $T$ is routed to the bin labeled bad, and one tuple is routed to the bin labeled good.

When both Sports Car and Marital Status are public attributes, the decision tree compromises $k$-anonymity. For example, the tuple John is the only one in the span containing the equivalence classes good, bad in the leaf $l_{\text{Married}}$. Note that in the special case that all the attributes in a decision tree are public and the Class attribute is private, the tree is $k$-anonymous if and only if every leaf contains at least $k$ learning examples or no learning examples at all.

If the Sports Car attribute is private, the decision tree implies just two spans: \{l_{\text{Married/good}}, l_{\text{Married/bad}}, l_{\text{no/good}}, l_{\text{no/bad}}\} for John, Ben, and Laura (since the attacker can route these tuples to any of the leaves $l_{\text{No}}, l_{\text{Married}}$), and \{l_{\text{Unmarried/good}}, l_{\text{Unmarried/bad}}, l_{\text{no/good}}, l_{\text{no/bad}}\} for Lisa, Robert, and Anna (since the attacker can route these tuples to any of the leaves $l_{\text{No}}, l_{\text{Unmarried}}$). As each of these spans contains 3 tuples, the decision tree maintains 3-anonymity.

### 2.4.2 Clustering

Assume that a data owner has the data shown in Table 2.5 and generates the clustering model shown in Figure 2.3. Now, he wishes to release the knowledge that his customers form four major groups: One in zip code 11001, comprising customers with various income levels; a second group, of high income customers, living mainly in zip codes 13010 and 14384;
a third group, of low income customers, living mainly in zip codes 13010, 14384 and 15012; and a fourth group in zip code 15013, comprising medium and high income customers. This knowledge is released by publication of four centroids, $c_1, c_2, c_3, c_4$, which represent those groups, and imply a partitioning of the domain into four areas, $C_1, ..., C_4$, by assigning the nearest centroid to each point in the domain.

The zip code of each individual is common knowledge, but the income level is private data held only by the data owner. We ask whether the data owner can release this knowledge while retaining 2-anonymity for the data in the table.

Each of the areas $C_i$ implied by the centroids constitutes an equivalence class, and every tuple $x$ is assigned an equivalence class according to its Zip Code and Income attribute values. The span of a tuple $x$ consists of all the areas that $x$ may belong to when the income corresponding to that tuple is varied across the full range of the data.

The span of John and Cathy is $\{C_1\}$, because no matter what their income is, any tuple whose zip code is 11001 would be associated (according to Figure 2.3) with $c_1$. Because this span has two tuples, it maintains their anonymity.

The span of Ben, Laura and William is $\{C_2, C_3\}$, because unless their income is known, each tuple in the span can be related to either of the two centroids $c_2, c_3$. This ambiguity maintains the anonymity of these tuples.

The span of Lisa is $\{C_3, C_4\}$. It can be seen that this span is not shared by any other tuple; thus, by our definitions, this clustering model compromises 2-anonymity. To see why, consider an attacker who attacks the model with a public table which includes individual names and zip codes. Given the populations of the equivalence classes, the attacker knows that at least one individual has to be related to $c_4$. The attacker concludes that Lisa’s tuple is the only candidate, and thus Lisa’s income level is high. Hence Lisa’s privacy has been breached.

### 2.4.3 $k$-Anonymity of Association Rules

Assume that a retailer providing both grocery and pharmaceutical products wishes to provide association rules to an independent marketer. While everyone can see what grocery products
<table>
<thead>
<tr>
<th>Name</th>
<th>Zip Code</th>
<th>Income</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>11001</td>
<td>98k</td>
</tr>
<tr>
<td>Cathy</td>
<td>11001</td>
<td>62k</td>
</tr>
<tr>
<td>Ben</td>
<td>13010</td>
<td>36k</td>
</tr>
<tr>
<td>Laura</td>
<td>13010</td>
<td>115k</td>
</tr>
<tr>
<td>William</td>
<td>14384</td>
<td>44k</td>
</tr>
<tr>
<td>Lisa</td>
<td>15013</td>
<td>100k</td>
</tr>
</tbody>
</table>

Table 2.5: Individuals’ data

Figure 2.3: Clustering model

a customer purchased (i.e., such items are public knowledge), pharmaceutical products are carried in opaque bags whose content is known only to the customer and the retailer.

After mining the data of some 1,000 customers, the retailer discovers two rules: (Cherries $\Rightarrow$ Viagra), with 8.4% support and 75% confidence, and (Cherries, Birthday Candles $\Rightarrow$ Tylenol), with 2.4% support and 80% confidence. Can these rules be transferred to the marketer without compromising customer anonymity?

Given a rule and a tuple, the tuple may contain just a subset of the items on the left-hand side of the rule; all of the items on the left-hand side of the rule; or all of the items on both the left-hand side and the right-hand side of the rule. Applying these three options for each of the two rules results in a model with nine equivalence classes\(^1\).

By looking at customers’ shopping carts, any attacker would be able to separate the customers into three groups, each constituting a span:

$S_1$: Those who did not buy Cherries. The model does not disclose any information about the private items of this group (this span contains a single equivalence class).

$S_2$: Those who bought both Cherries and Birthday Candles. Using the confidence and

\(^1\)In fact there are only seven equivalence classes, since the rules overlap: A tuple that does not contain items from the left-hand side of the first rule (‘no cherries’) cannot be classified as containing the items on the left-hand side or on both sides of the second rule.
support values of the rules, the attacker can learn private information about their 
Viagra and Tylenol purchases (this span contains four equivalence classes).

$S_3$: Those who bought Cherries and did not buy Birthday Candles. Using the confidence
and support values of the rules, the attacker can learn private information about their 
Viagra purchases (this span contains two equivalence classes).

We can now compute the implied population of every span. There are \( \frac{1000 \cdot 0.084}{0.75} = 112 \)
customers who bought cherries (with or without birthday candles). There are \( \frac{1000 \cdot 0.024}{0.8} = 30 \)
customers who bought both cherries and birthday candles, and \( 112 - 30 = 82 \) customers who bought cherries and did not buy birthday candles. There are \( 1000 - 112 = 888 \) customers who did not buy cherries at all. Therefore, a linking attack would link 888, 30 and 82 individuals to $S_1$, $S_2$ and $S_3$ respectively. Using the confidence of the rule (Cherries, Birthday Candles $\Rightarrow$ Tylenol), an attacker can deduce that of the 30 customers linked to $S_2$, 24 bought Tylenol and 6 did not, which is a breach if the retailer wishes to retain $k$-anonymity for \( k > 30 \).

We conclude that if the objective of the retailer is to retain 30-anonymity, then it can
safely release both rules. However, if the retailer wishes to retain higher anonymity, the
second rule cannot be released because it would allow an attacker to link a small group of
customers to the purchase of Tylenol.

2.5 $k$-Anonymity Preserving Data Mining Algorithms

In the previous section we used our definition of $k$-anonymity to test whether an existing
model violates the anonymity of individuals. However, it is very probable that the output
of a data mining algorithm used on non-anonymized data would cause a breach of anonymity. Hence the need for techniques to produce models which inherently maintain a
given anonymity constraint. We now demonstrate data mining algorithms which guarantee
that only $k$-anonymous models will be produced.

2.5.1 Inducing $k$-Anonymized Decision Trees

We present an algorithm that generates $k$-anonymous decision trees, given a set of tuples $T$, assuming $|T| > k$. The outline is given in Algorithm 2. We accompany the description of
the algorithm with an illustration of a 3-anonymous decision tree induction, given in Figure
2.4. It shows an execution of the algorithm using the data in Table 2.4 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 2.2.

The algorithm is based on concepts similar to those of the well-known ID3 decision tree
induction algorithm [67]. The algorithm begins with a tree consisting of just the root and a
set of learning examples associated with the root. Then it follows a hill climbing heuristic that splits the set of learning examples according to the value the examples have for a selected attribute. Of all the given nodes and attributes by which it can split the data, the algorithm selects the one which yields the highest gain (for a specific gain function – e.g., Information Gain or the Gini Index), provided that such a split would not cause a breach of k-anonymity. Note that unlike the ID3 algorithm, our algorithm does not use recursion; we consider instead all the splitting possibilities of all the leaves in a single queue, ordered by their gain. That is because splitting leaves might affect the k-anonymity of tuples in other leaves.

For simplicity, we embed generalization in the process by considering each possible generalization of an attribute as an independent attribute. Alternatively, e.g., for continuous attributes, we can start with attributes at their lowest generalization level. Whenever a candidate compromises anonymity and is removed from the candidate list, we insert into the candidate list a new candidate with a generalized version of the attribute. In that case, when generating candidates for a new node, we should consider attributes at their lowest generalization level, even if they were discarded by an ancestor node.

**Algorithm 1 Inducing k-Anonymous Decision Tree**

1: procedure MakeTree(T,A,k)
2: {T – dataset, A – list of attributes, k – anonymity parameter}
3:   r ← root node.
4:   candList ← {(a,r) : a ∈ A}
5: while candList contains candidates with positive gain do
6:   bestCand ← candidate from candList with highest gain.
7:   if bestCand maintains k-anonymity then
8:     Apply the split and generate new nodes N.
9:     Remove candidates with the split node from candList.
10:    candList ← candList ∪ {(a,n) : a ∈ A, n ∈ N}.
11:  else
12:   remove bestCand from candList.
13:  end if
14: end while
15: return generated tree.
16: end procedure

To decide whether a proposed split in line 1 would breach k-anonymity, the algorithm maintains a list of all tuples, partitioned to groups $T_s$ according to the span $s$ they belong to. Additionally, at every bin on every leaf, the span containing that bin $s(b)$ is stored. Lastly, for every span there is a flag indicating whether it is pointed to by a single bin or by multiple bins.
Initially, in line 1, the following conditions hold:

- the only leaf is the root;
- there are as many bins as class values;
- there is just one span if the class is private;
- there are as many spans as class values if the class is public.

If the class is private, the population of the single span is $T$ and its flag is set to *multiple*. If it is public, the population of every span is the portion of $T$ which has the respective class value, and the flag of every span is set to *single*.

In Figure 2.4, we begin with the root node $I_0$, which contains two bins, one for each class value. As the class value is private, only one span $s_0$ is created: it contains the two bins and its flag is set to *multiple*. $Ts_0$, the population of $s_0$, is comprised of all the tuples.

Figure 2.4: Inducing a 3-anonymous decision tree

When a leaf is split, all of its bins are also split. The algorithm updates the data structure as follows:

- If the splitting attribute is public, then the spans are split as well, and tuples in $Ts$ are distributed among them according to the value of the splitting attribute. Every new bin will point to the corresponding span, and the flag of every new span will inherit the value of the old one.

- If the splitting attribute is private, then every new bin will inherit the old span. The flag of that span will be set to *multiple*.

If splitting a leaf results in a span with population smaller than $k$ and its flag set to *multiple*, $k$-anonymity will be violated. In that case the splitting is rolled back and the algorithm proceeds to consider the attribute with the next largest gain.
In the example, there are two candidates for splitting the root node: the *Sports Car* attribute and the *Marital Status* attribute. The first one is chosen due to higher information gain. Two new leaves are formed, $l_{\text{Yes}}$ and $l_{\text{No}}$, and the bins are split among them according to the chosen attribute. Since the *Sports Car* attribute is private, an attacker will not be able to use this split to distinguish between tuples, and hence the same span $s_0$ is maintained, with the same population of size $> 3$ (hence 3-anonymous). There are two remaining candidates. Splitting $l_{\text{No}}$ with *Marital Status* is discarded due to zero information gain. The node $l_{\text{Yes}}$ is split using the public *Marital Status*. As a consequence, all the bins in $s_0$ are also split according to the attribute, and $s_0$ is split to two new spans, $s_1$ and $s_2$, each with a population of three tuples, hence maintaining 3-anonymity.

### 2.5.2 Inducing $k$-Anonymized Clusters

We present an algorithm that generates $k$-anonymous clusters, given a set of tuples $T$, assuming $|T| > k$. The algorithm is based on a top-down approach to clustering [75].

The algorithm starts by constructing a minimal spanning tree (MST) of the data. This tree represents a single equivalence class, and therefore a single span. Then, in consecutive steps, the longest MST edges are deleted to generate clusters. Whenever an edge is deleted, a cluster (equivalence class) $C_i$ is split into two clusters (two equivalence classes), $C_{i1}$ and $C_{i2}$. As a consequence, every span $M = \{C_1, ..., C_i, ..., C_m\}$ that contained this equivalence class is now split into three spans:

1. $S_1 = \{C_1, ..., C_{i1}, ..., C_m\}$, containing the points from $C_i$ which, according to the public attribute values, may belong only to $C_{i1}$;
2. $S_2 = \{C_1, ..., C_{i2}, ..., C_m\}$, containing the points from $C_i$ which, according to the public attribute values, may belong only to $C_{i2}$;
3. $S_3 = \{C_1, ..., C_{i1}, C_{i2}, ..., C_m\}$, containing the points from $C_i$ which, according to the public attribute values, may belong either to $C_{i1}$ and $C_{i2}$.

The points that belonged to $M$ are now split between the three spans. If we can link to each of the new spans at least $k$ points, or no points at all, then the split maintains anonymity. Otherwise, the split is not performed. Edges are deleted iteratively in each new cluster, until no split that would maintain anonymity can be performed. When this point is reached, the algorithm concludes. The algorithm can also be terminated at any earlier point, when the data owner decides that enough clusters have been formed.

To see how the algorithm is executed, assume a data domain that contains two attributes. The attribute *age* is public, while the attribute *result*, indicating a result of a medical examination, is private. Figure 2.5 shows several points in the domain, and an MST that was constructed over these points.
The algorithm proceeds as follows: At first, the MST forms a single cluster (equivalence class) $C_1$, containing all the points, and a single span $S_1 = \{C_1\}$. Then the edge CD, which is the longest, is removed. Two clusters form as a result: $C_2 = \{A,B,C\}$ and $C_3 = \{D,E,F,G\}$. Consequently, we get three spans: $S_2 = \{C_2\}$, to which the points $A,B,C$ are linked; $S_3 = \{C_3\}$, to which the points $D,E,F,G$ are linked; and $S_4 = \{C_2,C_3\}$, to which no point is linked, and can therefore be ignored. In $C_3$, the longest edge is DE. Removing it will split the cluster into $C_4 = \{D\}$ and $C_5 = \{E,F,G\}$. Then the span $S_3$, which contains the split cluster $C_3$, is split into three spans: $S_5 = \{C_4\}$, to which no point is linked; $S_6 = \{C_5\}$, to which no point is linked; and $S_7 = \{C_4,C_5\}$, to which the points $D,E,F,G$ are linked. Note that although point $D$ is the only one in equivalence class $C_4$, this does not compromise $k$-anonymity, because the public attributes do not reveal enough information to distinguish it from the points $E,F,G$ in cluster $C_5$. Although the algorithm may continue to check other possible splits, it can be terminated at this point, after forming three clusters.

### 2.6 Related Work

The problem of $k$-anonymity has been addressed in many papers. The first methods presented for $k$-anonymization were bottom-up, relying on generalization and suppression of the input tuples [71, 73, 85]. Heuristic methods for $k$-anonymization that guarantee optimal $k$-anonymity were suggested in [11, 50]. Iyengar [44] suggested a metric for $k$-anonymizing data used in classification problems, and used genetic algorithms for the anonymization process. A top-down approach, suggested in [37], preserves data patterns used for decision tree classification. Another top-down approach, suggested in [12] utilizes usage metrics to bound generalization and guide the anonymization process. When the above methods are compared to specific implementations of ours (such as the one described in Section 2.5.1), several differences are revealed. First, while all these methods anonymize an attribute across all tuples, ours selectively anonymizes attributes for groups of tuples. Second, our method
is general and thus can be easily adjusted for any data mining task. For example, one could think of applying our method in one way for classification using decision trees and in another for classification using Bayesian classifiers. In both respects, using our method is expected to yield better data mining results, as will be demonstrated in Section 3.3 for decision tree induction.

A recent work [43] studies how data mining based on computing the distance between instances can be applied to anonymized datasets. The focus of this work is not on the anonymization method, but rather on how to process the data and build the data mining model once anonymization has complete.

A recent independent work [51] discusses multi-dimensional global recoding techniques for anonymization. Anonymity is achieved by mapping the domains of the quasi-identifier attributes to generalized or altered values, such that each mapping may depend on the combination of values over several dimensions. The authors suggest that multi-dimensional recoding may lend itself to creating anonymizations that are useful for building data mining models. Indeed, the methods we presented in this chapter can be classified as multi-dimensional global recoding techniques, and complement the aforementioned work. Another multi-dimensional approach is presented in [5]. The authors provide an \(O(k)\)-approximation algorithm for \(k\)-anonymity, using a graph representation, and provide improved approximation algorithms for \(k = 2\) and \(k = 3\). Their approximation strives to minimize the cost of anonymization, determined by the number of entries generalized and the level of anonymization. One drawback of applying multi-dimensional global recoding before mining data is the difficulty of using the anonymized results as input for a data mining algorithm. For example, determining the information gain of an attribute may not be trivial when the input tuples are generalized to different levels. Our approach circumvents this difficulty by embedding the anonymization within the data mining process, thus allowing the data mining algorithm access to the non-anonymized data.

One essential difference between previous works and this one is in the approach to measuring utility. One approach is to obtain the privacy constraint while minimizing information loss, usually measured by the extent of suppression or generalization required to obtain the \(k\)-anonymized outcome. For example, the Loss Metric (LM) [44] assigns to each entry in a record a penalty between 0 and 1, depending on the required amount of generalization. Another approach is provided by the Discernibility Metric (DM) [11], which expresses the desire to maintain the differences between records in the dataset, to the extent allowed by the \(k\)-anonymity constraints. It assigns to each generalized record a penalty according to the number of other generalized records identical to it, where suppression incurs a penalty \(|D|\) (the size of the dataset). The Classification Metric (CM) [44] favors generalizations that maintain homogeneous values of private attributes in each group of generalized records. In each such group, it penalizes the records that take a different private value than the majority of the records in the group. A more recent work [41] presents the Mutual In-
formation Utility (MI) and the Private Mutual Information Utility (PMI) measures, which are information-theoretic measures of information loss. The first (MI) measures the mutual information between the public attributes in the original database and the ones in the anonymized version. It expresses how much information is revealed about the original data by the generalized data. The second (PMI) incorporates also the private attribute values, and measures the amount of information that the generalized public data reveals on the private data. All the methods described above regard anonymization as a standalone operation that is executed separately from any following analysis. The various proposed metrics allow to guide the anonymization process such that the outcome optimizes the metric. In contrast, in this work we consider anonymization and data mining simultaneously, and we measure utility by means of the performance (e.g., expected accuracy) of the resulting data mining model. By dropping the metric as a mediator between anonymization and data analysis, we can make anonymization choices in the context of a specific data mining algorithm. In the evaluation section of the next chapter we take a closer look into this subject.

Embedding $k$-anonymity in data mining algorithms was discussed in [8] in the context of pattern discovery. The authors do not distinguish between private and public items, and focus on identifying patterns that apply for fewer than $k$ transactions. The authors present an algorithm for detecting inference channels in released sets of itemsets. Although this algorithm is of exponential complexity, they suggest an optimization that allows running time to be reduced by an order of magnitude. A subsequent work [7] shows how to apply this technique to assure anonymous output of frequent itemset mining. In comparison, our approach allows breaches of $k$-anonymity to be detected and $k$-anonymization to be embedded in a broader range of data mining models. We intend to further explore the implications of our approach on itemset mining in future research.

Several recent works suggest new privacy definitions that can be used to overcome the vulnerability of $k$-anonymity with respect to data diversity. Wang et al. [84] offer a template-based approach for defining privacy. According to their method, a data owner can define risky inference channels and prevent learning of specific private attribute values, while maintaining the usefulness of the data for classification. This kind of privacy is attained by selective suppression of attribute values. [56] presents the $\ell$-diversity principle: Every group of individuals that can be isolated by an attacker should contain at least $\ell$ “well-represented” values for a sensitive attribute. As noted in [56], $k$-anonymization methods can be easily altered to provide $\ell$-diversity. We will show in the next chapter (Section 3.4) that our method can also be applied to $\ell$-diversification by adding private value restrictions on spans. Our definitions can be easily augmented with any further restriction on private attributes values, such as those presented in [84]. Kantarcioglu et al. [47] suggest another definition for privacy of data mining results, according to the ability of the attacker to infer private data using a released “black box” classifier. While this approach constitutes a solution to the inference vulnerability of $k$-anonymity, it is not clear how to apply it to data mining algorithms such
that their output is guaranteed to satisfy privacy definitions.

A different approach for privacy in data mining suggests that data mining should be performed on perturbed data [2, 13, 18, 21, 29, 31, 47]. This approach is applied mainly in the context of statistical databases.

Cryptographic methods were proposed for privacy-preserving data mining in multiparty settings [24, 40, 46, 55, 78]. These methods deal with the preservation of privacy in the process of data mining and are thus complementary to our work, which deals with the privacy of the output.

We refer the interested reader to [81] and [3] for further discussion of privacy preserving data mining.

2.7 Conclusions

Traditionally, the data owner would anonymize the data and then release it. Often, a researcher would then take the released data and mine it to extract some knowledge. However, the process of anonymization is oblivious to any future analysis that would be carried out on the data. Therefore, during anonymization, attributes critical for the analysis may be suppressed whereas those that are not suppressed may turn out to be irrelevant. When there are many public attributes the problem is even more difficult, due to the curse of dimensionality [1]. In that case, since the data points are distributed sparsely, the process of $k$-anonymization reduces the effectiveness of data mining algorithms on the anonymized data and renders privacy preservation impractical.

Using data mining techniques as a basis for $k$-anonymization has two major benefits, which arise from the fact that different data mining techniques consider different representations of data. First, such anonymization algorithms are optimized to preserve specific data patterns according to the underlying data mining technique. While this approach is more appealing when the data owner knows in advance which tool will be used to mine the data, our experiments show that these anonymizations may also be adequate when this is not the case. Second, as illustrated in section 2.3, anonymization algorithms based on data mining techniques may apply different generalizations for several groups of tuples rather than the same generalization for all tuples. In this way, it may be possible to retain more useful information. This kind of anonymization, however, has its downsides, one of which is that using different generalization levels for different tuples requires that the data mining algorithms be adapted. Therefore, we believe that this model will be particularly useful when the anonymity constraints are embedded within the data mining process, so that the data mining algorithm has access to the non-anonymized data.

To harness the power of data mining, our work proposes extended definitions of $k$-anonymity that allow the anonymity provided by a data mining model to be analyzed.
owners can thus exchange models which retain the anonymity of their clients. Researchers looking for new anonymization techniques can take advantage of efficient data mining algorithms: they can use the extended definitions to analyze and maintain the anonymity of the resulting models, and then use the anonymity preserving models as generalization functions. Lastly, data miners can use the definitions to create algorithms guaranteed to produce anonymous models.
Chapter 3

$k$-Anonymous Decision Tree Induction
3.1 Introduction

This chapter continues the discussion of $k$-anonymous data mining and takes an in-depth look at inducing $k$-anonymous decision trees. We take a direct approach to the combination of $k$-anonymity and data mining. Rather than asking how data can be anonymized so that it is useful for data mining, we ask how data can be mined so that the resulting model is guaranteed to maintain $k$-anonymity. We consider scenarios in which the data owner wishes to release knowledge in the form of a data mining model, while protecting $k$-anonymity of the underlying data. We specifically discuss this problem in the context of decision tree induction. In this context, anonymity may be at risk, e.g., when the decision tree overfits a small set of learning examples and allows the attacker to link private attribute values to the individuals represented in those learning examples.

As an example, we revisit the case of re-identification by linking presented in [74]. The Group Insurance Commission (GIC) in Massachusetts published medical data pertaining to state employees. The published data were believed to be anonymous since they did not contain names or addresses of patients. In the paper, the author describes how she purchased the voter registration list for Cambridge Massachusetts for twenty dollars and linked medical information to individuals in the voter list using ZIP code, birth date and gender, which appeared in both data sets. For instance, according to the Cambridge voter list, only six people had the governor’s particular birth date; only three of them were men; and, the governor was the only one in his 5-digit ZIP code. Therefore, the author was able to re-identify the medical records pertaining to the governor. $k$-anonymity was designed to thwart this kind of attack.

In this chapter, we consider a slightly different scenario. Assume that instead of being published in relational form, the data were to be released in the form of a decision tree, allowing a diagnosis be made using patient information. The decision tree is induced from the state employees’ medical records. Therefore, it would be susceptible to a similar attack: knowing that the decision tree is based on these records, an attacker can follow the decision tree nodes using the data from the voter registration list. When the attribute values along the path from the root to a leaf match the values available in the voter list, the attacker can associate the medical information in the leaf with individuals. For example, in Figure 3.1, an attacker could link the identity of a 51 year old male living in West Cambridge with the diagnosis appearing in the leaf. If less than $k$ state employees share the same attribute values appearing along the path from the root to the leaf, then the diagnosis in the leaf can be linked distinctly to those individuals and their $k$-anonymity compromised. Our goal is to induce decision trees that cannot be used for linking sensitive information to groups of less than $k$ individuals. One of the challenges is to do so in a manner that preserves the utility of the decision tree, in the sense that the tree should maintain good predictive value. For example, the usefulness of a decision tree can be measured according to its prediction
In this chapter we describe a decision tree induction algorithm whose output is guaranteed not to compromise the $k$-anonymity of the data from which it was induced. An independent work [4] discusses a similar concept based on clustering algorithms. Another work [8] presents a similar concept in the context of itemset mining. Neither work differentiates public attributes from private attributes. In the latter, only binary attributes are handled. In addition, anonymity is achieved by iterative postprocessing of the data mining output, while we suggest an integration of the two processes. Privacy in the context of itemset mining was also discussed in [82], but the authors study a different problem: how a data owner can alter and publish raw data such that sensitive knowledge cannot be inferred from it. Extension of $k$-anonymity to nonrelational models was studied also in [90], in the context of publishing a set of views.

Our approach is superior to existing methods such as [11, 37, 44], which guarantee $k$-anonymity of a data mining model by building it from a $k$-anonymized table. For the sake of efficiency, these methods generalize attributes homogeneously over all the tuples. This kind of anonymization was termed single-dimension recoding [50]. Using our method, however, attributes can be generalized differently in each tuple, depending on other attribute values. This kind of anonymization was termed multi-dimensional recoding. Furthermore, in the existing methods, heuristic cost metrics are the driving force. For example, a classification metric, which is in essence the classification error over the entire training data, may be used. Such metrics are not necessarily optimal for a specific data mining task. We show that a decision tree induced using our method can be more accurate than that induced by existing methods, yet both decision trees prevent the attacker from linking private information appearing in the release to fewer than $k$ individuals from the private database. A recent work [52] uses multi-dimensional recoding techniques to anonymize relational data, and can also make use of information gain to guide the anonymization process. However, the proposed algorithm, Mondrian InfoGain, handles only public attributes values. We show that better
results can be obtained when the impact of private attribute values is considered throughout the generalization process.

This chapter makes the following contributions:

- It presents decision tree induction algorithms based on ID3 and C4.5, which guarantees \( k \)-anonymous output and which performs more accurately than existing methods on standard benchmarks.

- It shows how attribute generalization can be used within decision tree induction.

- It discusses the privacy implications of using instances with missing values for decision tree induction.

The organization of this chapter is as follows: Section 3.2 demonstrates how the extended definitions of \( k \)-anonymity presented in the previous chapter can be incorporated within decision tree induction algorithms to guarantee \( k \)-anonymous output. Section 3.3 compares these new algorithms experimentally to previous work. Section 3.4 discusses the possibility of applying the proposed framework to privacy models that extend \( k \)-anonymity. Conclusions are drawn in Section 3.5.

### 3.2 Inducing \( k \)-Anonymous Decision Trees

This section presents an algorithm which induces \( k \)-anonymous decision trees. The algorithm is based on the well-known ID3 algorithm [67] and on its extension C4.5 [68]. 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 divided 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, \( k \)ADET (\( k \)-Anonymous Decision Tree, Algorithm 2), uses the same hill-climbing approach, with two changes: First, when considering all possible splits of a node, \( k \)ADET eliminates those that would lead to a \( k \)-anonymity breach. Second, the algorithm is not recursive. Rather, all the potential splits are considered in a single priority queue and the best one of all those that retain \( k \)-anonymity is chosen. This method is required since \( k \)-anonymity is defined in terms of spans, which may include bins from several decision tree nodes. A decision regarding one node may thus influence future decisions regarding other nodes. Nevertheless, the property of being \( k \)-anonymous is monotonic, meaning that once it is lost it can never be regained by future splits.
Algorithm 2 \( k \)-Anonymous Decision Tree (kADET)

1: **Input:** \( T \) – private dataset, \( A \) – public attributes, \( B \) – private attributes, \( C \) – the class attribute, \( k \) – anonymity parameter

2: **procedure** MAIN()

3: Create root node in Tree

4: Create in root one bin \( b_c \) for each value \( c \in C \), divide \( T \) among the bins

5: if \( C \in A \) then

6: Create one span \( S_c \) for every value \( c \in C \). \( S_c.Bins \leftarrow \{b_c\}, S_c.Population \leftarrow b_c.Population, S_c.Nodes \leftarrow \{root\} \)

7: set root.Spans to the list of all spans

8: else

9: Create a single span \( s \). Set \( s.Bins \) to the list of all bins, \( s.Population \leftarrow T, s.Nodes \leftarrow \{root\}, root.Spans \leftarrow \{s\} \)

10: if \( 0 < |s.Population| < k \) then

11: return nil

12: end if

13: end if

14: for \( att \in A \cup B \setminus \{C\} \) do

15: add \((root, att, gain(root, att))\) to Queue

16: end for

17: while Queue has elements with positive gain do

18: Let \((n, a, gain) = \arg\max_{gain} \{Queue\}\)

19: if \( n.sons \neq \emptyset \) then

20: continue

21: end if

22: if \( Breach(n, a, k) \) then

23: if \( a \) has generalization \( a' \) then

24: insert \((n, a', gain(n, a'))\) to Queue

25: end if

26: else

27: Split\((n, a)\)

28: end if

29: end while

30: Set the Class variable in each leaf to the value with the largest bin.

31: return Tree

32: end procedure
Algorithm 3 Supporting procedures for kADET

1:  \textbf{procedure} \textsc{Breach}(node, att, k)
2:      \textbf{if} \ att \in B \ \textbf{then}
3:          \textbf{return} \ false
4:      \textbf{end if}
5:      \textbf{for} \ v \in \textit{att.values} \ \textbf{and} \ \textit{span} \in \textit{node.Spans} \ \textbf{do}
6:          \textbf{if} \ |\textit{span.Bins}| > 1 \ \textbf{and} \ 0 < |\{t \in \textit{span.Population} : t[att] = v\}| < k \ \textbf{then}
7:              \textbf{return} \ true
8:          \textbf{end if}
9:      \textbf{end for}
10:     \textbf{return} \ false
11:  \textbf{end procedure}

12:  \textbf{procedure} \textsc{Split}(node, att)
13:      \textbf{for} \ v \in \textit{att.values} \ \textbf{do}
14:          Let node.sons[v] be a new descendant node
15:          Let node.sons[v].Bins[b] be a new bin, which refines node.Bins[b] s.t.
16:              node.sons[v].Bins[b].tuples \leftarrow \{t \in node.Bins[b].tuples : t[att] = v\}
17:          Let node.sons[v].Spans \leftarrow node.Spans
18:      \textbf{end for}
19:      \textbf{for} \ \textit{span} \in \textit{node.Spans} \ \textbf{do}
20:          replace each bin of the original node with its
21:              refinements, computed above, and add the new nodes to \textit{span.Nodes}
22:      \textbf{end for}
23:      \textbf{if} \ att \in A \ \textbf{then}
24:          \textbf{for} \ \textit{span} \in \textit{node.Spans} \ \textbf{do}
25:              Remove \textit{span} from every node \textit{n} \in \textit{span.Nodes}
26:                  \textbf{for} \ v \in \textit{att.values} \ \textbf{do}
27:                      Create a new span \textit{s}_v
28:                          \textit{s}_v.Nodes \leftarrow \textit{span.Nodes} \\setminus \{node.sons[u] : u \neq v\}
29:                          \textit{s}_v.Bins \leftarrow \textit{span.Bins} \\setminus \{\textit{bin} \in node.sons[u].Bins : \textit{u} \neq v\}
30:                          \textit{s}_v.Population \leftarrow \{t \in \textit{span.Population} : t[att] = v\}
31:                      Add \textit{s} to node.sons[v].spans
32:                          Add \textit{s} to every node \textit{n} \in \textit{span.Nodes} \setminus node.sons
33:          \textbf{end for}
34:      \textbf{end for}
35:  \textbf{end procedure}
### 3.2.1 kADET Algorithm

The input of kADET is a private database $T$, the public attributes $A$, the private attributes $B$, the class attribute $C$, and the anonymity parameter $k$. First, kADET computes the initial set of equivalence classes (bins) and spans: a single span containing all of the bins and populated by 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 if the class is public. In the first case, if the span contains less than $k$ tuples from $T$, kADET returns nil and terminates. After initializing the bins and the spans, kADET creates the initial queue of possible splits, where each candidate split contains the root node and an attribute from $A$ or $B$. The queue is ordered according to the gain from each split. kADET then enters its main loop.

The main loop of kADET has the following steps: first, the most gainful candidate split $(\text{node, attribute, gain})$ is popped out of the queue. If the node in this candidate is already split, the candidate is purged. Otherwise, kADET tests whether splitting the node according to the suggested attribute would breach $k$-anonymity. If it would, then, again, this candidate is purged. However, if the attribute for the purged candidate can be generalized, then a new candidate is inserted to the queue, this time with the generalized attribute. For example, if a five digit zipcode compromised anonymity, it can be replaced with a four digit zipcode (the last digit is suppressed). Section 3.2.2 explains in more detail how attribute generalization helps anonymization. Finally, if $k$-anonymity is not breached, the node is split.

Several actions are taken in the splitting of a node: first, every bin of the parent node is split between the descendant nodes, according to the value of the chosen attribute. Accordingly, every span that contains this bin is updated with the new list of bins. The descendant nodes inherit the list of spans from their parent, and are added to the lists of nodes of those spans. If the chosen attribute is private, no further action is required, as the attacker cannot distinguish between the new bins. However, if the chosen attribute is public, then the attacker can use the split to distinguish tuples. Specifically, tuples that are routed to one of the new nodes will not be routed to its sibling nodes. Therefore, each of the spans in the split node is split into new spans, one for each new descendant node. Each new span contains the bins from the original span, except for those of the sibling nodes. Likewise, the population of the original span is divided according to the value the tuples take on the chosen attribute. Nodes whose bins are contained in the new spans, and which are not descendants of the original node, are associated with all of the new spans.

Figure 3.2 demonstrates an execution of kADET with $k = 3$, using the data in Table 3.1 as input. Marital Status is a public attribute; Loan Amount and Loan Risk are private attributes. In the first step, the root node is created. Two bins are created within the root node, one for each value of the class attribute. Since the class attribute is private, there is only a single span comprised of both bins, populated by all the tuples. In the next step, the Loan Amount attribute is chosen for splitting the root node. Since the attribute is private,
Figure 3.2: Execution of $kADET$
<table>
<thead>
<tr>
<th>Name</th>
<th>Marital Status</th>
<th>Loan Amount</th>
<th>Loan Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lisa</td>
<td>Unmarried</td>
<td>$\geq 100K$</td>
<td>good</td>
</tr>
<tr>
<td>John</td>
<td>Married</td>
<td>$\geq 100K$</td>
<td>good</td>
</tr>
<tr>
<td>Ben</td>
<td>Married</td>
<td>$&lt; 100K$</td>
<td>bad</td>
</tr>
<tr>
<td>Laura</td>
<td>Married</td>
<td>$&lt; 100K$</td>
<td>bad</td>
</tr>
<tr>
<td>Robert</td>
<td>Unmarried</td>
<td>$\geq 100K$</td>
<td>bad</td>
</tr>
<tr>
<td>Anna</td>
<td>Unmarried</td>
<td>$&lt; 100K$</td>
<td>bad</td>
</tr>
</tbody>
</table>

Table 3.1: Mortgage company data

the span is not split (an attacker would not be able to distinguish which tuples should be routed to the left node and which should be routed to the right one). The span is populated by six individuals, so it maintains 3-anonymity. Finally, in the last step the Marital Status attribute is chosen for splitting the node $l_{\text{AmtLarge}}$. Since it is a public attribute, the attacker can use it to distinguish between tuples. One span is comprised of the bins $\{b_{111}, b_{211}, b_{12}, b_{22}\}$, and the other is comprised of the bins $\{b_{112}, b_{212}, b_{12}, b_{22}\}$. Each of the spans is populated by three individuals, so the split maintains 3-anonymity. There are no further useful candidates for splitting, so the decision tree induction is terminated after this step.

### 3.2.2 Generalization in Decision Tree Induction

Generalization is the most prevalent method for table anonymization, and it can also be valuable when inducing $k$-anonymous decision trees. In generalization, an attribute value is replaced with a less specific but semantically consistent value. In most works, attributes are generalized using a pre-determined hierarchy of values. For example, for a Zip Code attribute, the lowest level of the hierarchy could be a 5-digit zip code. This value could be generalized by omitting the rightmost digit to obtain a 4-digit zip code. Similarly, additional digits could be omitted up to a 1-digit zip code consisting only of the left-most digit. When this digit is omitted as well, the highest level of the generalization hierarchy is reached, with a single value (usually denoted *). Practically, replacing a value with * means suppression of the value.

![Figure 3.3: 2-Anonymous Mortgage company decision tree with generalization](image-url)
<table>
<thead>
<tr>
<th>Name</th>
<th>Marital Status</th>
<th>Zip Code</th>
<th>Loan Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lisa</td>
<td>Married</td>
<td>94138</td>
<td>good</td>
</tr>
<tr>
<td>John</td>
<td>Married</td>
<td>94138</td>
<td>good</td>
</tr>
<tr>
<td>Ben</td>
<td>Married</td>
<td>94139</td>
<td>good</td>
</tr>
<tr>
<td>Laura</td>
<td>Married</td>
<td>94139</td>
<td>bad</td>
</tr>
<tr>
<td>Robert</td>
<td>Married</td>
<td>94139</td>
<td>good</td>
</tr>
<tr>
<td>Anna</td>
<td>Unmarried</td>
<td>94138</td>
<td>bad</td>
</tr>
<tr>
<td>Emily</td>
<td>Unmarried</td>
<td>94139</td>
<td>bad</td>
</tr>
<tr>
<td>Chris</td>
<td>Unmarried</td>
<td>94138</td>
<td>bad</td>
</tr>
<tr>
<td>Karen</td>
<td>Unmarried</td>
<td>94141</td>
<td>good</td>
</tr>
<tr>
<td>Michael</td>
<td>Unmarried</td>
<td>94142</td>
<td>bad</td>
</tr>
</tbody>
</table>

Table 3.2: Mortgage company data – generalization example

To see how generalization is used within the kADET algorithm, consider the private database given in Table 3.2. Marital Status and Zip Code are public attributes, and the class attribute Loan Risk is private. 2-anonymous ID3 induction on this table results in the decision tree shown in Figure 3.3. We show how the algorithm considers candidates for splitting, taking into account generalizations when necessary:

1. At first, the root node is created, and two candidate attributes are considered for splitting: Marital Status and 5-digit Zip Code. Marital Status is chosen, because it yields higher information gain. The split maintains 2-anonymity (there are 5 individuals in each resulting span).

2. There are two candidates for the next split: split \( l_{Married} \) with 5-digit Zip Code, or split \( l_{Unmarried} \) with 5-digit Zip Code. The second one yields higher information gain, so it is chosen for the split. However, this split compromises 2-anonymity; for example, if the split is applied, then Karen can be identified as the only individual routed to the leaf \(<Marital.Status=Unmarried, Zip.Code=94141>\) and it can be linked to a good Loan Risk classification. Therefore, this split is canceled. Without generalization, this leaf could not be split further.

3. The candidate split of \( l_{Unmarried} \) with 5-digit Zip Code is removed, and instead a new candidate is considered: split of \( l_{Unmarried} \) with 4-digit Zip Code. The 5-digit Zip Code had four possible values: 94138, 94139, 94141, 94142. The 4-digit Zip code has only two: 9413* and 9414*. The split can be applied now, since it maintains 2-anonymity – there are two and three individuals respectively in each of the resulting spans.

4. Finally, the candidate split of \( l_{Married} \) with 5-digit Zip Code is applied, and it maintains 2-anonymity.

Note that this method allows the use of different generalization levels of attributes in different
sub trees: 5-digit Zip Code was used in one subtree, and 4-digit Zip code was used in another. This kind of generalization was termed multi-dimensional global recoding [50].

3.2.3 Correctness and Overhead Analysis

The key to proving the algorithm’s correctness is in showing that the computed population of each span is the same as the one defined in Section 2.3: the set of tuples which, without knowledge of private attribute values, can be routed to the same set of bins.

Claim 4 At every stage of the kADET algorithm, for every set of tuples $G \subseteq T_{ID}$ that can be grouped by the attacker in a linking attack on the current tree structure, there exists a single span $S$ such that:

1. $S.Population = G$.
2. $S.Bins$ is the set of equivalence classes to which any of the tuples of $G$ may be routed.
3. $S.Nodes$ is the set of leaves that include bins from $S.Bins$, and for each such leaf $l$, $S \in l.Spans$. For any leaf $l \notin S.Nodes$, it holds that $S \notin l.Spans$.

In addition, if $|s.Bins| > 1$, then $|S.Population| \geq k$ holds.

Proof. Proof by induction. As the basis of the induction, consider the creation of the root node. If the root node is released as a tree, then the only information available to the attacker is the distribution of class attribute values in the root node. If the class attribute is private, then any of the tuples may have any of the class attribute values, so an attacker can only link the whole population of $T_{ID}$ to this distribution of private attribute values. The check on Line 2 ensures that this population contains at least $k$ tuples. If the class attribute is public, then the attacker can distinguish between tuples with different class attributes, grouping together those whose class attribute is the same. In that case, every span contains only a single equivalence class, and no private information is revealed. Lines 2-2 in the algorithm apply this partitioning on the tuples, maintaining the properties detailed in the claim.

Now assume that the property holds for the induced tree at some stage of the algorithm, and let $G$ be a set of tuples from $T_{ID}$ that an attacker can group together using a linking attack on the tree. According to the assumption, there is a span $S$ related to this group, for which the properties in the claim hold. Consider a split on attribute $Att$ in node $N$. If the tuples in $G$ cannot be routed to node $N$, then an attacker cannot use this split to learn anything about the tuples in $G$. In this case, according to the assumption, $S \notin N.Spans$, and none of the statements in the procedure $Split$ will affect the properties of $S$. Thus, they will still be valid in the next iteration.
Otherwise, the split may affect the group. If the split is on a private attribute, then an attacker cannot use it to distinguish between tuples in \( G \), but it affects the bins to which the tuples may be routed. For every bin in the parent node \( N \) to which the tuples in \( G \) could be routed, they can now be routed to the respective bin in any of \( N \)'s descendant nodes. Lines 3-3 of the algorithm pertain to this case. Following the split:

1. \( S.Population = G \) holds, since no change to the population of \( S \) is applied.

2. \( S.Bins \) is updated to reflect the set of equivalence classes to which any of the tuples of \( G \) may now be routed.

3. \( S.Nodes \) is updated to include the newly created descendant nodes, as each of those nodes will contain at least one bin to which the tuples in \( G \) are routed. Correspondingly, the span \( S \) will be added to \( l.Spans \), for each of the new leaves \( l \). Spans that are not related to the parent node \( N \) will not be added to any of its descendants.

The split will be accepted only after passing the check on Line 2. This ensures that the property \( |S'.Population| \geq k \) holds for all the spans that now contain more than a single bin.

If the chosen attribute \( Att \) is public, then an attacker may use the split to break \( G \) into smaller groups; tuples in \( G \) which have different values on the attribute \( Att \) can be distinguished using this attribute value. Therefore, a subset of tuples from \( G \) can be routed by the attacker to each of the newly created children of \( N \). In other words, each such subset forms a new span, whose population is the subset of \( G \) with the relevant attribute value. Lines 3-3 pertain to this case. Following the split, the span \( S \) that is related to \( G \) will be eliminated. For each value of the attribute \( Att \) (respectively, each new descendant node of \( N \)), a new span \( S' \) is created, corresponding to an identifiable subset of tuples from \( G \). For each new span \( S' \):

1. \( S'.Population \) is updated to hold the subset of individuals from \( G \) identifiable by a unique value \( v \) of attribute \( Att \).

2. \( S.Bins \) is updated to reflect that the tuples in the subset can only be routed to the bins in the descendant node corresponding to the value \( v \). Routing of those tuples to bins in nodes external to \( N \) is not affected.

3. \( S.Nodes \) is updated to reflect that the tuples in the subset can only be routed to the leaf corresponding to the value \( v \). Routing to nodes external to \( N \) is not affected.

4. \( S' \) will be added only to \( l.Spans \) of the relevant leaf \( l \), and will not appear in any of the other new leaves. Leaves external to \( N \) will not be affected.
The split will be accepted only after passing the check on Line 2. This ensures that the property $|S'.\text{Population}| \geq k$ holds for all the newly created spans $S'$, if they have more than one bin. The inductive assumption asserts that the condition still holds for any of the other spans as well.

Since the spans imposed by the tree are maintained throughout the induction of the tree, and at each stage it is ensured that any span maintains the $k$-anonymity condition, it holds that the output tree maintains the $k$-anonymity condition as well.

**Corollary 2** *A linking attack on the tuples in $T_{\text{ID}}$ given the output of Algorithm 2 cannot link private data to fewer than $k$ individuals.*

We now present an upper bound on the complexity of $k\text{ADET}$, given $m$ learning examples. The learning examples are over a domain with $a$ attributes. Each attribute has up to $v$ different values. It is well known that the worst-case complexity of ID3 is exponential in $a$. Specifically, $O(v^a)$ entropy calculations are required to induce the decision tree. However, as noted in [77], a decision tree with $m$ examples will contain at most $m - 1$ decision nodes, because a node is split only if the split creates at least two nonempty leaves. Since the number of learning examples $m$ is typically far lower than $v^a$, it is acceptable to bound the entropy calculation complexity with $O(a \cdot m)$. The dominant complexity factor of ID3 is instance-count additions, so the worst case complexity of ID3 amounts to $O(a^2 \cdot m)$ [77].

$k\text{ADET}$ is different from ID3 in the overhead incurred by a split. In ID3, that overhead is simply the calculation of the information gain per attribute, and then the distribution of the learning example among the new leaves. $k\text{ADET}$, on the other hand, (i) stores every candidate in a priority queue, sorted according to information gain and (ii) decides for every candidate it removes from the queue whether it preserves $k$-anonymity. Storing and retrieving candidates from a priority queue (e.g., a binary heap) costs $\log(am)$ per candidate – the total number of candidates in the priority queue will be no higher than the number of decision nodes (bounded by $m$) times the number of attributes. There are up to $a$ candidates for every decision node, so the complexity of preparing the candidates and inserting them into the queue amounts to $O(am + a \log(am))$ for each decision node. To decide whether a candidate preserves $k$-anonymity, $k\text{ADET}$ needs to traverse all nonempty spans related to the candidate node, and decide for every one of them if splitting it will result in a new span that compromises $k$-anonymity. Note that because spans are, themselves, equivalence classes (from the attacker point of view), no overlap can exist between the learning examples associated with spans. Therefore, the overhead of splitting the learning examples of the spans associated with a node into new spans and deciding whether each of these has more than $k$ examples is bounded by the number of learning examples in the spans - which cannot exceed $m$. In the worst case, this check should be made for every candidate in the node, amounting to $O(am)$ for each decision node. Performing a split, once one that maintains anonymity is chosen, has time complexity $O(m)$. Overall, the overhead per decision node is
bounded by \( O(am + a \log(am)) \). Since the number of decision nodes is bounded by \( m \), the complexity of \( k\text{ADET} \) is \( O(a \cdot m^2 + am \cdot \log(am)) \).

### 3.2.4 From ID3 to C4.5

C4.5 was introduced by Quinlan [68] 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 extensions, other than the change of the scoring function – which has no effect on privacy – require careful analysis when used to extend \( k\text{ADET} \).

**Pruning.** C4.5 uses error-based pruning in two ways: discarding subtrees and replacing them with leaves, and replacing subtrees with one of their branches. Using the first method is safe – undoing a split unifies equivalence classes, and may unify spans, meaning that the population of a span can only increase. The second method, however, may cause a \( k \)-anonymous tree to become non-\( k \)-anonymous, as it induces different spans with different populations. Therefore we avoid this technique in our implementation.

**Continuous attributes in C4.5.** 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. We implemented the same approach, adding the constraint that a split point should not cause a breach of \( k \)-anonymity.

**Missing values.** C4.5 supports missing values in the training and the test data. The implications of missing values on \( k \)-anonymity are discussed in the next section.

### 3.2.5 Handling Missing Values

In this section we discuss how \( k \)-anonymity can be ensured when there are missing values in the learning examples. It is not uncommon for data to contain missing values in most or even all tuples. A privacy model which does not define the implications of missing values on privacy would not be applicable to this kind of data. The analysis that follows describes how the calculation of spans should be altered to support missing values.

Two cases are considered. In the first case, missing values are unknown both to the attacker and to the data owner. This is the case, for example, when missing values are inherent to the problem domain. In the second case, missing values are unknown to the data owner but may be known to the attacker if they pertain to public attributes. For example, an individual may have chosen not to disclose to the data owner whether or not she is married, yet the attacker may have obtained this information from some other source. To avoid relying on security by obscurity, we assume that the attacker also knows which
values the data owner does not have. Although the two cases are quite distinct, their effect on anonymity is similar. This is because in both cases the induced tree does not rely on values not known to the data owner. Even if the attacker knows public attribute values that are unknown to the data owner, he cannot use this to his advantage; the tree provides no information that can be used in conjunction with this knowledge to narrow down possible private values for a smaller population. Therefore, the analysis that follows applies to both cases. Note that other cases are also possible, but are not considered here. For example, it is possible that both attacker and data owner have missing values, but on different individuals or on different attributes.

The effect of missing values on model privacy depends on the way they have been treated during the model’s induction. If, as is customary in many anonymity preserving algorithms [11, 37, 44], tuples with missing values are deleted from the database before the algorithm is executed, then the missing values will have no effect on privacy. The suppression of tuples with missing values can be formalized by modelling it as splitting the root node on a binary public attribute that marks the existence of missing values. All tuples with missing values are routed to a distinct leaf in which no private information is provided. Since no private values are provided in this leaf, it comprises a span with a single equivalence class, thereby maintaining anonymity regardless of the number of suppressed tuples. Suppression of tuples with missing values, however, might strongly bias the input and might also produce a poorer outcome as a result, especially when missing values are abundant. Another possible approach would be to treat a missing value as a value in its own right, and then analyze the privacy of the model using the methods described earlier for regular values.

Traditionally, non-privacy preserving data mining algorithms have taken a third approach to missing values. C4.5 is a classic example for this approach: when a leaf is split according to a certain attribute, C4.5 divides the examples among the new leaves according to the value they take on this attribute. To deal with missing values, C4.5 attaches a weight to every example. An example which has a value for the attribute is moved to the corresponding leaf and retains its original weight. An example which has no value for the split attribute is moved to all leaves and has its weight divided between the new leaves, proportionally to the relative popularity of each leaf. When the joint distributions of attributes and class (the crosstables, as they are often called) are calculated, every example contributes its weight to the correct bin. If the crosstables refer to an attribute for which the example has no value, its weight is divided among the bins that match its class value.

When tuples with unknown values are split between new leaves, the private values associated with one individual may now be divided among several bins in different leaves. This is in contrast to tuples for which the values are known, and that will be routed to only one of the new leaves by the data owner and by a potential attacker. In other words, when missing values are present, the function of the decision tree is not from the domain to bins, but from the domain to sets of bins. As a consequence, tuples for which the value of the
chosen split attribute is missing, form a unique equivalence class. If the chosen attribute is public, these tuples can be identified by the attacker as a unique population associated with a unique span. Therefore, when checking for $k$-anonymity, the data owner should consider this span just as he considers the other spans, and avoid counting those tuples in any of the other spans associated with the new leaves.

<table>
<thead>
<tr>
<th>Name</th>
<th>Marital Status</th>
<th>Sports Car</th>
<th>Loan Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lisa</td>
<td>Unmarried</td>
<td>Yes</td>
<td>good</td>
</tr>
<tr>
<td>Philip</td>
<td>Unmarried</td>
<td>Yes</td>
<td>good</td>
</tr>
<tr>
<td>John</td>
<td>? (Married)</td>
<td>Yes</td>
<td>good</td>
</tr>
<tr>
<td>Ben</td>
<td>Married</td>
<td>No</td>
<td>bad</td>
</tr>
<tr>
<td>Laura</td>
<td>Married</td>
<td>No</td>
<td>bad</td>
</tr>
<tr>
<td>Robert</td>
<td>? (Unmarried)</td>
<td>Yes</td>
<td>bad</td>
</tr>
<tr>
<td>Anna</td>
<td>Unmarried</td>
<td>No</td>
<td>bad</td>
</tr>
</tbody>
</table>

Table 3.3: Data with missing values

For example, consider Table 3.3, in which the Marital Status attribute values are unknown for two individuals. We assume that Marital Status and Sports Car (denoting whether an individual owns a sports car) are both public attributes. Figure 3.4 is the decision tree resulting from a C4.5 induction, based on the learning examples in Table 3.3. Given $T_{ID}$, in the Sports Car = Yes node, the attacker knows for sure that Lisa’s and Philip’s tuples should be routed to the left-hand descendant node, since their Marital Status is Married. However, John and Robert may be routed to either of the descendant nodes, based on their unknown Marital Status attribute value. Thus the decision tree imposes three spans: one populated with Ben, Laura and Anna, comprised of the bins $\{b_1, b_2\}$, one populated with Lisa and Philip, comprised of the bins $\{b_3, b_4\}$, and one populated with John and Robert, comprised of the bins $\{b_5, b_6\}$. Therefore, this decision tree is 2-anonymous with respect to Table 3.3.
3.3 Evaluation

To conduct our experiments we implemented the algorithms using the Weka package [86]. We use as a benchmark the Adult database from the UC Irvine Machine Learning Repository [22], 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 (for comparison with other algorithms), there are 30,162 records for training and 15,060 records for testing (of which 24.5% are classified \( >50K \)). For the categorial attributes we use the same generalization hierarchies described in [37]. For the ID3 experiments we dropped the continuous attributes, because of ID3 limitations. All the experiments were performed on a 3.0GHz Pentium IV processor with 512MB memory.

The anonymized decision tree 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 anonymous ID3, and less than 12 seconds for anonymous C4.5.

3.3.1 Accuracy vs. Anonymity Tradeoffs

Our first goal is to assess the tradeoff between classification accuracy and the privacy constraint. In what follows, we refer to the statistical significance of the presented results. The statistical significance was established by the McNemar test [20].

![Classification error vs. \( k \) parameter for ID3](image)

Figure 3.5: Classification error vs. \( k \) parameter for ID3

Figure 3.5 shows the classification error of the anonymous ID3 for various \( k \) parameters, compared to the classification error for ID3. The ID3 algorithm (and its \( k \)-anonymous variation) can generate leaves with null classification, when no learning examples reach those leaves. For example, when the algorithm splits a decision tree node using a categorial attributes with several values, a leaf is created for each of the values. If one of the values
does not appear in any of the learning examples, then the corresponding leaf will have classification null. When classifying unseen examples, tuples which are routed to such leaves will be denoted by the algorithm as “unclassified.” Figure 3.5 shows two lines for the $k$-anonymous ID3: one in which unclassified test examples are treated as correct classifications (optimistic) and one in which they are treated as incorrect classifications (pessimistic). As $k$ rises, the induced decision trees are smaller and use more generalizations, so unclassified test examples become rare and the lines converge. For ID3 only the optimistic baseline is shown. The classification error for the pessimistic case is 22.278%, out of the scope of the graph.

In spite of the anonymity constraint, the $k$-anonymous classifier maintains good accuracy for most values of $k$. In fact, for $k = 10, 25$ the resulting decision tree significantly outperforms the ID3 baseline. For $k = 2$ and $k = 750$ the ID3 baseline is significantly better. For other values of $k$ there is no significant advantage to either of the algorithms. These results may seem surprising, since $kADET$ has privacy constraints limiting its ability to induce good decision trees, while regular ID3 faces no such constraints. However, the privacy constraints have a pruning effect on the decision trees induced by $kADET$, resulting in more general decision trees which can operate better on the test examples. ID3, on the other hand, may overfit the learning examples, and in some cases has worse accuracy on the testing examples.

The results in the graph highlight the impact of the heuristic nature of ID3. For example, at $k = 750$ there is a local deterioration in $kADET$ accuracy when the root node is split using the Relationship attribute. At $k = 1000$ this attribute is discarded because of an anonymity breach, and the Marital Status attribute is chosen instead, yielding better classification. The 2-anonymous decision tree also performs much worse than the 5-anonymous decision tree and the baseline. We relate this anomaly to the heuristic nature of ID3 as well. For example, we checked a test example with the attribute values: (Workclass = Self-emp-inc, education = HS-grad, marital status = Married-civ-spouse, occupation = sales, relationship = Husband, race = White, sex = Male, native-country = Mexico, income level: $\leq 50K$).

In the decision tree resulting from regular ID3, the algorithm routes the tuple along a path that checks in the following order: relationship, education, occupation, and finally native country. The tuple is classified by the tree as $\leq 50K$. In the 2-anonymous decision tree, the first three nodes in the path are the same, but native country was dumped because it compromised 2-anonymity, and the next checks are according to workclass and race instead. The tuple is classified by the tree as $> 50K$, which is an error. In the 5-anonymous decision tree, the first three nodes in the path are also the same, but workclass was dumped because it compromised 5-anonymity, and the next check is according to race instead. The tuple is classified by the tree as $\leq 50K$. This example demonstrates how the greedy nature of $kADET$ can lead to sub-optimal results, as in the case of $k = 2$.

We compared our results with those obtained using the top-down specialization (TDS) algorithm presented in [37], the goal of which is to produce anonymized data useful for
classification problems. The algorithm starts with the topmost generalization level and iteratively chooses attributes to specialize, using 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 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 3.5. In the case of TDS, it is enough that a single outlying tuple will have a unique attribute value to warrant generalization of this attribute over all the tuples in the data set. As a result, even for \( k = 2 \), TDS applies extensive generalization (both on learning and testing examples) – for example, workclass, relationship and native country were totally suppressed. Because of the high level of generalization, there were no unclassified test examples in the TDS experiments. However, this level of generalization reduces the accuracy of the resulting decision tree. In contrast to the TDS algorithm, our algorithm can apply different generalizations on different groups of tuples, so tuples with unique attribute values require attribute generalization only in the subtree in which they reside. Thanks to this advantage, \( kADET \) achieved an average reduction of 0.6% in classification error with respect to TDS. Regardless of how unclassified examples in \( kADET \) are treated, \( kADET \) significantly outperforms TDS for \( k = 25, 50, 75, 100, 150, 200, 250 \); TDS outperforms \( kADET \) when \( k = 2 \); and for other values of \( k \) neither of the algorithms has significant advantage over the other.

Finally, we compared \( kADET \) results with those obtained using the Mondrian InfoGain algorithm presented in [52]. Like TDS, the Mondrian algorithm is a top down algorithm, starting from the highest generalization level, and partitioning the data repeatedly to specialize attributes. However, unlike TDS, the Mondrian algorithm is a multi-dimensional recoding algorithm, meaning that it can generalize an attribute differently in each tuple, depending on other attribute values. This is enabled by allowing specialization of different attributes in each data partition. There are several variants of the Mondrian algorithm, depending on the metric that guides which attribute should be used for the next partitioning. Specifically, Mondrian InfoGain picks in every stage the partition with the highest information gain. In fact, the approaches taken by Mondrian InfoGain and \( kADET \) are very similar. The main difference between the algorithms is that the output of Mondrian InfoGain is a \( k \)-anonymous table, while the output of \( kADET \) is a \( k \)-anonymous decision tree. In addition, since table anonymization deals only with public attribute values, Mondrian InfoGain handles only public attribute values as well. Specifically, the Mondrian algorithm assumes that the set of predictor attributes, and especially the target attribute, are public. \( kADET \), on the other hand, also takes into account the impact of private attribute values appearing along the path from the root to the leaves, and allows the target attribute to be either public or private. To compare the results obtained by Mondrian InfoGain with those obtained by \( kADET \), we followed the approach outlined in [52]: anonymization is determined from the learning examples, and then applied on both learning and test examples. Regular ID3 is used to gen-
erate a decision tree from the anonymized learning examples, after which it is used to classify the anonymized test examples. As in ID3 and \textit{kADET}, some test examples are tagged as unclassified. Figure 3.5 shows the results obtained by applying Mondrian InfoGain on the Adult dataset, taking the optimistic approach of treating unclassified examples as correct classifications. The results of the two algorithms differ because of the way the class attribute \textit{income level} is treated – while \textit{kADET} treats this attribute as private, Mondrian InfoGain always treats the attributes as public. The difference is not statistically significant, with the exception of \(k = 2\). For \(k = 2\), the number of unclassified results in \textit{kADET} is large and the algorithm’s accuracy is reduced when unclassified examples are treated as misses.

![Graph showing classification error vs. k parameter for C4.5](image)

**Figure 3.6**: Classification error vs. \(k\) parameter for C4.5

Figure 3.6 shows the C4.5 variants of the algorithms, using all 14 attributes of the Adult dataset. The large size of the quasi-identifier affects the accuracy of the TDS generalization, and \textit{kADET} significantly outperforms TDS for all values of \(k\), reducing the classification error by an average of 3\% with respect to TDS. The results of TDS demonstrate again how the heuristic nature of the greedy algorithms affects the accuracy of the resulting classifier. For example, the different generalizations taken for \(k = 25, 50, 75\) result in different decision trees. The generalization obtained for \(k = 50\) fits better the patterns in the data, resulting in much better accuracy for this case.

For \(k = 25\) \textit{kADET} significantly outperforms C4.5, and obtains better (though not statistically significant) results for most lower values of \(k\). In those cases, the pruning effect of the privacy constraint is more effective in generalizing the decision tree than the pruning done by C4.5 using its default settings. For values of \(k = 250\) or above, the cost of anonymity is much higher and the results of \textit{kADET} are significantly worse than those of the original C4.5.

For evaluating Mondrian InfoGain we used J4.8, which is a Java version of C4.5 available in the Weka package [86]. As in the case of ID3, the treatment of the target attribute as public affects the obtained generalizations. The difference in the results can be attributed also to the different metrics used in the algorithms – while the C4.5 variant of \textit{kADET} uses
solely gain ratio, Mondrian InfoGain uses information gain to generalize the data, and then J4.8 uses gain ratio to induce the tree. For \( k = 2, 5, 8, 10, 500 \), \( kADET \) obtains significantly better results than Mondrian InfoGain. For other values of \( k \) the difference in results is not statistically significant.

### 3.3.2 Model-based \( k \)-Anonymization

In Section 2.3 we discussed the concept of equivalence between data mining models and their table representation. Based on this concept, we can use data mining techniques to anonymize data. Each span defines an anonymization for a group of at least \( k \) tuples.

When the data owner knows in advance which technique will be used to mine the data, it is possible to anonymize the data using a matching technique. This kind of anonymization would be very similar to embedding the anonymization within the data mining process. However, when the algorithm for analysis is not known in advance, would a data-mining-based anonymization algorithm still be useful? To answer this question, we next assess the value of the anonymous decision tree as an anonymization technique for use with other classification algorithms.

To make this assessment, we measured the classification metric (originally proposed in [44]) for the induced decision trees. This metric was also used in [11] for optimizing anonymization for classification purposes. In our terminology, the classification metric assigns a penalty 1 to every tuple \( x \) that does not belong to the majority class of \( S(x) \), 

\[
CM = \sum_{S} \text{Minority}(S).
\]

We compare our results with the \( k \)-Optimal algorithm presented in [11], which searches the solution domain to find an optimal anonymization with respect to a given metric. We discarded the \( Relationship \) attribute, since it is not used in [11]. Note also that we do not make use of the \( Age \) attribute, which is used in [11]. This puts our algorithm at a disadvantage. [11] reports several CM values, depending on the partitioning imposed on the \( Age \) attribute and the limit on number of suppressions allowed (our algorithm makes no use of suppressions at all). We present in Table 3.4 the ranges of CM values reported in [11] alongside the CM results achieved by our algorithm. Our algorithm obtains similar (sometimes superior) CM results in a shorter runtime.

<table>
<thead>
<tr>
<th>( k )</th>
<th>( k )-Optimal</th>
<th>Anonymous-DT</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>5230-5280</td>
<td>5198</td>
</tr>
<tr>
<td>25</td>
<td>5280-5330</td>
<td>5273</td>
</tr>
<tr>
<td>50</td>
<td>5350-5410</td>
<td>5379</td>
</tr>
<tr>
<td>100</td>
<td>5460</td>
<td>5439</td>
</tr>
</tbody>
</table>

Table 3.4: Classification metric comparison

Successful competition with optimal single-dimension anonymizations using multi-dimensional...
anonymizations has already been discussed in [51]. However, our results give rise to an additional observation: Intuitively it may seem that using a specific data mining algorithm to generalize data would “over-fit” the anonymization scheme to the specific algorithm, decreasing the ability to successfully mine the data using other algorithms. However, the CM results presented above suggest that this kind of anonymization may be at least as useful as metric-driven (and algorithm oblivious) anonymization.

3.3.3 Missing Values

To evaluate the possible benefit of using missing values while maintaining $k$-anonymity, two data sets were tested, using the following methodology:

1. One decision tree was induced from the entire data set, including tuples with missing values.

2. A second decision tree was induced from the data set, after all tuples with missing values were removed.

3. The resulting decision trees were tested against the same training set, including cases with missing values.

**Adult Database**

There are 32,561 tuples in the full Adult data set, of which 2399 contain missing values, concentrated in three of the 14 attributes.

Figure 3.7 shows the results of running the $kADET$ algorithm on the data set with and without missing values. For comparison, C4.5 baselines for the data set with and without missing values are shown as well. The results show that using tuples with missing values may in some cases improve the accuracy of the resulting tree, but in other cases accuracy may also deteriorate. While tuples with missing values provide additional information to rely on when inducing the tree, they also introduce additional privacy risk. Since tuples with missing values belong to spans which include equivalence classes from several subtrees, those tuples are affected by splits done in several subtrees. If the number of tuples with missing values is large and the $k$-anonymity parameter is relatively small, this poses no problem. However, as $k$ grows larger, few splits may compromise the privacy of tuples with missing values, so other, less informative splits should be chosen instead.

**Credit Approval Data Set**

To give another example of how missing values support affects decision tree accuracy, we conducted further experiments using the Credit Approval data set, also available on the UC
Irvine Machine Learning Repository. The data set contains 690 instances, 307 of which are positive examples and 383 of which are negative examples. In addition to the class attribute, there are 6 continuous attributes and 9 categorial attributes. 37 cases (5%) have one or more missing values. Since no test set is available for the Credit Approval data set, 10-fold cross validation was used for measuring accuracy instead.

Figure 3.8 compares the accuracy obtained with and without missing values with respect to the $k$-anonymity parameter. Although the size of the data set does not allow the statistical significance of the results to be determined, the outcome hints that, for most values of $k$, instances with missing values may prove useful without compromising privacy constraints.

### 3.4 Enhancements to $k$-Anonymity

$k$-anonymity makes no restriction regarding private attribute values. It only imposes constraints on the sizes of the groups of individuals to whom the private attribute values can be linked. Therefore, it is possible that a $k$-anonymous model would allow a complete inference of the private attribute values. $k$-anonymity is also vulnerable to background knowledge attacks, in which an attacker who has some kind of prior knowledge about private attribute values can use it to compromise the anonymity of individuals. These weaknesses of $k$-anonymity were studied in several works, and different modifications to $k$-anonymity were proposed to overcome this problem, e.g., [56, 87, 19, 54, 88].

In this section, our goal is to assess the extent to which individuals could be prone to inference attacks due to a release of a decision tree, in the context of the Adult data set experiments, and discuss how the $kADET$ algorithm can be extended using one of the proposed new privacy models to thwart such attacks.

We start by assessing the number of individuals in the adult data set for whom an attacker
may infer the class attribute value with full certainty. We refer to such individuals as \textit{exposed}. This number can be derived by looking at spans which are populated by tuples sharing a single (private) class attribute value, and counting the number of those tuples. Figure 3.9 shows the percentage of exposed individuals in the decision trees induced by \textit{kADET} for different values of \( k \). For the \( k \)-anonymous ID3, this number drops to zero only for values of \( k \) beyond 750, and even then the attacker may still be able to infer private attribute values with high probability. The inference problem is less acute in the case of the \( k \)-anonymous C4.5, because of pruning. The number of exposed individuals drops to zero at \( k = 75 \), and is very low (below 0.5\%) even for smaller values of \( k \).

![Figure 3.9: Exposed individuals in Adult data set](image)

The \( \ell \)-diversity model [56] suggests solving the inference problem by requiring a certain level of diversity in class values for every group of identifiable tuples. For example, \textit{entropy \( \ell \)-diversity} is maintained when the entropy of the class values for every such group exceeds a threshold value \( \log(\ell) \).

In general, applying the \( \ell \)-diversity constraint on induced decision trees requires determining the exact private attribute values linkable to every span. This extension of the model is out of the scope of this work. However, for the decision trees induced in the Adult data set experiments, determining the linked private attribute values is easier, and can provide some insights into the enforcement of \( \ell \)-diversity on decision tree induction. The only private attribute value in the experiments is the class attribute \textit{income level}; in every stage of the \textit{kADET} algorithm, every leaf constitutes a span with two bins: one bin per class attribute value. The population of the span can be linked with the distribution of private attribute values given by the populations of the two bins. The \( \ell \)-diversity constraint can be applied on this distribution to determine whether privacy is maintained. For example, if a leaf contains five individuals with \textit{income level} \( \leq 50K \) and two individuals with \textit{income level} \( > 50K \), then seven individuals are linked with a span consisting of two bins, and the entropy value of this distribution is \( (2/7) \times \log_2(7/2) + (5/7) \times \log_2(7/5) = 0.863 \). If this value exceeds \( \log(\ell) \), then entropy \( \ell \)-diversity is maintained for these individuals.

We altered our algorithms by replacing the \textit{Breach()} function with one that checks the entropy \( \ell \)-diversity constraint, ruling out splits that violate this constraint. Note that the parameters for \( k \)-anonymity and \( \ell \)-diversity are not comparable, e.g., comparing 10-
anonymity to 10-diversity is meaningless. 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. However, for \( \ell < 2 \), entropy \( \ell \)-diversity limits the attacker’s confidence in inference attacks. The confidence limit is the maximal probability of any private value for any individual. The data owner can control the confidence limit by manipulating the \( \ell \) parameter. For example, to deny the attacker the ability to infer a class value with confidence greater than 85\%, entropy higher than \( 0.85 \times \log_2(1/0.85) + 0.15 \times \log_2(1/0.15) = 0.61 \) should be maintained. This amounts to applying entropy \( \ell \)-diversity with \( \ell = 1.526 \) (\( \log_2(1.526) = 0.61 \)).

Following this discussion, Figures 3.10 and 3.11 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 greater than 85\%. Therefore, a confidence limit of 85\% or lower prohibits the induction of a useful decision tree. The additional numeric attributes available to the C4.5 algorithm in the Adult data set allow the boundary to be stretched to a lower confidence threshold.

### 3.5 Conclusions

In this chapter we presented decision tree induction algorithms which guarantee \( k \)-anonymous output. A data owner who wishes to publish a decision tree while maintaining customer anonymity can use this technique to induce decision trees which are more accurate than those acquired by anonymizing the data first and inducing the decision tree later. This way, anonymization is done in a manner that interferes as little as possible with the tree induction process.

Another problem addressed in this chapter is the effect of missing values on \( k \)-anonymity.
To the best of our knowledge, this problem was not studied before in the context of anonymization and privacy. According to our experiments, the utility of handling missing values may change in accordance with the specific data set at hand and the anonymity parameter. Nevertheless, the inability to handle missing values might render data sets with abundant missing values unusable.

The $k$-anonymity model has attained wide acceptance as a privacy model. Much of this can probably be attributed to its simplicity. Non-expert data owners can easily understand the privacy requirement and evaluate whether their data conform to it. In our work we have shown how this simple and intuitive notion of privacy can be brought into the world of privacy preserving data mining. However, the simplicity of the model comes at a cost. In some cases, a data set which conforms to a $k$-anonymity constraint might still allow inference of private information for a group of individuals sharing the same private data. Overcoming this weakness requires the introduction of more elaborate restrictions, such as those provided in the $\ell$-diversity model, thereby creating a tradeoff between the strength and the simplicity of the privacy model. One of the major challenges in privacy preserving data mining these days is the definition of rigorous privacy models that will fit real world privacy needs of real world applications, while maintaining the elegance, simplicity and ease of use that characterize the $k$-anonymity model.
Chapter 4

Data Mining with Differential Privacy
4.1 Introduction

In this chapter we consider data mining within the framework of differential privacy [25, 27]. Basically, differential privacy requires that computations be insensitive to changes in any particular individual’s record. Once an individual is certain that his or her data will remain private, being opted in or out of the database should make little difference. For the data miner, however, all these individual records in aggregate are very valuable.

Differential privacy has several advantages over prior approaches to privacy: first, it relies on a mathematical definition, making it possible to rigorously prove whether a mechanism conforms to differential privacy, and to deduce which calculations can or cannot be made in this framework. Second, differential privacy does not make any assumptions on an adversary’s background knowledge or computational power. This independence frees data providers who share data from concerns about past or future data releases and is adequate given the abundance of personal information shared on social networks and public Web sites. Thus, differential privacy is very well suited to a world in which companies own immense computing resources and can store and access huge amounts of personal information. Third, differential privacy maintains composability [38], meaning that differential privacy guarantees hold also when two independent differentially private data releases are combined by an adversary. This property frees the data provider from concerns about external or future data releases and their privacy implications on a current data release. Moreover, data providers that let multiple parties access their database can evaluate and limit any privacy risks that might arise from collusion between adversarial parties or due to repetitive access by the same party. Finally, one of the first steps toward practical implementation of differential privacy was the introduction of the Privacy Integrated Queries platform (PINQ [58]), a programmable privacy preserving layer between the data analyst and the data provider. Such a platform allows data analysts safe access to the data through the provided application interface without needing to worry about enforcing the privacy constraints, and without requiring expert knowledge in the privacy domain.

Most research on differential privacy so far has focused on theoretical properties of the model, providing feasibility and infeasibility results [49, 30, 14]. However, real-world application of differential privacy to data mining poses new challenges. One limitation of working with differential privacy is that every query to the privacy mechanism incurs some loss of privacy, controlled by a parameter $\epsilon$. Smaller values of $\epsilon$ mean that privacy is better preserved, but they also require masking the output with higher magnitudes of noise, diminishing accuracy. For example, in the PINQ platform, privacy loss is accounted for through the concept of privacy budgeting. Each interactive query to the PINQ mechanism incurs a privacy cost ($\epsilon$), and when the analyst exhausts a predetermined privacy budget, access to the database is blocked and no additional queries are allowed. Unfortunately, this means that the analyst must carefully manage how the privacy budget is spent, to avoid being cut off from the
dataset before the analysis is complete.

For many calculations, the magnitude of noise required by differential privacy depends only on the value of $\epsilon$ and the characteristics of the calculation and is independent of the data [27]. Therefore, from a theoretical standpoint, the privacy budget problem can be simply eradicated by working with a larger dataset. For example, adding noise of magnitude $\pm 100$ to a count query will have less (relative) effect on accuracy when executed on a dataset with 5000 instances than when executed on a dataset with 500 instances. However, in practice, this course of action may not always be feasible. First of all, gathering and accessing larger amounts of information may incur higher costs. Moreover, even when an extremely large corpus is accessible, the studied phenomenon (e.g., patients who suffer from a certain medical condition) may apply only to a small subset of the population, thereby limiting the number of records available. Therefore, practical use of differential privacy requires efficient use of the privacy budget, a subject which so far has not attracted much attention from the research community.

To address this problem, we focus in the next chapter on decision tree induction as a sample data mining application. As noted in [58], different methods for measuring a quantity with a differential privacy mechanism can provide results of varying quality. We demonstrate this notion both from theoretical and practical perspectives. Specifically, we study different approaches to decision tree induction over a differential privacy querying mechanism (such as PINQ), and their effect on privacy costs and the quality of the resulting decision trees. We show that different methods for decision tree induction incur different privacy costs, thereby motivating algorithm redesign with privacy costs in mind, and extension of the privacy-preserving data access layer to support more powerful calculation primitives.

### 4.2 Background

#### 4.2.1 Differential Privacy

Differential privacy [25, 26] is a recent privacy definition that guarantees the outcome of a calculation to be insensitive to any particular record in the data set.

**Definition 8** We say a randomized computation $M$ provides $\epsilon$-differential privacy if for any datasets $A$ and $B$ with symmetric difference $A \Delta B = 1$ ($A$ and $B$ are treated as multisets), and any set of possible outcomes $S \subseteq \text{Range}(M)$,

$$\Pr[M(A) \in S] \leq \Pr[M(B) \in S] \times e^\epsilon .$$

The parameter $\epsilon$ allows us to control the level of privacy. Lower values of $\epsilon$ mean stronger privacy, as they limit further the influence of a record on the outcome of a calculation.
The values typically considered for $\epsilon$ are smaller than 1 [26], e.g., 0.01 or 0.1 (for small values we have $\epsilon' \approx 1 + \epsilon$). The definition of differential privacy maintains a \textit{composability} property [58]: when consecutive queries are executed and each maintains differential privacy, their $\epsilon$ parameters can be accumulated to provide a differential privacy bound over all the queries. For example, if we perform on the dataset a computation $M_1$ that maintains $\epsilon$-differential privacy, followed by another computation $M_2$ that also maintains $\epsilon$-differential privacy, then the sequential application of the computations will maintain $2\epsilon$-differential privacy. Therefore, the $\epsilon$ parameter can be treated as a privacy cost incurred when executing the query. These costs add up as more queries are executed, until they reach an allotted bound set by the data provider (referred to as the privacy budget), at which point further access to the database will be blocked. This property enables the data provider to define a privacy budget $\epsilon'$ that would limit the accumulated privacy loss over all data accesses, while giving the data miner the freedom to choose how to spend this budget. For example, the data owner can require that an algorithm that executes on the dataset must maintain 0.5-differential privacy. The data miner can then choose to send to the dataset one query that preserves 0.5-differential privacy, five queries that preserve 0.1-differential privacy, or any other combination that sums to $\epsilon' = 0.5$. The composition property also provides some protection from collusion: collusion between adversaries will not lead to a direct breach in privacy, but rather cause it to degrade gracefully as more adversaries collude, and the data provider can also bound the overall privacy budget (over all data consumers).

Typically, differential privacy is achieved by adding noise to the outcome of a query. One way to do so is by calibrating the magnitude of noise required to obtain $\epsilon$-differential privacy according to the \textit{sensitivity} of a function [27]. The sensitivity of a real-valued function expresses the maximal possible change in its value due to the addition or removal of a single record:

\textbf{Definition 9} Given a function $f : D \rightarrow \mathbb{R}^d$ over an arbitrary domain $D$, the sensitivity of $f$ is

$$S(f) = \max_{A,B \text{ where } A \Delta B = 1} \| f(A) - f(B) \|_1 .$$

Given the sensitivity of a function $f$, the addition of noise drawn from a calibrated Laplace distribution maintains $\epsilon$-differential privacy [27]:

\textbf{Theorem 1} Given a function $f : D \rightarrow \mathbb{R}^d$ over an arbitrary domain $D$, the computation

$$M(X) = f(X) + (\text{Laplace}(S(f)/\epsilon))^d$$

provides $\epsilon$-differential privacy.
4.2.2 Noisy Counts

One of the basic functionalities studied in the scope of differential privacy is that of a noisy count [27], and it is also one of the core functionalities supported by PINQ. Let $S$ be a set of elements for which we want to evaluate the function $f(S) = |S|$. By calculating the exact result and then perturbing it with noise $N \sim \text{Laplace}(0, \frac{1}{\epsilon})$, we obtain the result $\text{NoisyCount}(S) = |S| + N$, which maintains $\epsilon$-differential privacy. Noisy counts provide accurate results with high probability, since in Laplace distribution noise levels exceeding $t/\epsilon$ are exponentially small in $t$. Note that the added noise depends in this case only on the privacy parameter $\epsilon$. Therefore, the larger the set $S$, the smaller the relative error introduced by the noise. Blum, Dwork, McSherry and Nissim [13] have shown that noisy counts can be used as building blocks for implementing private versions of a variety of algorithms, including principal component analysis, $k$-means clustering, decision tree induction, the perceptron algorithm, and in general all algorithms in the statistical queries learning model.

4.2.3 The Exponential Mechanism

Another approach to obtaining differential privacy is through the exponential mechanism [60]. This mechanism was originally introduced to demonstrate how differential privacy can support truthful mechanisms in game theory. In later works, it was adapted to construct private PAC (Probably Approximately Correct [80]) learners [49], to generate useful synthetic datasets while conforming to differential privacy [14] and to privately evaluate calculations such as average and median [58]. Intuitively, the exponential mechanism is an algorithm that picks at random one outcome from a set of possible outcomes. It is provided a query function $q$ that grades the possible outcomes. The probability distribution over the outcomes is induced from the query function, which expresses how desirable each outcome is – the higher the value of the query function, the higher the probability that this outcome will be picked. On the other hand, the probability distribution is calibrated with the sensitivity of the query function, such that adding or removing a single record in the input will change the probability of any outcome by a factor of at most $\exp(\epsilon)$, thereby maintaining differential privacy.

**Theorem 2 (Exponential Mechanism [60])** Given a domain $D$, a set of outcomes $\mathcal{R}$, a base measure $\mu$, and a query function $q : (D^n \times \mathcal{R}) \to \mathbb{R}$ which grades the outcomes for each instance of the domain, define:

$$\epsilon^*_{q} := \text{Choose } r \text{ with probability proportional to } \exp(eq(d, r) \times \mu(r)).$$
Then \( \epsilon \Delta q \) gives \((\epsilon \Delta q)\)-differential privacy\(^1\), where \( \Delta q \) is the largest possible difference in the query function when applied to two datasets with symmetric difference 1, for all \( r \in \mathcal{R} \).

The base measure \( \mu \) accounts for preliminary assumptions on \( \mathcal{R} \). In this work we will assume a uniform base measure and will always assign \( \mu(r) = 1 \). We proceed to show a theoretical (and impractical) application of the exponential mechanism to learning decision trees using the private PAC learner introduced in [49].

### Learning Decision Trees with the Exponential Mechanism

Kasiviswanathan, Lee, Nissim, Raskhodnikova and Smith have shown [49] that the exponential mechanism can be used to implement PAC (Probably Approximately Correct [80]) learning in a private manner. In private PAC learning we assume a domain \( D \), and a set of concepts \( C \) that can be applied to points in \( D \). We are provided a set of training examples \((x, \text{class}(x))\), drawn from a given distribution \( \mathcal{D} \), where \( \text{class()} \) is a concept from \( C \). We wish to learn a matching concept from a domain of concepts \( \mathcal{H} \), which may or may not coincide with the original set of concept classes \( C \). Given learning parameters \( \alpha, \beta \), and a differential privacy parameter \( \epsilon \), a private PAC learning algorithm \( A \) outputs a hypothesis \( h \in \mathcal{H} \) satisfying

\[
\Pr[\text{error}(h) \leq \alpha] \geq 1 - \beta
\]

where \( \text{error}(h) = \Pr_{x \sim \mathcal{D}}[h(x) \neq \text{class}(x)] \). In this approach, a data miner with full access to the raw data can perform private PAC learning using a private version of Occam’s razor, as described in Algorithm 4. The algorithm defines a query function \( q \) that grades the hypotheses in \( \mathcal{H} \), and applies the exponential mechanism to choose one of them.

**Algorithm 4** Private version of Occam’s Razor [49]

1. **Input:** \( \alpha, \beta \) – learner parameters, \( \epsilon \) – privacy parameter, \( q : D^n \times \mathcal{H}_d \rightarrow \mathbb{R} \) – a query function
2. **procedure** PRIVATE OCCAM’S RAZOR()
   3. Obtain a dataset \( S \) containing \( n = O((\ln |\mathcal{H}_d| + \ln \frac{1}{\beta}) \cdot \max\{\frac{1}{\alpha \epsilon}, \frac{1}{\alpha^2}\}) \) samples from the domain \( D \) and their labels \( \text{class}(s_i) \) for any \( s_i \in S \).
   4. \( \forall h \in \mathcal{H}_d \) define \( q(S, h) = -|\{i : s_i \in S \text{ is misclassified by } h, \text{i.e., } \text{class}(s_i) \neq h(s_i)\}| \)
   5. Sample a random hypothesis from \( \mathcal{H}_d \), where each hypothesis \( h \) has probability proportional to \( \exp(\epsilon q(S, h)) \).
3. **end procedure**

While private PAC learning is a powerful tool for reasoning about what can be learned privately, the private version of Occam’s razor is not necessarily efficient, and therefore may

---

\(^1\)Note that the original definition required an additional multiplicative factor of 2. This factor is redundant here due to the difference in the privacy definition with respect to measuring the distance between datasets \( A \) and \( B \). The same proof from [60] applies also to the theorem used herein.
not be practical. For example, assume a domain \( D \) with attributes \( A = \{A_1, \ldots, A_n\} \), where each attribute has up to \( t \) distinct values, and with a class attribute \( C \), with \( |C| \) distinct values. Let \( H_k \) be the hypotheses space consisting of trees of depth \( k \), for which the query function \( q \) should be computed. We can derive \( |H_k| \) in a similar way to that shown in [66] for binary trees:

\[
|H_k| = \text{Number of decision trees of depth } k \\
|H_0| = |C| \\
|H_{k+1}| = \text{choices of root attribute} \times (\text{possible subtrees})^t \\
\downarrow \\
|H_k| = n^{t^{k-1}} \times |C|^t
\]

For nontrivial values of \( t, |C| \) and \( n \), straightforward evaluation of the query function \( q \) is impractical even for a limited domain of trees of depth 2.

### 4.2.4 PINQ

PINQ [58] is a proposed architecture for data analysis with differential privacy. It presents a wrapper to C#’s LINQ language for database access, and this wrapper enforces differential privacy. A data provider can allocate a privacy budget (parameter \( \epsilon \)) for each user of the interface. The data miner can use this interface to execute over the database aggregate queries such as count (\( \text{NoisyCount} \)), sum (\( \text{NoisySum} \)) and average (\( \text{NoisyAvg} \)), and the wrapper uses Laplace noise and the exponential mechanism to enforce differential privacy.

Another operator presented in PINQ is \textbf{Partition}. When queries are executed on disjoint datasets, the privacy costs do not add up, because each query pertains to a different set of records. This property was dubbed \textit{parallel composition} [58]. The \textbf{Partition} operator takes advantage of parallel composition: it divides the dataset into multiple disjoint sets according to a user-defined function, thereby signaling to the system that the privacy costs for queries performed on the disjoint sets should be summed separately. Consequently, the data miner can utilize the privacy budget more efficiently.

Note that a data miner wishing to develop a data mining algorithm using the privacy preserving interface should plan ahead the number of queries to be executed and the value of \( \epsilon \) to request for each. Careless assignment of privacy costs to queries could lead to premature exhaustion of the privacy budget set by the data provider, thereby blocking access to the database half-way through the data mining process.
4.3 Approaches to Differentially Private Computation

There are several ways in which differential privacy could be applied to computations in general and to data mining in particular, depending on the way the user (data miner) can access the raw data.

![Figure 4.1: Data access with differential privacy: (a) the data miner has direct access to data; (b) the data miner can access data through a data access mechanism (DP) that enforces differential privacy; (c) the data miner works on an offline synthetic dataset, which was generated by an algorithm (DP) that preserves differential privacy.](image)

In the first approach (Figure 4.1(a)), the data miner is trusted with full access to the raw data, and there are no limitations on the operations the miner can carry out on the database. In this approach, the data miner is responsible for ensuring that the outcome conforms to differential privacy. For example, Chaudhuri and Monteleoni [17] proposed and compared two differentially-private algorithms for logistic regression. The algorithms have access to the raw data, and ensure differential privacy by adding noise to the outcome of the logistic regression, or by solving logistic regression for a noisy version of the target function. McSherry and Mironov studied the application of differential privacy to collaborative recommendation systems [59] and demonstrated the feasibility of differential privacy guarantees without a significant loss in recommendation accuracy.

In a second approach, featured by McSherry in PINQ [58], access to the data store is mandated through a privacy-preserving layer (Figure 4.1(b)). This layer offers a limited set of query operators that can be carried out on the dataset. It monitors the requested queries and manages a privacy budget. When the privacy budget is exhausted, access to the data is
blocked. Therefore, the data miner is required to plan ahead how the privacy budget is to be spent, to avoid an abrupt denial of the access before the algorithm has finished executing. On the upside, the data miner does not need to worry about enforcing privacy requirements, as the privacy-preserving data access layer takes care of that. A more recent work by Roth and Roughgarden [69] presents the median mechanism. This mechanism allows to answer exponentially more queries than the existing approach, which perturbs each query result independently. The median mechanism distinguishes between “easy” and “hard” queries, depending on the accuracy of answering a query according to the majority of the databases consistent with answers to previous queries. Easy queries are answered by taking the median value over these databases, while hard queries are answered by querying the real database and adding noise.

Another work, by Jagannathan, Pillai, and Wright [45], considers inducing differentially private random decision trees. An ensemble of trees is generated randomly, and then noisy counts are used to determine the class values in the leaves, making it easy to implement the method over a data access layer such as PINQ.

Finally, a recent approach to differentially-private computations is to create synthetic datasets [14, 33, 30, 28] (Figure 4.1(c)) through a differentially-private processing of the raw data. Once the dataset is created, it can be published for external analysis, and the data miner gets full access to the synthetic dataset. From a data miner’s perspective, this approach could be the preferable one, as the synthetic dataset can be analyzed just as if it were the original. However, initial results suggest that the synthetic dataset should be specially crafted to suit the analysis to be performed on it, in order to ensure its usability. Several works studied the special case of creating differentially private contingency tables and histograms [53, 89, 9]. Machanavajjhala et al. [57] applied a variant of differential privacy to create synthetic datasets from U.S. Census Bureau data, with the goal of using them for statistical analysis of commuting patterns in mapping applications.

In the next chapter we focus on the second approach. In this approach, the data miner need not worry about enforcing the privacy requirements nor be an expert in the privacy domain. The access layer enforces differential privacy by adding carefully calibrated noise to each query. Depending on the calculated function, the magnitude of noise is chosen to mask the influence of any particular record on the outcome. This approach has two useful advantages in the context of data mining: it allows data providers to outsource data mining tasks without exposing the raw data, and it allows data providers to sell data access to third parties while limiting privacy risks. We show how decision tree induction can be carried out using a limited set of operations available through a privacy-preserving layer. We propose additional operators that can be provided by this layer to better support the decision tree induction process. A similar methodology can be applied also to other data mining algorithms: different methods for building the data mining model can be evaluated to determine their privacy cost, after which the data analyst can use the method that provides
the best balance between this cost and the quality of the resulting model.

However, while the data access layer ensures that privacy is maintained, the implementation choices made by the data miner are crucial to the accuracy of the resulting data mining model. In fact, a straightforward adaptation of data mining algorithms to work with the privacy preserving layer could lead to suboptimal performance. Each query introduces noise to the calculation and different functions may require different magnitudes of noise to maintain the differential privacy requirements set by the data provider. Poor implementation choices could introduce larger magnitudes of noise than necessary, leading to inaccurate results.
Chapter 5

Decision Tree Induction with Differential Privacy
5.1 Introduction

In this chapter, we investigate several approaches to inducing decision trees over a data access layer that enforces differential privacy (e.g., PINQ [58]). The data provider sets a privacy budget $\epsilon$, and the data miner tries to learn a decision tree while maintaining $\epsilon$-differential privacy.

The ID3 algorithm presented by Quinlan [67] applies greedy hill-climbing to construct a decision tree, given a set of training examples $T$. The examples are drawn from a domain with attributes $A = \{A_1, \ldots, A_d\}$ and a class attribute $C$, where each record pertains to a single individual. Throughout this chapter we will use the following notation: $T$ refers to a set of records, $|T|$ to the number of records, $r_A$ and $r_C$ refer to the values that record $r \in T$ takes on the attributes $A$ and $C$ respectively, $T_j^A = \{r \in T : r_A = j\}$, $\tau_j^A = |T_j^A|$, $\tau_c = |r \in T : r_C = c|$, and $\tau_{j,c} = |r \in T : r_A = j \land r_C = c|$. To refer to noisy counts, we use a similar notation but substitute $N$ for $\tau$. All the $\log()$ expressions are in base 2 ($\ln()$ is used for the natural base).

Starting from a root that holds the entire training set, ID3 chooses the attribute that maximizes the information gain with respect to the given samples in $T$, and splits the current node into several new nodes. The training set is then divided among the new nodes according to the value each record takes on the chosen attribute, and the algorithm is applied recursively on the new nodes. The information gain of an attribute $A \in A$ is given by

$$\text{InfoGain}(A, T) = H_C(T) - H_{C|A}(T) ,$$

where $H_C(T) = -\sum_{c \in C} \frac{\tau_c}{|T|} \log \frac{\tau_c}{|T|}$ is the entropy of the instances in $T$ with respect to the class attribute $C$, and $H_{C|A}(T) = \sum_{j \in A} \frac{\tau_j^A}{|T|} \cdot H_C(T_j^A)$ is the entropy obtained by splitting the records in $T$ according to the value they take on the attribute $A$.

The chapter is organized as follows. Section 5.2 revisits a prior theoretical algorithm for differentially-private tree decision induction [13]. Section 5.3 explores different practical approaches to decision tree induction with differential privacy. It investigates how different methods for inducing the decision tree affect the privacy costs of the algorithm, presents a differentially-private algorithm based on this analysis, and suggests how the data miner can tradeoff privacy and accuracy when inducing the decision tree. Section 5.4 describes our experiments, in which we evaluate and compare the performance of the different approaches.

5.2 Revisiting Decision Tree Induction with Sub-Linear Queries

One of the works predating differential privacy presented a noisy computational primitive called SuLQ (Sub-Linear Queries) [13], which was the ancestor of the noisy count presented
in Section 4.2.2. The authors showed how this primitive can be used to implement private versions of several algorithms, including ID3.

The SuLQ-based ID3 algorithm (Algorithm 5) relies on the observation that maximizing the information gain is equivalent to maximizing

\[ V(A) = -\tau \cdot H_{C|A}(T) = \sum_{j \in A} \tau_{j}^{A} \cdot H_{C}(T_{j}^{A}) . \]

Therefore, the algorithm approximates the information gain by evaluating the quantities \( \tau_{j}^{A} \) and \( \tau_{j,c}^{A} \) with NoisyCount to obtain \( N_{j}^{A} \) and \( N_{j,c}^{A} \), and then evaluating for each attribute \( A \in \mathcal{A} \):

\[ V_{A} = \sum_{j \in A} \sum_{c \in C} N_{j,c}^{A} \cdot \log \frac{N_{j,c}^{A}}{N_{j}^{A}} . \] (5.1)

When a new node is reached, a noisy count is used to evaluate the number of training instances in the node. If the count is below a certain threshold (more on that later), or there are no more attributes for splitting, the node is turned into a leaf. In that case, the algorithm then uses noisy counts to evaluate the distribution of the class values, and labels the leaf with the dominant class.

The following approximation guarantee for the SuLQ-based algorithm is adapted from [13]:

**Lemma 1** Given a gain bound \( \Delta g \) and a probability bound \( p \) as input to the SuLQ-based algorithm, the gain of the attribute \( \bar{A} \) chosen by the algorithm when splitting a node differs from the maximum gain by \( \Delta g \) with probability \( 1 - p \).

In Algorithm 5, we replaced the asymptotic notation used in the original SuLQ algorithm for ID3 with concrete terms. Specifically, we introduced the term \( \rho \) in lines 5 and 9 as a coefficient that ensures that the approximation guarantees hold. Note that our aim is not to find or present tight bounds for the terms used in the algorithm. However, these terms provide a rough estimate of the size of the dataset required to obtain approximation guarantees for the information gain. While the proof that follows is based on the original one [13], it accounts for the new concrete terms introduced into the algorithm, and amends two flaws (which some of the new terms were introduced to fix).

To prove bounds on the calculation of \( V_{A} \), we will use the following auxiliary claims:

**Claim 5** For \( b > 1 \):

\[ \max_{a \in [1,b]} \left| a \cdot \log \frac{a}{b} \right| = \frac{b \log(e)}{e} . \]

**Proof.** We prove by inspecting the first and second derivatives of \( a \cdot |\log \frac{a}{b}| \) in the given
Algorithm 5 SuLQ-based algorithm

1: **procedure** SuLQ-ID3($T, A, C, \epsilon, \Delta g, p$)
2: **Input:** $T$ – private dataset, $A = \{A_1, \ldots, A_d\}$ – a set of attributes, $C$ – class attribute, $\epsilon$ – differential privacy parameter, $\Delta g$ – desired bound on gain difference, $p$ - probability bound for $\Delta g$
3: $t = \max_{A \in A} |A|$
4: $\mu = \frac{1}{\epsilon} \ln \frac{d(t|C| + t) + 1}{p}$
5: $\rho = \frac{12d|C|}{\Delta g \ln 2}$
6: Find $\delta \in [0, \Delta g/2]$ such that $2\delta \log \frac{2|C|}{\Delta g} = \Delta g$.
7: $\gamma = \log(e) \cdot \frac{t \cdot |C|}{\delta}$
8: $N_T = \text{NoisyCount}_\epsilon(r \in T)$
9: if $A = \emptyset$ or $N_T < (p + 1)\mu \gamma$ then
10: $\forall c \in C : N_c = \text{NoisyCount}_\epsilon(r \in T : r_C = c)$
11: return a leaf labeled with $\arg \max_c(N_c)$
12: end if
13: for every attribute $A$ do
14: $\forall j : N_j^A = \text{NoisyCount}_\epsilon(r \in T : r_A = j)$
15: $\forall j, c : N_{j,c}^A = \text{NoisyCount}_\epsilon(r \in T : r_A = j \land r_C = c)$
16: $V_A = \sum_{j \in A} \sum_{c \in C} N_{j,c}^A \cdot \log \frac{N_{j,c}^A}{N_j^A}$ where terms for which $N_j^A$ or $N_{j,c}^A$ are smaller than $N_T/\gamma$ are skipped.
17: end for
18: $\bar{A} = \arg \max_{A \in A} V_A$
19: $\forall i \in \bar{A} : \text{Subtree}_i = \text{SuLQ-ID3}(T \land (r_A = i), A \setminus \bar{A}, C, \epsilon, \Delta g, p)$.
20: return a tree with a root node labeled $\bar{A}$ and edges labeled 1 to $|\bar{A}|$ each going to $\text{Subtree}_i$.
21: **end procedure**

range:

$$\left(a \left| \log \frac{a}{b} \right| \right)' = \left(a' \cdot \left| \log \frac{a}{b} \right| + a \cdot \left| \log \frac{a}{b} \right|' = \left| \log \frac{a}{b} \right| + a \cdot \frac{\log a}{\log \frac{a}{b}} \cdot \frac{1}{a \ln 2} = - \log \frac{a}{b} - \frac{1}{\ln 2} = 0 .$$

From that we get that the extremum is reached at $a = b/e$. The second derivative is negative, which means we reached a maximum point. Hence, $\max \left| a \cdot \log \frac{a}{b} \right| = b \cdot \log(e)/e$.  

**Claim 6** Given $r \in (0, 0.5]$, for any $x$ such that $|x| \leq r$:

$$\frac{x - 2r^2}{\ln(2)} \leq \log(1 + x) \leq \frac{x + 2r^2}{\ln(2)} .$$
Proof. Taylor series expansion for the function \( f(x) = \log(1 + x) \) provides:

\[
\log(1 + x) = \ln(1 + x) = \frac{1}{\ln 2} \left( x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \ldots \right) = \frac{1}{\ln 2} (x + R_1(x)),
\]

where \( R_1(x) \) is the remainder function. According to Taylor’s theorem, for \( x \in [-r, r] \) the remainder satisfies the inequality \( |R_1(x)| \leq M_1 \cdot \frac{x^2}{2} \), where \( M_1 \) is a bound such that \( |f''(x)| \leq M_1 \) for all \( x \in [-r, r] \). Taking \( M_1 = 4 \) as an upper bound on \( |f''(x)| = \frac{1}{(x+1)^2} \) in the range \( |x| \leq \frac{1}{2} \), the claim follows.

Following is the proof for Lemma 1:

Proof. In the NoisyCount operations, we use noise distributed \( X \sim \text{Laplace}(0, \frac{1}{\epsilon}) \). For this distribution we have:

\[
\Pr[|X| \leq \mu] = 1 - \exp(-\mu\epsilon) = 1 - \frac{p}{d(t|C| + t) + 1}.
\]

Since for each of the \( d \) attributes there are \( t|C| + t \) noisy count queries, and since there is an additional query for \( N_T \), we get that with probability \( 1 - p \) all the counts are within \( \mu \) from the actual counts. In what follows we assume this is the case for all the queries.

We need to show that \( V_A - V_A \leq \tau\Delta g \). We will do so by showing that for each of the components in the sum:

\[
\left| \tau_{j,c}^A \cdot \log \frac{\tau_{j,c}^A}{\tau_j^A} - N_{j,c}^A \cdot \log \frac{N_{j,c}^A}{N_j^A} \right| \leq \frac{\tau\Delta g}{t|C|}.
\]

We will consider three cases:

1. \( N_j^A < N_T/\gamma \).
2. \( N_{j,c}^A < N_T/\gamma \).
3. \( N_{j,c}^A \geq N_T/\gamma \) and \( N_j^A \geq N_T/\gamma \).

In cases 1 and 2, the algorithm will skip the terms in the calculation (line 16), so we only need to bound the expression \( \left| \tau_{j,c}^A \cdot \log \frac{\tau_{j,c}^A}{\tau_j^A} \right| \).

**Case 1:** \( N_j^A < N_T/\gamma \)

Since all counts are within \( \mu \) from the exact result, we have that:

\[
\tau_j^A \leq N_j^A + \mu < \frac{N_T}{\gamma} + \mu \leq \frac{\tau + \mu}{\gamma} + \mu \leq \frac{\tau + \mu\gamma + \mu}{\gamma} \leq 2\tau \leq \frac{2\tau \delta}{\log(e)t|C|}.
\]

The transition marked with asterisk (*) is due to \( N_T \geq (\rho + 1)\mu\gamma \geq \mu(\gamma + 2) \) (the condition in line 9), which ensures that \( \tau \geq \mu\gamma + \mu \).
Because only terms for which $\tau_{j,e}^A \geq 1$ contribute to $V_A$, and in addition $\tau_{j,e}^A \leq \tau_j^A$, we can apply Claim 5, and using also $\delta \leq 2\Delta g$ we obtain:

$$\max_{\tau_{j,e}^A} \left| \tau_{j,e}^A \cdot \log \frac{\tau_{j,e}^A}{\tau_j^A} \right| \leq \tau_j^A \log(e)/e \leq \frac{2\tau \delta}{t|C|} \leq \frac{\tau \Delta g}{t|C|}.$$ 

**Case 2:** $N_{j,e}^A < N_T/\gamma$

We differentiate between two sub-cases$^1$. In the first, the extremum point of $\left| \tau_{j,e}^A \cdot \log \frac{\tau_{j,e}^A}{\tau_j^A} \right|$ is reached. Since $N_{j,e}^A < N_T/\gamma$, then $\tau_{j,e}^A < \frac{2\tau \delta}{\log(e)|C|}$ (as in case 1). Because the extremum is reached when $\tau_j^A = e \cdot \tau_j^A < e \cdot \frac{2\tau \delta}{\log(e)|C|}$, the maximal value is $\frac{\tau \Delta g}{t|C|}$.

Alternatively, the term does not reach its extremum point. In that case, the term gets the highest value when $\tau_j^A$ is highest (at most $\tau$) and $\tau_{j,e}^A$ is closest to the posed limit. In that case we get:

$$\left| \tau_{j,e}^A \log \frac{\tau_{j,e}^A}{\tau_j^A} \right| \leq \frac{2\tau \delta}{t|C|} \cdot \log \frac{2|T| \delta}{|T|} \leq \frac{\tau}{t|C|} \cdot 2\delta \cdot \log \frac{t|C|}{2\delta} = \frac{\tau \Delta g}{t|C|},$$

where the last transition is based on how $\delta$ was chosen. Note that given $2\delta \log \frac{t|C|}{2\delta} = \Delta g$, $\delta$ can be evaluated using the product log function. Unfortunately, this function may return values that are not in the range $[0, \Delta g/2]$ (there may be several solutions to the equation). However, $\delta$ can be found quickly using a simple binary search.

**Case 3:** $N_{j,e}^A \geq N_T/\gamma$ and $N_j^A \geq N_T/\gamma$

In this case, we evaluate the impact of the noise, of magnitude $\mu$, added in $N_j^A$ and $N_{j,e}^A$.

$$N_{j,e}^A \cdot \log \frac{N_{j,e}^A}{N_j^A} = (\tau_{j,e}^A \pm \mu) \cdot \log \frac{\tau_{j,e}^A \pm \mu}{\tau_j^A \pm \mu} = (\tau_{j,e}^A \pm \mu) \cdot \left( \log \frac{\tau_{j,e}^A}{\tau_j^A} + \log \frac{1 \pm (\mu/\tau_{j,e}^A)}{1 \pm (\mu/\tau_j^A)} \right). \quad (5.3)$$

Note that since $N_T \geq (\rho + 1)\mu \gamma$, and since $N_{j,e}^A$ and $N_j^A$ are greater than $N_T/\gamma$, we get that $^2 \tau_{j,e}^A \geq \frac{N_T}{\gamma} - \mu \geq \frac{(\rho + 1)\mu \gamma - \mu}{\gamma} = \rho \mu$ and likewise $\tau_j^A \geq \rho \mu$. Therefore, the expressions $\mu/\tau_j^A$ and $\mu/\tau_{j,e}^A$ evaluate to at most $1/\rho$. Consequently, we can apply Claim 6:

$$\log \frac{1 \pm (\mu/\tau_{j,e}^A)}{1 \pm (\mu/\tau_j^A)} \leq \frac{2(1/\rho + 2/\rho^2)}{\ln 2} \leq \frac{6/\rho}{\ln 2} \leq \frac{\Delta g}{2t|C|} \leq \frac{1}{8}.$$

---

$^1$This case was not accounted for in the original proof, as it is possible that $N_{j,e}^A < N_T/\gamma$, while log($N_j^A$) is larger than $O(\tau \Delta g/(t|C|))$.

$^2$Note that in the original proof, while the expression log$\frac{1 \pm (\mu/\tau_{j,e}^A)}{1 \pm (\rho/\tau_j^A)}$ is recognized as constant, it is not bounded. However, since this constant is multiplied by $\tau_{j,e}^A$, it must be bounded to ensure that the expression will not exceed $O(\tau \Delta g/(t|C|))$. 

75
Next, we evaluate a lower bound on $\tau_{j,c}^{A}$. First, we have $\tau_{j,c}^{A} \geq N_{j,c}^{A} - \mu \geq \frac{N_{j}}{\gamma} - \mu \geq \frac{\tau - \mu - \mu}{\gamma}$ (line 16). In addition, since $\tau \geq (\rho + 1)\mu\gamma - \mu \geq \rho\mu\gamma$ (line 9) we have $\mu\gamma \leq \tau / \rho$ and $\mu \leq \frac{\tau}{\rho}\gamma$. Therefore, we can evaluate:

$$\tau_{j,c}^{A} \geq \frac{\tau - \mu - \mu}{\gamma} \geq \frac{\tau(\rho \gamma - \gamma - 1)}{\rho \gamma} \geq \frac{\tau}{\rho \gamma}.$$  

Combining this with $\tau_{j}^{A} \leq \tau$, we get:

$$\left| \log \frac{\tau_{j,c}^{A}}{\tau_{j}^{A}} \right| \leq \left| \log \frac{|T|}{\rho \gamma |T|} \right| \leq \log(\rho \gamma) . \tag{5.4}$$

In addition, we have:

$$\frac{\Delta g \mu \gamma}{2t|C|} \geq \frac{\Delta g \mu \gamma}{2t|C|} \geq \mu \frac{12t|C|\Delta g \gamma}{2t|C|\Delta g \ln 2} \geq \mu \frac{6 \gamma}{\ln 2} \geq \mu \cdot 2 \gamma \geq \mu \left( \log(2\gamma) + \frac{1}{8} \right) . \tag{5.5}$$

Combining equations 5.3, 5.4 and 5.5 we obtain:

$$N_{j,c}^{A} \cdot \log \frac{N_{j,c}^{A}}{N_{j}^{A}} \leq (\tau_{j,c}^{A} \pm \mu) \cdot \left( \log \frac{\tau_{j,c}^{A}}{\tau_{j}^{A}} \pm \frac{\Delta g}{2t|C|} \right) \leq \tau_{j,c}^{A} \log \frac{\tau_{j,c}^{A}}{\tau_{j}^{A}} \pm \frac{\Delta g}{2t|C|} \tau \pm \mu \left( \log(2\gamma) + \frac{1}{8} \right) \leq \tau_{j,c}^{A} \log \frac{\tau_{j,c}^{A}}{\tau_{j}^{A}} \pm \frac{\Delta g \tau}{t|C|} .$$

The suggested implementation in SuLQ-based ID3 is relatively straightforward: it makes direct use of the NoisyCount primitive to evaluate the information gain criterion. However, this implementation also demonstrates the drawback of a straightforward adaptation of the algorithm: because the count estimates required to evaluate the information gain should be carried out for each attribute separately, the data miner needs to split the overall privacy budget $B$ between those separate queries to avoid running out of budget halfway through the execution of the algorithm. Consequently, the budget per query $\epsilon$ is small, resulting in large magnitudes of noise which must be compensated for by larger datasets.
5.2.1 Improving the SuLQ-based Algorithm with the Partition Operator

In the SuLQ-based algorithm, for each internal node, we make up to \(d(t|C| + t) + 1\) noisy counts to choose an attribute, and for each leaf we make \(|C| + 1\) noisy counts to determine the dominant class value. Therefore, when each query uses budget \(\epsilon\), the privacy cost of inducing a tree of depth \(D\), consisting of \(2^D - 1\) internal nodes and \(2^D\) leaves, can be up to \(\epsilon \cdot \left[2^D \cdot (1 + |C|) + (2^D - 1) \cdot (d(t|C| + t) + 1)\right]\).

By taking advantage of the Partition operator discussed in Section 4.2.4, this privacy cost can be reduced. The privacy cost of inducing a tree will be comprised of the cost of choosing the attribute for splitting the root plus the maximal cost incurred by any of the subtrees, since the subtrees pertain to disjoint sets of records. Similarly, the privacy cost of evaluating the counts in the leaves is the maximal cost incurred by any of the counts on specific class values. Note that the Partition operator is executed using the fixed list of values possible for a given attribute and is not dependent on the values in the dataset. Otherwise, the operator could reveal the values in the dataset and violate differential privacy.

Using the Partition operator, the privacy cost is \(\epsilon(1 + 2d)\) for inducing an internal node and \(2\epsilon\) for inducing a leaf node, incurring an overall privacy cost of \(\epsilon((D - 1)(1 + 2d) + 2)\).

Several recent works [69, 53, 89] suggest methods that allow to improve further the accuracy of the count queries while lowering privacy costs. However, since eventually our goal is to find the “best” attribute rather than the accurate counts, we take a direct approach instead and consider in Section 5.3.2 the application of the exponential mechanism to this problem.

5.3 Privacy Considerations in Decision Tree Induction

5.3.1 Stopping Criteria

The recursion in ID3 stops either when all the attributes were already used, when an empty node was reached, or when all the instances in the node have the same class. The last two stopping criteria cannot be applied in the differentially private algorithm because of the introduced noise. Instead, the algorithm evaluates whether there are “enough” instances in the node to warrant further splits. The threshold should be large enough such that the information gain approximation guarantees can be maintained (as done in Algorithm 5). However, in practice, for any nontrivial domain, this number is required to be prohibitively large. Therefore, in our experiments, which were performed on significantly smaller datasets, we gave up the approximation guarantee, and followed a heuristic approach instead. Given a noisy estimate for the number of instances \(N_T\), we checked whether \(N_T/|C| \leq \sqrt{2}/\epsilon\), where \(\epsilon\) is the privacy parameter allocated for the noisy counts in the leaves. In other words, if on
average each class count in a node will be equal to or smaller than the standard deviation of the noise in the \textbf{NoisyCount}, we turn this node into a leaf. Otherwise, we try to split it. While this requirement is quite arbitrary, in the experiments it provided reasonable accuracy in the average case, despite lacking any formal accuracy guarantees. There are two conflicting factors that influence the decision to split or to stop. On one hand, the deeper the tree, the fewer samples reach each leaf, thereby increasing the chance that the noise added to the counts in the leaves would alter the classification. On the other hand, the shallower the tree, the weaker the ability to take advantage of attribute information and obtain better classifications. The point of balance between these factors may change from dataset to dataset, as it depends on the dataset size, the distribution of attribute values, the distribution of class values, and the predictive value of each attribute. After conducting several experiments with different threshold values, we found that on average the heuristic mentioned above did reasonably well, and we used it in all our experiments. For particular datasets, it is possible to tune the threshold value to be higher or lower to obtain better accuracy.

When evaluating the threshold-based stopping criterion, we have two possible outcomes: \( r_1 \) – continue (split), \( r_2 \) – stop (turn to leaf), with probabilities \( p_1 \) and \( p_2 \) respectively. Using the properties of the Laplace noise added to the count query, we can explicitly state the tradeoff between privacy and accuracy in this decision. Say that we wish to stop the tree induction with probability at least \( \hat{p}_2 \) if the accurate count is \( \Delta c \) records or more below the threshold. According to the Laplace cumulative distribution function, the probability to stop is given by \( p_2 = 1 - \exp(-\epsilon \Delta c)/2 \). To ensure that this expression evaluates to at least \( \hat{p}_2 \), we should set

\[
\epsilon_{\text{counts}} \geq -\ln(2(1 - \hat{p}_2))/\Delta c .
\]

Alternatively, the exponential mechanism can be applied to choose between the two outcomes. Given an accurate count \( c \) and a threshold \( t \), we set the query function as follows:

\[
q_{\text{threshold}}(c, r_1) = c - t, \quad q_{\text{threshold}}(c, r_2) = t - c.
\]

The sensitivity of this function is 1. In that case, for \( \Delta c = t - c \), the probability to stop is:

\[
p_2 = \frac{\exp(\epsilon \Delta c)}{\exp(-\epsilon \Delta c) + \exp(\epsilon \Delta c)} .
\]

We require this quantity to be above \( \hat{p}_2 \), from which follows \( \frac{\exp(2\epsilon \Delta c)}{1 + \exp(2\epsilon \Delta c)} \geq \hat{p}_2 \). From that we can extract the requirement

\[
\epsilon_{\text{exp}} \geq \ln \left( \frac{\hat{p}_2}{1 - \hat{p}_2} \right) / (2\Delta c) .
\]

Figure 5.1 illustrates the privacy budget (\( \epsilon \)) required to obtain the correct stopping
decision for counts that are 10 or more records below the threshold with different probability levels. Given any accuracy requirement, the exponential mechanism can provide the same level of accuracy while consuming a smaller privacy budget.

![Graph](image)

Figure 5.1: Evaluating a threshold-based stopping criterion with noisy counts and the exponential mechanism.

Despite the better privacy/accuracy tradeoff offered by the exponential mechanism, we chose to use noisy counts in our experiments for two reasons. First of all, the noisy counts are useful if pruning is to be applied (see Section 5.3.4). In addition, if several trees are induced to obtain a classifier committee (e.g., as done in [45]), the distribution information can be useful in deriving the combined classification.

### 5.3.2 Choosing an Attribute with the Exponential Mechanism

In this section we investigate how the exponential mechanism can be utilized to choose an attribute when splitting a node. When we choose a class with NoisyCount, the calculation can utilize the Partition operator to reduce privacy costs. In contrast, when we approximate the information gain with count queries, the calculation is performed separately for each attribute (the sets are not disjoint), and therefore the privacy costs accumulate across the separate calculations. Consequently, taking advantage of the exponential mechanism could significantly improve the privacy budget use, since we will be charged only for a single query operation. The idea is to let the differential privacy mechanism score the attributes (through direct access to the raw data), and then use the exponential mechanism to choose an attribute, with exponentially higher probabilities for attributes with higher scores. To this end, we propose extension of the data access layer with a new operator, ExpMech, which the user will be able to invoke with one of several predetermined scoring functions.

A crucial factor in applying the exponential mechanism is determining the query function $q$ and evaluating its sensitivity $S(q)$. Although many studies have compared the performance of different splitting criteria for decision tree induction, their results do not, in general, testify to the superiority of any one criterion in terms of tree accuracy, although the choice may affect the resulting tree size (see, e.g., [63]). Things change, however, when the splitting criteria
are considered in the context of algorithm 6. First, the depth constraint may prevent some splitting criteria from inducing trees with the best possible accuracy. Second, the sensitivity of the splitting criterion influences the magnitude of noise introduced to the exponential mechanism, meaning that for the same privacy parameter, the exponential mechanism will have different effectiveness for different splitting criteria. We consider several query functions and their sensitivity.

**Information gain:** following the discussion leading to equation 5.1, we take the query function for information gain to be

\[ q_{IG}(\mathcal{T}, A) = V(A) = -\sum_{j \in A} \sum_{c \in C} \tau_{j,c}^A \cdot \log \frac{\tau_{j,c}^A}{\tau_j^A}. \]

To evaluate the sensitivity of \( q_{IG} \), we will use the following property:

**Claim 7** For \( a > 0 \):

\[ \left| a \log \frac{a+1}{a} \right| \leq \frac{1}{\ln 2}. \]

**Proof.** The function \( a \log \frac{a+1}{a} \) does not have extremum points in the range \( a > 0 \). At the limits, \( \lim_{a \to \infty} a \log \frac{a+1}{a} = \frac{1}{\ln 2} \) and \( \lim_{a \to 0} a \log \frac{a+1}{a} = 0 \) (L'Hospital).

Let \( r = \{r_1, r_2, \ldots, r_d\} \) be a record, let \( \mathcal{T} \) be some dataset, and \( \mathcal{T}' = \mathcal{T} \cup \{r\} \). Note that in the calculation of \( q_{IG}(\mathcal{T}, A) \) over the two datasets, the only elements in the sum that will differ are those that relate to the attribute \( j \in A \) that the record \( r \) takes on the attribute \( A \). Therefore, given this \( j \) we can focus on:

\[ q'_{IG}(\mathcal{T}, A_j) = \sum_{c \in C} \tau_{j,c}^A \cdot \log \frac{\tau_{j,c}^A}{\tau_j^A}, \]

\[ = \sum_{c \in C} (\tau_{j,c}^A \log \tau_{j,c}^A) - \log \tau_j^A \sum_{c \in C} \tau_{j,c}^A, \]

\[ = \sum_{c \in C} (\tau_{j,c}^A \log \tau_{j,c}^A) - \tau_j^A \log \tau_j^A. \]

The addition of a new record \( r \) affects only one of the elements in the left-hand sum (specifically, one of the elements \( \tau_{j,c}^A \) increases by 1), and in addition, \( \tau_j^A \) increases by one as well.
Hence we get that:

\[
S(q_G) \leq (\tau_{j,c}^A + 1) \log (\tau_{j,c}^A + 1) - \tau_{j,c}^A \log \tau_{j,c}^A + \tau_j^A \log \tau_j^A - \tau_j^A + 1 \log (\tau_j^A + 1) = \\
= \left| \tau_{j,c}^A \log \frac{\tau_{j,c}^A + 1}{\tau_{j,c}^A} + \log (\tau_{j,c}^A + 1) - \tau_j^A \log \frac{\tau_j^A + 1}{\tau_j^A} - \log (\tau_j^A + 1) \right|.
\]

The expressions \( \tau_{j,c}^A \log \frac{\tau_{j,c}^A + 1}{\tau_{j,c}^A} \) and \( \tau_j^A \log \frac{\tau_j^A + 1}{\tau_j^A} \) are both of the form \( a \log \frac{a+1}{a} \). Therefore, we can apply Claim 7 and get that for a node with up to \( \tau \) elements,

\[
S(q_G) \leq \log(\tau + 1) + 1/\ln 2.
\]

Bounding the sensitivity of \( q_G(T,A) \) requires an upper bound on the total number of training examples. In our experiments, we assume that such a bound is given by the data provider. An alternate approach is to evaluate the number of training examples with a NoisyCount before invoking the exponential mechanism. The downside of this approach is that negative noise may provide a value for \( \tau \) which is too small, resulting in insufficient noise. Therefore, in this approach there is a small chance that the algorithm would violate \( \epsilon \)-differential privacy.

**Gini index:** this impurity measure is used in the CART algorithm [15]. It denotes the probability to incorrectly label a sample when the label is picked randomly according to the distribution of class values for an attribute value \( t \). Taking \( p(j|t) \) to be the fraction of records that take the value \( t \) on attribute \( A \) and the class \( t \), the Gini index can be expressed as \( \text{Gini} = \sum_{j \neq i} p(j|t)p(i|t) = 1 - \sum_j p^2(j|t) \). When determining the Gini index for a tree, this metric is summed over all the leaves, where each leaf is weighted according to the number of records in it. Minimizing the Gini index is equivalent to maximizing the following query function:

\[
q_{\text{Gini}}(T,A) = -\sum_{j \in A} \tau_j^A \left(1 - \sum_{c \in C} \left( \frac{\tau_{j,c}^A}{\tau_j^A} \right)^2 \right).
\]

To minimize the Gini index, we observe that:

\[
\min \text{Gini}(A) = \min \sum_{j \in A} \frac{\tau_j^A}{\tau} \left(1 - \sum_{c \in C} \left( \frac{\tau_{j,c}^A}{\tau_j^A} \right)^2 \right) = \min \sum_{j \in A} \tau_j^A \left(1 - \sum_{c \in C} \left( \frac{\tau_{j,c}^A}{\tau_j^A} \right)^2 \right).
\]

As in information gain, the only elements in the expression that change when we alter a record are those that relate to the attribute value \( j \in A \) that an added record \( r \) takes
on attribute $A$. So, for this given $j$, we can focus on:

$$q'_{\text{Gini}}(T,A) = \tau_j^A \left( 1 - \sum_{c \in C} \left( \frac{\tau_{j,c}^A}{\tau_j^A} \right)^2 \right) = \tau_j^A - \frac{1}{\tau_j^A} \sum_{c \in C} (\tau_{j,c}^A)^2 .$$

We get that:

$$S(q'_{\text{Gini}}(T,A)) \leq \left| \frac{\tau_j^A + 1}{\tau_j^A} - \frac{\sum_{c \notin c_r} (\tau_{j,c}^A)^2}{\tau_j^A} \right| = 1 + \frac{\tau_j^A + 1}{\tau_j^A} \left( \frac{\tau_j^A}{\tau_j^A + 1} \right)^2 - \frac{\tau_j^A \left( (\tau_j^A + 1)^2 + \sum_{c \notin c_r} (\tau_{j,c}^A)^2 \right)}{\tau_j^A} = 1 + \frac{\tau_j^A \left( (\tau_{j,c}^A)^2 - \tau_j^A (2\tau_{j,c}^A + 1) \right)}{\tau_j^A} = 1 + \frac{\sum_{c \in C} (\tau_{j,c}^A)^2 - \tau_j^A (2\tau_{j,c}^A + 1)}{\tau_j^A(\tau_j^A + 1)} = 1 + \frac{\sum_{c \in C} (\tau_{j,c}^A)^2}{\tau_j^A(\tau_j^A + 1)} - \frac{(2\tau_{j,c}^A + 1)}{\tau_j^A + 1} .$$

In the last line, $0 \leq \frac{\sum_{c \in C} (\tau_{j,c}^A)^2}{\tau_j^A(\tau_j^A + 1)} \leq \frac{\sum_{c \in C} (\tau_{j,c}^A)^2}{\tau_j^A(\tau_j^A + 1)} \leq 1$ due to $c \sum_{c \in C} \tau_{j,c}^A = \tau_j^A$ and the triangle inequality. In addition, since $\tau_{j,c}^A \leq \tau_j^A$, we get that $0 \leq \frac{2\tau_{j,c}^A + 1}{\tau_j^A + 1} \leq \frac{2\tau_{j,c}^A + 1}{\tau_j^A + 1} \leq 2$. Therefore,

$$S(q'_{\text{Gini}}) \leq 2 .$$

**Max:** (based on the resubstitution estimate described in [15]) this function corresponds to the node misclassification rate by picking the class with the highest frequency:

$$q_{\text{Max}}(T,A) = \sum_{j \in A} \left( \max_c (\tau_{j,c}^A) \right) .$$

This query function is adapted from the resubstitution estimate described in [15]. In a given tree node, we would like to choose the attribute that minimizes the probability of misclassification. This can be done by choosing the attribute that maximizes the total number of hits. Since a record can change the count only by 1, we get that $S(q_{\text{Max}}) = 1$.

**Gain Ratio:** the gain ratio [68] is obtained by dividing the information gain by a measure called *information value*, defined as $\text{IV}(A) = - \sum_{j \in A} \frac{\tau_j^A}{\tau_j^A} \cdot \log \frac{\tau_j^A}{\tau_j^A}$. Unfortunately, when $\text{IV}(A)$ is close to zero (happens when $\tau_j^A \approx \tau$), the gain ratio may become undefined or very large. This known problem is circumvented in C4.5 by calculating the gain ratio only for a subset of attributes that are above the average gain. The implication
is that the sensitivity of the gain ratio cannot be bounded and consequently, the gain ratio cannot be usefully applied with the exponential mechanism.

To ensure $\epsilon$-differential privacy when using the exponential mechanism with a query function $q$, we invoke it with the privacy parameter $\epsilon' = \epsilon/S(q)$.

Using the suggested query functions, we propose three variants of the differentially-private ID3 algorithm, presented in Algorithm 6. The variants differ in the query function $q$ passed to the exponential mechanism in line 13: we can pass either $q_{IG}$, $q_{Gini}$ or $q_{Max}$. The sensitivity of the query functions listed above suggests that information gain will be the most sensitive to noise, and the Max operator will be the least sensitive to noise. In the experimental evaluations we compare the performance of these query functions, and the influence of the noise is indeed reflected in the accuracy of the resulting classifiers. For notational simplicity, we assume the same privacy parameter $\epsilon$ in all queries.

Algorithm 6 Differential Private ID3 algorithm

1. **procedure** DIFFPID3($T, A, C, d, B$)
2. **Input:** $T$ – private dataset, $A = \{A_1, \ldots, A_d\}$ – a set of attributes, $C$ – class attribute, $d$ – maximal tree depth, $B$ – differential privacy budget
3. $\epsilon = \frac{B}{2(d+1)}$
4. Build_DiffPID3($T, A, C, d, \epsilon$)
5. **end procedure**

6. **procedure** BUILD_DiffPID3($T, A, C, d, \epsilon$)
7. $N_T = \text{NoisyCount}_\epsilon(T)$
8. if $A = \emptyset$ or $d = 0$ or $\frac{N_T}{|C|} < \frac{\sqrt{2}}{\epsilon}$ then
9. $T_c = \text{Partition}(T, \forall c \in [C] : r_C = c)$
10. $\forall c \in C : N_c = \text{NoisyCount}_\epsilon(T_c)$
11. **return** a leaf labeled with $\arg\max_c(N_c)$
12. **end if**
13. $\bar{A} = \text{ExpMech}_\epsilon(A, q)$
14. $T_i = \text{Partition}(T, \forall i \in \bar{A} : r_{\bar{A}} = i)$
15. $\forall i \in \bar{A} : \text{Subtree}_i = \text{Build}_\text{DiffPID3}(T_i, A \setminus \bar{A}, C, d - 1, \epsilon)$.
16. **return** a tree with a root node labeled $\bar{A}$ and edges labeled 1 to $|\bar{A}|$ each going to $\text{Subtree}_i$.
17. **end procedure**

Note that the calculations carried out by the exponential mechanism are similar to the ones used for regular, nonprivate evaluation of the splitting criteria, so the differentially private version of ID3 does not incur a penalty in computational costs. In addition, while the private algorithm cannot deterministically terminate the induction when no more instances are available, the threshold-based stopping criterion makes it exponentially unlikely to continue as the counts drop further below the threshold. Moreover, the imminent exhaustion of
the privacy budget ensures termination even in those unlikely cases (due to parallel composition, the exhaustion of the budget would not affect the induction process in other branches). If the data analyst does not wish to rely on the exhaustion of the privacy budget to this end, the maximal depth of the tree can be easily restricted.

5.3.3 Continuous Attributes

One important extension that C4.5 added on top of ID3 was the ability to handle continuous attributes. Attribute values that appear in the learning examples are used to determine potential split points, which are then evaluated with the splitting criterion. Unfortunately, when inducing decision trees with differential privacy, it is not possible to use attribute values from the learning examples as splitting points; this would be a direct violation of privacy, revealing at least information about the record that supplied the value of the splitting point.

The exponential mechanism gives us a different way to determine a split point: the learning examples induce a probability distribution over the attribute domain; given a splitting criterion, split points with better scores will have higher probability to be picked. Here, however, the exponential mechanism is applied differently than in Section 5.3.2: the output domain is not discrete. Fortunately, the learning examples divide the domain into ranges of points that have the same score, allowing for efficient application of the mechanism. The splitting point is sampled in two phases: first, the domain is divided into ranges where the score is constant (using the learning examples). Each range is considered a discrete option, and the exponential mechanism is applied to choose a range. Then, a point from the range is sampled with uniform distribution and returned as the output of the exponential mechanism. The probability assigned to the range in the first stage takes into account also the sampling in the second stage. This probability is obtained by integrating the density function induced by the exponential mechanism over the range. For example, consider a continuous attribute over the domain \([a, b]\). Given a dataset \(d \in \mathcal{D}^n\) and a splitting criterion \(q\), assume that all the points in \(r \in [a', b']\) have the same score: \(q(d, r) = c\). In that case, the exponential mechanism should choose this range with probability

\[
\frac{\int_{a'}^{b'} \exp(\epsilon q(d, r)/S(q))dr}{\int_{a}^{b} \exp(\epsilon q(d, r)/S(q))dr} = \frac{\exp(\epsilon \cdot c) \cdot (b' - a')}{\int_{a}^{b} \exp(\epsilon q(d, r))dr}.
\]

In general, given the ranges \(R_1, \ldots, R_m\), where all the points in range \(R_i\) get the score \(c_i\), the exponential mechanism sets the probability of choosing range \(R_i\) to be \(\frac{\exp(\epsilon c_i) \cdot |R_i|}{\sum \exp(\epsilon c_i) \cdot |R_i|}\), where \(|R_i|\) is the size of range \(R_i\). Note that this approach is applicable only if the domain of the attribute in question is finite, and in our experiments we define the domain for each attribute in advance. This range cannot be determined dynamically according to the values observed...
Table 5.1: Applying the exponential mechanism to choose a split point for a continuous attribute with Max scorer, $\epsilon = 1$.

<table>
<thead>
<tr>
<th>Range</th>
<th>Max score (per point)</th>
<th>Score proportion (for range)</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0 \leq att &lt; 2$</td>
<td>3</td>
<td>40.2</td>
<td>0.063</td>
</tr>
<tr>
<td>$2 \leq att &lt; 3$</td>
<td>4</td>
<td>54.6</td>
<td>0.085</td>
</tr>
<tr>
<td>$3 \leq att &lt; 5$</td>
<td>5</td>
<td>296.8</td>
<td>0.467</td>
</tr>
<tr>
<td>$5 \leq att &lt; 7$</td>
<td>4</td>
<td>109.2</td>
<td>0.172</td>
</tr>
<tr>
<td>$7 \leq att &lt; 10$</td>
<td>3</td>
<td>60.3</td>
<td>0.095</td>
</tr>
<tr>
<td>$10 \leq att &lt; 11$</td>
<td>4</td>
<td>54.6</td>
<td>0.086</td>
</tr>
<tr>
<td>$11 \leq att \leq 12$</td>
<td>3</td>
<td>20.1</td>
<td>0.032</td>
</tr>
</tbody>
</table>

in the learning examples, as this would violate differential privacy.

A split point should be determined for every numeric attribute. In addition, this calculation should be repeated for every node in the decision tree (after each split, every child node gets a different set of instances that require different split points). Therefore, supporting numeric attributes requires setting aside a privacy budget for determining the split points. To this end, given $n$ numeric attributes, the budget distribution in line 3 of algorithm 6 should be updated to $\epsilon = \frac{B}{(2+n)^{d+2}}$, and the exponential mechanism should be applied to determine a split point for each numeric attribute before line 13. An alternative solution is to discretize the numeric attributes before applying the decision tree induction algorithm, losing information in the process in exchange for budget savings.

Example We demonstrate the application of the exponential mechanism to choose a split point for a continuous attribute that takes values in the range $[0,12]$. We evaluate possible split points according to the Max splitting criterion for the following six learning examples labeled as positive (+) or negative (-):

---+-+---+---+-----+-+--
2 3 5 7 10 11
+ + - - - +

Table 5.1 shows the scoring for each point in each range according to the Max scorer. The probability proportion for each range is given by $exp(\epsilon \cdot q(r)) \cdot |R_i|$, where we take in the example $\epsilon = 1$. Finally, the table presents the induced probabilities. After a range is picked, a split point from the range is sampled with uniform probability.
5.3.4 Pruning

One problem that may arise when building classifiers is overfitting the training data. When inducing decision trees with differential privacy, this problem is somewhat mitigated by the introduction of noise and by the constraint on tree depth. Nonetheless, because of the added noise, it is no longer possible to identify a leaf with pure class values, so the algorithm will keep splitting nodes as long as there are enough instances and as long as the depth constraint is not reached. Hence, the resulting tree may contain redundant splits, and pruning may improve the tree.

We avoid pruning approaches that require the use of a validation set, such as the minimal cost complexity pruning applied by CART [15] or reduced error pruning [68], because they lead to a smaller training set, which in turn would be more susceptible to the noise introduced by differential privacy. Instead, we consider error based pruning [68], which is used in C4.5. In this approach, the training set itself is used to evaluate the performance of the decision tree before and after pruning. Since this evaluation is biased in favor of the training set, the method makes a pessimistic estimate for the test set error rate: it assumes that the error rate has binomial distribution, and it uses a certainty factor $CF$ (by default taken to be 0.25) as a confidence limit to estimate a bound on the error rate from the error observed on the training set. C4.5 estimates the error rate in a given subtree (according to the errors in the leaves), in its largest branch (pruning by subtree raising), and the expected error if the subtree is turned into a leaf. The subtree is then replaced with the option that minimizes the estimated error.

Since error based pruning relies on class counts of instances, it should be straightforward to evaluate the error rates using noisy counts. The error in the subtree can be evaluated using the class counts in the leaves, which were obtained in the tree construction phase. To evaluate the error if a subtree is turned into a leaf, the counts in the leaves can be aggregated in a bottom-up manner to provide the counts in upper level nodes. However, this aggregation would also add up all the noise introduced in the leaves (i.e., leading to larger noise variance). Moreover, subtrees split with multi-valued attributes would aggregate much more noise than those with small splits, skewing the results. Executing new NoisyCount queries to obtain class counts in upper level nodes could provide more accurate results. However, this would require an additional privacy budget at the expense of the tree construction phase. For similar reasons, error estimations for subtree raising would also incur a toll on the privacy budget.

As a compromise, we avoid making additional queries on the dataset and instead use the information gathered during the tree construction to mitigate the impact of the noise. We make two passes over the tree: an initial top-down pass calibrates the total instance count in each level of the tree to match the count in the parent level. Then a second bottom-up pass aggregates the class counts and calibrates them to match the total instance counts from
the first pass. Finally, we use the updated class counts and instance counts to evaluate the error rates just as in C4.5 and prune the tree. Algorithm 7 summarizes this approach. In the algorithm, \( N_T \) refers to the noisy instance counts that were calculated in algorithm 6 for each node \( T \), and \( N_j \) refers to the class counts for each class \( c \in C \).

5.4 Experimental Evaluation

In this section we evaluate the proposed algorithms using synthetic and real data sets. The experiments were executed on Weka [86], an open source machine learning software. Since Weka is a Java-based environment, we did not take advantage of the PINQ framework, which relies on the .NET LINQ framework. Instead, we wrote our own differential privacy wrapper to the \texttt{Instances} class of Weka, which holds the raw data. The algorithms were implemented on top of that wrapper. We refer to the implementation of algorithm 6 as \texttt{DiffPID3}, and to its extension that supports continuous attributes and pruning as \texttt{DiffPC4.5}.

5.4.1 Synthetic Datasets

To evaluate the performance of the algorithms on synthetic datasets, we follow the approach presented in [23] for generating learning and testing samples over ten attributes and an additional class attribute. We define a domain with ten nominal attributes and a class attribute. At first, we randomly generate a decision tree up to a predetermined depth. The attributes are picked uniformly without repeating nominal attributes on the path from the root to the leaf. Starting from the third level of the tree, with probability \( p_{\text{leaf}} = 0.3 \) we turn nodes to leaves. The class for each leaf is sampled uniformly. In the second stage, we sample points with uniform distribution and classify them with the generated tree. Optionally, we introduce noise to the samples by reassigning attributes and classes, replacing each value with probability \( p_{\text{noise}} \) (we used \( p_{\text{noise}} \in \{0, 0.1, 0.2\} \)). The replacement is chosen uniformly, possibly repeating the original value. We used training sets of varying size, and test sets with 10,000 records, which were generated in a similar way (but without noise).

Comparing Splitting Criteria Over a Single Split

In the first experiment we isolated a split on a single node. We created a tree with a single split (depth 1), with ten binary attributes and a binary class, which takes a different value in each leaf. We set the privacy budget to \( B = 0.1 \), and by varying the size of the training set, we evaluated the success of each splitting criterion in finding the correct split. We generated training sets with sizes ranging from 100 to 5000, setting 5000 as a bound on the dataset size for determining information gain sensitivity. For each sample size we executed 200 runs, generating a new training set for each, and averaged the results over the
Figure 5.2: Splitting a single node with a binary attribute, $B = 0.1$, $p_{\text{noise}} = 0.1$

Figure 5.3: Splitting a single node with a continuous attribute, $B = 0.1$, $p_{\text{noise}} = 0.1$

runs. Figure 5.2 presents the results for $p_{\text{noise}} = 0.1$, and Table 5.2 shows the accuracy and standard deviations for some of the sample sizes. In general, the average accuracy of the resulting decision tree is higher as more training samples are available, reducing the influence of the differential privacy noise on the outcome. Due to the noisy process that generates the classifiers, their accuracy varies greatly. However, as can be seen in the cases of the Max scorer and the Gini scorer, the influence of the noise weakens and the variance decreases as the number of samples grows. For the SuLQ-based algorithm, when evaluating the counts for information gain, the budget per query is a mere 0.00125, requiring Laplace noise with standard deviation over 1000 in each query. On the other hand, the Max scorer, which is the least sensitive to noise, provides excellent accuracy even for sample sizes as low as 500. Note that ID3 without any privacy constraints does not get perfect accuracy for small sample sizes, because it overfits the noisy learning samples.

Figure 5.3 presents the results of a similar experiment carried out with a numeric split. Three of the ten attributes were replaced with numeric attributes over the domain $[0, 100]$, and one of them was used to split the tree with a split point placed at the value 35. The results of the J48 algorithm (Weka’s version of C4.5 v8) are provided as a benchmark, with and without pruning. The relation between the different scores is similar for the numeric and the nominal case, although more samples are required to correctly identify the split point given the privacy budget.

In Figure 5.4 we compare the performance of DiffPC4.5 on a numeric split as opposed to discretizing the dataset and running DiffPID3. The dataset from the former experiment was discretized by dividing the range of each numeric attribute to 10 equal bins. Of course, if the split point of a numeric attribute happens to match the division created by the discrete domains, working with nominal attributes would be preferable, because determining split points for numeric attributes consumes some of the privacy budget. However, we intention-


<table>
<thead>
<tr>
<th>Number of samples</th>
<th>ID3</th>
<th>DiffPID3-InfoGain</th>
<th>DiffPID3-Gini</th>
<th>DiffPID3-Max</th>
<th>SuLQ-based ID3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>90.2 ± 1.4</td>
<td>62.6 ± 21.6</td>
<td>93.5 ± 16.8</td>
<td>100 ± 0.0</td>
<td>59.7 ± 19.8</td>
</tr>
<tr>
<td>2000</td>
<td>92.6 ± 0.9</td>
<td>73.5 ± 25.0</td>
<td>100 ± 0.0</td>
<td>100 ± 0.0</td>
<td>57.8 ± 18.1</td>
</tr>
<tr>
<td>3000</td>
<td>94.9 ± 0.7</td>
<td>85.5 ± 22.6</td>
<td>100 ± 0.0</td>
<td>100 ± 0.0</td>
<td>56.7 ± 17.1</td>
</tr>
<tr>
<td>4000</td>
<td>96.5 ± 0.6</td>
<td>94.2 ± 16.0</td>
<td>100 ± 0.0</td>
<td>100 ± 0.0</td>
<td>58.5 ± 18.8</td>
</tr>
<tr>
<td>5000</td>
<td>97.6 ± 0.5</td>
<td>98.0 ± 9.8</td>
<td>100 ± 0.0</td>
<td>100 ± 0.0</td>
<td>63.0 ± 21.9</td>
</tr>
</tbody>
</table>

Table 5.2: Accuracy and standard deviation of single split with binary attribute and binary class, $B = 0.1$

ally used discrete domains that mismatch the split point (placed at the value 35), to reflect the risk of reduced accuracy when turning numeric attributes to discrete ones. The results show that for smaller training sets, the budget saved by discretizing the dataset and switching to nominal attributes allows for better accuracy. For larger training sets, the exponential mechanism allows, on average, split points to be determined better than in the discrete case.

Figure 5.4: Comparing numerical split to discretization, $B = 0.1$, $p_{\text{noise}} = 0.1$

Inducing a Larger Tree

We conducted numerous experiments, creating and learning trees of depths 3, 5 and 10, and setting the attributes and class to have 2, 3 or 5 distinct values. We used $B = 1.0$ over sample sizes ranging from 1000 to 50,000 instances, setting 50,000 as the bound on the dataset size for determining information gain sensitivity. For each tested combination of values we generated 10 trees, executed 20 runs on each tree (each on a newly generated training set), and averaged the results over all the runs. In general, the results exhibit behavior similar to that seen in the previous set of experiments. The variance in accuracy, albeit smaller than that observed for a single split, was apparent also when inducing deeper trees. For example, the typical standard deviation for the accuracy results presented in Figure 5.5 was around ±5% and even lower than that for the results presented in Figure 5.6.
When inducing shallower trees or using attributes with fewer distinct values, we observed an interesting pattern, which is illustrated in Figures 5.5 and 5.6. When the size of the dataset is small, algorithms that make efficient use of the privacy budget are superior. This result is similar to the results observed in the previous experiments. However, as the number of available samples increases, the restrictions set by the privacy budget have less influence on the accuracy of the resulting classifier. When that happens, the depth constraint becomes more dominant; the Max scorer, which with no privacy restrictions usually produces deeper trees than those obtained with InfoGain or Gini scorers, provides inferior results with respect to the other methods when the depth constraint is present.

5.4.2 Real Dataset

We conducted experiments on the Adult dataset from the UCI Machine Learning Repository [22], which contains census data. The data set has 6 continuous attributes and 8 nominal attributes. The class attribute is income level, with two possible values, \( \leq 50K \) or \( > 50K \). After removing records with missing values and merging the training and test sets, the dataset contains 45,222 learning samples (we set 50,000 as the bound on the dataset size for determining information gain sensitivity). We induced decision trees of depth up to 5 with varying privacy budgets. Note that although the values considered for \( B \) should typically be smaller than 1, we used larger values to compensate for the relatively small dataset size\(^3\) (the privacy budget toll incurred by the attributes is heavier than it was in the synthetic dataset experiments). We executed 10 runs of 10-fold cross-validation to evaluate the different scorers. Statistical significance was determined using corrected paired t-test with confidence 0.05. Figure 5.7 summarizes the results. In most of the runs, the Max

---

\(^3\)In comparison, some of the datasets used by previous works were larger by an order of magnitude or more (census data with millions of records [57], Netflix data of 480K users [59], search queries of hundreds of thousands users [58]).
scorer did significantly better than the Gini scorer, and both did significantly better than the InfoGain scorer. In addition, all scorers showed significant improvement in accuracy as the allocated privacy budget was increased. The typical measured standard deviation in accuracy was ±0.5%.

Figure 5.7: Accuracy vs. privacy ($B$) in the Adult dataset
Algorithm 7 Pruning with Noisy Counts

1:  **Input:** $UT$ - an unpruned decision tree, $CF$ - Certainty factor
2:  **procedure** `PRUNE(UT)`
3:    TopDownCorrect($UT, UT.N_T$)
4:    BottomUpAggregate($UT$)
5:    C4.5Prune($UT, CF$)
6:  **end procedure**

7:  **procedure** `TopDownCorrect(T, fixedN_T)`
8:    $T.N_T \leftarrow fixedN_T$
9:    **if** $T$ is not a leaf **then**
10:       $T_i \leftarrow \text{subtree}_i(T)$
11:          **for** all $T_i$ **do**
12:              $fixedN_{T_i} \leftarrow T.N_T \cdot \frac{T_i.N_T}{\sum_i T_i.N_T}$
13:              TopDownCorrect($T_i, fixedN_{T_i}$)
14:          **end for**
15:    **end if**
16:  **end procedure**

17:  **procedure** `BottomUpAggregate(T)`
18:    **if** $T$ is a leaf **then**
19:       **for** all $c \in C$ **do**
20:          $T.N_c \leftarrow T.N_T \cdot \frac{T.N_c}{\sum_{c \in C} T.N_c}$
21:      **end for**
22:    **else**
23:       $T_i \leftarrow \text{subtree}_i(T)$
24:       \ $\forall T_i : \text{BottomUpAggregate}(T_i)$
25:       **for** all $c \in C$ **do**
26:          $T.N_c \leftarrow \sum_i T_i.N_c$
27:      **end for**
28:    **end if**
29:  **end procedure**
Chapter 6

Summary and Discussion
In this work we considered the tradeoff between privacy and utility when mining data, and investigated algorithms that allow for better tradeoffs. We summarize below some of the main challenges and aspects of this work.

Measuring privacy Defining what privacy means in the context of data mining is a difficult task, and at large it is still an open research question. In this work we focused on two definitions of privacy that have drawn significant attention from the research community in the last few years: $k$-anonymity and differential privacy. $k$-anonymity and derived approaches rely on a syntactic definition to ensure that one released record is indistinguishable from a set of other records. The size $k$ of the indistinguishability set is used as the privacy measure in $k$-anonymity. We extended the definition of $k$-anonymity to the realm of data mining and used the extended definition to generate data mining models that conform to $k$-anonymity. While $k$-anonymity is relatively easy to understand and verify, recent research has pointed out that it may not offer suitable protection when an adversary has access to abundant auxiliary information or even other $k$-anonymized data sets. Differential privacy, on the other hand, provides a semantic definition of privacy with worst-case guarantees that hold regardless of the adversary’s resources or available auxiliary information. A privacy parameter $\epsilon$ controls how much information can be leaked by a differentially private mechanism. Unfortunately, the strong guarantees provided by differential privacy come at a cost. Differential privacy poses harsh restrictions on data use – interactive mechanisms for differential privacy manage a privacy budget and block access to data once it is exhausted; Non-interactive mechanisms should be tailored to a predetermined set of queries, limiting future possible uses. These limitations may inhibit wide adoption of differential privacy unless breakthroughs in differential privacy research manage to overcome them.

Measuring utility Different works on privacy proposed different utility measures. In the context of $k$-anonymity, some of the anonymization algorithms rely on utility metrics to guide the anonymization process, such that the anonymized data is more useful for future analysis, which may or may not be known in advance. Measures such as Loss Metric, Discernibility Metric and Classification Metric were used as a benchmark for utility, with the purpose of optimizing the metric within given anonymity constraints. In the context of differential privacy, one of the common techniques to maintain privacy is to add noise to the result of a query. Consequently, the accuracy by which a mechanism can answer a family of queries for a given privacy budget $\epsilon$ provides a measure for the utility of the mechanism. In contrast to these approaches, we took a straightforward approach to measuring utility in the context of data mining by evaluating the accuracy of the resulting data mining models. This was possible because we focused on a specific data mining application to optimize, while former approaches targeted
more general scenarios. Despite the narrowed focus, this does not necessarily mean that the results would be useless for any other purpose. For example, we have shown that $k$-anonymized decision trees could be used to generate $k$-anonymizations with classification metric comparable (and sometimes superior) to that of the $k$-Optimal anonymization algorithm.

**Considering privacy and utility simultaneously** Previous works consider privacy protection and data mining as two separate mechanisms, each to be studied and carried out in isolation. In many $k$-anonymity works the anonymization process is guided by utility metrics, regardless of the actual data mining algorithm to be executed on the anonymized data. In the context of differential privacy, the PINQ framework was suggested as a programming interface that provides access to data while enforcing privacy constraints. In theory, PINQ should allow a programmer to write privacy preserving algorithms without requiring privacy expert knowledge. The PINQ layer enforces differential privacy, and the programmer gains a considerable amount of flexibility in designing privacy preserving algorithms. Unfortunately, a data mining algorithm can be implemented in several ways on top of this interface, and accuracy may vary considerably between these implementations.

In contrast, we argue that to improve the tradeoff between privacy and utility, these two goals should be considered simultaneously within a single process. In the context of $k$-anonymity, we showed how privacy considerations can be interleaved within the execution of a data mining algorithm, allowing to switch rapidly between utility-oriented decisions and privacy-oriented decisions. For example, when inducing decision trees, a splitting criterion (utility) is used to pick an attribute to split a node. If this would result in a breach of $k$-anonymity, the attribute is generalized (privacy) and the algorithm will re-evaluate (utility) the candidate attributes to make a new decision. This kind of interaction between utility and privacy considerations is not possible when the anonymization and data mining processes are distinct. For differential privacy we demonstrated that it is not only important what is the calculated functionality, but also how it is calculated. For example, a splitting criterion for decision tree induction, such as information gain, can be evaluated in several ways on top of a privacy preserving data interface, and choosing a good implementation is crucial to the effectiveness of the resulting algorithm. In addition, when choosing the data mining algorithm, the data miner should balance utility considerations with privacy considerations. Functionalities that are comparable in terms of utility, may have a very different privacy impact. For example, the Max, Information Gain and Gini Index criteria for choosing an attribute to split a decision tree node provide decision trees with comparable accuracy when no privacy considerations are involved. However, in a privacy preserving algorithm, privacy considerations should be taken into account as well. The Max crite-
rion for choosing an attribute has low sensitivity, so it has an advantage over the other criteria, especially when working on small data sets or with a small privacy budget. On the other end, Gini Index and Information Gain tend to generate shallower trees than the Max criterion, so given depth constraints on the induced decision tree, they may outperform a differentially private decision tree generated with the Max criterion. Hence the utility and privacy considerations should both be taken into account to obtain the best tradeoff.

Algorithm-oriented approach  Different data mining algorithms take different approaches to data analysis and therefore present different utility considerations. Therefore, we claim that to improve the tradeoff between privacy and utility, a “one size fits all” solution would not be optimal. For example, using a classification metric to guide an anonymization algorithm could provide anonymized data that can be reasonably used for classification tasks. However, to reach better accuracy, a better approach would be to adapt a specific data mining algorithm such that it provides a $k$-anonymous outcome. Similarly, a framework such as PINQ has the huge advantage that it makes privacy-preserving data analysis approachable for programmers who have no expertise in privacy research. However, to achieve a good tradeoff between privacy and accuracy, it is better to develop a differentially-private version of the desired data mining algorithm.
Bibliography


כריית מידע משמורת פרטיות

אריה פרידמן
כריית מידע משמרת פרטיות

תיבור על מחקר

לשם מילוי חלקי של הדרישות לקבלת התואר
דוקטור לפילוסופיה

אריה פרידמן

הנהלת סטטס הטכניון – מרכז טכנולוגיה לישראל
שבט התחשב"א
ת isArray
ינואר 2011
המחק נעשה בהנחיית פרופ’ אסף שוסטר בפקולטה מחשב.

ברצוני להודות למנהלתショップ, אסף,اشפר, על התמיכה המקצועית וה技能ית ויזאות הדמדום. ברצוני להודות לד”ר רן וולף על עזרתו בראשית המחקר ועל חלוקה שעבודה עם. בברצוני הניחה למ想起了י על התמיכת המחקר ואתת מתן עמית, עם עסף. וברצוני להודות למשפחה המקצועית בין היתר על מתן כל תמיכה למאגר וסמי שבעアクセס בולש שקדתיי על לימוד.

אני מודה לטכנון על התמיכת המפורשת והגדולה בשתי התוכניות.
"יש להם אפים פרטיו בתכיפה כלープנה. עכרי הלאה!"

סקוט מקנילי (מכן"ל סאן, 1999)

"אם יש המשה שלחה לא מעוניין שContextHolder אתך, אוolah эта לא עריך לעשות אתו כלכתולה!".

אריק שמידט (מכן"ל גוגל, 2009)

"לא נשיא נועשה נoha לא רק לשתייה יחרבי מידה ופוסיגה, שלום, אלא גם באורי פיתוח יחרבי עד יחרבי אנשיים. הנורמות החברתיות הם לי פשיט משה השפעה她说ה לאותך.

מאירק צוקברג (מכן"ל פייסבוק, 2010)

"לא נשיא כתה פועם פוגה פוגה סכלה.hו הצר עבור סטטוסים מוייחס יכרי על מות הפרטיות, הפרטיות לא иметь."

דנה בוז (חוקרת מדיה חברית, מיקרוסופט, 2010)
Total Information Awareness

As the technologies of information and the internet became available in the last two decades, the abundance of information has created a challenge for organizations and governmental bodies. As the amount of data increases, it becomes more and more important for organizations to use the best practices in handling the information they gather to make informed decisions.

One of the concerns that have arisen is the ability of big companies, such as major corporations and government agencies, to misuse information and endanger the privacy of their customers. A debate has been going on for years about whether government agencies could perform investigations using the data collected by companies.

For example, in 2006, AOL faced criticism after it was revealed that the company had collected data on a massive scale and had used it to create a user profile database, with a total of 20 million unique searches. Although AOL claimed that it had anonymized the data, journalists from The New York Times exposed the identities of those who had searched for certain keywords.

Another example is Netflix, which in 2009 released a database containing over 480 million reviews for movies and TV shows. The database was used to select the best algorithms for creating recommendations, but it also revealed that the company had the ability to track individual customers and manipulate their ratings.

These incidents have raised concerns among researchers, who now focus on developing methods to balance the need for information with the protection of privacy. The question is how to achieve the best results with the least amount of data, while ensuring that the algorithms used are not biased against certain groups.

In this context, the Netflix Prize was established in 2006, with the goal of improving the recommendation algorithms to make them more accurate and fair. The prize was won by the team of researchers from Bell Labs, who developed an algorithm that increased the accuracy of recommendations by 10%.

This success has led to the creation of new challenges in the field of information retrieval, as companies and governments are increasingly collecting and using large amounts of data. The question is how to balance the need for information with the protection of privacy, and how to ensure that the algorithms used are fair and unbiased.

In order to achieve this balance, researchers have developed a number of methods and techniques, including data anonymization, differential privacy, and federated learning. These methods allow companies and governments to use data while ensuring that individual identities cannot be linked to the data.

The key is to develop algorithms that are both accurate and fair, and that respect the privacy of individuals. This is a complex challenge, but one that is essential if we are to use data effectively and responsibly.

The goal is to develop algorithms that can provide useful insights while protecting individual privacy. This requires a careful balance of data analysis and privacy protection, and a commitment to ethical and responsible data use.

In conclusion, the challenge of balancing the need for information with the protection of privacy is a complex one, but it is crucial if we are to use data effectively and responsibly. As technology continues to advance, it is essential that we develop methods and techniques that allow us to use data effectively while respecting individual privacy.
The graph illustrates the trade-off between privacy and accuracy for three algorithms: Algorithm 1, Algorithm 2, and Algorithm 3. The Pareto frontier indicates the best possible trade-off between privacy and accuracy for each algorithm.

1. **Diagram Description:**

   - **Axes:**
     - X-axis: Weak Privacy to Strong Privacy
     - Y-axis: Average Error Rate

   - **Graph:**
     - Points represent the performance of each algorithm.
     - Algorithm 1 is marked with a square, Algorithm 2 with a circle, and Algorithm 3 with a triangle.
     - The Pareto frontier is highlighted, showing the optimal solutions for the trade-off.

2. **Text Description:**

   - Strong Privacy: High accuracy but low privacy.
   - Weak Privacy: Low accuracy but high privacy.

   - Algorithm 1 offers the best trade-off, followed by Algorithm 2 and then Algorithm 3.

---

The thesis explores methods for balancing privacy and accuracy in learning algorithms. It discusses the concept of the Pareto frontier, which represents the best possible trade-off between privacy and accuracy. The thesis also covers various methods for evaluating the accuracy of models, such as cross-validation. It highlights the importance of protecting sensitive data while ensuring useful results.

---

3. **Methodology:**

   - **Cross-validation:**
     - 10-fold cross-validation is used to assess the performance of the algorithms.

   - **Privacy Protection:**
     - Various techniques are employed, including cryptographic methods to protect data privacy.

---

The thesis is a comprehensive study on the interplay between privacy and accuracy in machine learning, offering insights and methods for developers and researchers to balance these critical aspects of data analysis.
Differential Privacy \[25\] is a model of privacy that generalizes the notion of "privacy" to a world where data is shared by many different parties. The model of differential privacy guarantees that the results of a query on a database are indistinguishable from the results of a query on a database with one record removed.

The definition of differential privacy is based on the idea of a "privacy budget" parameter \( \epsilon \), which quantifies the amount of privacy loss that is acceptable. If \( \epsilon \) is small, then the privacy guarantees are stronger.

The key properties of differential privacy are:
1. Composition: If \( k \) queries are executed on the same database, then the privacy budget is multiplied by \( k \).
2. Post-processing: If a differentially private mechanism is applied to the output of another differentially private mechanism, then the resulting output is also differentially private.

Differential privacy is a powerful tool for ensuring privacy in data analysis, especially in the context of large-scale data sharing.

Technion - Computer Science Department - Ph.D. Thesis  PHD-2011-01 - 2011
electronically processed and then applied to the data, it is clear that the method of our choice can also be used to create substitute decisions that are more accurate than those derived from anonymization. In fact, anonymization is performed in a way that prevents the full generation of the substitute decisions. In other words, anonymity is not sufficient to prevent the full generation of the substitute decisions.

In contrast, the method presented in this thesis is applied to the data, it is clear that the method of our choice can also be used to create substitute decisions that are more accurate than those derived from anonymization. In other words, anonymity is not sufficient to prevent the full generation of the substitute decisions.

In conclusion, the algorithm can achieve the same level of privacy as the naive method, but with fewer examples in the right-hand side. The algorithm enhances the method of our choice, and the method presented in this thesis is applied to the data, it is clear that the method of our choice can also be used to create substitute decisions that are more accurate than those derived from anonymization. In other words, anonymity is not sufficient to prevent the full generation of the substitute decisions.

However, with the availability of new technology, the method presented in this thesis can achieve the same level of privacy as the naive method, but with fewer examples in the right-hand side. The algorithm enhances the method of our choice, and the method presented in this thesis is applied to the data, it is clear that the method of our choice can also be used to create substitute decisions that are more accurate than those derived from anonymization. In other words, anonymity is not sufficient to prevent the full generation of the substitute decisions.

In conclusion, the algorithm can achieve the same level of privacy as the naive method, but with fewer examples in the right-hand side. The algorithm enhances the method of our choice, and the method presented in this thesis is applied to the data, it is clear that the method of our choice can also be used to create substitute decisions that are more accurate than those derived from anonymization. In other words, anonymity is not sufficient to prevent the full generation of the substitute decisions.

In conclusion, the algorithm can achieve the same level of privacy as the naive method, but with fewer examples in the right-hand side. The algorithm enhances the method of our choice, and the method presented in this thesis is applied to the data, it is clear that the method of our choice can also be used to create substitute decisions that are more accurate than those derived from anonymization. In other words, anonymity is not sufficient to prevent the full generation of the substitute decisions.

In conclusion, the algorithm can achieve the same level of privacy as the naive method, but with fewer examples in the right-hand side. The algorithm enhances the method of our choice, and the method presented in this thesis is applied to the data, it is clear that the method of our choice can also be used to create substitute decisions that are more accurate than those derived from anonymization. In other words, anonymity is not sufficient to prevent the full generation of the substitute decisions.

In conclusion, the algorithm can achieve the same level of privacy as the naive method, but with fewer examples in the right-hand side. The algorithm enhances the method of our choice, and the method presented in this thesis is applied to the data, it is clear that the method of our choice can also be used to create substitute decisions that are more accurate than those derived from anonymization. In other words, anonymity is not sufficient to prevent the full generation of the substitute decisions.

In conclusion, the algorithm can achieve the same level of privacy as the naive method, but with fewer examples in the right-hand side. The algorithm enhances the method of our choice, and the method presented in this thesis is applied to the data, it is clear that the method of our choice can also be used to create substitute decisions that are more accurate than those derived from anonymization. In other words, anonymity is not sufficient to prevent the full generation of the substitute decisions.

In conclusion, the algorithm can achieve the same level of privacy as the naive method, but with fewer examples in the right-hand side. The algorithm enhances the method of our choice, and the method presented in this thesis is applied to the data, it is clear that the method of our choice can also be used to create substitute decisions that are more accurate than those derived from anonymization. In other words, anonymity is not sufficient to prevent the full generation of the substitute decisions.

In conclusion, the algorithm can achieve the same level of privacy as the naive method, but with fewer examples in the right-hand side. The algorithm enhances the method of our choice, and the method presented in this thesis is applied to the data, it is clear that the method of our choice can also be used to create substitute decisions that are more accurate than those derived from anonymization. In other words, anonymity is not sufficient to prevent the full generation of the substitute decisions.

In conclusion, the algorithm can achieve the same level of privacy as the naive method, but with fewer examples in the right-hand side. The algorithm enhances the method of our choice, and the method presented in this thesis is applied to the data, it is clear that the method of our choice can also be used to create substitute decisions that are more accurate than those derived from anonymization. In other words, anonymity is not sufficient to prevent the full generation of the substitute decisions.