Data Mining and Privacy: Modeling
Sensitive Data with Differential Privacy

by

Sam Fletcher

A thesis submitted in partial fulfillment for the
degree of Doctorate of Philosophy

in the
Faculty of Business
School of Computing and Mathematics

June 2017
In the data-driven society of the 21st century, mining data to discover information about people is becoming increasingly valuable. The information can be used to learn more about society and humanity, or to build models that enable us to predict future events. Applications of data mining range from commercial endeavors, to contributing to the common good through demographic and medical studies. Unfortunately, sometimes there are real-world considerations that conflict with the goals of data mining; sometimes the privacy of the people being data mined needs to be considered. This necessitates that the output of data mining algorithms be modified to protect sensitive information, while simultaneously not ruining the informative or predictive power of the outputted model.

Many techniques have been developed to preserve privacy over the years, but one stands out above the rest: differential privacy. Differential privacy is an enforceable definition of privacy that can be used in data mining algorithms, guaranteeing that nothing will be learned about the people in the data that could not already be discovered without their personal information.

In this thesis, we focus on one particular data mining algorithm – decision trees – and how differential privacy interacts with each of the components that constitute decision tree algorithms. We analyze the conflicts that arise when balancing privacy requirements with the utility of a model. We view “utility” as a two-sided coin; on one side there is prediction accuracy, and on the other there is knowledge discovery. Optimal results for both sides cannot be achieved at the same time, and the importance of each side is dependent on the user’s needs. We explore the trade-offs that need to be made when prioritizing one side over the other.
Certificate of Authorship

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the award of any other degree or diploma at Charles Sturt University or any other educational institution, except where due acknowledgment is made in the thesis.

Any contribution made to the research by colleagues with whom I have worked at Charles Sturt University or elsewhere during my candidature is fully acknowledged. I agree that this thesis be accessible for the purpose of study and research in accordance with the normal conditions established by the Executive Director, Library Services or nominee, for the care, loan and reproduction of theses.
Acknowledgements

My utmost gratitude goes to my supervisor, Associate Professor Zahid Islam. Without his constant guidance and advice, not only would the last four years of my life have been far less successful, but also far less rewarding and enjoyable. For that I will be forever grateful, and I look forward to the coming years of continued collaboration.

I would also like to thank my secondary supervisor, Associate Professor Oliver Burmeister. Constantly ready to offer support and advice, Oliver had my back. His presence was the rock I could build my PhD upon, confident in the knowledge that I wasn’t building on sand.

Thank you to my friends on “the outside”; without you reminding that a world exists outside of my PhD bubble, I would have gone crazy years ago.

I would also like to thank my parents for providing me with the tools necessary to reach this point in my life, and for always being there for me in times of hardship. I continue to discover the subtle ways in which your parenting sculpted the attitudes and values I hold today. I am who I am because of you. Thank you so much.
Contents

Abstract i
Certificate of Authorship ii
Acknowledgements iii
Contents iii
List of Publications viii
Symbols ix
List of Figures x
List of Tables xiv

I Exordium 1

1 Introduction 3
  1.1 Overview of the Thesis 4

2 Preliminaries 7
  2.1 Data 7
  2.2 Data Mining 8
    2.2.1 Patterns 10
    2.2.2 Prediction Accuracy 11
  2.3 Decision Trees 12
    2.3.1 Greedy Decision Trees 16
      2.3.1.1 Information Gain 16
      2.3.1.2 Gain Ratio 17
      2.3.1.3 Gini Index 17
      2.3.1.4 Max Operator 18
    2.3.2 Random Decision Trees 18
    2.3.3 Factors that Affect Tree-Building 19
  2.4 Privacy 19
    2.4.1 Accessing Private Data 22
  2.5 Differential Privacy 23
4.3.4.2 Metric Properties ........................................... 85
4.4 Example Scenario: Comparing Classifiers ......................... 86
4.5 Conclusion ..................................................... 89

5 Measuring Pattern Retention after Anonymization .................. 90
5.1 Chapter Outline ............................................... 90
5.2 The Setting ................................................... 91
5.3 Related Work .................................................. 94
5.4 A Framework for Measuring Pattern Retention .................... 98
5.5 Implementations of our Framework ............................... 101
5.5.1 Pattern Accuracy .......................................... 101
5.5.2 Pattern Support Distance (PSD) ............................ 102
5.5.3 Pattern Label Distance (PLD) ................................ 104
5.6 A Thought Experiment ........................................ 106
5.7 Experiment Methodology ....................................... 107
5.7.1 Toy Privacy Preservation Techniques ......................... 109
5.7.2 Real-World Differential Privacy ............................ 109
5.7.3 Pearson’s Correlation Coefficient ........................... 110
5.8 Our Implementations in Practice ............................... 110
5.8.1 Analyzing Individual Patterns .............................. 110
5.8.2 Detecting Pattern Retention as Noise Increases .............. 112
5.8.3 Correlations Between Utility Measures ...................... 114
5.8.4 Real-World Differentially Private Data ..................... 115
5.9 Discussion .................................................... 117

III Decision Trees .................................................. 118

6 Differentially-Private Greedy Trees ................................. 120
6.1 Our Contributions ............................................. 121
6.2 A Differentially Private Decision Forest ......................... 121
6.2.1 The Local Sensitivity of the Gini Index ...................... 124
6.2.2 Pruning the Tree .......................................... 127
6.2.3 Outputting the Rules and Subrules .......................... 128
6.3 Related Work .................................................. 130
6.4 Experiments and Results ...................................... 130
6.5 Summary ....................................................... 135

7 Differentially-Private Random Trees ................................ 136
7.1 Our Contributions ............................................. 136
7.2 Previous Work .................................................. 137
7.3 Signal-to-Noise Ratio ........................................ 138
7.3.1 Signal Averaging .......................................... 138
7.4 Our Differentially Private Random Decision Forest ............. 139
7.4.1 Overview of Our Algorithm, DP-RF ......................... 139
7.4.2 Querying the Leaves of a Tree .............................. 141
7.4.3 Building a Random Tree ................................... 141
7.4.4 Defining the Number of Trees .............................. 143
List of Publications


Symbols

\( x \) A dataset
\( x_i \) A subset of \( x \), \( x_i \subseteq x \)
\( x_i^v \) A subset of \( x_i \), limited to the records that have value(s) \( v \)
\( y \) A neighboring dataset of \( x \)
\( w \) A test dataset, drawn from the same distribution as \( x \)
\( z \) A modified or anonymized version of \( x \)
\( n \) The number of records in a dataset
\( m \) The number of attributes in a dataset
\( A \) The set of attributes in a dataset
\( A \) An attribute in a dataset
\( v \) A value of an attribute (either discrete or continuous)
\( r \) A record in a dataset
\( c \) A class value, i.e. a label
\( C \) A class attribute, containing several labels
\( T \) A decision tree
\( F \) A decision forest
\( \tau \) The number of trees in a decision forest
\( b \) The number of branches made when splitting with an attribute
\( d \) The depth of a tree
\( \delta \) The maximum allowable depth of a tree
\( \psi \) A decision rule or pattern, made up of antecedents/tests
\( \Psi \) A set of decision rules or patterns
\( f \) A function
\( M \) A non-deterministic mechanism/function
\( \epsilon \) The privacy budget spent on a mechanism
\( \beta \) The total privacy budget allocated to a user
List of Figures

2.1 Clustering records on an $x$ and $y$ axis. Each axis is a different continuous attribute, and each point in the plot is a different record. Two clusters were discovered by the clustering algorithm, or perhaps the user told the algorithm to find the best clustering solution when using only two clusters. 10

2.2 The data mining pipeline for building and testing a classification model. 12

2.3 An example of a decision tree, with a depth of $d = 3$. Tags are included, identifying the different components that make up a decision tree. 13

2.4 High-level representation of the analyst’s interface with private data, using differential privacy (DP). 24

2.5 The Laplace probability distribution, with mean = 0 and scale = 1. It has higher probability at 0 than the Normal (Gaussian) distribution, and fatter tails than the Normal distribution. 25

3.1 A generalization taxonomy tree of an attribute. 42

3.2 A Venn diagram of possible outcomes when filtering records – True Positives, False Positives, True Negatives, and False Negatives. 46

3.3 The prediction accuracy achieved when building Friedman and Schuster’s decision tree with three different splitting criteria, using the Adult dataset. Based on results presented in Friedman and Schuster [2010]. Details of Adult can be found in Table 5.5 and Appendix A. 56

3.4 Comparing the prediction accuracy of Jagannathan et al.’s random decision forest algorithm (JPW) to Friedman and Schuster’s greedy decision tree algorithm (FS) with seven datasets, when $\epsilon = 1$. 59

4.1 The Jaccard index compared to the prediction accuracy difference for each dataset. Note the scope of the axes – the maximum prediction accuracy difference is small, while the Jaccard index results make use of the full range between the bounds. 88

5.1 A high-level diagram of the scenario discussed in this chapter, where a data owner is using an anonymization technique to output an anonymized version of their data to the public. The anonymization technique might add noise, or generate new records, or use another method; the anonymization technique itself is out of the scope of this chapter. We focus on measuring the comparative utility of $z$ compared to $x$, highlighted in bold. 92
5.2 A diagram of the datasets and pattern sets involved when comparing the utility of anonymized data to the original data. The test dataset $w$ is drawn from the same distribution that $x$ is drawn from, without replacement. Dataset $z$ is an anonymized version of $x$. Unique values of attributes $A_0$ and $A_1$ are distinguished from each other with unique notation, such as $b_0$. $\Psi_x$ is a set of patterns (in the form $\psi \rightarrow c$) discovered in $x$, and similarly for $\Psi_z$ and $z$. $\Psi_x | z$ is our generalized notation for any assessment of $\Psi_x$ when $z$ is inputted into it (and similarly for other pattern sets and inputs). An example of what $\Psi_x$ and $\Psi_z$ could look like is presented in Table 5.2 and Table 5.3, respectively.

5.3 The Support and Chi-squared Histogram Distance of the example patterns shown in Table 5.1 (discovered in the Adult dataset), as UN increases. The left $y$-axis measures the Support and the right $y$-axis measures the Chi-squared Histogram Distance.

5.4 The mean results of PSD, PLD and pattern accuracy error as UN increases. The left-hand $y$-axis corresponds to PLD and PSD. The right-hand $y$-axis corresponds to pattern accuracy error. The $x$-axis is the percentage of noise from 0% to 30%.

5.5 The mean results of pattern accuracy, prediction accuracy, PLD and PSD as the privacy budget decreases. We use the differentially-private technique proposed by Xiao et al. [2010] to generate new data based on the Banknotes dataset.

6.1 A comparison of our technique (DPDF) to FS using the Nursery dataset, with Random Forest included as a benchmark. We test two parameter settings for DPDF: $\tau = 1$ and $\tau = 4$.

6.2 A comparison of our technique (DPDF) to FS using the Tic-Tac-Toe dataset, with Random Forest included as a benchmark. We test two parameter settings for DPDF: $\tau = 1$ and $\tau = 4$.

6.3 A comparison of our technique (DPDF) to FS using the Connect4 dataset, with Random Forest included as a benchmark. We test two parameter settings for DPDF: $\tau = 1$ and $\tau = 4$.

6.4 A comparison of our technique (DPDF) to FS using the Card dataset, with Random Forest included as a benchmark. We test two parameter settings for DPDF: $\tau = 1$ and $\tau = 4$.

6.5 A comparison of our technique (DPDF) to FS using the Chess dataset, with Random Forest included as a benchmark. We test two parameter settings for DPDF: $\tau = 1$ and $\tau = 4$. Note that there are 18 class values, so randomly guessing would give a prediction accuracy of 5.55%; hence the low prediction accuracy.

6.6 A comparison of our technique (DPDF) to FS using the Mushroom dataset, with Random Forest included as a benchmark. We test two parameter settings for DPDF: $\tau = 1$ and $\tau = 4$.

7.1 Comparing the prediction accuracy of our technique (DP-RF) to JPW [Jagannathan et al., 2012] with different privacy budgets, using three datasets from [Bache and Lichman, 2013].

7.2 Comparing the prediction accuracy of our technique (DP-RF) to JPW [Jagannathan et al., 2012] with different privacy budgets, using three more datasets from [Bache and Lichman, 2013].
7.3 Comparing the prediction accuracy of our technique (DP-RF) to JPW [Jagannathan et al., 2012] with different privacy budgets, using three more datasets from [Bache and Lichman, 2013].

8.1 The average prediction accuracy of two possible versions of our technique with a privacy budget of $\epsilon = 1$. One version uses our proposed smooth sensitivity of the scoring function in the Exponential mechanism. The other version uses the global sensitivity of the scoring function (i.e., $\Delta(u) = 1$).

8.2 The (a) average prediction accuracy and (b) average smooth sensitivity of our proposed algorithm, with and without using disjoint data in each decision tree. All other parameters remain constant, using the default settings described in the other sections. The budget is $\epsilon = 1$ and the number of trees is $\tau = 100$. Recall that lower sensitivity is better.

8.3 The average prediction accuracy of our proposed algorithm for three different tree depths: $m/2$; our proposed depth defined by Theorem 8.2; and $m$. The results are for when the budget is $\epsilon = 1$.

8.4 The average prediction accuracy of our proposed algorithm when building different numbers of trees, with privacy budget (a) $\epsilon = 0.2$ and (b) $\epsilon = 1.0$. We recommend building 100 trees when using our algorithm, seen in blue.

8.5 The percentage of non-empty leaf nodes that had their majority labels changed by the Exponential mechanism, when building different numbers of trees with a budget of $\epsilon = 1$.

8.6 The average prediction accuracy of our proposed algorithm, when applied to dataset SynthF with different numbers of generated records $n$. The results of three privacy budgets are shown, $\epsilon = 0.01, 0.1, 1.0$.

8.7 The average prediction accuracy of our private technique as $\epsilon$ increases for the Adult dataset, approaching the prediction accuracy of a non-private extremely random forest, with all the same parameters as our technique except that no noise is added to the most frequent labels. We also include the accuracy of Breiman [2001a]'s Random Forest as context.

8.8 The average prediction accuracy of three differentially-private tree algorithms with $\epsilon = 1$: our proposed technique, JPW [Jagannathan et al., 2012], and FS [Friedman and Schuster, 2010]. We also provide the prediction accuracy of Breiman [2001a]'s (non-private) Random Forest for context, portrayed as an outlined bar.

8.9 The average area under the ROC curve of our algorithm, compared to Jagannathan et al. [2012]'s private JPW and Breiman [2001a]'s non-private Random Forest, when $\epsilon = 1$.

8.10 The average F1 score of our algorithm, compared to Jagannathan et al. [2012]'s private JPW and Breiman [2001a]'s non-private Random Forest, when $\epsilon = 1$.

8.11 The average prediction accuracy of our technique compared to RGV [Rana et al., 2016] when $\epsilon = 1$. Note that RGV uses a weaker definition of differential privacy.

9.1 The similarity between the sets of patterns produced by our algorithm in Chapter 6 and CART, compared to the similarity between the patterns produced by FS and CART.
9.2 The prediction accuracy of a tree built using our algorithm proposed in Chapter 6, compared to FS [Friedman and Schuster, 2010] and CART [Breiman et al., 1984]. .................................................. 186
9.3 The prediction accuracy of our three algorithms on three datasets, with CART and Breiman [2001a]’s Random Forest included as non-private benchmarks. .................................................. 188
List of Tables

2.1 An example dataset. .............................................. 8
2.2 The notation used throughout the thesis. ......................... 32
3.1 A comparison of the main properties of the differentially-private decision tree algorithms. .............................................. 69
4.1 Some examples of patterns (i.e., rules) discovered by CART in the WBC dataset. ..................................................... 77
4.2 The encoded versions of the rules shown in Table 4.1. ................. 81
4.3 The Jaccard index and prediction accuracy difference between CART-G and CART-I for 19 datasets. ................................. 87
5.1 A selection of patterns discovered in the Adult dataset [Bache and Lichman, 2013]. ................................................. 95
5.2 An example of what $\Psi_x$ could look like: some patterns discovered from $x$ in Figure 5.2. We include the confidence and support of the patterns as well. These patterns could be manually discovered, or discovered by a data mining algorithm, such as a decision tree. ................................. 100
5.3 An example of what $\Psi_z$ could look like: some patterns discovered from $z$ in Figure 5.2. We include the confidence and support of the patterns as well. These patterns could be manually discovered, or discovered by a data mining algorithm, such as a decision tree. ................................. 100
5.4 The results of six measures when the two patterns seen in Table 5.2 undergo changes so that they now resemble what is seen in Table 5.3. .................. 106
5.5 Details of the datasets used in our experiments. The columns are, in order: the number of records in $x$; the number of continuous attributes in $x$; the number discrete attributes in $x$; the number of labels (class values) for $C$ in $x$; and the relative frequency of the most common label in $C$. ................................. 108
5.6 A matrix of correlations for each combination of two measures, for all two noise types. We include the $p$ value of each correlation in brackets (i.e., the probability of observing a result at least as extreme as the one reported by chance, assuming there is zero correlation). ................................. 114
6.1 The size of $\epsilon$ per query, depending on the technique used and the total privacy budget $\beta$. In the example shown, the depth of the trees is $\delta = 5$. .................. 131
6.2 The main properties of the differentially-private decision tree algorithm presented in this chapter. .............................................. 135
7.1 The number of trees built for each dataset used in our experiments. ................................. 150
7.2 The main properties of the differentially-private decision tree algorithm presented in this chapter. ................................. 151

8.1 Parameters for the synthetic datasets we use throughout the chapter. . . . 155

8.2 Example smooth sensitivities of our scoring function $u$ using Theorem 8.1, when $\epsilon = 0.01, 0.1, 1.0$. Note how $j$ and $\epsilon$ have an equal impact on the result – increasing $j$ tenfold is the same as increasing $\epsilon$ tenfold. ............... 161

8.3 The percentage of leaf nodes with no records in them, when building random trees with eight real-world datasets and seven synthetic datasets. We present the results for one tree and 30 trees, when using disjoint data in each tree, and a privacy budget of $\epsilon = 1$. We include one standard deviation for each result. The depth of the trees is defined by a novel theorem that we present in Section 8.4.3. ........................................... 166

8.4 The depths calculated using Theorem 8.2 for the synthetic and real datasets used in our experiments. ........................................ 167

8.5 The main properties of the differentially-private decision tree algorithm presented in this chapter. ........................................ 180

9.1 A comparison of the main properties of all differentially-private decision tree algorithms, extending the table presented in Table 3.1. ................. 184

9.2 In broad terms, the possibility for knowledge discovery compared to the model accuracy of our three differentially-private decision forest algorithms. 187
Part I

Exordium
A problem well put is half-solved.

JOHN DEWEY
Chapter 1

Introduction

Imagine, for a moment, that you are asked to participate in a survey for a scientific study. The survey asks for personal information about your eating habits, with the study hoping to uncover patterns in the general population about factors that increase the risk of several heart conditions. The researchers will monetarily compensate you for your time, and they promise to preserve your privacy. You agree. Years later, you hear from your health insurer that based on the results of a recent scientific study, it has been discovered that eating six certain kinds of food increases the risk of a heart attack, and that since you personally often eat those six types of food, your insurance premium has risen.

Three questions grab our attention from the above example:

1. Did your participation in the survey cause your privacy to be breached? How or how not?
2. How did the researchers even find such a complicated pattern in the data they collected?
3. Is it an accurate pattern? What does “accurate” even mean in this context, and how can we measure it?

The answer to the first question might seem obvious – “yes!” – however it might not be that simple; what if the insurance company never saw the raw data from the study, but instead bought shopping data from the supermarket company that you buy your groceries from? The survey is still responsible for discovering the connection between certain food and heart attacks, but note the wording of our first question above; did your participation in the survey cause your privacy to be breached, or would the pattern have been found even if you had not participated at all?
Chapter 1. Introduction

The answer to the second question has many possible answers, most of which require the processing power of many more human minds than is practical. This is where computers enter the picture; computers allow us to process data at a rate that far exceeds many thousands of human minds, provided that they are taught what to look for. It is easy to teach a computer to calculate \( \pi \) to one million digits, but it is much harder to teach a computer to tell the difference between a cat and a dog, or to play Chess. How exactly does a computer find patterns such as in our example above, connecting certain food to heart disease?

The third question is fundamentally a question about trust and truth – can we trust that the patterns the researchers found match what is true in reality? What percentage of the time does someone who eats the six specific types of food actually develop a heart condition? How does this percentage compare to the prevalence of heart conditions in people who do not eat those types of food, or eat some fraction of them? Are these questions even the right questions to ask, to find out what we are interested in? Asking the right questions is only half the story as well; measuring the quantities we require in order to answer our questions necessitates precise data collection and statistics.

It is these three questions, and the many questions they beget, that we explore in this thesis. We propose several computer algorithms that find patterns in data and enable us to predict future events, while simultaneously preserving the privacy of everyone involved by making small changes to the data. We investigate what it means to have “good patterns”, and how to detect if the discovered patterns are affected by the small changes we need to make to the data to preserve privacy.

1.1 Overview of the Thesis

Our treatise is divided into four parts:

Part I: Exordium provides context, background information, and a review of the surrounding literature that relates to our work. It is here that we elucidate what it means to preserve a person’s privacy, and how the practical implications change for different definitions of “privacy”.

Part II: Metrics focuses on how we can measure different aspects of patterns, such as their accuracy, and how much those aspects change because of the privacy-preserving processes we apply to the data.

Part III: Decision Trees puts forward three computer algorithms, called “decision trees”, that find patterns in data and output models that can be used to predict
future events. At the same time, the algorithms also preserve the privacy of all the participants in the data.

**Part IV: Denouement** brings together all of the different threads explored throughout the thesis, weaving them into a tapestry that enables the reader to confidently and safely find high-quality patterns, without risking the privacy of those who have entrusted you with their sensitive data.

Part I covers the first three chapters, starting with this introductory chapter.

**Chapter 2: Preliminaries** lays the foundation upon which the rest of the thesis builds. We describe and define the necessary computer science that the reader will need to understand and evaluate everything we discuss in the following chapters. This includes topics such as data, data mining, decision trees, utility measures, privacy, and a particular definition of privacy known as differential privacy. We also define the terminology and notation we use throughout the thesis, as well as the terms used in the titles of the chapters in Part I and Part II.

**Chapter 3: Literature Review** thoroughly investigates the related work that came before us. We discuss several archetypes of privacy definitions, and specific implementations of each, before focusing in on the definition we use in our work: differential privacy. We analyze a variety of utility measures that have been proposed over the years, each aiming to measure something different about the utility of data, or models built from data. We then dive into the area most related to our work in Part III – differentially-private decision trees – as well as other types of differentially-private models and data.

Part II is separated into two chapters.

**Chapter 4: Measuring the Similarity between Sets of Patterns** begins by exploring the desirable properties of a utility measure. What practical considerations are there to make when designing a new way of measuring something? What mathematical considerations are there? Using these ideas, we propose a new way of measuring the similarity between two sets (i.e., collections) of patterns. We then apply this measure in a scenario where one set of patterns is found in the original data, and the other set is found in an “anonymized” version of the data; that is, a version of the data that has been modified to preserve privacy.

**Chapter 5: Measuring Pattern Retention after Anonymization** develops a generalized framework for measuring whatever changes in patterns the user is interested in. This framework is tailored to privacy-preservation, measuring changes
in patterns before and after anonymization. We provide three example measures that use our proposed framework and demonstrate how they work in practice.

Part III contains one chapter for each of the three decision tree algorithms that we propose.

**Chapter 6: Differentially-Private Greedy Trees** proposes a decision tree algorithm that is built using a “greedy” approach. We explain what this means in Section 2.3.1, but put succinctly, it refers to a tree built using a local objective function. The algorithm outputs a decision tree model with good utility, while simultaneously preserving the privacy of every participant in the data.

**Chapter 7: Differentially-Private Random Trees** proposes an algorithm with the same aims as the algorithm in Chapter 6, but does so using a “random” approach. This is explained in Section 2.3.2, but essentially it builds a decision tree while using the data as little as possible, making it much easier to preserve privacy.

**Chapter 8: Harnessing Smooth Sensitivity** extends the work done in Chapter 7 to take advantage of the idea of “smooth sensitivity”, described in Section 2.5.1. Smooth sensitivity allows for substantially less noise to be added when using differential privacy, leading to a decision tree with much higher utility than the ones presented in Chapter 6 and Chapter 7.

Part IV finishes the thesis with two chapters.

**Chapter 9: Discussion** gives us an opportunity to synthesize everything we have discovered and developed in the preceding chapters, and discuss advantages, disadvantages, implications, and possibilities for the future. We discuss the inescapable trade-offs that must be made between privacy and utility, and how to make the best of it, given the needs of any particular scenario.

**Chapter 10: Conclusion** summarizes the contributions to science made in our thesis, and offers some concluding thoughts.

We also include an Appendix A before our bibliography, which provides some information about the data used in our experiments.
Chapter 2

Preliminaries

Our thesis builds on work done in several fields, namely data mining, utility analysis, decision trees, and differential privacy. We start by laying the groundwork on which the rest of the thesis will be built, expounding the theory that we draw on later.

We start by defining precisely what we mean by “data”, and how that data can be “mined” for useful information. We focus in on a particular kind of data mining algorithm – decision trees – and explain how they are built and used. We also analyze what we mean when we talk about privacy, and offer a practical definition of it: differential privacy. After introducing the fundamentals of differential privacy and how we can design algorithms with it, we finish with a summary of the terminology and notation we will be using throughout the thesis.

2.1 Data

At the most general level, “data” refers to facts about the universe. Fortunately, we can be more specific for the purposes of this thesis: when we talk about data, we are talking about the contents of a dataset $x$, where a dataset is a two-dimensional matrix of rows and columns, with each cell in the dataset containing a single datum (i.e., piece of data, or value). Each row $r$ represents the record of a unique participant in the dataset, and each column $A$ represents an attribute or feature of the participants. The value $v$ that a record $r$ has for attribute $A$ is the value that corresponds to some real-life fact about the participant. An attribute $A$ can use either discrete (i.e., categorical or unordered) values, or continuous (i.e., numerical or ordered) values. We write $r_A$ when referring to the value that $r$ has for attribute $A$. Table 2.1 presents an example of a dataset. We use $n$ to denote the number of records in the whole dataset, and $m$ to denote the number of attributes.
There is one more important property of the datasets we use: they have a “class attribute” $C$. This is a special attribute that will be the target of the models we build. We explain what we mean in more detail in Section 2.2, but basically we will be building models that aim to correctly classify new records, using the classification labels $c \in C$. Each record $r$ has a class label, referred to as $r_C$. The labels will always be discrete in our work; continuous labels are possible, but involve enough differences in the way the model needs to be built that it is considered a different topic (namely, “regression” instead of “classification” [Breiman et al., 1984]).

### 2.2 Data Mining

Gathering and analyzing information about people is becoming increasingly important in the data-driven society of the 21st century. Technology continues to facilitate new and efficient ways of collecting data, but extracting knowledge from the data remains a difficult and nuanced topic. While discovering patterns or building models can be done manually using expert domain knowledge, the size and complexity of modern datasets has led to increasing reliance on computer-driven, human-independent techniques. Many fields of science intersect when “mining” data for information; machine learning, statistics and database systems all play a role in producing useful information from (potentially enormous) repositories of data [Breiman, 2001b]. The intersection of these fields is often referred to as “data mining”, and data mining algorithms cover a wide range of applications; some output humanly-understandable patterns discovered in the data [Vellido et al., 2012], some produce a classification or regression model that can make predictions about the future [Criminisi et al., 2011], and others detect anomalies in the data [Chandola et al., 2009]. These techniques can be applied on a wide range of modern datasets, such as medical data, financial data, social data, and law-enforcement data, among others.

---

1. Machine learning is almost synonymous with artificial intelligence, and refers to the science of designing computer algorithms that can iterate upon past successes and failures to “learn”, and thus continue to improve over time [Wagstaff, 2012]. Data mining algorithms are, in many ways, machine learning algorithms [Aodha et al., 2014, Pedregosa et al., 2011].
Data mining can be considered a part of a larger data pipeline [Han et al., 2006]:

**Data cleaning**: The removal of natural noise and inconsistent data. An example of inconsistent data would be if different words are used to signify the same discrete attribute value, or if some values are missing or blank [Rahman and Islam, 2011].

**Data integration / data transformation**: The combining of multiple data sources into a uniform dataset. Sometimes two differently titled attributes are actually describing the same thing and one should be removed, or similar continuous attributes need to be normalized to a consistent scale [Peralta, 2006].

**Data selection**: Not all attributes are relevant all the time; unnecessary attributes should be removed to prevent them from being factors in finding patterns and correlations [Han and Embrechts, 2006, Kelarev, 2012].

**Data Mining**: The processing of a computer algorithm in order to analyze the data for patterns, and to build models and make predictions [Han et al., 2006].

**Pattern evaluation and knowledge presentation**: Sometimes, obvious or useless patterns are discovered when data mining, and are ideally detected and discarded. The remaining useful (or surprising, counter-intuitive or interesting) results then need to be presented in a meaningful way, so that they can be acted upon Freitas [2013], Geng and Hamilton [2006], Huysmans et al. [2011], Vaillant et al. [2004].

When mining a dataset, there are several broad types of techniques that can be circumstantially appropriate. This can depend on the type of data being analyzed and the desired focus of the results. Some common types of data mining are association rule learning, clustering, and classification [Han et al., 2006].

**Association rule learning** analyzes the relationship between attributes – whether an increase or appearance of particular attribute values increase or decrease the chance of a value of another attribute changing or appearing. A variety of “interestingness” measures can be used to find associations that are valuable to the analyst, depending on what they are looking for [Geng and Hamilton, 2006, Tan et al., 2002]. This technique is most commonly used for transactional datasets, where the buying patterns of products can be mined for association rules [Evfimievski et al., 2003, Oliveira and Zaïane, 2002, Ordonez and Zhao, 2011]. An example of an association rule can be seen below:

\[ \{milk, eggs\} \rightarrow \{bread\} \]
Clustering is an unsupervised learning technique; it does not check its results against a known label, or receive any other kind of success or failure signals. Instead, clustering algorithms aim to find structure within the data, such as areas of high density in the multi-dimensional hypercube formed by the data [Jain, 2010]. As a simple example, if there were only two attributes, the two-dimensional clustering might look like this:

![Figure 2.1: Clustering records on an x and y axis. Each axis is a different continuous attribute, and each point in the plot is a different record. Two clusters were discovered by the clustering algorithm, or perhaps the user told the algorithm to find the best clustering solution when using only two clusters.](image)

Classification is a supervised technique that uses labeled data to train itself to predict the labels of future data. Decision tree algorithms use supervised learning to find accurate patterns in the data. Other classification algorithms include support vector machines [Sindhwani et al., 2001, Valentini and Dietterich, 2004], k-nearest neighbor classification [Thomas and Hart, 1967], and naive Bayes methods [Zhang et al., 2005], among others Freitas [2013]. Decision trees are the model of choice in this thesis, and we explain them in detail in Section 2.3.

### 2.2.1 Patterns

Another important concept in data mining, and our thesis, is the concept of a pattern. By “pattern”, we refer to a set of criteria, where if a record meets all of the criteria, allow an analyst to infer additional information about the record that was not previously known. More formally, patterns take the form $\psi \rightarrow c \in C$. That is to say, if criteria $\psi$ is met, attribute $C$ is predicted to equal $c$. $C$ is known as the consequent in this context, but is functionally the same as a class attribute for the purposes of our thesis, and hence
the same notation is used. Functioning as the antecedent, ψ is a set of conditions for a
subset of the m attributes used in the data, where each attribute A has values v ∈ A. Conditions take the form A = v, or can use other operators such as A > v if A is ordered. An example would be

\{Education = PhD, Age > 45\} → Income = High ,

or even the association rule example presented above, where milk and eggs can be thought of as Boolean (binary) attributes.

### 2.2.2 Prediction Accuracy

There are many ways to assess the utility or quality of a model – and we explore many
of them in Section 3.2 – but for now we introduce some basic concepts and the most
common method of assessing a classification model: prediction accuracy.

A classification model (otherwise known as a classifier) is built using data. Put another
way, the classifier is trained using some dataset x. We then want to assess the model
to see how good it is. Usually for a classifier, what we are interested in is if it can accurately classify future data with one of the class labels. In other words, if it can accurately predict the labels of records it has not seen before. We could use the same
data we trained the model on, but this is dangerous; the model might be over-fitting itself
to the training data, finding idiosyncrasies that are not extrapolatable to future data.
For example, a trivial classifier might build a personalized pattern for each individual
record such as \{record ID = 1234\} → Healthy. If we tested to see how accurately such as model predicted the labels of the training data, we would find that it was 100%
accurate!

To avoid this problem, we use different data. Given data from some universe (i.e.
distribution) D, we separate the data into two subsets: the training data x, and the
testing data w. After training a model Ψ with x, we then test how well Ψ can predict
the class labels of the records in w. Because we know the labels of the records in w already, we know if the model makes correct or incorrect predictions. Figure 2.2
presents this scenario diagrammatically. Put more rigorously, prediction accuracy (also
known as classification accuracy) can be written as:

\[ α(Ψ|w) = \frac{1}{|w|} \sum_{r∈w} 1(Ψ(r) = r_C) \] (2.1)
where $\alpha(\Psi|w)$ is the accuracy of model $\Psi$ on data $w$, $|w|$ is the number of records in $w$, $\Psi(r)$ is the outputted label when record $r$ is inputted into $\Psi$, and $1(\bullet)$ is the indicator function, returning 1 if $\bullet$ is true and 0 otherwise.

We can then extrapolate the model’s performance on $w$ to all future data from the universe $D$, and can use that as a guideline for how much we should trust the predictions made by the model.

### 2.3 Decision Trees

Decision tree algorithms are a non-parametric supervised learning method used for classification [Han et al., 2006]. They make no assumptions about the distribution of the underlying data, and are trained on labeled data to correctly classify previously unseen data. Figure 2.3 is an example of a decision tree. A decision tree is an acyclic directed graph [Shotton et al., 2013], built using top-down recursive partitioning of the dataset [Han et al., 2006]. The records in the dataset are recursively divided into subsets using “tests” in each node of the tree. The test checks the value each record has for a chosen attribute $A$, sending records to the child node that matches the value they have. Which attribute $A$ is chosen in each node can based on a variety of reasons, but is usually based on that attribute’s ability to split records into subsets that have as homogeneous a class label as possible.

If a discrete attribute is selected, either branches are created for each value in its domain, or a single value is chosen to split the records based on having that value or not having that value. If a continuous attribute is selected, the best splitting value is chosen from the domain of the attribute, with one branch created for all values less than the splitting point, and a second branch created for all values greater than or equal to the splitting
Figure 2.3: An example of a decision tree, with a depth of \( d = 3 \). Tags are included, identifying the different components that make up a decision tree.

point. Child nodes are then created at the end of each new branch, and another attribute is selected to continue the process. A node will not repeat any discrete attributes chosen by its ancestors, since we assume each record can only have one value per feature and thus it would be a waste to test the same discrete feature more than once. Continuous attributes can be randomly selected any number of times though, with new splitting points chosen from the domain defined by any previous tests done with that attribute in ancestor nodes. For example if the left-most leaf node in Figure 2.3 was split again with the “Age” attribute, a new splitting point would be chosen from the \((25, 120]\) domain (assuming 120 is the maximum age). Methods for finding the “best” attribute (and splitting value) in each node are discussed in Section 2.3.1. An alternative to finding the best attribute is to simply select an attribute at random; this strategy is discussed in Section 2.3.2.

The recursive branching of nodes described above is terminated when one of the following conditions are met:

- All records \( x_i \) in node \( i \) have the same class value \( c \).
- All attributes have been used previously in the current recursion chain (an attribute can only be used once, since records in a subset \( x_i \) that was filtered using value \( v \in A \) cannot be further usefully split by using attribute \( A \) again).
• The number of records in $x_i$ is below a user-defined minimum size. This is often done to prevent the tree from over-fitting to the training data, mistaking idiosyncrasies in the data for real, extrapolatable patterns.

• The depth $d$ of node $i$ has reached a user-defined maximum (i.e., the current recursion chain has repeated the maximum number of times). This is often done to limit computational complexity of the algorithm and the average decision rule complexity (i.e., length, described later) of the tree.

The resulting tree can then undergo a small amount of back-propagation in the form of “pruning”, where untrustworthy leaf nodes are removed from the bottom of the tree [Han et al., 2006]. The “untrustworthiness” of a node can be defined in a variety of ways [Letham et al., 2013], but the most common methods involve “support” and “confidence”. Support refers to the size of the data subset (i.e., the number of records) in the node. Confidence refers to the homogeneity of the class labels in the node; that is, the percentage of records with the most common class label in the node. The larger the support and confidence, the more a node is normally trusted. Tangentially, a node’s “interestingness” is another active field of research, which aims to describe how much useful information is contained in a node [Geng and Hamilton, 2006, Guillet and Hamilton, 2007].

The tree is then ready for classifying, whereby unseen records can be filtered through the tree (starting at the root node) based on the tests in each node, finally ending up in the only leaf node for which it obeys all the preceding tests. The majority (i.e., most common) class label out of the training records in that leaf node is then the predicted class label of the new record. The path a record took from the root node to a leaf node is sometimes called the root-to-leaf path, or the “decision rule” that the record obeys [Menardi and Torelli, 2014]. Each root-to-leaf path in a decision tree is a different rule, and the set of rules can be considered a summary of the information conveyed by the dataset. The number of nodes in the root-to-leaf path (excluding the leaf itself) equals the number of tests that a record must obey in order to reach the leaf, and this is called the “length” of the decision rule.

Note that when written out, a decision rule is identical to a pattern. For example, the left-most decision rule in Figure 2.3 can be written as:

$$\{Eats\ Apples = Yes, \ Age > 25\} \rightarrow Diagnosis = Heart\ Attack$$

If the tree is built greedily (explained in Section 2.3.1), local optima are a possibility, which could lead to parts of the tree being of much lower quality than other parts.
of the tree. Small fluctuations in the training data can also cause different attributes
to be chosen in early nodes, cascading into a completely different tree structure from
what otherwise might be built. To avoid these problems, a forest (i.e., an ensemble or
collection) of trees is built, with new unseen records being filtered through every tree,
and the most common prediction out of all the trees being used as the final predicted
class label (a process commonly referred to as “voting” [Han et al., 2006]).

Decision trees have several advantages over other kinds of supervised learning methods
that make them appealing to data scientists:

**High human interpretability:** The decision rules discovered by decision tree algo-
rithms can be easily read and understood by human analysts. This makes it much
easier to act on the patterns, especially in professions such as medicine where doc-
tors are often required to be able to explain their diagnoses [Huysmans et al., 2011,
Letham et al., 2013].

**Non-parametric design:** Decision tree algorithms make no assumptions about the
distribution of the data, and handle heavily skewed data well [Murphy, 2012].

**Relatively low computational cost:** The computation time of decision tree algo-
rithms rises linearly with the number of records, and polynomially with the number
of attributes: $O(nm^2)$ [Han et al., 2006].

**Ability to discover non-linear relationships among attributes:** At each node in
a decision tree, the attributes are re-explored to find relationships in only the
subset of data in that node, as opposed to simply considering the whole dataset
[Han et al., 2006].

**Resilience to missing values:** If a record is missing a value for an attribute, the rest
of the record can still be used when testing the other attributes. The record can
simply be skipped over when testing the attribute that the missing value is from
[Quinlan, 1993].

**Ability to handle both continuous and discrete data:** Both types of data are han-
dled well, and can co-exist in the same decision tree [Quinlan, 1996].

**Ability to handle non-binary labels:** The class attribute can have any number of
discrete values, and the functions used in each node still perform well [Quinlan,
1996].

Their main disadvantages – tendency to over-fit the data and instability to small changes
in the data – are minimized by limiting how deep the trees can grow, pruning away
untrustworthy leaf nodes, building an ensemble of trees instead of just one, and using bootstrapped data samples in each tree [Breiman, 2001a]. Other disadvantages can make them unsuitable in some scenarios, such as their difficulty in finding non-orthogonal decision boundaries or express XOR relationships [Murphy, 2012]. Despite some disadvantages, their advantages make them an apt choice in a wide variety of data mining scenarios.

Within the basic tree-building process, there are two fundamentally different approaches that can be taken: greedy approaches and random approaches. We discuss the differences below. If the reader is interested in the details of any of the aspects of decision trees introduced in this section, we recommend Han et al. [2006]'s textbook on the subject.

### 2.3.1 Greedy Decision Trees

The process of top-down decision-making is sometimes referred to as “greedy” decision-making, due to its strategy of making the optimal decision for the immediate short-term situation, with no regard for long-term consequences. While clearly not perfect, considering all possible eventualities of a system is usually NP-hard, and greedy heuristics perform well in practice [Han et al., 2006]. Greedy decision trees use this heuristic strategy, where an objective function is maximized in each node of the tree to decide how to split the node into child nodes. Many different splitting functions have been proposed for decision trees in the past, such as information gain [Quinlan, 1993], gain ratio [Quinlan, 1996], and gini index [Breiman et al., 1984], and they all aim to maximize the discriminatory power of the child nodes in as unbiased [Quinlan, 1996] a way as possible. We briefly describe some of greedy splitting functions below:

#### 2.3.1.1 Information Gain

Informally, information gain uses Shannon’s definition of information as being the opposite of entropy [Shannon, 1949]: reducing entropy means gaining information. More formally, information gain can be expressed as the entropy of the current node $i$ minus the total entropy of the child nodes $J$ created by splitting $i$ with each value of an attribute $A$.

Since the aim is to find the attribute that best reduces entropy, and the entropy of $i$ is the same regardless of which attribute is selected to make child nodes $J$, only the entropy of $J$ needs to be calculated in order to find the attribute with the largest information gain. The attribute that will best split $i$ is the attribute that minimizes:

$$
InfoGain(x_i, A) = -\sum_{v \in A} \left( \frac{n_i^v}{n_i} \sum_{c \in C} \frac{n_i^{v,c}}{n_i^v} \log_2 \frac{n_i^{v,c}}{n_i^v} \right),
$$

(2.2)
where $x_i$ is the subset of $x$ in node $i$, $n_i^v$ is the number of records in $i$ with value $v \in a$, and similarly for $n_i^{v,c}$ with class label $c \in C$. In the case of continuous attributes, $v$ is instead a range of values. Usually a greedy decision tree algorithm will use brute-force to check every possible splitting value that divides a continuous attribute into two subsets, and define $v$ as the two ranges above and below any given splitting value. Information gain can be used with other strategies though; using more than two ranges $v$ is fine [Islam, 2012].

Unfortunately information gain is biased towards tests with many outcomes – it prefers to select attributes with a large number of values.

### 2.3.1.2 Gain Ratio

Observing some biases in information gain, Quinlan [1996] corrected for them with a function called “gain ratio”. Gain ratio applies a kind of normalization to information gain using a “split information” value, which represents the potential information generated by splitting $x_i$ into $b$ partitions:

$$
SplitInfo(x_i, A) = -\sum_{v \in A} \frac{n_i^v}{n_i} \log_2 \frac{n_i^v}{n_i},
$$

$$
GainRatio(x_i, A) = \frac{\text{InfoGain}(x_i, A)}{SplitInfo(x_i, A)}.
$$

While gain ratio overcomes information gain’s bias towards attributes with a large number of values, it comes with its own bias. Gain ratio is biased towards unbalanced splits, where one partition is much smaller than the others.

### 2.3.1.3 Gini Index

The Gini index [Breiman et al., 1984] is a measure of how often a randomly chosen record $r \in x_i$ would be incorrectly classified if the predicted class value was randomly drawn from the distribution of class labels in $x_i$. The best attribute to split with is the one that minimizes:

$$
GiniIndex(x_i, A) = -\sum_{v \in A} n_i^v \left(1 - \sum_{c \in C} \left(\frac{n_i^{v,c}}{n_i^v}\right)^2\right).
$$

The Gini index is not without it biases either – it is biased towards attributes with large domains, and tends to favor attributes that result in equal-sized partitions and homogeneity in both partitions (though this may not necessarily be a bad thing).
2.3.1.4 Max Operator

Proposed by Friedman and Schuster [2010] as a naive, simple splitting function, max operator represents the classification rate of a node $i$, summing the highest class frequencies in each child node if $i$ was split with $A$. The best attribute to split a node with is the one that maximizes:

$$MaxOp(x_i, A) = \sum_{v \in a} \max_{c \in C} n_{i,v,c}^c.$$  

While generally poorer performing than the other splitting functions discussed, max operator has its place, as will be seen in Chapter 3.

Examples of greedy decision tree algorithms are ID3 [Quinlan, 1986] (which was developed further into C4.5 [Quinlan, 1993]) and CART [Breiman et al., 1984]. These algorithms built a single tree, and were extended in later years to build an ensemble of trees. Random Forest [Breiman, 2001a] is one such extension, which selects a random subset of records and a random subset of attributes to build each decision tree with, often 100 or more times. The random selection of records and attributes adds diversity to the decision trees, helping prevent the greedy splitting function from getting trapped in local optima. It also prevents the final decision forest from being over-reliant on specific records, which can otherwise lead to fragile tree structures.

2.3.2 Random Decision Trees

We mentioned above how Breiman [2001a] took advantage of some randomness (hence the name, “Random Forest”) to improve the performance of decision trees. This idea has since been taken much further by removing greedy heuristics all-together [Fan et al., 2003, Geurts et al., 2006]. For these random decision trees, nodes are instead split by randomly selecting an attribute. In the case of continuous attributes, the splitting point can also be uniformly randomly selected from the attribute’s range. This approach works extremely poorly for a single tree; it only performs well when many random trees are used in combination. The computational cost of this approach is also much lower than an ensemble of greedy decision trees, since it avoids calculating the output of an objective function for every attribute in every node of every tree.
To build a random decision tree, an analyst only needs the attribute schema; in other words, the list of attributes and their respective domains (i.e. range of possible values). The training data itself is not used until the tree structure has been finalized.

The prediction accuracy of random trees was further improved by Fan et al. [2003] by using combinatorial reasoning to decide the optimal depth of the trees: equal to half the number of features, $m/2$. They also empirically demonstrated that 10 to 30 trees is often enough to achieve most of the possible prediction accuracy potential, with increases in accuracy flattening out beyond 30 trees. These two findings by Fan et al. [2003] were later taken advantage of by Jagannathan et al. [2012], discussed in Section 3.3.

### 2.3.3 Factors that Affect Tree-Building

The main factors to consider when designing a decision tree algorithm are:

- What kinds of dataset properties the algorithm is catered towards, such as if it can handle discrete attributes, continuous attributes, or both. The dimensionality of the dataset also plays a large role, mostly in terms of how the number of attributes $m$ affects tree depth, discussed in Section 3.3.4. The number of records $n$ also plays a role in defining termination criteria, but predominately due to the requirements of differential privacy, as mentioned in Section 2.5.2. The overall role of the data is discussed in Section 2.1.

- What splitting function to use (including random selection). We explore the effect of this decision in Section 3.3.2.

- What termination criteria to use. We discuss four different types of terminations, and several examples of each, in Section 3.3.4.

- Whether to include a pruning step, and what pruning would be most appropriate if so. We discuss pruning in Section 3.3.5.

- Whether to build multiple trees and use each as part of a larger ensemble, and how many trees to build if so. We explore these ideas in Section 3.3.6.

### 2.4 Privacy

In 1948, the United Nations declared that privacy was a basic human right [UN General Assembly, 1948]. Since then, many governments around the world have codified the right to privacy into law. Some examples include the United States enforcing privacy
law through the Health Insurance Portability and Accountability Act (HIPAA)\textsuperscript{2}, and Australia enforcing it with the Australian Privacy Principles under the Privacy Act\textsuperscript{3}.

Privacy plays a role in scenarios ranging from government projects like a census, to businesses collecting information about their present and future customers, to health organizations analyzing illnesses or hospitals admissions [Mohammed et al., 2015]. In some of these cases, the government may mandate certain privacy protections; in other cases, businesses may want to encourage customers to provide more personal data by promising to protect their privacy. Privacy is valuable to individuals for a variety of reasons, such as for financial and social reasons. Insurance premiums can depend heavily on what an insurer knows about your past behavior or health, for example. People also like to keep some information private from their social peers, such as if they have a serious illness, a criminal record, or engage in behavior that their peers or society considers taboo.

The consequences of a breach of personal privacy were seen in 2010, when Netflix settled a lawsuit filed against them by a mother whose unrevealed sexual orientation was made public [Singel, 2010]. Netflix had released a dataset of their customers’ movie preferences to the public as part of a competition to build a movie recommendation algorithm, which they claimed was anonymized, but in fact was not. It is unknown what techniques Netflix used to anonymize their data before they published it, or how much privacy-preservation was considered “safe”, but their attempt was inadequate enough that the company decided to settle the lawsuit [Singel, 2010].

This example raises an important point – how do we know if data is adequately private? How do we define what “privacy” even means, in the context of a dataset containing sensitive information? Several definitions have been proposed over the years. Dalenius [1977] defined privacy as:

\begin{quote}
If the release of a statistic $S$ makes it possible to determine the value $v$ [of an individual] more accurately than without access to $S$, a disclosure [i.e., privacy breach] has taken place.
\end{quote}

This is the definition used when implementing several common anonymization archetypes, such as generalization, suppression, and noise addition Fung et al. [2010a]. We discuss techniques that use these archetypes in Section 3.1, but for now we provide some preliminary explanations for context.

Generalization provides privacy by decreasing the granularity of values in a dataset, such as generalizing birth dates to just birth years, or rounding dollar amounts to the

\textsuperscript{2}https://www.hhs.gov/hipaa/
\textsuperscript{3}https://www.oaic.gov.au/privacy-law/
nearest thousand. Some techniques focus on building a taxonomy of values for each attribute, and then generalizing values low in the taxonomy to a higher level [Bayardo and Agrawal, 2005, Fung et al., 2005, 2007, Iyengar, 2002, LeFevre et al., 2005, Samarati, 2001, Sweeney, 2002, Wong et al., 2006, Xu et al., 2006]. One of the advantages of generalization over other forms of data modification is that all the generalization techniques “result in a less precise but consistent representation of the original data” [Fung et al., 2010a]. There is no risk of the generalized data being misleading or false; the data is merely vaguer. Counter-intuitively, generalization can sometimes increase data utility, as overly specific information such as exact dates can often act as noise when performing data mining [Fung et al., 2010a]. Defining a taxonomy of values for each attribute can prove difficult however; it often needs to be manually designed by domain experts, and can change depending on the expected workload of the data (that is, how the data is expected to be used).

Suppression involves replacing of some values with null or missing values. While the bluntest form of data modification, it can be very useful in certain scenarios. It results in a large amount of information loss and will be obvious to any malicious users looking for sensitive information, but it shares a similar benefit to generalization in that it prevents misleading or false patterns from being inferred. This can be of paramount importance in situations such as health or national security – acting based on a lack of data is preferable to acting on incorrect data [Saygin et al., 2001]. Some techniques suppress entire records [Bayardo and Agrawal, 2005, Iyengar, 2002, LeFevre et al., 2005, Samarati, 2001]; others suppress all occurrences of particular values [Wang et al., 2005, 2007]; and others suppress only some occurrences of values [Cox, 1980, Meyerson and Williams, 2004].

Noise addition involves modifying continuous values by a randomly generated amount, drawn from some appropriate distribution, or by randomly swapping some discrete values to different values with some probability [Agrawal and Aggarwal, 2001, Agrawal and Srikant, 2000, Islam, 2007, Liu et al., 2006]. This approach can lead to technically incorrect data being present, but can often have higher utility in the aggregate than generalized or suppressed data.

A problem exists with Dalenius [1977]’s definition, however; a problem demonstrated by our example at the start of Chapter 1. Imagine that you, the individual, decided to not participate in the study about eating habits, but that the connection between seven types of food and heart attacks was still discovered. Has your privacy been breached? According to Dalenius, yes it has, since something was learned about you (your increased risk of heart attack). This causes a problem though – how is a company like Netflix
meant to use this definition of privacy? If a person’s privacy can be breached independent of whether or not they are even in the dataset, the definition is vacuous and unusable.

Dwork [2006] rectified this problem by offering another definition of privacy:

If information is learned about someone because they participated in a study, and that information would not be learned if they did not participate, a privacy breach has occurred.

This definition is enforceable, and can offer guarantees to every participant. Known as “differential privacy”, it is the definition we use in our proposed decision tree algorithms in Part III. We define it more thoroughly below in Section 2.5, as well defining some of its other features.

2.4.1 Accessing Private Data

There are two main scenarios in which data may need to be anonymized: the “interactive” scenario and the “non-interactive” scenario. Interactive privacy preservation (sometimes titled as Privacy-Preserving Data Mining, or PPDM [Aggarwal and Yu, 2008, Islam, 2007]) involves inputting queries to a dataset owned by somebody else, and receiving anonymized outputs; the data itself is never publicized [Blum et al., 2005, Dinur and Nissim, 2003]. Non-interactive privacy preservation (or Privacy-Preserving Data Publishing, PPDP [Fung et al., 2010a]) is where the data owner applies an anonymization process to the data, then releases the anonymized version to the public [Adam and Worthmann, 1989, Brand, 2002]. We investigate this scenario in Part II, and propose methods for measuring aspects of anonymized data that were not previously detectable.

Differential privacy can be used in either approach. It can be used non-interactively, by generating a “synthetic” dataset where new records are created using information from the original dataset [Hardt et al., 2012, Xiao et al., 2010]. Data mining techniques have been developed for it in the interactive scenario in the past [Friedman and Schuster, 2010, Jagannathan et al., 2012], and we contribute to this pursuit in Part III.

It is worth mentioning that in almost all scenarios involving privacy preservation, the attribute schema (that is, the domains of the attributes) is considered to be public knowledge [Blum et al., 2005, Dwork and Roth, 2013, Friedman and Schuster, 2010, Jagannathan et al., 2012, Rana et al., 2016]. This information can be dependent on the data in the sense that the domain of a discrete attribute is the set of discrete values in the dataset $x$, but it can also contain values that are possible in the universe $D$ but lack any actual examples in $x$. This is the recommended approach for continuous attributes.
Chapter 2. Preliminaries

[Dwork and Roth, 2013]; defining the lower and upper bounds of a continuous attribute using the minimum and maximum values found in \( x \) can be a breach of privacy, and so instead it is better to define the bounds using the universe \( D \). For example in Figure 2.3, the domain of the “Age” attribute might be \([0, 120]\), even if the oldest person in \( x \) is 96. This approach can be used for discrete attributes as well, though defining the domain of an attribute like “Favorite Food” as all possible food on Earth is likely to make the attribute untenable when trying to query it. In the interactive scenario, it is therefore the responsibility of the data owner to provide a reasonable attribute schema to the analysts wishing to query the data. It is possible for analysts to use their best estimations to create a reasonable schema of their own, but this is susceptible to practical issues such as value formatting and spelling. Without a public attribute schema, it is difficult to imagine how queries can even be submitted to a database.

2.5 Differential Privacy

Differential privacy [Dwork, 2006, 2007, 2008, 2011, Dwork and Roth, 2013, Dwork et al., 2006, McSherry, 2009, McSherry and Talwar, 2007] makes a promise to each participant in a dataset: “Any information that could be discovered about you with your data in the dataset could also, with high probability, be discovered without your data in the dataset”. In other words, the output of any query \( f \) performed on dataset \( x \) will be indistinguishable from the output of the same query \( f \) performed on dataset \( y \), where \( y \) differs from \( x \) by at most one record (the record of any participant). It does not promise that nothing will be discovered about the participant, just that whatever is discovered about them would have been discovered even if their data was not in the dataset at all. It also promises that any supplementary data a malicious user might have about the participant is irrelevant; the attacker can know any amount of information about a participant, and even know every single other data point in the dataset, and still not be able to detect their presence in \( x \). This is not a 100% guarantee, but instead a very high probability guarantee. The exact probability is determined by a parameter \( \epsilon \); the smaller \( \epsilon \) is, the higher the privacy guarantee.

Figure 2.4 presents a high-level view of the pipeline used by differentially-private data mining algorithms; an algorithm submits a query to the dataset, the dataset calculates the answer to the query, and then a differentially-private mechanism modifies the answer in a way that protects the privacy of every individual person in the dataset.

An analyst is given limited access to dataset \( x \), in which they are allowed to query \( x \) in an \( \epsilon \)-differentially private way. For any given query \( f \), the value of \( \epsilon \) can be equal to or less than the amount provided to the analyst by the data owner. We define this
amount as the total privacy budget $\beta$, and will discuss how it can be divided into smaller parts for each query $f$. Each time the user queries the dataset, an amount of $\beta$ is spent depending on how invasive the query was to the privacy of the individuals described in the dataset, and how accurate the analyst wants the output to be. Once all of $\beta$ is spent, the analyst loses access to the data forever.

We write the following definitions in terms of some query (i.e., function) $f$ submitted to a dataset $x$ from a universe $D$. We compare $x$ to a neighboring dataset $y$, where “neighbor” describes how many records differ between $x$ and $y$ (i.e., their Hamming distance in terms of records).

**Definition 2.1 (Differential Privacy [Dwork, 2006]).** A non-deterministic function $M$ (i.e. a function with a randomized component) is $\epsilon$-differentially private if for all outputs $g \subseteq \text{Range}(M)$ and for all data $x, y \in D^n$ such that $||x - y||_1 \leq 1$:

$$\Pr(M(x) = g) \leq e^\epsilon \times \Pr(M(y) = g) .$$  

Function $M$ will often be “wrapper” function around a deterministic function $f$.

The parameter $\epsilon$ can be considered as a “cost”, with multiple costs summing together, described below:

**Definition 2.2 (Composition [McSherry and Talwar, 2007]).** The application of all functions $\{M_i(x)\}$, each satisfying $\epsilon_i$-differential privacy, satisfies $\sum_i \epsilon_i$-differential privacy.

If the same query (i.e. function) is submitted to multiple subsets of the data, with no overlapping records, the costs do not need to be summed:

**Definition 2.3 (Parallel Composition [McSherry, 2009]).** For disjoint subsets $x_i \subset x$, let function $M(x_i)$ satisfy $\epsilon$-differential privacy; then applying all functions $\{M(x_i)\}$ still satisfies $\epsilon$-differential privacy.
Differential privacy is a definition, not an algorithm; mechanisms need to be designed that allow an analyst to query data in a way that adheres to the definition. These mechanisms are often formulated as function $M$, which takes a non-private function $f$ as input and converts it into a differentially-private function. A popular mechanism for outputting a real number is to use the following Laplace mechanism, and a discrete value can be outputted with the Exponential mechanism:

**Definition 2.4** (Laplace Mechanism [Dwork et al., 2006]). A query $M$ successfully satisfies $\epsilon$-differential privacy if it outputs $M(x) = f(x) + L$, where $f : f(x) \to \mathbb{R}$ and $L \sim \text{Lap}(\Delta(f)/\epsilon)$ is an i.i.d. random variable drawn from the Laplace distribution with mean 0 and scale $\Delta(f)/\epsilon$ (see Figure 2.5). We shorten $L \sim \text{Lap}(\Delta(f)/\epsilon)$ to $\text{Lap}(\Delta(f)/\epsilon)$ when our meaning is clear from context. $\Delta$ is described below.

**Definition 2.5** (Exponential Mechanism [McSherry and Talwar, 2007]). Using a scoring function $u : u(z,x) \to \mathbb{R}$ where $u$ has a higher value for more preferable outputs $z \in Z$, a query $M$ satisfies $\epsilon$-differential privacy if it outputs $z$ with probability proportional to $\exp(\frac{\epsilon u(z,x)}{2\Delta(u)})$. That is,

$$\Pr(M(x) = z) \propto \exp\left(\frac{\epsilon \times u(z,x)}{2\Delta(u)}\right). \quad (2.5)$$

The above mechanisms add noise to the output that is scaled to the “sensitivity” $\Delta$ of the query, which is defined as the maximum amount that the output could change by if $x$ had one record added or removed. The original, conventional definition of sensitivity used is
known as “global sensitivity”, described below in Definition 2.6. Another definition of
sensitivity is discussed later in Section 2.5.1.

**Definition 2.6 (Global Sensitivity [Dwork et al., 2006]).** A query $f$ has global sensitivity
$\Delta(f)$, where:

$$\Delta(f) = \max_{x,y : ||x-y||_1 \leq 1} ||f(x) - f(y)||_1 .$$

(2.6)

Differential privacy has become the de-facto privacy standard around the world in recent
years, with the U.S. Census Bureau using it in their Longitudinal Employer-Household
Dynamics Program in 2008 [Machanavajjhala et al., 2008], and the technology company
Apple implementing differential privacy in their latest operating systems and applica-
tions [Greenberg, 2016]. If the reader is interested in delving deeper into differential
privacy, we recommend Dwork and Roth [2013]’s textbook on the subject.

### 2.5.1 Smooth Sensitivity

The global sensitivity of a function $f$ is the theoretically largest difference between the
output of $f(x)$ and $f(y)$, for any possible dataset $x$ and its neighbor $y$. Instead, we can
consider the local sensitivity of $f$, which takes into account a specific $x$:

**Definition 2.7 (Local Sensitivity [Nissim et al., 2007]).** For $f : D^n \rightarrow \mathbb{R}^d$ where $n,d \in \mathbb{N}$, and data $x \in D^n$, the local sensitivity of $f$ at $x$ (with respect to the $\ell_1$ metric) is

$$LS_f(x) = \max_{y : ||x-y||_1 \leq 1} ||f(x) - f(y)||_1 .$$

(2.7)

Unfortunately this definition of sensitivity is not differentially private. Nissim et al.
[2007] developed a method for making it differentially private in 2007, dubbing it smooth
sensitivity:

**Definition 2.8 (Smooth Sensitivity [Nissim et al., 2007]).** The local sensitivity of $f$, with distance $k$ between datasets $x$ and $y$, is

$$S_k(x) = \max_{y : ||x-y||_1 \leq k} LS_f(y) .$$

(2.8)

The smooth sensitivity of $f$ can now be expressed using $S_k(x)$:

$$S^*(f, x) = \max_{k=0,1,\ldots,n} e^{-\epsilon k} S_k(x)$$

(2.9)

where $\epsilon$ is the privacy budget of $f$. 

Smooth sensitivity allows for much less noise to be added while still achieving differential privacy, by analyzing the actual dataset $x$ instead of just assuming the worst-case scenario.

### 2.5.2 Factors that Affect Differentially-Private Algorithms

The main factors that need to be considered when designing a data mining algorithm are:

- How large of a privacy budget $\beta$ the data owner is providing the analyst with. The total budget dictates the overall constraints put on the data mining algorithm. We discuss the range of sizes $\beta$ can have in real-world scenarios in Section 3.5.

- The number of times the data needs to be queried. The more queries that the algorithm needs, the more the total privacy budget needs to be divided up to pay for them all. If each query uses $\epsilon_i$ amount of the budget, $\sum \epsilon_i \leq \beta$. The smaller $\epsilon_i$ is, the more noisy the outputs of the queries will be. Limiting the number of required queries is discussed in Section 3.3.2, Section 3.3.4 and Section 3.3.8.

- The sensitivity $\Delta$ of the queries. Sometimes a query that performs well in a non-private setting becomes unusable due to how sensitive it is to individual records, leading to overwhelming noise being added to the output. Instead, traditionally sub-optimal queries in non-private settings can be preferable in the private setting if they have low sensitivity. We explore this phenomena in Section 3.3.2.

- The size $n$ of dataset $x$ also plays an important role. The amount of noise that must be added to enforce differential privacy is independent of $n$, so the larger $n$ is, the smaller the relative amount of noise becomes.

### 2.5.3 Introducing Differentially-Private Decision Trees

Now that decision trees have been introduced in Section 2.3, and differential privacy in Section 2.5, a natural progression is to combine these two ideas.

When outputting a decision tree, privacy is leaked via the information describing each node. At its most basic level a decision tree algorithm is deciding which attribute to split each node with (e.g. with a splitting function; see Section 2.3.1.1), and this decision is dictated by the data in the node. Once the tree has finished being built, the leaf nodes can output some information about the class counts, which is also dictated by the data in the nodes (see Section 2.3). Since these decisions and outputs are directly
based on the data, differential privacy states that releasing the information can be a breach of privacy. These potential breaches are what a differentially-private decision tree algorithm aims to prevent. In Section 3.3 we explore the current state-of-the-art in differentially-private decision trees, and then propose extensions to the art in Part III.

2.6 Terminology

Below is most of the terminology that we use throughout the thesis. We provide it as a guide, should the reader wish to refer back to it while reading deeper into our work.

**Data**: Facts about the universe that can be interpreted by a human or computer.

**Information**: Transformations or re-interpretations of data into correlations, patterns, or other statistics.

**Knowledge**: Useful or actionable information that can be read and understood by humans.

**Model**: A representation or interpretation of some aspect of the universe, such as a weather system, a component of society, or a concept such as “privacy”. It might enable a user to understand something about the system being modeled, or to make predictions about future events.

**Record / Tuple / Row**: Data about a single participant in a dataset.

**Attribute / Feature / Column**: A property of the participants in the dataset, describing them with a variety of values.

**Dataset**: A two-dimensional matrix or table made up of rows and columns, containing data.

**Continuous / Numerical / Ordered**: Values of an attribute that exist in real space, \( v \in \mathbb{R} \), with some values being larger than others.

**Discrete / Categorical / Unordered**: Values of an attribute with no inherent ordering, that exist independently of each other.

**Domain / Range**: The set of all possible values for a discrete attribute, or the minimum and maximum possible values for a continuous attribute. When talking about these properties for all attributes as a whole (as well as any supplementary descriptions of each attribute), we call it the “attribute schema”.
Participant / Individual / Person: A person who has supplied data about themselves to a dataset.

Analyst / User: The person who will be using or querying the data to extract knowledge or build models. May be referred to as “the user” when clear from context that we are talking about the analyst.

Data Owner / User: The person or organization that owns the dataset, and is either releasing an anonymized version of the data or is providing analysts with secure access to it. May be referred to as “the user” when clear from context that we are talking about the data owner.

Private / Sensitive: Data about participants that they might want kept private.

Quasi-identifier: Data about participants that, when combined with other data, can uniquely identify them.

Non-Private / Public: Data that is either already publicly available, or is otherwise not sensitive.

Publishing / Data Release: Making data available to the general public, or to people who did not initially have access to it.

Anonymization / De-identification / Perturbation / Modification: The process of transforming or modifying sensitive data or information into a form that has a lower risk of breaching participants’ privacy.

Noise: Randomly generated continuous values that are added to otherwise accurate data, or the process of randomly changing a discrete value to another value in the attribute’s domain with some probability.

Attacker / Intruder / Malicious User: Someone pretending to be an analyst, who wants to discover information about one or more specific people who the attacker suspects is in the dataset.

Generalization & Specialization: The process of changing the granularity of a value, either making it more (specialization) or less (generalization) specific. Suppression is the most extreme method of generalization; changing a value to be null or missing.

Neighboring Dataset: A dataset $y$ that has all the same records as dataset $x$, except for $k$ records.

Workload: The job or application that some data or information is to be used for.
Classification & Regression: The process of ascribing or predicting a record to have a certain label (classification); or to be larger, smaller or equal to some continuous value (regression).

Utility / Quality / Performance: A measure of the usefulness or quality of some data, information, knowledge or model.

Training & Testing Data: Data that is used to build a model (training data) or test a model’s performance (testing data).

Prediction Accuracy / Testing Error: A specific measure of quality, in which the accuracy of a model at predicting (i.e., classifying) future events is estimated by using test data as a stand-in for future, currently unknown, data.

Class Labels: Values of a special attribute (the class attribute) that a model is built to classify or predict. Sometimes simply referred to as labels, or class values.

Majority: The most common value in some data out of all values of a specific attribute. Usually used when talking about the most common class label in a leaf node of a decision tree.

Decision Tree: A type of classification model built by recursively dividing up data with the aim of finding subsets of records that all share the same class label. Sometimes simply referred to as trees.

Decision Forest / Ensemble: Multiple decision trees, used in conjunction to form a higher-quality model than a single tree.

Graph: A model made up of nodes and edges, where nodes represent some piece of information and edges represent connections between two nodes that are related somehow. A decision tree is a type of graph.

Branch / Edge: A component of a graph or decision tree. Connects two nodes together. Can be either directed (where the edge goes from one node to another) or undirected. Edges in decision trees are directed, pointing from a parent node to a child node. Multiple branches are formed when a parent node is split into multiple child nodes, with one branch pointing to each child node.

Node: A component of a graph or decision tree. In a decision tree, nodes represent a subset of the data. Parent nodes contain more records than child nodes, with child nodes containing records that are ideally more closely related to each other than the records in the parent node.
Root & Leaf: The root node is the first node in a graph made up of directed edges, such as a decision tree. No edges point to a root node; only edges that go from it. Leaf nodes are the last nodes in a directed graph; no edges come out of leaf node. There is only one root node in a decision tree, but many leaf nodes.

Path: An unbroken connection of directed edges from one node to another, with any number of nodes in between. Since decision trees are acyclic (no circles), a path can never return to a previous node. We normally talk about “root-to-leaf” paths, which start at the root node and end at a leaf node.

Depth / Height / Level: How far a decision tree has grown from the root node. The root node is considered to have a depth of $d = 1$, with children of the root having a depth of $d = 2$, and so on.

Pattern / Rule: A formal, syntactically consistent description of a pattern found in data. Lists some requirements or conditions that when met, predicts an event or label to be true. In our context, it is usually the result of following a root-to-leaf path in a decision tree. Also called a decision rule or logic rule.

Antecedent & Consequent: The conditions or requirements in a rule (antecedent); or the predicted event or label in a rule (consequent).

Support & Confidence: The number of training records that match the conditions of a pattern (support); or the fraction of training records in a pattern that have the majority class label (confidence).

Query: A request for information or statistics from a database, containing a list of requirements or parameters that define what records to consider when calculating the answer.

2.7 Notation

Table 2.2 below presents the notation used throughout the thesis. Individual chapters may use additional notation on top of what is presented in Table 2.2, but everything here remains true and consistent in all chapters. This list is also provided in the opening pages of the thesis, before Part I.
Table 2.2: The notation used throughout the thesis.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>A dataset</td>
</tr>
<tr>
<td>$x_i$</td>
<td>A subset of $x$, $x_i \subseteq x$</td>
</tr>
<tr>
<td>$x_i'$</td>
<td>A subset of $x_i$, limited to the records that have value(s) $v$</td>
</tr>
<tr>
<td>$y$</td>
<td>A neighboring dataset of $x$</td>
</tr>
<tr>
<td>$w$</td>
<td>A test dataset, drawn from the same distribution as $x$</td>
</tr>
<tr>
<td>$z$</td>
<td>A modified or anonymized version of $x$</td>
</tr>
<tr>
<td>$n$</td>
<td>The number of records in a dataset</td>
</tr>
<tr>
<td>$m$</td>
<td>The number of attributes in a dataset</td>
</tr>
<tr>
<td>$A$</td>
<td>The set of attributes in a dataset</td>
</tr>
<tr>
<td>$A$</td>
<td>An attribute in a dataset</td>
</tr>
<tr>
<td>$v$</td>
<td>A value of an attribute (either discrete or continuous)</td>
</tr>
<tr>
<td>$r$</td>
<td>A record in a dataset</td>
</tr>
<tr>
<td>$c$</td>
<td>A class value, i.e. a label</td>
</tr>
<tr>
<td>$C$</td>
<td>A class attribute, containing several labels</td>
</tr>
<tr>
<td>$T$</td>
<td>A decision tree</td>
</tr>
<tr>
<td>$F$</td>
<td>A decision forest</td>
</tr>
<tr>
<td>$\tau$</td>
<td>The number of trees in a decision forest</td>
</tr>
<tr>
<td>$b$</td>
<td>The number of branches made when splitting with an attribute</td>
</tr>
<tr>
<td>$d$</td>
<td>The depth of a tree</td>
</tr>
<tr>
<td>$\delta$</td>
<td>The maximum allowable depth of a tree</td>
</tr>
<tr>
<td>$\psi$</td>
<td>A decision rule or pattern, made up of antecedents/tests</td>
</tr>
<tr>
<td>$\Psi$</td>
<td>A set of decision rules or patterns</td>
</tr>
<tr>
<td>$f$</td>
<td>A function</td>
</tr>
<tr>
<td>$M$</td>
<td>A non-deterministic mechanism/function</td>
</tr>
<tr>
<td>$\epsilon$</td>
<td>The privacy budget spent on a mechanism</td>
</tr>
<tr>
<td>$\beta$</td>
<td>The total privacy budget allocated to a user</td>
</tr>
</tbody>
</table>
Chapter 3

Literature Review

Before we can delve into the details of our contributions to data science, we first contextualize our contributions within the surrounding contributions of others that came before us.

In Section 3.1 we start with an overview of the research done into privacy-preserving data science, providing context for our privacy model of choice, differential privacy. We then analyze the nuances involved in measuring the utility of both anonymized raw data, and anonymized data mining models, in Section 3.2. We finish our literature review with several sections dedicated to differential privacy: Section 3.3 explores how differential privacy interacts with every component of decision tree algorithms; Section 3.4 looks at how decision trees can be used to publish differentially-private data, as opposed to building a data mining model; and Section 3.5 completes our literature review with proposed guidelines on how large or small a user’s privacy budget should be in real-world scenarios.

3.1 Alternate Privacy Models

In Chapter 2 we introduced differential privacy, the model of privacy that we will be using throughout our thesis. Differential privacy does not exist in a vacuum though, and privacy-preserving data mining and publishing has been an active area of research for

---


Other parts of the work in this chapter has been submitted to the following journal: Sam Fletcher and Md Zahidul Islam. Decision Tree Classification with Differential Privacy: A Survey. ACM Computing Surveys, 2016.
over two decades [Brankovic and Estivill-Castro, 1999, Clifton and Marks, 1996, Reiss et al., 1982]. In that time, many privacy models have been proposed, each aiming to preserve the privacy of every participant in a dataset. We outline some of these alternate privacy models here.

These models often categorize attributes into a few different types, depending on what they can reveal about participants. In general, attributes can be separated into four disjoint categories: explicit identifiers, quasi-identifiers, sensitive attributes and non-sensitive attributes [Fletcher, 2011, Fung et al., 2010a]. Explicit identifiers are attributes which can allow an attacker to directly identify the participant, such as driver licenses or phone numbers. Quasi-identifiers are attributes that could potentially identify a participant if the attacker possesses useful background or auxiliary knowledge [Fung et al., 2010a]. Sensitive attributes are those which are considered private and harmful in the wrong hands, such as disease or salary information. Non-sensitive attributes describe information that is already considered public knowledge (such as by being available in other public datasets), or is neither identifying nor sensitive. Explicit identifiers are almost always immediately removed, and non-sensitive attributes can usually be left as they are; how the other two types of attributes are handled depends on the data modification technique.

At first glance, one might think that removing explicit identifiers is enough to preserve privacy. Unfortunately, quasi-identifying attributes can be very powerful in enabling privacy breaches. Many other sources of data beyond any single data owner’s dataset exist in the world; even data that an attacker knows about their neighbors or friends can be used as auxiliary data! A poignant example of quasi-identifiers leading to a privacy breach was demonstrated by Sweeney [2002] when she proposed \(k\)-anonymity: she was able to uncover the medical records of the governor of Massachusetts by combining the “ZIP Code”, “Birth Date”, and “Sex” attributes in a (supposedly anonymized) medical dataset to the same attributes in the (public) voter registration dataset, thus uniquely identifying the governor. Many of the privacy models below focus on anonymizing the quasi-identifiers for this reason.

Note that the symbols in the titles of these models are the symbols used by their authors, and do not match the notation used in our thesis.

**\(k\)-Anonymity:** Harnessing the idea of generalization that we discussed in Section 2.4, \(k\)-anonymity generalizes the values of quasi-identifying attributes to prevent an attacker linking their auxiliary knowledge to a specific record in the dataset. A dataset is said to be \(k\)-anonymous if, for each distinct quasi-identifier value, there
are at least \( k - 1 \) other records that have that same quasi-identifier value [Samarati and Sweeney, 1998a, Sweeney, 2002]. This lends itself to an intuitive measure of privacy protection, with an easily understood user-defined threshold – the probability of a participant being linked to a specific record through the use of quasi-identifiers is at most \( 1/k \).

**Multi-relational \( k \)-Anonymity:** While \( k \)-anonymity focuses on anonymizing a single dataset, multi-relational \( k \)-anonymity [Nergiz et al., 2009] provides the same functionality but for larger database systems, where multiple relational tables exist.

\((X,Y)\)-Anonymity: One shortcoming of \( k \)-anonymity is that it assumes each record represents a unique individual, and does not handle multiple records belonging to the same individual. Wang and Fung [2006] address this shortcoming with their more general \((X,Y)\)-anonymity, of which \( k \)-anonymity is a special case. \( X \) and \( Y \) are defined as disjoint sets of attributes, where \( X \) are the quasi-identifiers and \( Y \) are the sensitive attributes. \((X,Y)\)-anonymity makes sure that each value in \( X \) is linked to at least \( k \) distinct values in \( Y \). This means each group of records with similar quasi-identifying values is associated with \( k \) sensitive values, making identifying a participant’s \( Y \) values based on their \( X \) values difficult. Using this definition of anonymity allows for a more flexible and uniform evaluation than \( k \)-anonymity [Wang and Fung, 2006].

**Bloom filters:** In some scenarios, such as with medical data, information about the same person needs to be linked while still preserving privacy. Probabilistic record linkage can be used to identify matching record pairs when there is no unique identifier. Of course, the data is also often encrypted due to privacy concerns. This leads to intractable computational demands, or unacceptable rates of false positives and false negatives. Bloom filters avoid these pitfalls, and offer a way to link records that contain details about the same person without breaching the privacy of the person themselves [Schnell et al., 2013].

While the above four models focus on preventing an attacker from linking a participant to a specific record (referred to as “record linkage” [Fung et al., 2010a]), an attacker does not always need that level of identification to extract information about someone. If an attacker knows that a participant belongs to a group of records that share the same quasi-identifying values (referred to as a quasi-identifier group), and that group has a high proportion of certain sensitive values, then the attacker can already infer a participant’s sensitive values with high certainty. There is no need for the attacker to narrow their search any further. A high certainty of a participant having a certain sensitive value is a privacy breach, and this is known as “attribute linkage” [Fung et al.,
Chapter 3. Literature Review

2010a]. To solve this, the correlation between quasi-identifiers and sensitive attributes needs to hidden.

**l-Diversity:** Machanavajjhala et al. [2007] proposed l-diversity as a way of preventing attribute linkage. l-Diversity states that every quasi-identifier group has to contain at least l “well-represented” sensitive values. While multiple definitions exist, the most common definition of “well-represented” is that there is a minimum threshold of distinct values for the sensitive attribute in each quasi-identifier group [Fung et al., 2010a]. In this case, that threshold is l.

**Recursive (c,l)-Diversity:** One limitation of l-diversity is that it assumes each sensitive value appears with equal frequency, which is rarely true in practice. Attempting to maintain l-diversity in this situation can lead to a large loss in data utility [Fung et al., 2010a]. Machanavajjhala et al. [2007] offered an extension to their l-diversity: recursive (c,l)-diversity. This extension makes sure that the most frequent value does not appear too often, and that the least frequent value does not appear to rarely. However, any value that is not the most or least frequent cannot be altered by recursive (c,l)-diversity.

**Confidence Bounding:** Wang et al. [2005, 2007] developed a technique aiming to bound an attacker’s confidence in guessing sensitive values in a quasi-identifier group to a user-defined threshold. Approaching the problem from this angle has two main advantages over the above “diversity” techniques. One is its intuitiveness in setting the threshold, as it can be equated to the probability of an attacker inferring a sensitive value. The second is its flexibility; it allows the data publisher to specify different thresholds for different quasi-identifier groups, and is capable of altering value frequencies other than the highest and lowest. However, a major drawback of confidence bounding is that if background knowledge is a factor, it fails to prevent attribute linkages. Recursive (c,l)-diversity does not have this problem.

**(X,Y)-Privacy:** Wang and Fung [2006] combined (X,Y)-anonymity and confidence bounding (both explained above), naming it (X,Y)-privacy.

**(k,e)-Anonymity:** For the most part, k-anonymity and its extensions are only intended for discrete attributes (or numerical attributes that are first discretized). Zhang et al. [2007] handled continuous sensitive attributes with (k,e)-anonymity, where records are partitioned into groups, where each group contains at least k unique sensitive values with a range of at least e. However, it fails to take into account the distribution of sensitive values, allowing attackers to potentially narrow down
the value to a small range within a group. This is known as a proximity attack [Li et al., 2008], and was corrected for with \((\epsilon, m)\)-anonymity [Li et al., 2008].

**t-Closeness:** Similarly to the distribution problem encountered by \((k, \epsilon)\)-anonymity, Li et al. [2007] found that \(l\)-diversity possessed the same weakness when a sensitive attribute’s distribution was skewed. To prevent skewness attacks, they proposed \(t\)-closeness, requiring the distribution of a sensitive attribute in each quasi-identifier group to be within \(t\) closeness of the attribute’s overall distribution in the dataset. Unfortunately, it possesses some weaknesses: it lacks the flexibility of tuning different protection levels for different sensitive values; and more importantly, enforcing similar distributions of attributes in all quasi-identifier groups leads to greatly reduced data utility [Fung et al., 2010a].

**Personalized Privacy:** as the name suggests, Xiao and Tao [2006a] proposed that each participant be allowed to set their own privacy level. Aside from this difference in tuning parameters, it functioned similarly to confidence bounding, and was shown to potentially have lower information loss than confidence bounding. However in practice, it is difficult to supply a participant with enough useful information to make an informed decision, leading to a high likelihood of participants “playing it safe” and selecting an unnecessarily strong level of privacy [Fung et al., 2010a].

A common theme one can extract from the above techniques is a heavy reliance on quasi-identifiers. While this is a solid principle, in practice it can be difficult to decide which attributes classify as quasi-identifiers. A quasi-identifier is any attribute that can be combined with auxiliary information to identify a participant, and this can be quite difficult to determine in many cases. A miscategorization of any attribute as being a quasi-identifier or sensitive or non-sensitive attribute will have serious effects on the security of the dataset (and utility, if an attribute is incorrectly anonymized) [Fung et al., 2010a]. While methods exist to solve this problem, such as Motwani and Xu [2007]’s attempt at determining the minimum set of quasi-identifiers, it remains an open issue.

Thus both record linking and attribute linking have imperfect, but arguably acceptable solutions. Unfortunately, an attacker does not even need to directly infer information about their victim, but can instead merely update their probabilistic beliefs about the victim based on whatever data is published. Aiming to prevent an attacker’s subsequent knowledge after accessing a dataset from differing from his initial knowledge is known as the uninformative principle [Machanavajjhala et al., 2007], and the following models represent a selection of the more prominent models in this area.

\((d, \gamma)\)-Privacy: Proposed by Rastogi et al. [2007], \((d, \gamma)\)-privacy provides preserves both privacy and data utility. It does so by bounding the difference between the initial
probabilities of the attacker and their subsequent probabilities after accessing the dataset, but must make assumptions about the attacker in order to do so. These assumptions are avoided by differential privacy [Dwork and Roth, 2013], which we discussed in Section 2.5.

**Distributional Privacy:** Blum et al. [2013] created a privacy model designed for non-interactive query models, named distributional privacy. It is based on the philosophy that a dataset drawn from a distribution should only reveal information about that distribution, and nothing else. While a much stronger privacy guarantee than differential privacy, it has narrow utility in real-world applications.

The above models all use generalization to preserve privacy. While less common, models have also been made that use noise addition instead:

**GADP:** General Additive Data Perturbation adds noise to continuous values in very careful ways, making sure to maintain statistical properties such as the variance of each attribute, and the correlations between each pair of attributes, among others [Muralidhar and Sarathy, 2005, Muralidhar et al., 1999, Sarathy et al., 2002].

**Tree-based Noise:** Islam and Brankovic [2011] proposed a noise-addition framework that, through a multi-step process, finds patterns in the data using decision trees and adds noise in a way that preserves the patterns, allowing for a tree with very similar structure to be built from the anonymized data $z$. Unlike all of the previous privacy models, this model is considering a specific workload – decision tree classification – when aiming for high utility. It makes no promises about the utility of $z$ when used for other workloads. Even though it publishes an anonymized dataset $z$, it shares some similarities with privacy-preserving data mining, where an anonymized model is built from private data. Similar strategies have also been researched by Du and Zhan [2003], Fong and Weber-Jahnke [2012] and Islam and Brankovic [2005].

Unlike differential privacy, none of the above privacy-preserving strategies are “guaranteed”; that is, they do provide privacy in a mathematically rigorous way that guarantees that every participant will have their privacy protected, and that no amount of auxiliary information will change that, now or in the future. A recent example of these weaknesses was seen in Australia, where the government released a health dataset describing one billion insurance claims since 1984 to researchers. The privacy of the people in the dataset was protected using “a suite of confidentiality measures including encryption, perturbation and exclusion of rare events” [Cowan, 2016a]. This approach received wide
criticism for not being rigorous enough, including from Australia’s Privacy Commissioner, Timothy Pilgrim [Cowan, 2016b].

The guarantee-able nature of differential privacy also means that the amount of privacy provided does not need to be empirically verified – it is mathematically proven to not disclose information about any individual that could not already be discovered without that individual’s data [Dwork and Roth, 2013]. This proof extends to cover any amount of prior or supplementary knowledge, both for present knowledge and potential future knowledge. This is not the case for the other privacy models discussed in this section; they are all susceptible to supplementary knowledge, and the level of disclosure risk needs to be empirically measured. This fundamental difference was investigated by McClure [2012], who noted “This difference makes a general mapping between the two paradigms elusive”. An exploration of how to empirically measure disclosure risk can be found in the work of Bertino et al. [2005], Domingo-Ferrer et al. [2001], Evfimievski et al. [2003] and Fung et al. [2010a].

3.2 Utility Assessment

While Section 3.1 explored the history of privacy model development, this section explores the other side of the privacy-quality trade-off: information quality. A lot of the time, simplistic measures are developed to provide an estimate of the information quality, or statistical techniques are borrowed from the SDC (Statistical Disclosure Control) community. While robust, these evaluation techniques often fail to capture the nuances that can be present when evaluating specific anonymization tasks, such as generalization. Utility measures that target specific anonymization tasks solve this problem, however comparing the results of different measures is an ongoing problem. If two datasets are anonymized with two different techniques, and each technique requires its own utility measure, comparing the quality of the datasets can be problematic [Bertino et al., 2008, Guo et al., 2010, Kifer and Gehrke, 2006].

In PPDP (privacy-preserving data publishing), the information quality of an anonymized dataset is most often evaluated by measuring the similarity between the anonymized dataset $z$ and the original dataset $x$. If the dataset could be used for a variety of reasons and there is no single workload in mind, the dataset is evaluated in a way that applies to any scenario – we refer to this as measuring the “dataset quality” or “dataset information loss”. These types of techniques are discussed in Section 3.2.1.

Alternatively, if the workload of the dataset is specific and known, the information quality can be measured with respect to that workload. PPDM (privacy-preserving
data mining) focuses on this type of data, where the quality of the dataset itself is less important than the quality of the outputted data mining results produced from the dataset. Common workloads are classification and clustering [Fung et al., 2010b], discussed in Section 2.2. Many patterns in a dataset can be lost after anonymization, even if the dataset itself appears to retain most of its statistical information [Islam and Brankovic, 2011, Islam et al., 2003, Lim et al., 2000]. For this reason, information measures have been designed that specifically look at the effect of anonymization on data mining results, and we discuss these in Section 3.2.2. We call this type of information quality, “data mining quality” or “data mining information loss”.

It should be noted that we make a distinction between “information loss” and “information quality” due to the implied comparative nature of the word “loss” – this section focuses on measures that compare a dataset before and after anonymization. “Information quality” could refer to this before-and-after comparison, but also to the quality of an isolated dataset (with no comparison). “Information loss” provides more specificity. Measures of information loss are also usable in scenarios outside of privacy preservation, such as data cleaning\(^1\). In these instances, information gain is the goal.

### 3.2.1 Dataset Information Loss

Dataset information loss refers to the loss of useful information in the dataset itself. Statistical tests are one way of measuring this, but additional insight can be gained through more targeted measures. Some measures target the quality of the user-defined QID groups (short for quasi-identifier groups, introduced in Section 3.1), while others target data modifications via generalization and suppression (Section 3.2.1.2).

#### 3.2.1.1 QID Group Quality

*Discernibility Metric* (DM) [Bayardo and Agrawal, 2005, Skowron and Cecylia, 1992] functions by penalizing each record for how many other records it is indiscernible from in each QID group, compared to the original dataset. A QID group is defined as any collection of records that have the same values for all the quasi-identifying attributes. If a record belongs to a QID group with \(n\) records, then the penalty for that record is \(n - 1\) – it is indiscernible from \(n\) records with respect to the attributes in the QID group. This naturally leads to considering the penalty per QID group, rather than per record: each QID group incurs a penalty of \(n^2\). Interestingly, it is the conceptual opposite of \(k\)-anonymity [Samarati and Sweeney, 1998a, Sweeney, 2002] – a well-known privacy

---

\(^1\)“Data cleaning” refers to estimating missing values in a dataset and removing misinformation/noise [Rahman and Islam, 2013].
technique that requires a user-defined minimum number of indistinguishable records per QID group.

DM is a commonly-used measure [Bayardo and Agrawal, 2005, LeFevre et al., 2006, Machanavajjhala et al., 2007, Xu et al., 2006] despite its inability to consider the data distribution. As is often the case, the lack of a single robust information measure has led many to adopt an ensemble approach, with multiple measures each addressing something missed by the others [Guo et al., 2010, Machanavajjhala et al., 2007, Nergiz and Clifton, 2007]. It is worth noting that ignoring the data distribution is a common shortcoming of information measures, and including statistical evaluations such as KL-divergence [Guo et al., 2010, Kifer and Gehrke, 2006, Machanavajjhala et al., 2007], chi-squared distance [Giggins, 2012, Guo et al., 2010] and covariance comparisons [Aggarwal and Yu, 2004] are possible solutions. Statistical tests are also often the solution to measuring information loss caused by noise, or changing discrete values to other values with a certain probability (e.g. “England” to “Australia”) [Evfimievski, 2002, Guo et al., 2010, Islam, 2007, Islam and Brankovic, 2011, Kargupta et al., 2003, Liu et al., 2006]. In addition to those mentioned above, common tests include regression analysis, mean square error and contingency tables. For further information on these tests, we refer the reader to Domingo-Ferrer et al. [2001], Hair et al. [1998].

3.2.1.2 Generalization and Suppression

Throughout most information measurement literature, the assumption is made that for the purposes of quality evaluation, suppression can be considered as generalizations that generalize a value to its most vague state [Bertino et al., 2008, Iyengar, 2002, Kifer and Gehrke, 2006, Nergiz and Clifton, 2007, Samarati, 2001, Samarati and Sweeney, 1998b, Sweeney, 1997, 2002, Wang and Fung, 2006, Xiao and Tao, 2006b]. We maintain this assumption, and hereafter only refer to generalization.

Minimal Distortion (MD) [Samarati, 2001, Sweeney, 1997, 2002] (or generalization height [Bertino et al., 2008, Kifer and Gehrke, 2006, LeFevre et al., 2005]) is a penalty-based system where whenever a value (for one record) is generalized, the distortion count is incremented. MD harnesses the taxonomy trees of attributes (also called domain generalization hierarchies [Nergiz and Clifton, 2007]), in which each value of an attribute is a leaf in the tree, and the higher nodes represent collective terms for their child nodes (and are thus more vague). Figure 3.1 is a simplified example of a taxonomy tree\(^2\), with the number in each leaf referring to the frequency of the discrete values appearing in the dataset. MD treats each level of generalization separately – if a value is generalized

\(^2\)Note that a taxonomy tree is not a decision tree. The only similarity between them is the basic structure.
Chapter 3. Literature Review

Figure 3.1: A generalization taxonomy tree of an attribute.

As an example, take Figure 3.1: if all instances of value F were generalized to A (a user-defined term that collectively describes B and C), then 50 records have moved up two levels, resulting in 100 units of distortion. Additionally, however, most algorithms [Giessing, 2004, Iyengar, 2002, LeFevre et al., 2005, Sweeney, 1997, 2002, Wang et al., 2004] do not allow multiple levels of generalization for an attribute to co-exist in a dataset (e.g. a record cannot have a value of “apple” if another record has a value of “fruit” for the same attribute), as this would cause problems with data mining algorithms. Therefore if F is generalized to A, so too are G and H, bringing the total distortion up to 50 + 100 + 150 + 300 = 600. If they were only generalized to C, \( MD = 50 + 100 + 150 = 300 \).

Along with MD, Iyengar’s loss metric (LM) [Iyengar, 2002] marked the first work in specifically targeting the information loss caused by generalization. LM is defined as the number of nodes a record’s value has been made indistinguishable from (via generalization) compared to the total number of original leaf nodes in the taxonomy tree. This is repeated for each record for the attribute in question, and each attribute’s loss is the average over all records.

For example in Figure 3.1, generalizing the 50 records in F to C (which collectively describes F, G and H) would result in the value of 50 records (for one attribute) being indistinguishable from 2 other values (nodes). With 5 leaf nodes in the taxonomy tree, \( LM = \frac{2}{5} \) for those 50 records in regards to the attribute shown in Figure 3.1.

For continuous attributes, if a value (for example, “5”) is generalized (e.g. “3−7”), LM compares the size of the generalized domain (e.g. 7−3 = 4) to the total domain size of the attribute (e.g. if the domain is [1,10], then the domain size is 9), giving a
The final result of $\frac{4}{9}$ in this example. The LM result for each attribute is averaged over all records. The loss of each attribute is then summed together, giving a final LM result. Unfortunately, LM does not take the distribution of the data into consideration [Kifer and Gehrke, 2006].

While both MD and LM take into account how many records are affected by a value being generalized, a major downside is that they treat each generalization as equally damaging [Machanavajjhala et al., 2007]. When considering discrete values and collective terms for discrete values, it is unlikely that a user (even an expert) could design each generalization in the taxonomy to have a real-world equivalence to each other. For example, something like “state $\rightarrow$ country” could have a far bigger impact on a dataset than “birth month $\rightarrow$ birth year”, and even with modification it could still never truly be equivalent. Defining and measuring differences in generalizations is an open question. Perhaps weighting each generalization based on changes in the information gain (using whatever measure fits the needs of the anonymization expert) of the dataset is a possibility.

ILoss [Xiao and Tao, 2006a] takes the same approach as LM when measuring the information lost to generalizations. It measures the fraction of domain values lost for an attribute by each generalization, just as LM does, and gives the same results for each generalized value as LM would. It differentiates itself by allowing each attribute to also possess a weighting, allowing for the major disadvantage of MD and LM discussed above to be partially solved. While each generalization is not treated differently, at least each attribute is treated differently based on their user-defined importance. The ILoss values of each record (taking into account the attribute weightings) are then averaged across the whole dataset, resulting in a final ILoss result.


Generalizations for continuous attributes can be found by finding the optimal binary split for the domain that maximizes the information gain (here, “information gain” is measured using algorithms commonly found in decision trees – see Section 2.3.1.1) [Fung et al., 2005]. Splitting the domain can be repeated until the desired number of generalization levels is achieved, for example a domain $[1, 10]$ might be split into $[1, 6]$ and $[7, 10]$, with $[1, 6]$ being further split into $[1, 3]$ and $[4, 6]$. Thus a value of “2” would now be “1−3” and a value of “7” would now be “7−10”. Alternatively continuous attributes
can be generalized using clustering techniques such as $iK$-Means, where adjacent values with high frequency are grouped together [Cordeiro de Amorim and Mirkin, 2012].

Discrete attributes prove much more difficult to automatically generalize due to the lack of a natural ordering. A possible solution is to dynamically combine “appropriate” categorical values together – for example, “apple” and “banana” could be generalized to a value called “apple_or_banana” [Nergiz and Clifton, 2007]. “Appropriateness” can be defined using any similarity measure at the discretion of the anonymization expert.

Not only is user-input vulnerable to human error, but even a perfectly reasonable taxonomy is a commitment that removes all other interpretations. It may seem intuitive to generalize “apple” and “banana” to “fruit”, but what if more information (or relevancy to a specific task) could be retained by sorting by sugar content or price or color? User-defined taxonomies create or strengthen certain semantic meanings, while destroying or weakening others [Nergiz and Clifton, 2007]. This can result in the anonymization expert making a drastic data mining decision, which is usually outside their job description.

Parallel to these measures of quality is an important concept that should be considered when debating the differences between generalization and randomization: “faithfulness” [Kifer and Gehrke, 2006]. Faithfulness, or truthfulness, refers to how confident a data miner can be about the quality of anonymized data, or whether it could be noisy (we discussed this briefly in Section 2.4). Trottini called this “perceived data utility” [Trottini, 2001, 2003] and warned of the dangers of false confidence – what if a doctor or federal security agency acts on an anonymized record that they falsely believe to be accurate? Generalization offers an advantage over randomization in this case: it can guarantee that each record is still as truthful as it was in the original dataset.

### 3.2.2 Data Mining Information Loss

“Data mining information loss” involves comparing the data mining results of an original dataset to the results of an anonymized version. Since the output of various data mining techniques differs greatly, targeted information measures are required. Guo et al. [2010] described this phenomenon: “utility of any dataset, whether randomized or not, is inherently dependent on the tasks that one may perform on it. Without a workload context, it is difficult to say whether a dataset is useful or not”. Data mining quality and dataset quality are not mutually exclusive – often both are tested for, and the empirical results support the differentiation between dataset quality and data mining quality [Aggarwal and Yu, 2004, Iyengar, 2002, Nergiz and Clifton, 2007]. The most common data mining techniques are clustering and classification, and these will be addressed below in turn.
3.2.2.1 Clustering

Thus far, the quality of clustering results has proven difficult to robustly capture due to the absence of a strict definition of clustering and a reference point for evaluating the results. The usefulness of a clustering result can easily vary depending on the purpose of clustering. Fung et al. [2008] described the problem succinctly: “the anonymity problem for cluster analysis does not have class labels to guide the generalization. It is not even clear what “information for cluster analysis” means and how to evaluate the quality of generalized data in terms of cluster analysis”.

This impacts cluster analysis in two ways. Firstly, it makes ensemble approaches even more vital due to each technique measuring a different aspect of the data mining results [Rahman and Islam, 2012]. Secondly, it makes it difficult to identify a direct loss of information when going from an original clustering to an anonymized clustering (i.e., the clustering result from the anonymized data). Therefore, “cluster information loss” is often defined as the difference in results from clustering evaluations when applied to the original clustering, and then separately applied to the anonymized clustering.

Common metrics used and included in ensembles are: Rand index [Rand, 1971], F-measure [van Rijsbergen, 1979], Fowlkes-Mallows index [Fowlkes and Mallows, 1983], Davies-Bouldin index [Davies and Bouldin, 1979], and Silhouette [Rousseeuw, 1987]. Many of them rely on the concepts described by Figure 3.2 – type I errors (false positives) and type II errors (false negatives). When executing a query or filtering records based on a decision rule, false positives (FP) refer to records that were retrieved but shouldn’t have been. False negatives (FN) refer to records that were not retrieved, but should have been. True positives (TP) and true negatives (TN) are the opposite: the records that were appropriately handled.

In order to apply these principles to a clustering scenario, a reference point is required that validates the obtained results (i.e. retrieved data) as either relevant or irrelevant. Here, “retrieved” is defined as “the records present in the cluster being assessed” and “relevant” is defined as “the records correctly belonging to that cluster”. A class attribute can serve as a reference point to evaluate the clustering results. Typically, the class attribute is removed from the dataset prior to the application of a clustering algorithm. Once the clustering is complete, the class values are reassigned to the records. The most common class value in a cluster is used to define records as either relevant or irrelevant: if a record has any class value other than the majority value, it is irrelevant [Han et al., 2006, Rahman and Islam, 2012]. A reference point is known as “external information”, since in real life scenarios a clustering algorithm is typically applied to
datasets that do not have a natural class attribute. Therefore, clustering information metrics that use external information are known as external metrics [Tan et al., 2005].

One such external metric is the \( \text{Rand index} \) (RI) [Rand, 1971]. It simply measures the fraction of correctly clustered records:

\[
RI = \frac{TP + TN}{TP + FP + FN + TN} \tag{3.1}
\]

Unfortunately it treats false positives and false negatives as being equally undesirable, which is sometimes not the case. For example, a security agency would much rather deal with the inconvenience of a false positive than the security breach caused by a false negative.

\( F\text{-measure} \) [van Rijsbergen, 1979] provides an easy solution to the weakness in RI, and is one of the most common clustering evaluation tool used by the PPDP and PPDM communities [Fung et al., 2008, Rahman and Islam, 2012]. It uses two expansions of the concepts described in Figure 3.2: precision and recall. Precision measures the fraction of retrieved results that are relevant compared to irrelevant:

\[
P = \frac{TP}{TP + FP} \tag{3.2}
\]

Recall measures the fraction of relevant results that were successfully retrieved:
\[ R = \frac{TP}{TP + FN} \] (3.3)

Using these concepts, F-measure can be defined as:

\[ F = \frac{2 \times P \times R}{P + R} \] (3.4)

And thus, it acts as a weighted average (harmonic mean) of the precision and recall. When treating false positives and false negatives differently, an expansion of the formula is used:

\[ F_\lambda = \frac{(\lambda^2 + 1) \times P \times R}{(\lambda^2 \times P) + R} \] (3.5)

Where \( 0 \leq \lambda \leq 1 \). When \( \lambda = 0 \), \( F_0 = P \), and recall has no impact on the F-measure result. F-measure is sometimes called “F_1 score”, referring to the common case of \( \lambda = 1 \). When \( P \) and \( R \) are expanded, the formula can be written as:

\[ F_\lambda = \frac{(\lambda^2 + 1) \times TP}{(\lambda^2 + 1) \times TP + \lambda^2 \times FN + FP} \] (3.6)

Thus \( w = 0 \) equates to false negatives holding no weight.

An alternative to F-measure is the *Fowlkes-Mallows index* (FMI) [Fowlkes and Mallow, 1983], otherwise known as G-measure. While F-measure is the harmonic mean of precision and recall, FMI is the geometric mean and is defined as:

\[ FMI = \sqrt{P \times R} \] (3.7)

There also exist a number of internal metrics that evaluate cluster quality without requiring a reference point. These metrics generally evaluate the results based on how compact each cluster is and how separated the clusters are from each other. One such metric is the *Davies-Bouldin index* (DBI) [Davies and Bouldin, 1979]. It defines a good clustering result as having low intra-cluster distances (i.e., high compactness) and high inter-cluster distances (i.e., high separation):

\[ DBI = \frac{1}{k} \sum_i \max_{i \neq j} \frac{d_i + d_j}{D(c_i, c_j)} \] (3.8)
Where $\Phi_i$ and $\Phi_j$ are two different clusters out of $k$ clusters, $\phi_i$ (and similarly for $j$) is the centroid of cluster $\Phi_i$, $d_i$ is the average distance of records $r \in \Phi_i$ to the centroid $\phi_i$, and $D(\phi_i, \phi_j)$ is the distance between $\phi_i$ and $\phi_j$.

Silhouette [Rousseeuw, 1987] measures how much more appropriate a record’s cluster is compared to its second-most appropriate cluster:

$$s(r) = \frac{b(r) - a(r)}{\max(a(r), b(r))}$$  \hspace{1cm} (3.9)

$$S = \frac{\sum_r^n s(r)}{n}$$  \hspace{1cm} (3.10)

Where $a(r)$ is the average distance of record $r$ from each other record in the same cluster $\Phi_i$, and $b(r)$ is the lowest average distance of $r$ from all the records of one other cluster $\Phi_j; \forall j \neq i$. Therefore $s(r)$ represents the “appropriateness” of record $r$’s chosen cluster, and $S$ is how appropriately all the records have been clustered. $s(r)$ ranges from $-1$ to $1$, with $0$ meaning that record $r$ is on the border of two clusters, and a negative value meaning that $r$ might be better off in its neighboring cluster. By comparing the result of these techniques before and after anonymization, one can make a more informed judgment on whether the clustering information has been preserved. An advantage of RI, F-measure and FMI is that they are simply reinterpretations of the same analysis, and so no extra computational time would be required if all three were to be calculated. DBI and Silhouette are clearly more computationally complex, but are arguably more explanatory and do not require reference points.

### 3.2.2.2 Classification

One of the main purposes of classification is to predict the value of a certain attribute for future records, where the values are not known. A “classifier” is any model that takes unlabeled data as input, and outputs a predicted label using some decision-making process. A decision tree is a good example of a classifier, and is discussed in detail in Section 2.3. The most common technique for measuring the information quality of a classifier is prediction accuracy [Aggarwal and Yu, 2004, Fung et al., 2005, 2007, Islam and Brankovic, 2011, Islam et al., 2003, Nergiz and Clifton, 2007, Wang et al., 2004, 2005], which measures the rate at which future records have their class values correctly predicted by a classifier. This is done by hiding the class value of some records not used when building the classifier (i.e., test data), and seeing if the class values are correctly predicted. Technical details can be found in Section 2.2.2.
One way of presenting the results is to invert the accuracy so it represents prediction error, and compare this to the baseline error [Fung et al., 2007, Wang et al., 2004, 2005]. Baseline error is the prediction error of the classifier when no anonymization has occurred. Another type of error is also sometimes used: worst error [Wang et al., 2004] or removal error [Wang et al., 2005]. This error can be defined in any way which represents a relevant worst-case scenario. For example, when every attribute in the QID group is suppressed (or generalized to the root of the taxonomy tree, e.g. “anything”) [Wang et al., 2004]; or when every sensitive attribute is removed from the dataset before a classification algorithm is applied [Wang et al., 2005].

These additional measures allow for further knowledge to be gained about the data mining information quality. As an example: the difference between the baseline error and worst (removal) error provides insight into the importance of the QID attributes in the classifier. A small difference would imply that the attributes do not influence the ability of the dataset to predict future cases. Perhaps some of the attributes can even be completely removed before publication if their utility (usefulness) does not warrant the privacy risk.

Unfortunately, prediction accuracy has some weaknesses. By simply measuring the percentage of records that have their class value correctly predicted, it fails to take into account any changes to the decision rules (patterns) or structure of a decision tree [Islam et al., 2003]. Sometimes an anonymized dataset can differ from the original dataset enough to result in significant structural differences between the trees obtained from the datasets, even if both trees have similar prediction accuracies [Islam et al., 2003, Lim et al., 2000]. If the original patterns discovered through classification are weakened or destroyed by the anonymization process, but other – potentially misleading or artificial [Bertino et al., 2008] – patterns are discovered, it is easily possible for the prediction accuracy to stay high, or even surpass the accuracy of the original classifier.

Some argue prediction accuracy is all that matters for a classifier, and if anonymization causes new patterns to be discovered and increases prediction accuracy then so much the better [Fung et al., 2005, 2007, 2010b, Wang et al., 2005]. However this represents a data mining decision made by the anonymization expert, and prevents data miners from exploring that possibility after the anonymized dataset is published. This is especially dangerous when considering that the alternate patterns may be artificial (fake), and not discovered in the original data because they do not exist [Bertino et al., 2008].

Empirically this is supported by other measures sometimes disagreeing with the results of prediction accuracy [Islam and Brankovic, 2011, Islam et al., 2003, Lim et al., 2000, Nergiz and Clifton, 2007]. One of the strongest advantages of classification trees is that they provide humanly-readable patterns that can then be acted on or investigated.
In other words, the decision rules themselves are valuable to a data miner, not just the predictive power of the classifier as a whole. Relying on prediction accuracy alone unnecessarily narrows the utility of the classifier. Unfortunately many do solely rely on prediction accuracy when measuring data mining information loss [Fung et al., 2005, 2007, Wang et al., 2004, 2005].

Some alternatives to prediction accuracy that can be used in the same scenarios are F-measure (which we described above in the context of clustering; it can also be used in classification) and measuring the area under the ROC curve. The ROC (short for “Receiver Operating Characteristic”) curve describes the trade-off between TP (benefits) and FP (costs). Often it is plotted on axes with the TP Rate \( \frac{TP}{TP+FN} \) as the y-axis and the FP Rate \( \frac{FP}{FP+TN} \) as the x-axis. The area under this curve, known as AUC [Hanley and McNeil, 1982], represents the probability that a classifier is more likely to predict a positive label as positive than to predict a negative label as positive. While neither AUC [Hand, 2009, Lobo et al., 2008] nor F-measure [Hand, 2017] are ideal when comparing different classifiers, they both more effectively handle skewed distributions than prediction accuracy does. AUC and F-measure are most appropriate when the class attribute is binary (i.e., \(|C| = 2\)), and where \( c_1 \in C \) is the “important” class label, or the class label that the user is trying to correctly predict, and \( c_2 \in C \) is unimportant. These labels are often referred to as the “Positive” and “Negative” labels, respectively.

Another classification quality measure is information-gain-to-privacy-loss ratio (IGPL) [Fung et al., 2005, 2007, Wang et al., 2005]. This measure differs from previous measures in that it is considered a trade-off measure, or a search measure. A search measure is actively used during the anonymization process to guide it in sacrificing as little information as possible while gaining as much privacy/security as possible [Duncan et al., 2001, Fung et al., 2010b].

In this instance of a search measure, IGPL is used as the filtering algorithm during decision tree construction, replacing the usual measures such as gain ratio (see Section 2.3.1.2). Unlike an ordinary decision tree, the Fung et al. [2005] propose a strategy they call “top-down specialization”, where they first generalize all attributes to their most vague state, and then using the decision tree to choose which attributes to make more specific (in other words, specialized – the opposite of generalization). The concept is surprisingly simple and effective: rather than defining a node split by the best information gain (IG), it can be defined by the best trade-off between information gain and privacy loss (\( \frac{IG}{PL} \)). Here, IG and PL can be defined as any information measure and any privacy measure that the anonymization expert feels appropriate. If an attribute is calculated to provide a large increase in IG and a low increase in PL, it is likely to be chosen as the filter for a node, and thus specialized.
A possible extension that remains unanswered is for IGPL to handle several different attributes being used as class attributes in order to maintain additional decision rules [Bertino et al., 2008, Kifer and Gehrke, 2006]. We suggest an exploration into decision forests, using good alternate IGPL trade-offs in each tree, and perhaps weighting attributes.

3.2.3 The Big Picture

In the pursuit of quality, it is important that we can define and measure what “quality” really is. Due to anonymization, measures have been developed over the years to measure changes in the quality of a dataset, and this is likely to become increasingly important as society becomes ever-more data-driven and privacy-aware [Gellman and Poitras, 2013].

It has been shown that different scenarios require different approaches to information quality evaluation, and it is important that an anonymization expert is confident in selecting the most applicable measures. We have discussed a variety of methods for quality analysis, and the scenarios they are best suited for. Ensemble approaches are also important to keep in mind: in many cases, a single measure cannot robustly evaluate the utility of a dataset in all scenarios.

Another possibility is for information measures to be used as search measures. By guiding the anonymization process towards the optimal solution (as defined by the measure), it removes the need for iterative testing with estimated parameters. This can be useful for anonymization techniques or information measures with high computational cost, where heavy testing might be infeasible. Some algorithms are more computationally complex than others; but in many cases this should not be considered a deciding factor. Often the goal is to create one anonymized version of a dataset for a single release, and this is rarely time-sensitive. Here, maximizing data quality is far more important than computational cost.

Comparing the effect of different types of anonymization techniques is still an open question. Perhaps the only solution is to judge the techniques based on their principles, rather than their empirical results. For example, the faithfulness of generalization might outweigh the benefits of randomization, such as not relying on user-defined attribute taxonomies.

The common trade-off in PPDP and PPDM is privacy vs. utility, and measuring utility has received much less attention. Further research would prove beneficial for not only identifying effective anonymization techniques, but also for better understanding the factors that affect dataset quality and data mining quality.
3.3 Differentially-Private Decision Trees

In Part III we propose several strategies for building differentially-private decision tree models. Before demonstrating how we improve the state-of-the-art, we first contextualize our work within the current state-of-the-art in this section.

Several previous works have surveyed the conflict between data mining and differential privacy [Ji et al., 2014, Sarwate and Chaudhuri, 2013]. We differentiate ourselves in this chapter from their work by focusing on decision trees in particular, going into greater depth on the intricacies of differentially-private decision tree algorithms than is possible in a more general survey. Sarwate and Chaudhuri [2013] provided a general overview of machine learning with differential privacy, briefly discussing classification, regression, dimensionality reduction, time series, filtering, and a suite of some of the basic “building blocks” of differential privacy, such as the differences between input, output and objective perturbation, the Exponential mechanism, and robust differentially-private statistics. Ji et al. [2014] focused more on specific differentially-private data mining algorithms, providing an overview of the work done with naive Bayes models, linear regression, linear support vector machines [Rubinstein et al., 2012], logistic regression, kernel support vector machines [Jain and Thakurta, 2013], decision trees, online convex programming, clustering [Chen et al., 2015], feature selection, PCA [Chaudhuri et al., 2013], and statistical estimators.

3.3.1 Accessing Differentially-Private Data

Part of the privacy budget $\beta$ needs to be spent whenever the private data is queried. This means it is important to identify when exactly the data needs to be queried in order to build a decision tree. The less times the data needs to be queried, the less budget we need to spend, or the more budget we can spend per query. Sometimes the data will not necessarily need to be queried, but doing so will still be worth the privacy cost due to how much better the classifier performs because of it. Spending part of the budget on optional queries is explored in Section 3.3.8.

Which queries should be considered “compulsory” depends on one major characteristic of the tree-building algorithm: whether it builds the tree greedily or randomly. By “greedily”, we refer to an algorithm using an objective function locally in each node; heuristically finding the best attribute to divide the local subset of data contained in a node (see Section 2.3.1). No matter what objective function is used (be it information gain, gini index, or any other), it requires querying the local data at least once. We refer to these as “non-leaf queries”, since they are performed in each non-leaf node in a tree,
and explore them in detail in Section 3.3.2. The other compulsory query for greedy trees is the same as the only compulsory query for random trees. By “random” we refer to algorithms that do not use an objective function in each node, and instead pick attributes randomly (thus having no need for the data; see Section 2.3.2). The compulsory query that both categories of decision trees share is a query that enables the prediction of class labels for unseen, unlabeled data. In almost all cases, this takes the form of querying the distribution of class counts in each leaf node, with the most common (i.e., majority) class label being used as the predicted label for future records (see Section 2.3). We refer to these queries as “leaf queries” and discuss them in Section 3.3.3.

### 3.3.2 Non-Leaf Queries

Deciding which attribute to split a node with to optimize its ability to discriminate between class labels is at the core of a greedy decision tree algorithm. It is what Blum et al. [2005] focused on during differential privacy’s inception in 2005, where they proposed a simple proof-of-concept decision tree algorithm. This algorithm demonstrated how a traditional non-private algorithm could be converted to achieve differential privacy, by rephrasing the splitting function in terms of queries that could be made differentially-private. Specifically, Blum et al. [2005] made Information Gain (see Equation 2.2) differentially-private by breaking it down into two counting queries for each attribute $a$, and then making the counting queries differentially-private with the Laplace mechanism (Section 2.5):

**Blum et al.’s Query 1:** $n_i^{v,c} + \text{Lap}(1/\epsilon); \forall c \in C, \forall v \in A$ and

**Blum et al.’s Query 2:** $n_i^v + \text{Lap}(1/\epsilon); \forall v \in A$.

where $n_i$ is the number of records (i.e., support) in node $i$, $n_i^v$ (and $n_i^{v,c}$) is the number of records in $i$ with value $v$ (and class label $c$), and $\epsilon$ is the privacy budget spent on that query. With these two queries, Equation 2.2 can then be calculated client-side. Differentially-private outputs are immune to post-processing privacy breaches, so nothing more needs to be done to satisfy differential privacy in the non-leaf nodes.

It is, however, a costly approach to take when considering the privacy budget. If we consider the user’s total privacy budget to be $\beta$, then the two queries listed above can only receive a small fraction of $\beta$ each. When querying any given attribute $A$ with the above queries, the counts for each combination of value $v$ and label $c$ involves a disjoint subset of $x_i$ (i.e., the data in node $i$), allowing Query 1 to be composed in parallel $\forall c \in C, \forall v \in A$ (Definition 2.3). Query 2 uses the same data as the first query though,
and so its privacy cost is summed with the cost of the first query (Definition 2.2). This then needs to be repeated for every attribute $A$, at every level of the tree (sibling nodes contain disjoint subsets of $x$ and can be composed in parallel). When all is said and done, each query in the algorithm has a fraction of the total privacy budget $\beta$ equal to:

$$\epsilon = \frac{\beta}{2md},$$

where $m$ is the number of attributes and $d$ is the tree depth (including the root and leaf levels). This expenditure of the budget is inefficient for several reasons. One inefficiency can be immediately seen when looking at the two queries listed above; the second query could be calculated by simply summing the outputs from the first query! That is, $n_i^v = \sum_{c \in C} n_i^{v,c}$, allowing the user to skip the second query entirely. This would involve summing the $\text{Lap}(1/\epsilon)$ noise added to each $n_i^{v,c}$ count, but this actually results in lower noise due to a lot of the noise “canceling out” (explored in Chapter 7). We explore the idea of summed noise canceling itself out in Section 3.3.5, and further budget-saving strategies in Section 3.3.8.

Converting a splitting function into differentially-private counting queries is fortunately not the only approach one can take when building a greedy tree. Rather than submitting counting queries via the Laplace mechanism (Definition 2.4), one of the major advances Friedman and Schuster [2010] made over the preliminary work done by Blum et al. [2005] was to submit a single query to find the best splitting attribute in each node via the Exponential mechanism (Definition 2.5).

Friedman and Schuster [2010] proposed building a decision tree by querying the dataset twice at each node:

**Friedman and Schuster’s Query 1:** first, get a count of how many records are in the node, followed by

**Friedman and Schuster’s Query 2:** find the best attribute to split the records with.

These two non-leaf queries are also used by Patil and Singh [2014] and Rana et al. [2016] in their greedy tree algorithms, which we discuss later in this section. For any particular level (i.e., depth) of the decision tree, all the nodes contain disjoint subsets of the dataset, allowing for parallel composition to be used (Definition 2.3). The privacy budget spent on the two parallel queries at each level of the tree can then be summed using composition (Definition 2.2). This means that for a given total privacy budget $\beta$,

\footnote{Note that strictly speaking, the leaf nodes only require one query; the total class counts in each leaf node. We discuss leaf queries in Section 3.3.3}
each query submitted to the dataset has a portion of the budget equal to

\[ \epsilon = \frac{\beta}{2d}, \]

an \( m \)-fold improvement over the proof-of-concept algorithm offered by Blum et al. [2005]. Note that the first query (i.e., the support of a node) is only used as part of the algorithm’s termination criteria (Section 3.3.4) and for pruning (Section 3.3.5), and is not discussed here.

Using the Exponential mechanism, the best attribute to split records in a node with can be outputted with high probability. The probability of any given output being chosen by the Exponential mechanism depends on the scoring function, which in this case is the splitting function. Common splitting functions are information gain [Quinlan, 1993], gain ratio [Quinlan, 1996], and gini index [Breiman et al., 1984]. Friedman and Schuster analyzed each of these; not in terms of utility performance (which has been done many times in non-private scenarios [Islam, 2012, Katz et al., 2012, Li and Belford, 2002]), but in terms of how sensitive they were to individual records. The sensitivity of a function directly impacts how much noise needs to be added to achieve differential privacy. In their experiments Friedman and Schuster found that due to its low sensitivity, using max operator [Breiman et al., 1984] (see Section 2.3.1) as their splitting function achieved the highest prediction accuracy, despite it being the least sophisticated of the tested functions. Their findings can be seen in Figure 3.3. This demonstrates that taking a function that performs well in non-private scenarios and making it differentially-private does not necessarily continue to perform well in private scenarios. Gini index also achieved competitive results, having much lower sensitivity than Information Gain.

Interestingly, Friedman and Schuster [2010] were unable to test Gain Ratio due to its erratic behavior when the denominator of the ratio approaches zero. Quinlan’s C4.5 algorithm [Quinlan, 1996] solved this problem with additional heuristics, but ultimately the sensitivity of Gain Ratio cannot be bounded and thus cannot be implemented with the Laplace mechanism or the Exponential mechanism.

### 3.3.2.1 Continuous Attributes

Another contribution made by Friedman and Schuster [2010] was a preliminary exploration into using splitting functions on continuous attributes. Using an attribute value from among the records in a node is a breach of privacy, so another solution is needed. Friedman and Schuster propose using the Exponential mechanism to select the best range of attribute values for splitting, where all the values in the range output the same
score from the splitting function. A data-independent value can then be uniformly randomly selected from the chosen range. Unfortunately, this approach requires a far higher number of queries than simply selecting the best discrete attribute to split a node with, requiring the privacy budget to be divided such that

$$\epsilon = \frac{\beta}{(2 + n)d}$$

(3.11)

where $n$ is the number of continuous attributes. For even small numbers of continuous attributes, this is a substantial increase in the amount of noise added to each query. To the best of our knowledge, there has not been a more budget-efficient method proposed for selecting the splitting point of continuous attributes in a (strongly) differentially-private way, other than discretizing the attribute first. This is a weakness that would need to be addressed before greedy heuristics such as splitting functions can be considered a viable way to build a differentially-private classifier when continuous attributes are involved.

Rana et al. [2016] explored one possible way of efficiently finding a continuous splitting point, but it required weakening the definition of differential privacy. For a given continuous attribute $x$, rather than spending a large amount of $\beta$ to find the optimal splitting value like Friedman and Schuster, the authors instead find the average $x$ value for all the records in that node that have class label $c_1$, and similarly for $c_2$. The chosen splitting value is defined as halfway between those two average values. While this approach does
not maximize the discriminatory power of the child nodes, it increases the budget per query from Friedman and Schuster’s $\epsilon = \frac{\beta}{(2+n)d}$ to

$$\epsilon = \frac{\beta}{3d},$$

thus reducing noise.

To make this query (weakly) differentially-private, Rana et al. [2016] add Laplace noise $\text{Lap}(\Delta/\epsilon)$, where the sensitivity $\Delta$ is based on the smaller of the two class counts, $\Delta = 3/\min(n_{c_1}, n_{c_2})$. Instead of using smooth sensitivity or a strictly-correct global sensitivity, Rana et al. opt to assume that the continuous attributes have a normal distribution and estimate the sensitivity within 3 standard deviations, which is adequate in 99.7% of cases (assuming the assumption holds). While not strictly differentially private, Rana et al. keep in the spirit of their weakened definition of differential privacy (analyzed in detail in Section 3.3.6) and instead heuristically add an amount of noise that could be considered “adequately private” in most scenarios. The assumption they choose to make is somewhat self-defeating however, as it negates one of the well-known advantages of decision trees; their non-parametric independence from the underlying distribution of the data.

### 3.3.3 Leaf Queries

The information required from the data in a leaf node is quite different from the information required in a non-leaf node. Rather than learning the best attribute to partition the data with, the purpose of a leaf node is to predict the class label of unlabeled records that are filtered to the leaf. A different query from the ones discussed in Section 3.3.2 is therefore required; one related to the class labels of the training records in the leaf node.

A counting query is the most straight-forward solution, and is the solution used by almost every differentially private decision tree algorithm that we know of [Blum et al., 2005, Friedman and Schuster, 2010, Jagannathan et al., 2012, Mohammed et al., 2015, Patil and Singh, 2014, Rana et al., 2016]. Conceptually, one can consider the query as a single query submitted in parallel to all $j$ leaf nodes, returning a histogram of the class counts in those nodes:

$$\{n^c_j + \text{Lap}(1/\epsilon); \forall c \in C\}; \forall j.$$  

Alternatively, one can recognize (in much the same way that [Dwork et al., 2006] originally discovered) that since each count in a histogram is disjoint from the other counts,
each class count in each leaf is simply being queried in parallel. These counts are made differentially-private by adding $\text{Lap}(1/\epsilon)$ (see Definition 2.4).

The size of $\epsilon$ in Equation 3.13 depends on how much of the privacy budget is remaining after querying the non-leaf nodes. This is where random decision trees, described in Section 2.3.2, gain a unique advantage; they do not query the non-leaf nodes at all. Jagannathan et al. [2012] were the first to investigate the idea of an ensemble of $\tau$ differentially-private random decision trees, spending the budget allocated to each tree on one query: Equation 3.13. In other words, each class count in each leaf node in each tree has $\text{Lap}(1/\epsilon)$ added to it, where

$$\epsilon = \frac{\beta}{\tau}. \quad (3.14)$$

Deciding how large $\tau$ should be is discussed later in Section 3.3.6; suffice to say that Jagannathan et al. recommend $\tau = 10$. This is close to the portion of the budget allocated to the leaf nodes of the greedy trees proposed by Friedman and Schuster [2010] when using their recommended depth $d = 5$ (assuming few continuous attributes; see Equation 3.11) and Rana et al. [2016] (see Equation 3.12).

Jagannathan et al. [2012]’s strategy proved successful; empirically, their random approach produces higher quality classifiers than Friedman and Schuster [2010]’s greedy approach. Figure 3.4 presents prediction accuracy comparisons with several datasets from the UCI Machine Learning Repository [Bache and Lichman, 2013]. Similarly to how the max operator splitting function outperformed more sophisticated functions in Section 3.3.2, these results suggest that the benefits of using a splitting function that is more sophisticated than randomness may be lost when the output is so noisy. Instead, the ability to make multiple trees proves to be more valuable than building a single tree.

What random trees sacrifice is a lot of the discriminatory power created in each node by the splitting functions of greedy decision trees, and this is clearest when handling continuous attributes; rather than finding the optimal splitting point like Friedman and Schuster [2010], or an average value that tries to separate two binary class labels like Rana et al. [2016], Jagannathan et al. simply randomly choose a splitting point from the attribute’s domain with uniform probability. This is the same strategy taken by traditional, non-private random decision trees [Fan et al., 2003, Geurts et al., 2006]. Despite the average random node being less discriminatory than the average greedy node, the overall prediction accuracy of an ensemble of random trees has been shown to be very similar to the accuracy of an ensemble of greedy trees [Geurts et al., 2006].
3.3.4 Termination Criteria

Knowing when to stop growing a decision tree is important in traditional non-private algorithms, but is even more important for a differentially-private decision tree algorithm. This is due to two main reasons:

1. When querying non-leaf nodes, only sibling nodes can be queried in parallel; different generations (i.e., levels) of nodes must have their privacy costs composed (see Section 2.5). The larger the depth, the more queries there are that need to be paid for.

2. The data is split into smaller partitions as a tree gets deeper, and the smaller the class counts are in a leaf node, the larger the effect of the $\text{Lap}(1/\epsilon)$ noise added to them.

The first reason affects only greedy trees, while the second affects both greedy trees and random trees. Throughout the literature, four main types of termination criteria are used when building either type of differentially-private tree:

**Criteria 1:** a user-defined criteria, such as maximum depth $d$ [Friedman and Schuster, 2010, Patil and Singh, 2014] or minimum node support $n'$ [Rana et al., 2016];

**Criteria 2:** a maximum depth based on the number of attributes $m$ [Friedman and Schuster, 2010, Jagannathan et al., 2012];
Criteria 3: defining a minimum node support $n'$ based on the ratio between the support and the added noise, and then either

- **a:** using an estimation $\tilde{n}_i$ of node $i$’s support to decide whether to terminate [Jagannathan et al., 2012]; or

- **b:** querying the actual support $n_i$ of node $i$ to more accurately determine whether to terminate [Friedman and Schuster, 2010].

Having the user define a maximum depth is the simplest approach, but it is also a naive approach that does not take into account any differences between datasets (unless the user employs some personal heuristics when making their decision). Friedman and Schuster [2010] used this criteria in conjunction with two other termination criteria. Other researchers [Patil and Singh, 2014] use it as a placeholder criteria; their research does not focus on this component of the algorithm, but $d$ nonetheless needs to be defined in order to divide the budget $\beta$ among the queries. Many of the equations for dividing the budget $\beta$ in Section 3.3.2 refer to a maximum tree depth $d$, such as Equation 3.11 and Equation 3.12.

Rana et al. [2016] propose having the user define a minimum node support to act as the termination criteria, rather than a maximum tree depth. They define $n'$ as a some small fraction (such as 0.1%) of the total number of records $n$, where a node will not be split if any of the subsequent child nodes would have any class counts less than $n'$. This has a similar result to other minimum node support criteria (such as Friedman and Schuster [2010]’s, discussed later), except that Rana et al. rely on the user to manually define $n'$ rather than using a heuristic. Unfortunately, this approach leads to a problem: if we don’t know how deep the tree will grow before we start querying the data, how do we decide what fraction of the privacy budget $\beta$ each query gets? For this reason, relying solely on $n'$ to terminate tree growth is not enough, and we include a maximum tree depth $d$ when discussing the privacy budgeting of Rana et al.‘s approach (such as in Equation 3.12). Note that using a minimum support criteria also requires querying the data in each node, which we discuss at the end of this section.

Friedman and Schuster [2010]’s algorithm uses two more termination criteria, the next of which is based the number of attributes $m$. They only use it to avoid a logic error though; since discrete attributes can only be used once, the tree-building algorithm terminates if all discrete attributes have been tested (i.e. if $d = m$). For most datasets, $m$ will be larger than the user-defined maximum depth, leading to this termination criteria rarely being used.

The number of attributes can be used for more than just preventing logic errors though. As discussed in Section 2.3.2, Fan et al. [2003]’s combinatorial reasoning suggests that
an ensemble of random trees has optimal diversity (and thus good discriminatory power) when each tree has a depth of \( d = m/2 \), and Jagannathan et al. [2012] propose using the same idea with differentially-private random trees. Because random trees do not query the data in non-leaf nodes, the portioning of the budget is independent of \( d \) (see Equation 3.14), allowing random trees to use whatever depth is considered optimal for non-private random trees.

Note that using \( d = m/2 \) assumes that all the attributes can only be selected once in any root-to-leaf path. This is true for discrete attributes, but not continuous attributes. We update Fan et al. [2003]’s tree depth calculation to take into account continuous attributes in Chapter 8, so that the number of different attributes tested in each root-to-leaf path remains optimal at \( m/2 \) different attributes tested.

There is one caveat with just using the same depth as non-private random trees though; if \( m \) is large (and the tree is therefore deep), the tree might partition the data so many times that each leaf node ends up having a very small fraction of the total dataset size \( n \). If the average support of the leaf nodes is small, adding \( \text{Lap}(1/\epsilon) \) to each class count (see Section 3.3.2) might completely overwhelm them with noise. Jagannathan et al. [2012] therefore proposed a heuristic that uses the third type of termination criteria (i.e. estimating the nodes’ support) and combined it with the \( m/2 \) criteria:

\[
d = \min\left(\frac{m}{2}, \log_b n - 1\right)
\]

(3.15)

where \( m \) is the number of attributes, \( n \) is the number of records in the dataset, and \( b \) is the average “branching factor” of the attributes. The branching factor of an attribute refers to how many child nodes will be created if a node is split using that attribute. This is the same as the domain size of the attribute if it uses discrete values, and is equal to \( b = 2 \) if the attribute is continuous. Functionally, \( \log_b n - 1 \) is setting the maximum depth to one less than the number of times \( n \) can be partitioned evenly by each node (where each node has the average number of child nodes \( b \)) until the bottom nodes have support equal to one.

Jagannathan et al. [2012] use \( b \) to estimate the support of nodes, making the assumption that the data will be roughly uniformly divided among all branches. This ensures that tree-building will terminate before the noise outweighs the class counts if the records are evenly distributed. If this assumption does not hold, and the records clump together in a small number of leaf nodes, this is still a good outcome if future unseen records follow the same distribution as the training records. The noise might completely overwhelm the class counts in small leaf nodes, but this will only affect a proportionally equally small number of future predictions.
Equation 3.15 acknowledges that small class counts are more susceptible to noise than larger counts, but it does not take into account how much noise is being added. Depending on how small $\epsilon$ is, the class counts in the leaf nodes might still be overwhelmed by the noise when using Equation 3.15. Friedman and Schuster [2010] solve this shortcoming by not using the branching factor $b$ at all. Instead, they directly query the data in the nodes as they build the tree. Recall that Friedman and Schuster [2010] propose asking two queries in each node: one to return the support of the node, and the other to either return the best attribute to split the node with or to return the class counts if the node is a leaf. The latter query was addressed in Section 3.3.2 and Section 3.3.3; the former query is used to enable Friedman and Schuster’s third and final termination criteria.

This termination criteria, along with Rana et al. [2016]’s minimum support criteria discussed earlier, are the only data-dependent criteria proposed in the literature. Querying the support of each node allows the growth of different parts of the tree to be more precise, but it comes at a cost however; part of the privacy budget $\beta$ must be spent on this query. It is debatable whether increasing how precise a termination criteria is is worth part of the budget; especially if it costs as much as half the budget, as seen in Equation 3.11 and Equation 3.12. Fortunately the extra query that Friedman and Schuster ask in each node is not only used for the termination criteria; they also use it when pruning the tree. We discuss this below in Section 3.3.5.

### 3.3.5 Pruning

After a tree has finished being built, pruning (described in Section 2.3) can then be applied. In order to account for differential privacy and the modifications to a decision tree algorithm that that entails, conventional pruning strategies also need to be modified. Friedman and Schuster [2010] propose one such modification for greedy decision trees.

Friedman and Schuster [2010] use the error-based pruning approach employed by C4.5 since it does not require additional data (data that could instead be used during the tree-building process, reducing the impact of the noise). Before performing pruning, they observe that due to the noise added to the support and class counts in each node, the sum of these counts in each level of the tree do not necessarily match the total size of the dataset. They therefore first “normalize” the support of each node in the tree so that the sum of all the nodes on each level of the tree match the number of records in the whole dataset. They then “normalize” the class counts in a similar fashion, making sure that the sum of the class counts in each node match the support of that node. The pruning is then applied as normal. The support $n_i$ of the nodes is normalized using the
noisy counts gathered from the additional query the authors ask in each node (discussed in Section 3.3.2 and Section 3.3.4).

### 3.3.6 Multiple Trees

Another design decision of any decision tree algorithm is the number of random decision trees to build. If the trees are built randomly, a large number of trees are required to result in a good classifier (i.e., a decision forest, see Section 2.3.2). This is a core concept behind traditional, non-private random decision forests [Fan et al., 2003, Geurts et al., 2006], and is still true for differentially private random decision forests [Jagannathan et al., 2012]. If the trees are built greedily, building more trees is not as necessary, but still beneficial [Islam and Giggins, 2011]. This is especially true if they are built using bootstrapped samples and random subsets of the attributes, increasing the diversity of the trees and decreasing the model’s susceptibility to small changes in the training data, which in turn reduces over-fitting [Breiman, 2001a].

Unfortunately in the private scenario, each tree added to a forest comes with a cost; if each tree is built using the full dataset, the trees are not disjoint and cannot be queried in parallel (see Section 2.5). If the records are divided into disjoint subsets for each tree, the leaf nodes will have much lower support, and therefore less reliable majority class labels. The smaller a class count is, the larger the relative effect of adding \( \text{Lap}(1/\epsilon) \) to it is. Only dividing the privacy budget has been tried [Jagannathan et al., 2012, Patil and Singh, 2014]; we experiment with dividing the data instead of the budget in Chapter 8. We explore what conclusions can be made about building multiple differentially-private decision trees below.

Patil and Singh [2014] were the first to take a differentially-private greedy decision tree, such as the one proposed by Friedman and Schuster [2010], and expanding it into a forest. Their approach was to use bootstrapped samples in each differentially-private tree, in the same way as Breiman’s Random Forest algorithm [Breiman, 2001a]. In the eyes of differential privacy, sampling a dataset of size \( n \) with replacement \( n \) times (i.e. bootstrapping) is almost identical to using the same data in every tree, and thus the privacy costs of the queries must be composed (Section 2.5). Some work has been done on calculating what privacy savings can be made by using bootstrapped data, since it is not exactly the same as querying the original \( n \) records [Rana et al., 2016], however thus far only approximations have been made, which do not guarantee differential privacy in the strongest sense. We explore these approximations later in Section 3.3.7.

In their empirical results, Patil and Singh [2014] demonstrated the infeasibility of their approach; even with a modest forest size of \( \tau = 20 \) trees and a tree depth of \( d = 5 \),
the resulting classifier has poor prediction accuracy with very high variance. If part of the privacy budget $\beta$ is needed for one query in each level of a tree, for every tree, each query only receives

$$\epsilon = \frac{\beta}{d\tau}$$  \hspace{1cm} (3.16)

of the budget, leading to a very large amount of noise being added to each query for any reasonable budget $\beta$.

Random decision trees, on the other hand, are only useful when part of a larger forest [Fan et al., 2003, Geurts et al., 2006]. In the context of differential privacy, their greatest advantage over greedy trees is the complete removal of queries in the non-leaf levels of the trees. This allows the budget for each query to rise from that of Equation 3.16 to:

$$\epsilon = \frac{\beta}{\tau}.$$  \hspace{1cm} (3.17)

Jagannathan et al. [2012] used this advantage and searched for the ideal number of random decision trees to build differentially-privately. They chose to use all the data in each tree, and divide the budget as shown in Equation 3.17. Experimentally, the authors concluded that $\tau = 10$ was the ideal number of random trees to build. Unfortunately, there is no mathematical or heuristical explanation for this number; their only advice is that in their experiments, datasets with less than 500 records worked best with only $\tau = 5$ trees, and that otherwise ten trees worked well.

Some theory was later developed by Bojarski et al. [2014] that offered some probabilistic bounds on the prediction accuracy of a differentially-private random forest, based on the number of trees and the depth of those trees. The authors released a preliminary version of their research on arXiv.org, presenting theorems for bounding both training error and testing error (in other words, classifying training data and non-training data). Notably, they propose that to correctly classify most new (non-training) data, a logarithmic number of random decision trees $\tau$ is needed, relative to the number of training records $n$. In their experiments, the number of trees went as high as 21, but also as low as one. This lines up reasonably well with Jagannathan et al. [2012]’s recommendation of using $\tau = 10$. Rana et al. [2016] implemented the findings of Bojarski et al. [2014] as a comparative decision forest in their experiments, with mixed results. Since proofs of the proposed bounds have not yet been released, it is difficult to discern their efficacy.
3.3.7 Weakening the Privacy Requirements

Jagannathan and Pillaiapakkamnatt, this time joined by Monteleoni, extended the work they did with Wright the previous year to make a semi-supervised classifier in a semi-private scenario [Jagannathan et al., 2013]. The scenario they describe is one where the user has a large number of unlabeled non-private (i.e., public) records, and a small number of labeled private (i.e., sensitive) records. The authors propose an algorithm that uses the unlabeled records in two ways. First, to find dense regions of the data hypercube and partition those regions when building decision trees, aiming to spread the records evenly among the leaf nodes. Second, to use a small differentially-private random forest built from the private (labeled) data to classify the non-private (unlabeled) data, and then build a large non-private decision forest with the non-private data. The final classifier is then the union of the two decision forests.

The main weakness of this approach is its applicability in the real world. We question how often a user would find themselves in a situation that meets all of the criteria: a large number of records that are both unlabeled and non-private, and a small number of records that are both labeled and private, with both types of records being from the same distribution.

Taking a different approach, Rana et al. [2016] proposed a greedy decision forest algorithm that used a weaker definition of differential privacy. By weakening the requirements of differential privacy, it allowed the forest to be much larger than previous forests made of greedy decision trees [Patil and Singh, 2014]. Rather than hiding whether a participant’s record is in a dataset with high probability, Rana et al.’s weakened definition of differential privacy makes sure that any estimate a user makes about a specific attribute value has the same variance as it did before the user made any queries, with high probability. In other words, their definition prevents a participant’s values being discovered (attribute linkage), but not whether or not the participant is in the dataset (table linkage) [Fung et al., 2010a]. In the paper, the scope of this definition is limited to an ensemble of decision trees using bootstrapped samples; no work has yet been done to extend it to other scenarios.

Similar to Patil and Singh [2014], the authors propose a greedy decision forest algorithm that builds \( \tau \) trees in the same fashion as Friedman and Schuster [2010], where each tree uses a random sub-sample of the total dataset. Rather than using a budget of \( \epsilon = \beta / \tau \) in each tree, Rana et al. distinguish their algorithm from others by allowing the user to manually define \( \epsilon \) (where \( \epsilon \leq \beta \)), with each tree using that amount. They demonstrate that their weakened definition of differential privacy is preserved by the
resulting decision forest as long as the number of trees is limited to:

\[
\tau \leq \frac{1}{p(1-p)} \left( n'(n' + 1) + \frac{1}{\epsilon^2} \right), 
\tag{3.18}
\]

where \( n' \) is the minimum support of leaf nodes, \( p \) is the probability of any randomly chosen class label being the least common class label, and \( \epsilon \) is the user-defined privacy budget per tree. As discussed in Section 3.3.4, \( n' \) is user-defined here. Note that Rana et al. only present formulas for scenarios with a binary class.\(^4\) Equation 3.18 only bounds the forest size in terms of preserving the class attribute; Rana et al. also define an upper bound on the number of trees to ensure no leakage of non-class attribute values, however in practice this bound is much higher than the bound based on the class attribute so we do not include it here.

\[3.3.8\] Query Efficiency

In this section, we use Friedman and Schuster [2010]’s differentially-private greedy decision tree as a case study in how to efficiently query a dataset when designing a data mining algorithm, and how to save parts of the privacy budget where possible.

In Section 3.3.2, we discussed how Friedman and Schuster [2010] asked two queries per node. The authors divided the privacy budget \( \beta \) evenly among all the queries, with each query having \( \epsilon = \beta/2d \), where \( d \) is the depth of the tree. One natural question might be, “is the first query in each node (which outputs the support) useful enough to be worth half the budget?”. While no research has directly explored this question, a differentially-private decision tree proposed by Mohammed et al. [2015] takes a very similar approach to Friedman and Schuster, but removes the first query in each node. This allows the splitting function in each node to have twice as much of the budget, \( \epsilon = \beta/d \). Mohammed et al. demonstrated that not only can a differentially-private greedy tree be built with only one query per node, but that it actually achieves higher prediction accuracy than Friedman and Schuster’s version that has the more precise termination criteria and pruning.

While ultimately resulting in lower prediction accuracy, Friedman and Schuster [2010] demonstrated an important concept when designing differentially-private algorithms: using queries that can achieve multiple purposes at once. By using the support of each node in two separate components of their algorithm (termination criteria and pruning),

\[\text{They also provide several additional parameters in the paper, which we generalize here to avoid minutiae.}\]
the authors got more “bang for their buck”. Differential privacy is immune to post-processing [Dwork and Roth, 2013], so outputs that can be used multiple times to improve a classifier are essentially “free” optimizations.

We can also observe that in Section 3.3.4, Friedman and Schuster’s Criteria 3b will never be used by the algorithm if $n/b^d$ is larger than the standard deviation of the Laplace noise, $\sqrt{2}/\epsilon$; one of the other two criteria will always occur first. If this is the case for a particular dataset, querying the support of each node is no longer useful for the termination criteria, and is only useful for pruning. The query was of questionable utility when it had two purposes (termination and pruning), let alone one, and should probably be skipped in this case, saving half the budget. We can take this idea further and ask ourselves: would we even want to terminate tree-building at $d \leq 2$? If not, then why not only query the support of nodes when $d > 2$, thus reducing the number of queries by two?

We should also check the two data-independent criteria (Criteria 1 and Criteria 2) first at each level, and maybe not query the support if one of the other criteria is already terminating the tree’s growth. On the topic of the other two criteria, note how terminating because all the attributes $m$ have been tested is unlikely to occur unless $m$ is less than the user-defined maximum depth $d$. However, this termination criteria differs from the minimum support criteria in that it does not cost us anything, since it does not rely on the data. We can afford to have redundant components in the algorithm, as long as they do not cost us.\footnote{Of course, computational complexity may also want to be considered, however in privacy scenarios we often want to maximize the utility-to-privacy trade-off, and are unwilling to compromise for the sake of other trade-offs like utility-to-complexity.}

If the user still wishes to query the support of each node (or perhaps ask some other non-compulsory query), they can consider whether every query should have an even proportion of the total budget. To the best of our knowledge, all research around differentially-private trees has divided $\beta$ evenly, such as $\epsilon = \beta/2d$ or $\epsilon = \beta/\tau$. This is not required however, and it is worth considering if it is more important that certain queries are less noisy, and that other queries can afford to be noisy without sacrificing too much utility.

These sorts of heuristics can lead to large budget savings, and thus a better classifier. While we used Friedman and Schuster [2010]’s algorithm as a case study, similar ideas can be used when designing any differentially-private algorithm.
3.3.9 Bringing it all Together: Implementations

All the algorithms we discuss in this section achieve the same basic goal of outputting information about nodes in a differentially-private way, but do so with a wide variety of strategies. Some algorithms intelligently find patterns in the data, providing the user with knowledge and not just a black-box classifier. Other algorithms sacrifice everything that is not absolutely necessary in order to build an accurate classifier, submitting as few queries as possible. These two extremes, and everything in between, offer different trade-offs between knowledge, accuracy, noise, and privacy costs. Cleverly using the privacy budget is imperative when designing an effective differentially-private decision tree classifier.

Table 3.1 summarizes the main properties of all the algorithms discussed. While no table could properly capture all the details of these algorithms, Table 3.1 provides a good overview and can act as a quick reference.

In some cases, such as some of the cells in the “Tree Depth $d$” and “Forest Size $\tau$” columns, the values provided are not inherently part of the proposed algorithm, but are instead the recommended values for parameters that are user-defined. For example, Mohammed et al. [2015] recommend setting the user-defined depth to $d = 4$, but since they only propose a decision tree algorithm and not a forest, their forest size is inherently $\tau = 1$. In the case of Rana et al. [2016]’s algorithm, the authors do not specify $d$ in their paper, and the size of their forest is defined by Equation 3.18 but is usually $\tau > 500$.

Some cells are labeled as “Maximized Diversity”, by which we refer to the combinatorial reasoning put forth by Fan et al. [2003] to maximize the attribute diversity of the trees, discussed in Section 3.3.4. The inclusion of “(Discrete)” in these cells indicates that the authors only accounted for discrete attributes, and not continuous attributes. For Jagannathan et al. [2013]’s work, we describe the forest size as “5 then 200”, referring to the fact that the algorithm builds two separate decision forests; a small differentially-private one, and then a large non-private one. The “Handles Cont. Attributes” column lists whether an algorithm handles continuous attributes, or only discrete attributes.

The details of all the algorithms summarized in Table 3.1 are discussed and compared throughout Section 3.3.

3.4 Differentially-Private Data Publishing with Trees

When differentially-privately querying a dataset, it is assumed that the data owner is maintaining a server that allows users to submit queries. This scenario is an example of
Table 3.1: A comparison of the main properties of the differentially-private decision tree algorithms.

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Budget per Query</th>
<th>Tree Type</th>
<th>Tree Depth $d$</th>
<th>Forest Size $\tau$</th>
<th>Handles Cont. Attributes</th>
<th>Prediction Accuracy$^1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blum, Dwork, McSherry &amp; Nissim (2005)</td>
<td>$\frac{\beta}{2^{m-1}}$</td>
<td>Greedy</td>
<td>$m$</td>
<td>1</td>
<td>No</td>
<td>Very Low$^2$</td>
</tr>
<tr>
<td>Friedman &amp; Schuster (2010)</td>
<td>$\frac{\beta}{(2^d+1)^2}$</td>
<td>Greedy</td>
<td>5</td>
<td>1</td>
<td>Poorly</td>
<td>Low</td>
</tr>
<tr>
<td>Mohammed, Barouti, Alhadidi &amp; Chen (2015)</td>
<td>$\frac{\beta}{d}$</td>
<td>Greedy</td>
<td>4</td>
<td>1</td>
<td>No</td>
<td>Medium$^2$</td>
</tr>
<tr>
<td>Patil &amp; Singh (2014)</td>
<td>$\frac{\beta}{(2^{d-1})}$</td>
<td>Greedy</td>
<td>5</td>
<td>20</td>
<td>No</td>
<td>Very Low</td>
</tr>
<tr>
<td>Rana, Gupta &amp; Venkatesh (2016)</td>
<td>$\frac{\beta}{2d}$ or $\frac{\beta}{3d}$</td>
<td>Greedy</td>
<td>Unspecified</td>
<td>500+</td>
<td>Yes</td>
<td>High$^3$</td>
</tr>
<tr>
<td>Jagannathan, Pillaiappakkamnatt &amp; Wright (2012)</td>
<td>$\frac{\beta}{\tau}$</td>
<td>Random</td>
<td>Maximized Diversity (Discrete)</td>
<td>10</td>
<td>Yes</td>
<td>Medium</td>
</tr>
<tr>
<td>Jagannathan, Montealeoni &amp; Pillaiappakkamnatt (2013)</td>
<td>$\frac{\beta}{\tau}$</td>
<td>Random</td>
<td>Maximized Diversity (Discrete)</td>
<td>5 then 200</td>
<td>Yes</td>
<td>High</td>
</tr>
</tbody>
</table>

$^1$ Prediction accuracy is categorized into quartiles of average relative empirical performance, calculated using the prediction accuracy results presented by each of the authors across a variety of datasets and $\beta$ values.

$^2$ Our best estimation; the authors did not empirically test the algorithm.

$^3$ With a weaker definition of differential privacy.

Privacy-preserving data mining [Aggarwal and Yu, 2008], where data mining is applied directly to the original data using a private mechanism. Sometimes maintaining a server is impractical, such as due to financial or logistical reasons, and the data owner would prefer to release a private version of the data to the public. This scenario is an example of privacy-preserving data publishing [Fung et al., 2010a], where the data is kept as similar as possible to the original data while still protecting privacy. We explore what “similar” means in this context in Section 3.2.1 and Chapter 5. We take a moment to briefly discuss the other side of the coin: how traditional, non-private decision trees can be used to publish private data. This includes privacy-preservation strategies other than differential privacy as well.

While differential privacy is becoming the de-facto standard for modern privacy preservation [Clifton and Tassa, 2013], other forms of privacy preservation exist, such as
those presented in Section 3.1. One of the main competing approaches is \( k \)-anonymity [Sweeney, 2002]. Fung et al. [2005, 2007] proposed a practical implementation of \( k \)-anonymity called “top-down specialization” (discussed briefly in Section 3.2.2.2) that preserves privacy by first making every value in the dataset as general as possible, and then building a greedy decision tree that iteratively increases the granularity of attributes. Attributes were selected to be made more granular (i.e., specialized) if they had high information gain compared to the amount of privacy they leaked. In other words, \( k = N \) at the root node, and then \( k \) is reduced with each split as efficiently as possible, terminating when \( k \) reached the user-defined minimum. After the decision tree was built, a synthetic \( k \)-anonymous dataset could be released that used the specializations chosen by the tree.

Interestingly, top-down specialization was re-purposed by Mohammed et al. [2011] to provide differential privacy instead of \( k \)-anonymity. Instead of specializing attributes based on their utility-privacy trade-off (in the node splitting function), only utility is considered. The resulting tree is then used to create segments of generalized records, whose frequency can be differentially-privately outputted. Thus a frequency table of generalized records can be released. The reasoning for this approach was that generalization offers something that adding Laplace noise does not; values in the dataset cannot be outright incorrect, merely less granular. In scenarios where misleading information must be avoided at all costs, this approach can be very appropriate. When building a non-private decision tree from the generalized data, the prediction accuracy was similar to the accuracy achieved by Friedman and Schuster [2010]’s algorithm.

Another type of privacy preservation is noise addition, in which noise is selectively added to attribute values to hide sensitive information [Agrawal and Srikant, 2000, Islam, 2007]. Several privacy-preserving decision tree algorithms have been proposed that add noise in such a way that the original patterns found by the decision trees are preserved in the anonymized data [Islam and Brankovic, 2011]. Noise addition offers less rigorous protection of people’s privacy than \( k \)-anonymity however, and neither technique offers mathematical guarantees like differential privacy. Perhaps most egregiously, they provide no protections against malicious users who have auxiliary information (information from other sources about the people in the dataset). Without clearly definable guarantees for each participant, it is difficult for noise addition or \( k \)-anonymity to be implemented in a way that meets the real-world privacy policy requirements of governments and businesses. A detailed comparison of the advantages and disadvantages between \( k \)-anonymity and differential privacy was performed by Clifton and Tassa in 2013 [Clifton and Tassa, 2013].
Chapter 3. Literature Review

Mivule et al. [2012] also proposed a framework that iteratively checks the quality of a differentially-private dataset with an AdaBoost decision forest [Freund and Schapire, 1999], updating the dataset until the forest achieves acceptable prediction accuracy. Their framework lacked detail however, and their results were mixed.

Many other techniques have been proposed for publishing differentially-private data, though not with decision trees and therefore outside the scope of this section. They often involve partitioning the original dataset in a way that balances utility with privacy, then creating aggregated records from noisy histograms of the partitions [Blum et al., 2013, Li et al., 2014]. The utility of these techniques suffers exponentially when the number of attributes increases beyond four, making them unusable in most real-world scenarios (barring an equally exponential increase in the number of records). We refer the reader to Ji et al. [2014]’s survey for more details.

3.5 Choosing the Privacy Budget

In all of the algorithms talked about in this chapter, the privacy parameter $\epsilon$ determines how much noise is added to the output of queries. This extends to all applications of differential privacy; $\epsilon$ is at the core of its definition. Despite this, very little work has been done in how to choose an appropriate value for $\epsilon$ in the real world, both for ensuring comparable utility between private and non-private queries, and for providing an appropriate amount of privacy protection to each of the participants in the data. Dwork and Roth [2013] themselves have left these questions up to the discretion of the user and their particular needs. To the best of our knowledge, two papers have provided practical guidelines for choosing $\epsilon$; one from the perspective of utility [Vu and Slavkovic, 2009] and one from the perspective of privacy [Hsu et al., 2014].

Using the context of clinical trials, Vu and Slavkovic [2009] provided several results in 2009 for how to use hypothesis testing in a scenario where sensitive information needs protecting. They focus on the sample size of the test, and demonstrate how many more records are needed to produce the same quality of results that the test would produce if privacy was not a concern. The authors do so in terms of two main factors: the confidence level of the hypothesis test (type I error) and the power of the hypothesis test (type II error). The equations for this scaling of the number of records, dubbed the “sample size correction factor”, can be found in Vu and Slavkovic [2009]. If a user is able to estimate how many samples they would need to achieve strong results for their hypothesis test, they can use Vu and Slavkovic’s work to achieve the same results, but differentially-privately.
On the other side of the utility-privacy trade-off, Hsu et al. [2014] proposed the first practical methodology for deciding how small $\epsilon$ needs to be to appropriately protect people’s privacy for a given scenario. They do so by modeling the problem in terms of money; the monetary budget of the user collecting or using the sensitive data. In any given scenario, each participant included in the data is at a certain amount of risk. By estimating the probability of a participant’s privacy being maliciously breached, and the monetary cost associated with that breach, the user can calculate the average risk per participant in the data. Since the probability of a breach is directly tied to $\epsilon$ (i.e., the presence of a record can change the output with probability proportional to $\exp(\epsilon)$, see Definition 2.1), the user can balance how many samples they collect/use, how large $\epsilon$ can realistically be, and how much money they are willing to spend (either on paying people to participate in the study or on compensating the participants whose privacy is breached). Such a model offers practical guidelines for how large or small the privacy budget can realistically be.

### 3.6 Real-World Implementations

Despite differential privacy being a relatively new invention (first proposed in 2006 [Dwork, 2006], and the first textbook being written in 2013 [Dwork and Roth, 2013]), government and industry adoption has already begun. The first significant implementation of differential privacy in a real-world scenario occurred in 2008, when the U.S. Census Bureau used it in their Longitudinal Employer-Household Dynamics Program [Machanavajjhala et al., 2008]. This program resulted in a public-facing product that enables users to create demographic maps of people living in user-specified suburbs in the United States.

In 2009, Microsoft released an API called Privacy Integrated Queries (PINQ) for querying datasets in a differentially-private manner [McSherry, 2009]. PINQ facilitates other organizations granting analysts access to sensitive data in a way that preserves privacy, without the analyst needing to be an expert in differential privacy themselves.

More recently the technology company, Apple, announced that they would be using differential privacy in their current and future operating systems and applications [Greenberg, 2016]. At the keynote address of Apple’s Worldwide Developers’ Conference in 2016, Senior Vice President Craig Federighi said:
“Differential privacy is a research topic in the areas of statistics and data analytics that . . . enables crowd-sourced learning while keeping the data of individual users completely private. Apple has been doing some super-important work in this area to enable differential privacy to be deployed at scale.”

[Greenberg, 2016]
Part II

Metrics
We tend to overvalue
the things we can measure
and undervalue the things we cannot.

JOHN HAYES
Chapter 4

Measuring the Similarity between Sets of Patterns

The discovery of patterns in data is the cornerstone of data mining; sometimes for predicting the future, and other times for extracting meaning. It is the latter case that this chapter will concern itself with. By extracting meaning, statisticians and data analysts are able to elevate otherwise meaningless data into usable information; information that can be acted on or used to discover knowledge [Han et al., 2006]. The extraction of meaning from data is most often explicated as mining patterns from data, preferably in a form that is interpretable by humans. Many branches of data mining have concerned themselves over the years with this endeavor, not limited to frequent pattern (itemset) mining [Han et al., 2007], decision trees [Quinlan, 1993] and forests [Breiman, 2001a], and association rule mining [Ordonez and Zhao, 2011]. Constructing models from data in order to make accurate predictions is useful, but it is important to remember that the accuracy of a model does not guarantee the truthfulness of the patterns contained in the model. The history of science is littered with untruthful models that accurately predicted observations, from Bohr’s model of the atom to the geocentric model of the Solar System.

Assessing the patterns discovered by data mining algorithms is often overlooked in the current zeitgeist, instead favoring a dogged pursuit of higher prediction accuracy [Hand, 2006, Wagstaff, 2012]. We argue that this is to the detriment of the discipline and provide a simple but powerful method for comparing sets of patterns, aiding a renewed

---

The work in this chapter has been published in the following paper: Sam Fletcher and Md Zahidul Islam. Measuring the Similarity between Rule Lists. In 14th Australasian Data Mining Conference, pages 1–8, Canberra, Australia, 2016.
focus in pattern assessment seen in recent years [Aodha et al., 2014, Letham et al., 2013].

We make no distinction between the words “pattern” and “rule” in this chapter, and use them interchangeably.

We refresh the reader on what we mean by “patterns” or “rules”; we refer to a set of criteria, where if a record meets all of the criteria, allow an analyst to infer additional information about the record. More formally, patterns take the form $\psi \rightarrow c \in C$. That is to say, if criteria $\psi$ is met, attribute $C$ is predicted to equal $c$. $C$ is known as the consequent or class attribute, and can be found dynamically or be user-defined. Functioning as the antecedent, $\psi$ is a set of conditions for a subset of the $m$ attributes used in the data, $A \in A$, where each attribute $A$ has values $v \in A$. Conditions take the form $A = v$, or can use other operators such as $A > v$ if $A$ is ordered. An example would be $\{Education = PhD, Age > 45\} \rightarrow Income = High$, or $\psi$ might detail the presence of items such as $\{milk, bread\}$ and predict that the consequent $eggs$ will also be present. Other examples can be seen in Table 4.1. Patterns in any of these forms are often referred to as decision rules or association rules, and when collected together can be called a decision list or rule list [Letham et al., 2013].

1For example, a decision tree might be “flattened” so as to no longer have roots or leaves, and it merely becomes an unordered set (a rule list) of unordered sets of attribute conditions (rules). The decision tree would then be indistinguishable from a rule list.
4.1 Problem Statement

Consider the following question:

Given two rule lists $\Psi_1$ and $\Psi_2$, how similar are they?

To the best of our knowledge, this question has yet to be answered in the literature, and is answered in this chapter. The ability to answer this question has clear benefits in many areas of data mining and knowledge discovery, such as:

- comparing different classifiers, or classifiers with different parameters [Islam and Giggins, 2011, Shotton et al., 2013];
- comparing patterns discovered manually by statisticians, to patterns discovered with machine learning techniques [Breiman, 2001b];
- comparing the quality of the patterns discovered in data before and after applying privacy-preserving techniques to the data [Islam et al., 2003]; and
- finding differences in different samples of data, including temporal scenarios with time series [Baron et al., 2003].

Answering the above question requires a measurement of some kind, and it is a measure (more specifically, a metric) that we propose in this chapter. We convert the rules in $\Psi_1$ and $\Psi_2$ into discrete elements, and take advantage of the Jaccard index to measure the similarity between sets made up of these elements. This is discussed in full in Section 4.3. In order to be a useful measure, we consider several external factors to be part of the problem statement. These are factors that often determine whether researchers and data miners willfully choose to use a measure:

- conceptual simplicity;
- computational simplicity;
- interpretability; and
- wide applicability.

We discuss how our proposed measure fulfills these external factors in Section 4.3. Section 4.2 provides additional background information and related work. In Section 4.4, we demonstrate our measure in action with an illustrative real-world scenario in which a data miner wishes to compare two classifiers. We conclude with Section 4.5.
4.2 Related Work

Patterns play a key role in the knowledge discovery and decision-making process. Several fields of machine learning – notably frequent pattern mining [Han et al., 2000, 2007] – focus specifically on finding patterns. Fields such as classification can also find patterns by using decision forests [Breiman, 2001a] or other classifiers [Han et al., 2006]. Patterns can be assessed for their usefulness [Geng and Hamilton, 2006] and their interpretability [Letham et al., 2013], or monitored for any changes in temporal scenarios [Baron et al., 2003]. While differing in methodology, approaches such as these agree on the importance of patterns and attest to the value of patterns for gaining knowledge.

It is important to note that assessing the quality of patterns in these ways is not the same as measuring the performance of a model at achieving a goal [Caruana and Niculescu-Mizil, 2004, Sokolova and Lapalme, 2009]. Prediction accuracy is often used to measure a model’s performance [Cheng et al., 2007, Letham et al., 2013], but is disconnected from any reliable assessments of pattern quality [Islam et al., 2003]. A user should be aware of the specific goals they wish their model to achieve and how important the truthfulness [Kifer and Gehrke, 2006] or interpretability of the patterns in the model are, and then use multiple measures to assess if their needs are met.

Our proposed application of the Jaccard index assesses if two lists of patterns are similar. Attempts to measure the similarity between trusted patterns and newly discovered patterns have been made in the past [Islam and Brankovic, 2011], but the measure used was designed for a specific problem, lacking any applicability in a wider context. We discuss the value of widely applicable measures in Section 4.3.3.

The field of privacy-preserving data mining [Dwork, 2008, Fung et al., 2010a] is known for struggling with the inherent trade-off that must be made between the amount of privacy provided and the quality of the perturbed data [Fung et al., 2005, Nergiz and Clifton, 2007]. A naive approach would be to use measures like RMSE to compare the original data to the perturbed data directly [Willmott, 1982, Willmott et al., 2009], however this ignores the correlations in the data necessary for information discovery [Agrawal and Aggarwal, 2001]. Just like how frequent pattern mining and other fields use prediction accuracy heavily, so too does privacy-preserving data mining [Chaudhuri et al., 2011, Friedman and Schuster, 2010, Fung et al., 2005, Mohammed et al., 2011], but the same problems encountered by the former when assessing patterns also plague the latter [Islam et al., 2003]. We explore directly measuring the loss of pattern retention as privacy needs are increased in Chapter 5.

There is therefore a need for insightful ways to compare between patterns gained by different means, regardless of what data mining algorithms or data are used. We do
so in this chapter, taking both practical and mathematical considerations into account [Meila, 2007].

4.3 Comparing Rule Lists with the Jaccard Index

The Jaccard index [Jaccard, 1901] is a well-known measurement of the similarity between two sets \( S \) and \( T \), defined as the size of the intersection divided by the size of the union of the two sets:

\[
J(S, T) = \frac{|S \cap T|}{|S \cup T|},
\]

(4.1)

where we say \( J(S, T) = 1 \) if \( |S \cup T| = 0 \). By reinterpreting a rule list (a collection of patterns) as a set of elements, we are able to use the Jaccard index to measure the similarity between two rule lists. In Section 4.3.1 we describe how we convert rules in rule lists \( \Psi_1 \) and \( \Psi_2 \) into elements for sets \( S \) and \( T \). In Section 4.3.3 and Section 4.3.4 we outline the practical and mathematical benefits of using the Jaccard index to measure the difference between two rule lists.

4.3.1 Converting Rules into Elements in a Set

Our aim is to compare two rule lists and describe their similarities with an intuitive, quantitative number. To do so with the Jaccard index, we must first translate each rule \( \psi \rightarrow c \) (such as those seen in Table 4.1) into element \( s \) of set \( S \). Each antecedent \( \psi \) is made up of attributes – continuous, discrete or binary attributes – that specify the conditions that must be met in order for the consequent \( C \) to be predicted to equal \( c \). To condense the set \( \psi \) as well as \( c \) into a single element \( s \), we use the following equation:

\[
s = 1_\psi(A_1), 1_\psi(A_2), \ldots, 1_\psi(A_m), c_\psi
\]

(4.2)

where we use \( c_\psi \) to simply refer to the class value (consequent) of rule \( \psi \), and \( 1_\psi(A) \) is the indicator function:

\[
1_\psi(A) := \begin{cases} 
1 & A \in \psi \\
0 & A \notin \psi 
\end{cases}
\]

(4.3)

Essentially, Equation 4.2 is recording the presence or absence of each attribute in rule \( \psi \), as well as recording the consequent \( c \). Table 4.2 illustrates this with converted versions of the rules seen in Table 4.1. The three 1’s in \( \psi_0 \) refer to the positions in the WBC
Table 4.2: The encoded versions of the rules shown in Table 4.1.

<table>
<thead>
<tr>
<th>i</th>
<th>$s_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>110001000a</td>
</tr>
<tr>
<td>1</td>
<td>011000000a</td>
</tr>
<tr>
<td>2</td>
<td>111001001b</td>
</tr>
<tr>
<td>3</td>
<td>011000100b</td>
</tr>
</tbody>
</table>

dataset of the three attributes used by $\psi_0$ in Table 4.1, and similarly for the other rules. We include the consequent in $s$ because of the role it plays in the definition of a pattern – without a consequent, an antecedent hardly means anything at all.

Note that it is not possible to include the attribute values in the encoding without a large number of heuristics to handle the edge cases, as well as heuristics for handling the fundamental difference between ordered and unordered data. For example, “PhD” and “Bachelors” are clearly different, but what about “45” and “4 4”? Imagine the following pair of rules: \( \{\text{Age} > 45\} \rightarrow \text{Income} = \text{High} \) and \( \{\text{Age} \leq 44\} \rightarrow \text{Income} = \text{Low} \); it is computationally infeasible to detect that pairs of rules such as these are, in fact, the same.

### 4.3.2 Steps

The steps for calculating the similarity between two rule lists are as follows:

1. Take $\Psi_1$ and $\Psi_2$ as input.

2. Convert $\Psi_1$ and $\Psi_2$ into sets $S$ and $T$ respectively, using the process in Section 4.3.1.

3. For each element in $S$, scan $T$ to see if that element exists in $T$. The number of pairs found equals $|S \cap T|$.

4. Deduce the number of unpaired elements in $S$ and $T$ using the results from Step 3. This number plus $|S \cap T|$ equals $|S \cup T|$.

5. Calculate $J(S, T)$.
4.3.3 Practical Considerations

As discussed in Section 4.1, a good measure should not be difficult for data analysts to harness effectively. We assess our proposed application of the Jaccard index with four factors that influence an analyst’s decision to use a measure:

4.3.3.1 Conceptual Simplicity

A good measure is one that can be easily and intuitively understood. If a measure has too many moving parts or variables, it can quickly become a “black box” of sorts, where analysts can no longer conceptualize all the possible outputs that the measure could produce. Our implementation of the Jaccard index avoids these risks by being very straight-forward – it is the number of patterns two rule lists share, divided by the number of unique patterns across both lists. There are no variables beyond the two rule lists, and there are no parameters that need expert knowledge to properly adjust. Our encoding of the rules into single elements is simple and intuitive, making the conceptualization of the processes involved in the measure no more difficult than in a standard application of the Jaccard index. Incorporating attribute value conditions into the encoding process would require a robust definition of “similarity” or “distance” for both numerical and categorical attribute values that did not induce any biases in the calculation, and would undoubtedly increase the conceptual complexity of the measure. This is one avenue for future work, but currently appears to be infeasible due to the fundamental differences between numerical and categorical attributes.

4.3.3.2 Computational Simplicity

A measure can become infeasible if it does not scale well as the variables become large. The computation time of our measure is very satisfactory, mostly due to the fact that it does not use the underlying raw data in any way, or even a classifier – it just needs the rule lists. Encoding all the rules in \( \Psi_1 \) and \( \Psi_2 \) as the sets \( S \) and \( T \) requires each rule in both lists to be read and converted once, with the presence of each of the \( m \) attributes being checked once in each rule. Therefore the computational complexity is \( O(m(|\Psi_1| + |\Psi_2|)) \). Once \( S \) and \( T \) have been generated, the intersection and union in the Jaccard index can both be calculated by comparing each element in \( S \) to every element in \( T \), where each element is \( m \) digits long. This gives a computational complexity of \( O(m \cdot |S| \cdot |T|) \). Since the length of the respective rule lists and sets will always be equal, the total computation time of our measure is \( O(m(|\Psi_1| + |\Psi_2| + |\Psi_1| \cdot |\Psi_2|)) \). As a rough example, classification algorithms such as Random Forest [Breiman, 2001a] might
generate several hundred rules, and the kinds of datasets that Random Forest might be applied to generally have less than 1000 attributes.

4.3.3.3 Interpretability

In order for analysts to judge the result of a measure and act on it in a meaningful way, the result needs to be interpretable. This can be as simple as being able to parse the result into a sentence. In our case, our measure can be interpreted as “out of all the rules that appear in rule lists $\Psi_1$ and $\Psi_2$, $(J(S, T) \times 100)\%$ can be found in both lists”. An addendum could be added about how the rules are compared at a level of granularity that ignores differences in attribute value conditions, without making the measure any more difficult to understand and interpret.

4.3.3.4 Wide Applicability

Measures that accurately encapsulate specific scenarios are undeniably useful, but measures capable of crossing academic discipline boundaries are far more appealing. They provide ways for researchers and professionals to interface with work outside of their personal scope and build connections between otherwise isolated fields of science. Historically, the Statistics and Machine Learning disciplines have learned this lesson the hard way, with the relatively new field of Machine Learning often recreating mathematics and methods invented previously by statisticians [Breiman, 2001b]. The ubiquity of prediction accuracy as a measure of performance and the role it plays in creating dialogue among researchers demonstrates the advantages of measures with wide applicability. Our proposed encoding method and application of the Jaccard index is general enough to be viable in any situation involving two groups of patterns discovered from data with the same attributes. Our measure is completely independent of how the patterns were discovered, built, or connected. It can handle non-binary consequents – a common constraint for other measures [Felkin, 2007]. It can handle continuous consequents as well, but only if the range of values are first discretized into “buckets” [Kotsiantis and Kanellopoulos, 2006]. It can even work in scenarios without a $C$ component, where $\psi_i$ might merely represent a collection of attributes that appear together frequently. In this situation, $s$ would intuitively drop the $c$ component and become equal to $1_\psi(A_1), 1_\psi(A_2), \ldots, 1_\psi(A_m)$.

Our measure is conceptually and computationally simple, it produces easily interpretable results, and it is an appropriate measure in a wide variety of scenarios. Not constraining ourselves by requiring definitions of similarity or distance for numerical and categorical attribute value conditions allows these qualities to be improved even further.
### 4.3.4 Mathematical Properties

A good measure should be capable of distinguishing between slightly different rule lists and very different rule lists. The Jaccard index possesses several properties that make this possible, as well as possessing properties that strengthen the versatility of the measure. Our encoding method is designed in a way that does not interfere with these properties.

#### 4.3.4.1 Bounds

The Jaccard index has the bounds $0 \leq J(S, T) \leq 1$. We can narrow down the upper bound further by using the difference in size between the lists. In situations where $|S| > |T|$, the maximum similarity is when $T \subseteq S$, in which case $J(S, T) = |T|/|S|$. The larger the size difference between the sets, the smaller the upper bound is. To put it formally:

$$J(S, T) \leq \frac{|T|}{|S|}, \text{ where } |S| \geq |T|. \quad (4.4)$$

It is reasonable to assume that the user will know the size of sets, making this upper bound easy to incorporate when interpreting the measure’s result, and possibly being quite informative. A similar situation exists for the popular prediction accuracy measure, where the lower bound is equal to the relative frequency of the most common class value.$^2$

If the flexibility of the Jaccard index’s upper bound is undesirable, a user is free to shorten $S$ until both lists have equal length. The Frequent Pattern Mining discipline does this often, only selecting rules with high support [Han et al., 2000, 2007]. Many other measures could be used to remove the least valuable rules depending on how the user defines “valuable” [Geng and Hamilton, 2006]. A user could also divide the Jaccard index result by the upper bound and reinterpret the result as “the ratio between the actual result and the best possible result”, but this approach could easily misguide the user about how similar the rule lists actually are and is not recommended.

The lower bound can also be narrowed in very specific scenarios: when the number of rules in $S$ and $T$ is more than the number of unique rules that could exist. This indicates that $|S \cap T| > 0$; some overlap between the sets must exist. This can occur when the number of unique rules that could exist given the number attributes and class values in the dataset (i.e. the number of ways you could write $s$, $|C|(2^m - 1)$) is less than $|S| + |T|$.\(^3\)

---

$^2$Theoretically it can go lower, but then the model is providing no benefit to the user and is actually causing harm – it is worse than a random guess.

$^3$We subtract one from $2^m$ because an empty $\psi$ is not a legal antecedent. Also note that $|S \cup T| \leq |S| + |T|$. 

Chapter 4. *Measuring the Similarity between Sets of Patterns*

It is also possible that the pattern creation process puts a constraint $k$ on the number of attributes that could be present in any one pattern [Webb and Brain, 2002]. It has been demonstrated that increased rule length can actually decrease the information gained due to the decreased generality of the patterns [Cheng et al., 2007], as well as decreasing the interpretability of the rules [Freitas, 2013, Huysmans et al., 2011, Letham et al., 2013, Vellido et al., 2012]. Note that the data mining processes used to generate $\Psi_1$ and $\Psi_2$ might have different $k$’s, denoted $k_{\Psi_1}$ and $k_{\Psi_2}$, but the combinations possible with a smaller $k$ are a subset of a larger $k$’s combinations. We can write the number of combinations possible in a scenario with constraint $k$ as $\sum_{i=1}^{k} \binom{m}{i}$, where $\binom{m}{i}$ is the binomial coefficient and equals $m! / i!(m-i)!$.

Strictly speaking, the lower bound of our measure is

$$J(S, T) \geq \max\left(\frac{|S| + |T| - |C| \cdot \sum_{i=1}^{k} \binom{m}{i}}{|C| \cdot \sum_{i=1}^{k} \binom{m}{i}}, 0\right),$$

where $k = \max(k_{\Psi_1}, k_{\Psi_2}) \leq m$. If $k = m$ (i.e. No constraint was put on the length of the rules),

$$|C| \cdot \sum_{i=1}^{m} \binom{m}{i} = |C|(2^m - 1).$$

To use some example numbers, if $|S| + |T| = 100$ and there are five attributes and two class values, there are $2(2^5 - 1) = 62$ possible combinations of attributes and class values. The lower bound then becomes 0.61, ruling out over half the original range. Having so many rules for so few attributes is unusual though, and in most real-world situations the lower bound will simply equal 0. These simple upper and lower bounds allow the user to interpret the measure’s results with far more insight than would be possible if the bounds were ignored.

### 4.3.4.2 Metric Properties

Metrics are a subset of measures, defined by four mathematical properties they possess: non-negativity; identity of indiscernibles; symmetry; and triangle inequality. We can describe each of these, respectively, with the Jaccard index:

- $J(S, T) \geq 0$
- $J(S, T) = 1 \iff S = T$
- $J(S, T) = J(T, S)$
• \( J(S, U) \geq J(S, T) + J(T, U) \). Note that we use “\( \geq \)” since the Jaccard index is traditionally a metric of similarity, not distance. If a distance is preferred, the Jaccard Distance can be very easily used: \( d_J(S, T) = 1 - J(S, T) \).

It is straightforward to see how the Jaccard index satisfies the first three properties, since its maximum bounds are \( 0 \leq J(S, T) \leq 1 \) and Equation 4.1 does not change if \( S \) and \( T \) are swapped. A proof of the Jaccard index satisfying the triangle inequality has been previously constructed [Levandowsky and Winter, 1971, Lipkus, 1999]. Because the Jaccard index has these properties, mathematicians can use them when constructing proofs and can use more assumptions that are guaranteed to hold true. The triangle inequality especially is well-known as a strong mathematical property when analyzing metrics [Chawla et al., 2005] and designing efficient data structures and algorithms [Meila, 2007]. Measures that are not metrics often output results that require unintuitive interpretations [Meila, 2007, Willmott et al., 2009].

4.4 Example Scenario: Comparing Classifiers

In this section we perform a short case study on one of the example scenarios given in Section 4.1: comparing classifiers with different parameters. Specifically, we use the implementation of the CART classifier [Breiman et al., 1984] found in the scikit-learn software [Pedregosa et al., 2011], and we compare two different objective functions for the splitting criteria: the Gini index [Breiman et al., 1984] and Information Gain [Quinlan, 1996]. This comparison represents a reasonably straight-forward scenario where a common question is being asked: “How different would our results be if we changed classifier?”.

By using single decision trees (rather than decision forests or other classifiers), we generate a manageable number of rules directly from the classifier⁴, rather than needing to filter a larger number of rules down using additional processes. By using a simple experimental set-up, our results are easily reproducible.

We apply the CART classifier on 19 datasets from the UCI Machine Learning Repository [Bache and Lichman, 2013] using five-fold cross-validation with stratified folds⁵, and measure the Jaccard index between CART with the Gini index (referred to as CART-G) and CART with information gain (referred to as CART-I). We also measure the prediction accuracy of both CART-G and CART-I on the test data from the unused fold. Our aim is to see whether the user could learn anything new by going beyond a

---

⁴In a decision tree, each path from the root to a leaf is considered a rule, predicting the most common class value found in the leaf.

⁵Stratified folds have the same distribution of class values as the dataset as a whole.
simple comparison of prediction accuracies and also comparing the classifiers with the Jaccard index.

For each dataset in our experiments, the minimum support threshold of each leaf in the decision trees is 2% of the records. The lower bound of the Jaccard index is 0 for all datasets. The lower bound for prediction accuracy depends on the relative frequency of the majority class label in each dataset. The upper bound for the Jaccard index depends on the number of rules found with each classifier and is reported along with the Jaccard index results in Table 4.3. Since our question is “How different would our results be if we changed classifier?”, we report the absolute difference between the prediction accuracy of CART-G and CART-I, also in Table 4.3. We display the same information as a scatter
Figure 4.1: The Jaccard index compared to the prediction accuracy difference for each dataset. Note the scope of the axes – the maximum prediction accuracy difference is small, while the Jaccard index results make use of the full range between the bounds.

We can see that the detected differences in prediction accuracy are small – too small to trust that the detected difference is solely due to the classifiers [Hand, 2006], or that the result is precise enough to remain consistent after repeated runs of the experiment. If a user found themselves in the scenario described above, the only thing they could reasonably conclude from the prediction accuracy results is “the two classifiers are very similar”. However the Jaccard index results inform the user that this is not true at all – the rules uncovered by the two classifiers are similar for some datasets, but on others they can have very few rules in common. It turns out that while there is almost no difference in prediction accuracy when choosing either the Gini index or information gain, the structure of the trees can be very different for some datasets! The Jaccard index allows the user to learn information such as “only 40% of the rules in the Adult dataset found by either classifier were found by both classifiers”, while prediction accuracy tells the
user “for the Adult dataset, the two classifiers differ by only a tenth of a percent when predicting the consequent of unseen records”. Our proposed measure does not replace prediction accuracy, but instead provides information that prediction accuracy (or any other measure, to the best of our knowledge) is incapable of providing.

\section*{4.5 Conclusion}

In this chapter we propose a method for measuring the similarity between two lists of rules. The method was designed with a focus on conceptual simplicity, computational simplicity, interpretable results, and applicability in a wide variety of scenarios. One such scenario was explored in detail to demonstrate the information a user could learn from our proposed measure. Strong mathematical properties are provided to aid users in interpreting their results and making comparisons between results. For example if a third classifier was added to the scenario portrayed in Section 4.4, the triangle inequality allows users to use their intuition that the similarity between classifier X and Z cannot be lower than the sum of the similarities between classifiers X and Y and classifiers Y and Z.

Our use of the Jaccard index successfully measures an aspect of data mining results that no pre-existing measure is able to do. As the data mining community continues to put more focus on the discovery of interpretable patterns in data, the ability to distinguish between different sets of patterns will be highly useful. In the next chapter, we explore a related idea: what if we only have only one set of patterns Ψ₁, but two datasets x and z, where z is an anonymized version of x? If Ψ₁ are patterns found in x, what aspects of Ψ₁ are retained in z?
“Utility of the data” can be a nebulous term, with the definition of “utility” often depending on the expected workload of the data. In this chapter, we propose a framework for designing workload-specific utility measures that evaluate patterns discovered in the data. More precisely, we compare an original dataset $x$ to an anonymized version $z$, and measure how the patterns discovered in the original dataset have changed after anonymization. We introduced the idea of patterns in Section 2.2.1, where a set of antecedents predicts a consequent, $\psi \rightarrow c$. We implement our proposed framework with three straightforward measures of pattern retention, but they are by no means exhaustive. Which patterns a data owner deems valuable can vary wildly depending on the owner’s context and goals, and it would be misguided to blindly apply a “one size fits all” measure to all privacy-preservation scenarios, devoid of context. Our framework enables data owners to design customized measures to meet their needs.

5.1 Chapter Outline

Our contributions in this chapter can be summarized as follows:

The work in this chapter is undergoing the second round of review at the following journal: Sam Fletcher and Md Zahidul Islam. Measuring pattern retention in anonymized data – where one measure is not enough. Transactions on Data Privacy, 2016.

• We propose a novel framework for measuring the pattern retention of a dataset after it has been anonymized with a privacy-preservation technique.

• We implement and test three measures that use our framework and demonstrate their sensitivity to changes in pattern retention.

• Using a thought experiment, we demonstrate that other pre-existing measures are not sensitive to changes in pattern retention, while our measures are.

• We also provide a correlation matrix of our three measures and three pre-existing measures, showing that there is almost no correlation between the performance of a classifier built from the anonymized data, and the retention of the original patterns. This demonstrates that no single measure can be expected to inform the user (e.g. the data owner) about every change in the data after anonymization, and that our framework captures information that no pre-existing measure does.

• We analyze the effect of anonymization on four specific patterns found in the Adult dataset.

• We use a real-world differentially-private anonymization technique [Xiao et al., 2010] and measure the utility of the resulting dataset.

We also make the code for our three implementations of our proposed framework available online.\footnote{The code can be found at \url{http://samfletcher.work/code} or \url{http://csusap.csu.edu.au/~zislam/}.}

In Section 5.3, we discuss related work, as well as three pre-existing utility measures that we will use in our experiments. In Section 5.4, we propose a generalized framework for measuring the retention of patterns in anonymized data. In Section 5.5, we present three implementations of the proposed framework. In Section 5.6, we use a thought experiment to explore our measures alongside pre-existing measures. In Section 5.7, we detail our experiments, and in Section 5.8, we present our empirical findings. We summarize our thoughts and conclude the chapter in Section 5.9.

## 5.2 The Setting

In this chapter, we frame the problem from the perspective of the data owner, where the data owner is wishing to release an anonymized version of their data to the public that retains the patterns that exist in the original version of the data. Figure 5.1 presents a high-level view of the scenario. The data owner applies an anonymization technique to their dataset $x$, outputting an anonymized version $z$. They can then assess the quality...
Figure 5.1: A high-level diagram of the scenario discussed in this chapter, where a data owner is using an anonymization technique to output an anonymized version of their data to the public. The anonymization technique might add noise, or generate new records, or use another method; the anonymization technique itself is out of the scope of this chapter. We focus on measuring the comparative utility of $z$ compared to $x$, highlighted in bold.

(that is, utility) of the anonymized data, and re-do the anonymization process with different parameters or a different technique if they are not satisfied with $z$’s quality. When they are satisfied, the anonymized dataset is released to the public.

In keeping with the definitions introduced in Chapter 2, we define a dataset $x$ (or $z$) as a two-dimensional table made up of independent rows, each defined by the values it possesses in each column. Each row represents a record $r \in x$, and each column represents an attribute $A \in A$. Each $A$ is made up of its own set of values, with each $r$ possessing one value $v$ per $A$, written as $r_A = v; \forall A \in A$. When $x$ is anonymized to preserve privacy, we denote the anonymized dataset as $z$. To simulate the process of predicting the labels of future records, some records are excluded from $x$, with a model being built using only the “training dataset” $x$. The model’s performance can then be tested with the “test dataset” $w$. Since the labels of these excluded records are already known, the user can tell if the patterns discovered in $x$ predicted the labels in $w$ correctly [Han et al., 2006].

In order to preserve privacy, a dataset can be anonymized in a variety of ways. We explored many of these ways in Section 3.1. In this chapter we focus on the non-interactive scenario, where the data owner releases an anonymized version of the dataset to the public and then forgets about it, having no more say in how the dataset will be used.
Note that the specific method of anonymization is not the focus, rather the focus is on how to measure the utility of the data once the anonymization method has been used. It is also worth mentioning that for many of these anonymization techniques, such as $k$-anonymity and differential privacy, the level of privacy preservation achieved is determined by the parameters used during the anonymization process, not via measurement after the fact. For example, the level of privacy achieved by an anonymization technique using differential privacy is determined by the size of the privacy budget $\epsilon$ used, which mathematically bounds the probability of a participant being detected in $z$ [Dwork and Roth, 2013, Jagannathan et al., 2012]. Measurements are only required to evaluate the utility of the data; not the privacy.

Regardless of which anonymization technique is used, some degradation of dataset $x$’s utility is unavoidable, due to $z$ being not as truthful as $x$ by definition. Balancing this loss of utility with privacy requirements is known as the privacy-utility trade-off, and optimizing this trade-off is key to a successful privacy-preservation technique. In order to assess the utility of a dataset anonymized to preserve privacy, it is currently common practice [Brickell and Shmatikov, 2008, Fung et al., 2005, Islam and Brankovic, 2011] to use a variety of data mining techniques to discover patterns in the anonymized dataset $z$, and then see if those patterns can correctly predict the labels of future records. Data mining techniques such as decision forests [Breiman, 2001a, Islam and Giggins, 2011], association rule mining [Evfimievski et al., 2004] and frequent pattern mining [Han et al., 2007] are some of the methods that can be used to extract patterns, where the patterns are in the form $\psi \rightarrow c$.

To refresh the reader about the terms we use in this chapter: $\psi$ is a collection of antecedents (in other words, conditions or requirements) that when true for a record $r$, predict that a consequent $c \in C$ (in other words, a label or class value) is also true for that record. The antecedents in $\psi$ will contain conditions for specific values $v \in A$ of attributes $A \in A$ that some records will meet and others will not. These conditions can be of the form $r_A = v$ or $r_A \in V$ for discrete attributes, where $V$ is some subset of $A$; or in the case of continuous attributes, be of the form $r_A > v$, $r_A < v$, or contain some other operator. Negation versions of these operators can also be used, such as $\neq$ and $\notin$. See Table 5.1 for some examples of what antecedents might look like.

If a record $r$ meets every condition in $\psi$ (in other words, $r \in \sigma_\psi(x))^2$, it is predicted to have a label $C = c$ either with certainty or with some probability (in other words,

\[ \sigma_\psi(x) \]

\text{Read $\sigma$ as the mathematical symbol for selection. For example, $\sigma_\psi(q)$ is the subset of elements in $q$ for which $\psi$ is true. $\psi$ can either be a statement such as $C = c$, or a set of statements such as $\psi$, in which case all statements in set $\psi$ must be true for an element in $q$ in order for that element to be in the set $\sigma_\psi(q)$.}
0 ≤ P(ρC = c) ≤ 1. Note that we abuse notation and simplify ψ → c to just ψ when it is clear from context that we mean the whole pattern.

Which patterns the data owner deems important enough to assess for changes in utility is outside the scope of this chapter. What makes a pattern valuable can vary depending on the expected workload; that is, the needs of the user. Measures have been developed to assess different aspects of patterns, such as a pattern’s support or coverage [Tan et al., 2002, Webb and Brain, 2002], confidence [Dasseni et al., 2001], conciseness [Padmanabhan and Tuzhilin, 2000], peculiarity [Zhong et al., 2003], or many other aspects depending on the user’s needs [Fletcher and Islam, 2015a, Geng and Hamilton, 2006, Tan et al., 2004, Vaillant et al., 2004]. These measures are often collectively referred to as interestingness measures. How the patterns are discovered is also outside of the scope of this chapter – any patterns in the form ψ → c are applicable to the solution proposed in this chapter. In our experiments, we arbitrarily use the CART decision tree algorithm [Breiman et al., 1984] to generate a collection of patterns. The number of patterns that are assessed for changes in utility can be as high as the data owner likes.

We refer to the set of all patterns \{ψ_i; \forall i\} discovered in a dataset \(x\) as \(Ψ_x\). If \(Ψ_x\) is used to predict \(C\) for all records in \(w\), we write the average accuracy \(α\) of these predictions (that is, the “prediction accuracy”) as \(α(Ψ_x|w)\). In words, this can be read as “The accuracy of \(Ψ_x\) at correctly predicting class labels of records in \(w\)”. Some examples of patterns can be seen in Table 5.1, including the probability of \(ρ\) having a label \(c\). Our proposed framework and its implementations are independent of methods for discovering patterns – any patterns in the form \(ψ → c\) are applicable, regardless of whether they were manually found, found with a decision tree or via association rule mining or frequent pattern mining, or any other method.

## 5.3 Related Work

One of the most popular methods for assessing the utility of \(z\) is to build a model from it, and compare its performance to the performance of a model built from \(x\) [Agrawal and Srikant, 2000, Bertino et al., 2008, Brickell and Shmatikov, 2008, Fletcher and Islam, 2015b, Fung et al., 2005, 2010a, Islam and Brankovic, 2011]. Prediction accuracy, for example, can tell the data owner how accurate the two models are predicting the class labels of previously-unseen records \(r \in w\) [Han et al., 2006]. For models that can be written in terms of patterns, such as decision tree classifiers, we can denote the two models as \(Ψ_z\) and \(Ψ_x\). Prediction accuracy is sometimes reversed to instead represent Prediction Error\(^3\) [Fung et al., 2010a], also known as Error Rate [Mancuhan and Clifton, 2003].

\(^3\)In other words, 1 − prediction accuracy = prediction error.
Table 5.1: A selection of patterns discovered in the Adult dataset [Bache and Lichman, 2013].

<table>
<thead>
<tr>
<th>i</th>
<th>Antecedent, $\psi_i$</th>
<th>Consequent, $Pr(C = c)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$206134 &gt; \text{Census Weighting} \leq 346177$ AND $\text{Capital Gains} \leq 4737$ AND $\text{Age} &gt; 40.5$ AND $\text{Capital Loss} \leq 1836.5$ AND $\text{Hours per Week} &gt; 52.5$</td>
<td>$Pr(\text{Income} \leq 50,000) = 0.57$, $Pr(\text{Income} &gt; 50,000) = 0.43$</td>
</tr>
<tr>
<td>1</td>
<td>$24440.5 &lt; \text{Census Weighting} \leq 249542$ AND $\text{Capital Gains} \leq 4737$ AND $\text{Age} \leq 40.5$</td>
<td>$Pr(\text{Income} \leq 50,000) = 0.92$, $Pr(\text{Income} &gt; 50,000) = 0.08$</td>
</tr>
<tr>
<td>2</td>
<td>$\text{Census Weighting} \leq 206134$ AND $\text{Capital Gains} &gt; 5316.5$</td>
<td>$Pr(\text{Income} \leq 50,000) = 0.05$, $Pr(\text{Income} &gt; 50,000) = 0.95$</td>
</tr>
<tr>
<td>3</td>
<td>$116388 &gt; \text{Census Weighting} \leq 200855.5$ AND $\text{Capital Gains} \leq 5316.5$ AND $\text{Capital Loss} \leq 1198$ AND $\text{Hours per Week} \leq 53.5$</td>
<td>$Pr(\text{Income} \leq 50,000) = 0.81$, $Pr(\text{Income} &gt; 50,000) = 0.19$</td>
</tr>
</tbody>
</table>

2016]. Other common measures are F-measure [Fung et al., 2008, 2010b, Mukherjee et al., 2006] and AUC [Herranz et al., 2010, Rana et al., 2016, Yu et al., 2008], where again $\Psi_z$ is compared to $\Psi_x$ using $w$ (refer to Figure 5.2 for a graphical representation of these datasets and sets of patterns). These measures are not limited to measuring the utility of patterns, and can be used in any classification model, such as $k$-Nearest Neighbor models [Maneeshan and Clifton, 2016]. We use prediction accuracy, F-measure and AUC in our experiments. These measures were introduced in Section 3.2.2, but are re-introduced here to aid clarity.

Formally, prediction accuracy can be written as:

$$\alpha(\Psi_x|w) = \frac{1}{|w|} \sum_{r \in w} 1(\Psi_x(r) = r_C) ,$$  \hspace{1cm} (5.1)

where $\Psi_x(r)$ is the outputted label when record $r$ is inputted into classifier $\Psi_x$ (that is, the set of patterns), and $1(\bullet)$ is the indicator function, returning 1 if $\bullet$ is true and 0 otherwise. Thus, prediction accuracy is the fraction of records that have their class label correctly predicted.

AUC and F-measure are most appropriate when $C$ is binary (in other words, $|C| = 2$), and where $c_1 \in C$ is the “important” class label, or the class label that the user is trying to correctly predict, and $c_2 \in C$ is unimportant. These labels are often referred to as the “positive” and “negative” labels, respectively. A “True Positive” (TP) label is therefore
a label that was correctly predicted to be positive, a “False Positive” (FP) is a label that was predicted to be positive but was not, and similarly for “True Negative” (TN) and “False Negative” (FN).

F-measure [van Rijsbergen, 1979] can be formally written in terms of precision \( \frac{TP}{TP+FP} \) and recall \( \frac{TP}{TP+FN} \) as:

\[
F_\beta = (1 + \beta^2) \times \frac{\text{precision} \times \text{recall}}{\beta^2 \times \text{precision} + \text{recall}},
\]

where \( \beta \) is very often equal to 1, so that recall and precision have equal weighting. We use \( \beta = 1 \) in our experiments.

Meanwhile, AUC [Hanley and McNeil, 1982] is shorthand for “Area under the ROC curve”, with “ROC” in turn being short for “Receiver Operating Characteristic”. The ROC curve describes the trade-off between TP (benefits) and FP (costs). Often it is plotted on axes with the TP Rate \( \frac{TP}{TP+FN} \) as the y-axis and the FP Rate \( \frac{FP}{FP+TN} \) as the x-axis. Thus, AUC is the area under this curve. It represents the probability that \( \Psi_x \) is more likely to predict a positive label as positive than to predict a negative label as positive. It has become popular in the machine learning community as of late, despite some problems it has when comparing different classifiers [Hand, 2009, Lobo et al., 2008].

Comparing the performance of \( \Psi_z \) to the performance of \( \Psi_x \) has a shortcoming, however: it cannot tell the user if the patterns discovered in \( z \) are the same patterns discovered in \( x \). There is no way of knowing if the anonymization process applied to \( x \) caused the original patterns to change (or disappear), or if weaker patterns became strong enough to become more prominent. Similar problems have been identified in the past, specifically that the performance of one type of classifier does not mean that other classifiers will perform similarly [Brickell and Shmatikov, 2008]. In fact, if the utility of an anonymized dataset is judged based on the performance of one or more classifiers, it is recommended that the dataset is not released at all, and instead just release the classifiers [Brickell and Shmatikov, 2008]. Our proposed framework solves this problem by measuring the utility of the anonymized data directly. Other problems arise from relying too heavily on prediction accuracy, such as when the data labels are imbalanced [Menardi and Torelli, 2014], or when the comprehensibility of the classifier needs to be considered as well [Freitas, 2013].

Less work has been done on utility measures that specifically focus on pattern retention, but not none. A measure known as “average relative error for aggregate counts”, or RE, has been used in the past [Wang et al., 2016, Xiao and Tao, 2006b], where datasets \( x \)

---

4 Using this notation, we can actually re-write prediction accuracy as \( \frac{TP+TN}{TP+TN+FP+FN} \).
and $z$ are both queried with the same count query $f$, and the outputs are compared:

$$RE = \frac{|f(x) - f(z)|}{f(x)}.$$ 

A query is very similar to a pattern except for the consequent; it uses a set of conditions to filter a dataset down to the subset of records that obey all the conditions. As we will demonstrate in Section 5.5.2, RE can be thought of as an implementation of our more general framework for measuring pattern retention.

A similar but different area of research related to our work is utility-based anonymization techniques, in which a utility measure is used as a cost function during the anonymization process [Hua and Pei, 2008]. Our proposed framework differs from this approach in that it is not for designing cost functions, but instead for designing standalone, workload-specific utility measures. The difference is best illustrated with an example: Information Gain is a common utility measure that is used as a cost function in decision tree algorithms [Quinlan, 1993], but using it to assess the utility of a finished decision tree would not be particularly insightful or useful. Conversely, prediction accuracy would perform poorly as a decision tree cost function [Friedman and Schuster, 2010], but is a good way to assess a decision tree’s performance at predicting labels. Our proposed framework is for designing utility measures closer in purpose to prediction accuracy than to Information Gain.

Some measures can be used in both of the above scenarios. Kullback-Leibler (KL) divergence [Kullback and Leibler, 1951], for example, has been proposed as a standalone utility measure for measuring the difference between two probability distributions; one of the original dataset $x$, and one generated from anonymized marginals (also known as frequency tables) of $x$ [Kifer and Gehrke, 2006]. KL-divergence has also been proposed as a cost function in a bottom-up, greedy anonymization algorithm for achieving $k$-anonymity [Kameya and Hayashi, 2016]. Other measures exist for measuring the difference between two distributions, such as Chi-squared histogram distance [Pele and Werman, 2010], which we will be using in Section 5.5.3. The most appropriate measure to use when comparing distributions depends on the properties of the distributions, such as if they are modeling continuous numerical data, discrete numerical data or discrete categorical data; and if discrete, whether or not both distributions have the same number of buckets [Pele and Werman, 2010].
5.4 A Framework for Measuring Pattern Retention

While prediction accuracy is an excellent measure when evaluating the utility of a classifier or model [Ferri et al., 2009, Sokolova and Lapalme, 2009], care needs to be taken when extending its use to the privacy-preservation domain. It has been common in the past for the impact of privacy-preservation techniques on the utility of the data to be measured with prediction accuracy [Agrawal and Srikant, 2000, Bertino et al., 2008, Brickell and Shmatikov, 2008, Fletcher and Islam, 2015b, Fung et al., 2005, 2010a, Islam and Brankovic, 2011]. This necessitates applying a data mining technique to the anonymized data to build a classifier (or discovering a set of patterns with another technique) and then testing how well the discovered patterns\(^5\) \(\Psi_z\) can accurately predict the class label of records in a test dataset \(w\). The accuracy \(\alpha\) of \(\Psi_z\) can then be compared to the accuracy of \(\Psi_x\) (in other words, \(\alpha(\Psi_x|w) - \alpha(\Psi_z|w)\)), and the difference is considered to be how much the privacy-preservation technique has affected the data. See Figure 5.2 for a graphical representation of the datasets and sets of patterns used in this discussion. Examples of patterns that could be discovered in the \(x\) and \(z\) datasets in Figure 5.2 are presented in Table 5.2 and Table 5.3, respectively.

We see two problems with this current framework:

**Problem 1** It only tells the user if the particular technique used to find the patterns in \(z\) produced a good classifier / model / set of patterns. Perhaps some amount of implicit assumptions can be made about the ability of other techniques to perform similarly well (in other words, “technique \(f\) did well, so techniques \(g\) and \(h\) probably produce similar results”), but there is nothing explicitly said by the quality of one \(\Psi_z\) about the quality of all data mining techniques applied to \(z\).

**Problem 2** The user cannot tell if the patterns in \(\Psi_z\) are the same patterns that can be discovered in \(x\) (such as the patterns that would be discovered if the same data mining technique was applied to \(x\), producing \(\Psi_x\)). Some patterns might still be there, while others might not, and the patterns that are still there might have changed in any number of ways (such as changes in the support or confidence of the pattern, or the values \(v\) used in the conditions in \(\psi\)).

A solution to Problem 1 is for the data owner to build a \(\Psi_z\) and \(\Psi_x\) with every possible data mining technique they think is worth checking [Brickell and Shmatikov, 2008]. This solution is extremely computationally expensive, and does not address Problem 2. If this solution is not used, then a user must either trust the implicit assumption that

\(^5\)Note that a classifier is semantically the same as a set of patterns if it can be broken down into antecedents and consequents.
other data mining techniques will perform similarly, or release the collection of patterns \(\Psi_z\) that they did test, and not release \(z\) to the public at all [Brickell and Shmatikov, 2008]. To the best of our knowledge, aside from our preliminary investigation [Fletcher and Islam, 2014], no solution to Problem 2 currently exists in the literature.

We therefore propose a framework that addresses both problems. Rather than discovering a collection of patterns in \(z\) (in other words, \(\Psi_z\)) that may or may not have any relation to the patterns in \(x\) (in other words, \(\Psi_x\)), we propose that the data owner defines a collection of patterns found in \(x\) and evaluates if the data in \(z\) still follows those patterns. That is to say, we propose designing measures of pattern retention using the following framework:

\[
\Psi_x|z
\]

No model \(\Psi_z\) needs to be computed in order to measure the pattern retention of \(z\). Nor is \(w\) required, unless it is desired for other, unrelated testing.
Chapter 5. Measuring Pattern Retention after Anonymization

Table 5.2: An example of what $\Psi_x$ could look like: some patterns discovered from $x$ in Figure 5.2. We include the confidence and support of the patterns as well. These patterns could be manually discovered, or discovered by a data mining algorithm, such as a decision tree.

<table>
<thead>
<tr>
<th>$i$</th>
<th>Antecedent, $\psi_i$</th>
<th>Consequent, $Pr(C = c)$</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$a_0$</td>
<td>$Pr(c_0) = 0.66, ; Pr(c_1) = 0.34$</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>$a_1$</td>
<td>$Pr(c_0) = 0.0, ; Pr(c_1) = 1.0$</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>$a_2 \text{ AND } b_0$</td>
<td>$Pr(c_0) = 1.0, ; Pr(c_1) = 0.0$</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>$a_2 \text{ AND } b_1$</td>
<td>$Pr(c_0) = 1.0, ; Pr(c_1) = 0.0$</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 5.3: An example of what $\Psi_z$ could look like: some patterns discovered from $z$ in Figure 5.2. We include the confidence and support of the patterns as well. These patterns could be manually discovered, or discovered by a data mining algorithm, such as a decision tree.

<table>
<thead>
<tr>
<th>$i$</th>
<th>Antecedent, $\psi_i$</th>
<th>Consequent, $Pr(C = c)$</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$a_0 \text{ AND } b_0$</td>
<td>$Pr(c_0) = 1.0, ; Pr(c_1) = 0.0$</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>$a_0 \text{ AND } b_1$</td>
<td>$Pr(c_0) = 1.0, ; Pr(c_1) = 0.0$</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>$a_1$</td>
<td>$Pr(c_0) = 1.0, ; Pr(c_1) = 0.0$</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>$a_2$</td>
<td>$Pr(c_0) = 0.34, ; Pr(c_1) = 0.66$</td>
<td>3</td>
</tr>
</tbody>
</table>

This framework nullifies the problem described in Problem 1 by not restricting the definition of “utility” to the context of a particular data mining technique being applied to $z$, and instead raises the level of abstraction when defining the “utility” of $z$ to a technique-independent level. It also solves Problem 2 since the same patterns that were found in $x$ are being used to evaluate $z$.

Of course, the framework used by prediction accuracy and other measures of model performance can be written similarly: $\Psi_z|w$. When using this framework, an important part of the process is to acknowledge that a model trained on $z$ cannot be tested on $z$, lest risk over-estimating the model’s performance when used in the real world. This is due to the phenomenon of over-fitting: when a machine learning algorithm is taught to distinguish between labels, the algorithm will tend to reinforce any patterns it finds that correctly classify more labels, even if the “patterns” are merely idiosyncrasies of the training data, such as noise, and do not model real life. We therefore use separate testing data $w$ to judge if the patterns the algorithm discovered are, in fact, real. Due to how similar $\Psi_x|z$ and $\Psi_z|w$ first appear based on their notation, the reader may wonder...
if the same risk of over-fitting needs to be ascribed to $\Psi_x|z$. Our framework represents a fundamentally different way of measuring properties of $z$ however – in terms of pattern retention – and the concept of over-fitting does not exist in this framework. Rather than measuring how much a set of patterns discovered in a sample is generalizable to the universe, we are measuring how much a perturbed version of the sample still follows the original patterns. The utility of the original patterns is not what is being measured in $\Psi_x|z$; that is the role of measures of model performance, and can be measured in the course of defining $\Psi_x$, before moving on to measuring pattern retention in $z$.

The next natural question is: how exactly do we evaluate $z$ with $\Psi_x$? There are many potential implementations of our proposed framework, but we provide three examples below in Section 5.5. The first measure (in Section 5.5.1) evaluates how much of $\Psi_x$ as a whole has been retained in $z$. The next two measures (in Section 5.5.2 and Section 5.5.3) evaluate the presence of each pattern in $\Psi_x$ (in other words, $\psi_i; \forall i$) separately, offering the data owner the ability to check for changes in individual patterns, as well as seeing the average change. Other implementations can easily be designed to meet the needs of the data owner. Every dataset has its own nuances, and it is usually advantageous to take those nuances into account when measuring the effect of privacy-preservation techniques, rather than trying to use a “catch-all” approach. The release of data to the public will be a one-time event (once it’s out there, there’s no taking it back!), and so spending additional resources to properly evaluate $x$ and $z$ is likely worth it.

### 5.5 Implementations of our Framework

#### 5.5.1 Pattern Accuracy

Introduced by us in a 2014 conference [Fletcher and Islam, 2014] and not published in a journal until now, pattern accuracy is a simple measure that compares $x$ and $z$. Like its name might suggest, it is very similar to prediction accuracy in that it measures the average accuracy of a collection of patterns at predicting the class value $C$ of some data. However, instead of predicting the class value of some testing data $w$, it predicts the class labels of the anonymized data $z$. If the prediction accuracy of $z$ is written as $\alpha(\Psi_z|w)$, then the pattern accuracy of $z$ is written as $\alpha(\Psi_x|z)$. In a privacy-preservation scenario the point is to compare $z$’s performance to $x$, so prediction accuracy becomes $\alpha(\Psi_z|w) - \alpha(\Psi_x|w)$, and the pattern accuracy equivalent is therefore $\alpha(\Psi_x|x) - \alpha(\Psi_x|z)$.

Note that while $\alpha(\Psi_x|x)$ should not be used to assess the quality of a classifier due to the risk of over-fitting, in this chapter we are not concerned with the generality of the

---

6 Remember that $z$ is an anonymized version of $x$, with each record in $z$ corresponding to an unaltered version in $x$. 
patterns. How $\Psi_x$ is created or defined is outside the scope of this chapter. Instead, the difference between $\alpha(\Psi_x|x)$ and $\alpha(\Psi_x|z)$ tells us the difference in the number of records that are contributing to the prediction made by each pattern (where the prediction is the majority class label). Pattern accuracy is a way of measuring the presence of $x$’s patterns in $z$; if $\alpha(\Psi_x|x) - \alpha(\Psi_x|z)$ is close to zero, then the user knows that a similar number of records in $x$ and $z$ are contributing to the correct predictions made by the classifier built from $x$. Since the records in $z$ are just anonymized versions of the records in $x$, this is a valuable thing to know! If $\alpha(\Psi_x|x) - \alpha(\Psi_x|z)$ is closer to one, the user knows that the records were anonymized in a way that reduced the presence of the patterns found in $\Psi_x$. If $\alpha(\Psi_x|x) - \alpha(\Psi_x|z)$ is negative, this is actually just as bad as a positive result of similar magnitude, because $x$ is trusted data – any random modifications made to $x$ is further from the trusted data by definition, even if some quality metrics increase. Ideally we want every pattern in $\Psi_x$ to be just as prevalent in $z$ as it is in $x$; no more, no less. Thus we define pattern accuracy as:

$$\text{Pattern Accuracy} = |\alpha(\Psi_x|x) - \alpha(\Psi_x|z)|. \quad (5.3)$$

Pattern accuracy evaluates whether $\Psi_x$, as a whole, can correctly predict $C$ for records in $z$. Since it uses an identical process to prediction accuracy (with the user simply having to redirect the measure to check $z$ rather than $w$), it gains all of the benefits of prediction accuracy such as low computation time and conceptual simplicity. What it does not do, however, is evaluate the presence of each pattern individually (in other words, $\psi_i; \forall i$). There are almost always multiple patterns that predict the same $c \in C$, so it is possible that some patterns no longer have records in $z$ that follow them (and instead those records follow different patterns) without the pattern accuracy result changing. As long as the record’s new pattern still correctly predicts $c$, the pattern accuracy measure is insensitive to this change. The following two measures avoid this insensitivity by evaluating the pattern retention in $z$ on a per-pattern basis, rather than evaluating the entire pattern list as a whole.

### 5.5.2 Pattern Support Distance (PSD)

The “support” of a pattern is the number of records in a dataset that a pattern covers [Dasseni et al., 2001, Webb and Brain, 2002], and can be represented as $|\sigma(\psi|x)|$ when describing the support of pattern $\psi$ in dataset $x$. Whether $C$ is predicted correctly is irrelevant when measuring support. To measure the support for $\psi$ in $x$ compared to $z$, we can calculate $|\sigma(\psi|x)|$ and $|\sigma(\psi|z)|$. By comparing these results, a user knows how much the presence of an individual pattern $\psi$ has changed due to the modifications
made to $x$ (resulting in $z$). This level of granularity allows the user to use their domain knowledge to make specific assessments of the status of each $\psi$. This can naturally be repeated for all $\psi \in \Psi_x$. To summarize the overall support retention of $\Psi_x$ for a dataset $z$, the mean difference can be calculated:

$$PSD = \frac{1}{|\Psi_x| \times |x|} \sum_{\psi \in \Psi_x} \left| |\sigma_\psi(x)| - |\sigma_\psi(z)| \right| .$$  \hspace{1cm} (5.4)

Note that each pattern contributes equally to the mean difference. Patterns with higher support are not assumed to be more important, since each $\psi$ in $\Psi_x$ should have already been assessed by the user as being important enough to worry about preserving in $z$, and we are now only interested in if the original support has changed. It should be noted though that patterns in $\Psi_x$ with very low support cannot reduce in size by as much as patterns with high support – support cannot go below 0 – so the presence of many patterns with low support risks “diluting” PSD.\(^7\) However it is normal for such small patterns to be considered as idiosyncrasies of $x$, and not generalizing to future records (such as $w$), and so most data mining algorithms automatically remove them from $\Psi_x$ [Han et al., 2006]. This is sometimes referred to as the “minimum support threshold”.

Pattern Support Distance (PSD) has a defined lower and upper limit of $0 \leq PSD \leq 1$, allowing for an intuitive interpretation of the result, such as: “The average percentage change in the prevalence of a pattern when anonymizing $x$ to create $z$”.

PSD is similar to measuring the average relative error for aggregate count queries (RE) [Wang et al., 2016, Xiao and Tao, 2006b], since counting the number of records that match a query is the same thing as measuring the support of a pattern. RE can be categorized as another implementation of our proposed generalized framework for measuring pattern retention. The main difference between PSD and RE is the problem domain; RE can be used when dealing with count queries, and PSD can be used when dealing with $\psi \rightarrow c$ patterns.

The aim of privacy preservation is to (1) make any individual record difficult to identify, while (2) leaving the patterns as unaffected as possible [Agrawal and Srikant, 2000, Estivill-Castro and Brankovic, 1999, Islam and Brankovic, 2011]. If the user considers support to be an important component of patterns, then PSD can be used to monitor this component. The specific records that matched each $\psi$ in $x$ is irrelevant – $\psi$ will still be just as prevalent in $z$ as it was in $x$ if other records take the place of the records that no longer follow $\psi$. In order for a record to change which pattern it matches, its values

\(^7\)This effect is caused whenever many small differences are averaged alongside several large differences. The presence of near-zero numbers effectively reduces the average, diluting the larger differences. There is nothing inherently wrong with this, but it is usually undesirable.
must have changed during the anonymization process enough for it to legitimately meet the conditions of a different pattern.

While pattern accuracy indirectly measures the support of \( \psi_i \in \Psi_x \); \( \forall i \) in \( z \), PSD does so directly, removing any uncertainty about the presence (in other words, coverage or support) of each pattern in \( z \).

5.5.3 Pattern Label Distance (PLD)

Say a record \( r \in x \) meets the conditions of a certain pattern \( \psi_i \in \Psi_x \) (in other words, \( r \in \sigma_{\psi_i}(x) \)). When \( x \) is anonymized to \( z \), it is possible that \( r \) will be changed in a way that causes it to meet the conditions of a different pattern \( \psi_j \in \Psi_x \) (in other words, \( r \in \sigma_{\psi_j}(z) \)). If this occurs, the distribution of labels (in other words, \( C \)) will change for both \( \psi_i \) and \( \psi_j \), since \( r_C \) has been removed from \( \psi_i \)’s distribution of class labels and added to \( \psi_j \)’s. The purpose of a pattern is often to predict \( C \), and so it is important to know how much that prediction might have changed in \( z \). Pattern accuracy measures this to an extent, as \( \psi_i \) and \( \psi_j \) might predict different class labels and a maximum of one of those predictions can be correct for a record \( r \). But it is also possible that the two patterns will predict the same class label, leading to no change in the pattern accuracy of \( z \) compared to \( x \) (at least as far as \( r \) is concerned). The consequent of any pattern \( \psi \) is usually the most common class label to occur out of all the records in \( \sigma_{\psi}(x) \), with any other class labels being ignored [Han et al., 2006]. This has the effect of making \( \psi \)’s prediction of \( C = c \) appear identically confident\(^8\) regardless of how high or low the frequency of \( c \) is in \( \sigma_{\psi}(z) \) compared to \( \sigma_{\psi}(x) \), as long as it remains the most frequent class label.

To avoid these problems, we use the Chi-squared histogram distance [Pele and Werman, 2010] to measure differences in the distribution of \( C \) between \( \sigma_{\psi}(x) \) and \( \sigma_{\psi}(z) \):

\[
\chi^2(\sigma_{\psi}(x), \sigma_{\psi}(z)) = \frac{1}{2} \sum_{c \in C} \frac{(f(\sigma_{\psi}(x), c) - f(\sigma_{\psi}(z), c))^2}{f(\sigma_{\psi}(x), c) + f(\sigma_{\psi}(z), c)}, \tag{5.5}
\]

where \( f(\sigma_{\psi}(x), c) \) is the relative frequency of the class label \( c \) in \( \sigma_{\psi}(x) \) (that is, the fraction of records in \( \sigma_{\psi}(x) \) that have \( r_C = c \)), and similarly for \( f(\sigma_{\psi}(z), c) \) in respect to \( \sigma_{\psi}(z) \):

\[
f(\sigma_{\psi}(x), c) = \frac{\sum_{r \in \sigma_{\psi}(x)} \mathbf{1}(r_C = c)}{|\sigma_{\psi}(x)|}.
\]

\(^8\)“Confidence” refers to the certainty or reliability of a pattern – that is, how frequent the most frequent label is [Tan et al., 2002]. If 100% of the records in a pattern have the same class label, then that pattern can be considered highly reliable.
Just like with Chi-squared hypothesis testing, Chi-squared histogram distance becomes unstable if there are less than five samples. This limitation is automatically handled if a minimum support threshold for each pattern $\psi$ was applied when making $\Psi_x$; otherwise we recommend discounting any patterns that have less than five class labels (in other words, ignoring patterns $\psi \in \Psi_x$ where $|\sigma_\psi(x)| < 5$).

Other measures of distribution distance could be used, such as KL-divergence [Kameya and Hayashi, 2016, Kifer and Gehrke, 2006], however none is as appropriate as Chi-squared histogram distance. For example, KL-divergence works best with continuous numerical data, while Chi-squared histogram distance is specifically for two distributions consisting of a small, equal number of discrete categorical buckets [Pele and Werman, 2010], which is the situation we have here where we are comparing label frequencies.

Even if the majority $c$ value in $\sigma_\psi(x)$ occurs even more frequently in $\sigma_\psi(z)$ (and thus has increased confidence), this should not be considered as an improvement unless the anonymization process that created $z$ was aiming to improve pattern utility. In scenarios such as privacy preservation, the distribution of $C$ for $\sigma_\psi(x)$ is considered to be the ground truth. PLD (Pattern Label Distance) successfully captures this scenario, where any distance away from $x$ is a reduction in utility by definition. The mean Chi-squared histogram distance of all patterns in $\Psi_x$ can then be easily calculated:

$$PLD = \frac{1}{|\Psi_x|} \sum_{\psi \in \Psi_x} \chi^2(\sigma_\psi(x), \sigma_\psi(z)) .$$  \hspace{1cm} (5.6)

It should be noted that Chi-squared histogram distance is invariant to the number of records [Pele and Werman, 2010], and so the support of a pattern (both in $x$ and $z$) does not affect the result. If the support of each pattern is deemed relevant by the user, $|\sigma_\psi(x)|$ can easily be taken into account as well. We do not recommend combining a pattern’s support difference and label distribution distance into a single result, as the results are likely to be far more informative when separate. This is true for both single patterns and the mean results (PSD and PLD). Chi-squared histogram distance is also invariant to the number of labels, so it is not restricted to datasets or patterns with a particularly sized $C$. This is often a concern with popular measures such as AUC [Hanley and McNeil, 1982] and F-measure [van Rijswijkeren, 1979], where non-binary class attributes need to be treated with care [Felkin, 2007].
Table 5.4: The results of six measures when the two patterns seen in Table 5.2 undergo changes so that they now resemble what is seen in Table 5.3.

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Accuracy</th>
<th>PSD</th>
<th>PLD</th>
<th>Prediction Accuracy</th>
<th>AUC</th>
<th>F-measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.67</td>
<td>0.67</td>
<td>0.80</td>
</tr>
<tr>
<td>$z$</td>
<td>0.50</td>
<td>0.08</td>
<td>0.34</td>
<td>0.67</td>
<td>0.50</td>
<td>0.67</td>
</tr>
<tr>
<td>Change</td>
<td>0.50</td>
<td>0.08</td>
<td>0.34</td>
<td>0.00</td>
<td>0.17</td>
<td>0.13</td>
</tr>
</tbody>
</table>

5.6 A Thought Experiment

We use a thought experiment to demonstrate the sensitivity of our measures to changes in the data that are not detected by pre-existing measures. We will use the toy datasets seen in Figure 5.2. The patterns in $\Psi_x$ have been written out in Table 5.2, along with their support and confidence. After anonymizing $x$ with a privacy-preservation technique, the result is $z$ as seen in Figure 5.2. The set of patterns $\Psi_z$ was then discovered from that anonymized data; we present the patterns in Table 5.3. We then assess the quality of $z$ using six measures: our three implementations of our proposed framework, as well as prediction accuracy, AUC and F-measure. The results are tabulated in Table 5.4.

Several things have happened here. Firstly, prediction accuracy was completely incapable of detecting any changes in $z$ compared to $x$. It is possible that an analyst would not care that $z$ is different, and is only interested in being able to make good predictions on future data (and that is fine). However, if the analyst makes any assumption about the similarity between $z$ and $x$ with prediction accuracy, they have made a very dangerous mistake. As we can see in Table 5.2 and Table 5.3, $\Psi_z$ is radically different from $\Psi_x$.

Due to the changes present in $z$, the patterns discovered in $z$ are very different from the patterns discovered in $x$. Our proposed framework solves the issue of quantifying the intuition one has about the differences between $\Psi_z$ and $\Psi_x$. Pattern accuracy, PSD and PLD were all able to accurately identify the differences between $x$ and $z$ that they are designed to identify: the overall retention of $\Psi_x$’s patterns, the changes in the patterns’ support and the changes in the patterns’ class label distribution, respectively.

AUC and F-measure were able to detect some changes, but it is important to recognize that these changes do not represent any connection between $\Psi_x$ and $\Psi_z$. Both measures started by calculating the true and false predictions of the positive and negative labels of $\Psi_x$ using $w$, and then they made similar calculations of $\Psi_z$ using $w$. At no point was $\Psi_z$ actually compared to $\Psi_x$, except indirectly, in much the same way that prediction accuracy indirectly compares them. As demonstrated by prediction accuracy’s results
in Table 5.4 though, there is no guarantee that any of these indirect comparisons will
detect any differences at all. Even if they do, if a data owner is trying to decide whether
to release \( z \) to the public (as seen in Figure 5.1), how do they use those results? Pattern
accuracy, PSD and PLD offer concrete results about \( z \)'s retention of \( x \)'s patterns.

5.7 Experiment Methodology

To empirically evaluate our three measures, we carry out the below experiments and
present the results in Section 5.8.

1. We analyze individual patterns in Section 5.8.1.
2. We detect the degradation of pattern retention as higher anonymization require-
ments are used in Section 5.8.2.
3. We measure correlations between six different measures in Section 5.8.3.
4. We test the usefulness of PSD and PLD in a real-world scenario in Section 5.8.4.

For all experiments except for the analysis of individual patterns, 10-fold cross valida-
tion is used, with each real-world dataset being split into a training dataset \( x \) and a
testing dataset \( w \). We use 17 datasets publicly available in the UCI Machine Learning
Repository [Bache and Lichman, 2013], and list their details in Table 5.5. When an
anonymized dataset \( z \) is required in the experiments, we use one of three anonymiza-
tion techniques (described in Section 5.7.1 and Section 5.7.2), applying it to \( x \). The
anonymization process is repeated 10 times, creating 10 separate \( z \)'s, and the results of
each measure are aggregated.

For \( \Psi_x \) and \( \Psi_z \), we generate the patterns with decision trees. Note that the patterns
could just have easily been manually created, generated from a different classifier, filtered
using any number of interestingness measures, hand-picked from a list of generated
patterns, or by any other means that outputs patterns in the form \( \psi \rightarrow c \).

To generate a collection of patterns \( \Psi_x \) (and \( \Psi_z \)) for each dataset, we run the CART
algorithm [Breiman et al., 1984], with a minimum leaf size (in other words, minimum
support threshold) of \( |x| \times 0.02 \) and a maximum tree depth of 12. By generating patterns
in this way, we produce a set of realistic patterns for each dataset, with the patterns
also varying in length (that is, the number of conditions in \( \psi \)). Another advantage of
generating our patterns in this way is that the deterministic nature of CART allows the
reader to replicate our \( \Psi_x \)'s exactly. To analyze individual patterns in Section 5.8.1,
Table 5.5: Details of the datasets used in our experiments. The columns are, in order: the number of records in $x$; the number of continuous attributes in $x$; the number discrete attributes in $x$; the number of labels (class values) for $C$ in $x$; and the relative frequency of the most common label in $C$.

<table>
<thead>
<tr>
<th>Name</th>
<th>Records</th>
<th>Continuous Attributes</th>
<th>Discrete Attributes</th>
<th>Labels</th>
<th>Majority Label %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Banknotes</td>
<td>1372</td>
<td>4</td>
<td>0</td>
<td>2</td>
<td>55%</td>
</tr>
<tr>
<td>Vehicle</td>
<td>846</td>
<td>18</td>
<td>0</td>
<td>4</td>
<td>26%</td>
</tr>
<tr>
<td>RedWine</td>
<td>1599</td>
<td>11</td>
<td>0</td>
<td>6</td>
<td>43%</td>
</tr>
<tr>
<td>Spambase</td>
<td>4601</td>
<td>57</td>
<td>0</td>
<td>2</td>
<td>60%</td>
</tr>
<tr>
<td>Wilt</td>
<td>4839</td>
<td>5</td>
<td>0</td>
<td>2</td>
<td>95%</td>
</tr>
<tr>
<td>WallSensor</td>
<td>5456</td>
<td>4</td>
<td>0</td>
<td>4</td>
<td>40%</td>
</tr>
<tr>
<td>PageBlocks</td>
<td>5473</td>
<td>10</td>
<td>0</td>
<td>5</td>
<td>90%</td>
</tr>
<tr>
<td>OptDigits</td>
<td>5620</td>
<td>62</td>
<td>0</td>
<td>10</td>
<td>10%</td>
</tr>
<tr>
<td>PenWritten</td>
<td>10992</td>
<td>16</td>
<td>0</td>
<td>10</td>
<td>10%</td>
</tr>
<tr>
<td>GammaTele</td>
<td>19014</td>
<td>10</td>
<td>0</td>
<td>2</td>
<td>65%</td>
</tr>
<tr>
<td>Shuttle</td>
<td>58000</td>
<td>9</td>
<td>0</td>
<td>7</td>
<td>79%</td>
</tr>
<tr>
<td>Credit</td>
<td>653</td>
<td>6</td>
<td>9</td>
<td>2</td>
<td>55%</td>
</tr>
<tr>
<td>Parkinsons</td>
<td>1040</td>
<td>28</td>
<td>1</td>
<td>2</td>
<td>50%</td>
</tr>
<tr>
<td>Yeast</td>
<td>1484</td>
<td>7</td>
<td>1</td>
<td>10</td>
<td>31%</td>
</tr>
<tr>
<td>Cardio</td>
<td>2126</td>
<td>21</td>
<td>1</td>
<td>3</td>
<td>78%</td>
</tr>
<tr>
<td>Adult</td>
<td>30162</td>
<td>6</td>
<td>5</td>
<td>2</td>
<td>75%</td>
</tr>
<tr>
<td>Bank</td>
<td>45211</td>
<td>7</td>
<td>9</td>
<td>2</td>
<td>88%</td>
</tr>
</tbody>
</table>

Four patterns were manually selected from the first CART decision tree built from the Adult dataset, and presented in Table 5.1.

Our datasets range in size from 653 to 58000 records, 6 to 62 attributes, and 2 to 18 class labels, and include both numerical and categorical attributes. The number of patterns (in other words, $|\Psi_x|$) ranges from 11 to 37. The details of the datasets are summarized in Table 5.5. For experiments involving AUC and F-measure (e.g. Table 5.6, discussed later) we limit our experiments to datasets with binary labels, where these measures are known to work best.
5.7.1 Toy Privacy Preservation Techniques

To simulate various anonymization techniques applied to a dataset, we add noise to the data in two simple ways. Each type of noise represents a different scenario respectively: where attribute and multi-attribute (in other words, multivariate) distributions are flattened (in other words, made more uniform); and where attribute distributions and most multi-attribute distributions are preserved. We simulate these scenarios using additive noise. Using these two scenarios, we explore what a user can learn from our three implementations of our proposed framework, and how they compare to prediction accuracy, AUC and F-measure.

The two types noise addition we use are listed below. Note that these are simple toy noise addition techniques, and are not part of this thesis’ contribution. Neither of these noise types add noise to the label $C$. For each type of noise, we increase the percentage chance of changing a value in 2% increments, from 0% to 30%.

**Uniform Noise (UN):** A user-defined percentage of values in $x$ are changed, with the result being $z$. If a value $r_A$ is changed and $A$ is a continuous attribute, the new value is selected from a uniform distribution between the minimum and maximum values of $A$. If $A$ is a discrete attribute, $r_A$ is changed to any unique value in the set $A$, with each value having an equal probability of being selected. Values are randomly selected, with the original value having no effect on the new value. This has the effect of flattening the distribution of values for all attributes, as well as flattening all multivariate distributions.

**Gaussian Noise (GN):** A user-defined percentage of values in $x$ are changed, with the result being $z$. If a value $r_A$ is changed and $A$ is a continuous attribute, a random number is selected from a Gaussian distribution with a mean of zero and a variance equal to $A$’s variance, and added to $r_A$. If $A$ is a discrete attribute, $r_A$ is changed to a value that is randomly selected from $A$’s original set of values (including repeated values). GN therefore maintains the distribution of values for both numerical and categorical attributes. Additionally, continuous values are changed in a way that takes into account the original value. This means that each record’s continuous values are likely to remain close to their original values, and thus the multivariate distribution of the dataset is likely to be preserved.

5.7.2 Real-World Differential Privacy

To simulate a real-world example, we anonymize the data of the Banknotes dataset using the differentially-private technique proposed by Xiao et al. [2010]. This involves
partitioning the data into disjoint subsets, using noisy count queries to return the number of records in each subset, and then generating new records based on the attribute domains in each subset. The count queries are made noisy using Laplace noise (as is common in differential privacy Section 2.5), where the amount of noise is dictated by the size of the privacy budget; the smaller the budget, the more noise that needs to be added to maintain differential privacy. Similarly to how we can increase the amount of noise induced by UN and GN, we can increase the amount of noise added by differential privacy by decreasing the privacy budget.

### 5.7.3 Pearson’s Correlation Coefficient

For this experiment, we use the results of six different measures: pattern accuracy, PSD, PLD, prediction accuracy, AUC, and F-measure. By comparing the results of each measure as noise increases, for each dataset, we calculate the measures’ correlation to each other using Pearson’s correlation coefficient (in other words, Pearson’s $r$ value) [Pearson, 1901]. We calculate their correlation for each noise type separately. The coefficient has a range of $-1 \leq r \leq 1$, where a result close to 1 indicates a high positive correlation (as one measure increases, so does the other measure), a result close to -1 indicates a high negative correlation (as one measure increases, the other decreases), and a result close to 0 indicates low correlation (the result of one measure has little bearing on the result of the other). To standardize the results of different datasets, we look at the difference between each measure’s result on $z$ compared to $x$. In other words, for each level of noise, we subtract the result that the measure achieved when there was zero noise. This has no effect on our proposed measures (which always equal 0 when there is no noise), and simply causes prediction accuracy, AUC and F-measure to be reported as the difference between the “true” result and the “noisy” result.

### 5.8 Our Implementations in Practice

In this section, we run the experiments outlined in Section 5.7, demonstrating how the implementations of our proposed framework work in practice.

#### 5.8.1 Analyzing Individual Patterns

To demonstrate the information a user can learn about individual patterns, Figure 5.3 presents the support and Chi-squared histogram distance of the example patterns shown in Table 5.1, as UN increases. In this example, we can see that the four patterns are
Chapter 5. *Measuring Pattern Retention after Anonymization*

![Graphs showing support and chi-squared histogram distance for patterns ψ0, ψ1, ψ2, and ψ3 from Adult as UN increases.]

**Figure 5.3:** The Support and Chi-squared Histogram Distance of the example patterns shown in Table 5.1 (discovered in the Adult dataset), as UN increases. The left y-axis measures the Support and the right y-axis measures the Chi-squared Histogram Distance.

affected quite differently by the noise addition. Some of the observations a data scientist could make about these four patterns are:

- $\psi_3$ has gone from representing over 8000 of the 30162 records in Adult to representing only 3000 records in the anonymized version of Adult by the time UN has
reached 30%.

- Despite this massive change in support, the distribution of the class labels in $\psi_3$ is actually almost exactly the same at all noise levels.

- The same cannot be said for $\psi_2$, where a massive change in support (from less than 1000 to roughly 4000) has been accompanied by a massive change in the distribution of class labels as well.

  - If this observation caused the user to investigate further, they would find that $\psi_2$’s change in label distribution was enough to completely flip the prediction the pattern is making! At 30% noise, the reported probability of a record having each label is $Pr(Income \leq $50,000) = 0.68, $Pr(Income > $50,000) = 0.32, compared to the probabilities that we can see in Table 5.1: $Pr(Income \leq $50,000) = 0.05, $Pr(Income > $50,000) = 0.95. It would be incredibly damaging to any analysis performed with $z$ if the user trusted this pattern.

- $\psi_0$ and $\psi_1$ represent a much smaller proportion of the Adult dataset, and have undergone moderate changes in support. $\psi_0$ has grown larger, while $\psi_1$ has become smaller, but neither saw enough change in label distribution to cause concern.

- These patterns in Adult were discovered with a decision tree, along with 31 other patterns that underwent a variety of changes in support and label distribution similar to the changes shown in Figure 5.3.

### 5.8.2 Detecting Pattern Retention as Noise Increases

After averaging the support distance and Chi-squared histogram distance of all patterns and thus calculating PSD and PLD, we can compare their assessments of $z$’s pattern retention for each dataset. We also compare PSD and PLD to the assessment made by pattern accuracy. For each dataset, we present the results of PSD, PLD and pattern accuracy as UN increases in Figure 5.4. Note that for pattern accuracy, we present the percentage of cases where $\Psi_x$ incorrectly predicts the label of records in $z$ so that lower values signify better pattern retention for all three measures.$^9$ As more noise is added, we observe that all three measures trend up as expected, but upon closer inspection we can see that they do not do so at identical rates.

$^9$That is, pattern accuracy error = 1 – pattern accuracy.
Figure 5.4: The mean results of PSD, PLD and pattern accuracy error as UN increases. The left-hand y-axis corresponds to PLD and PSD. The right-hand y-axis corresponds to pattern accuracy error. The x-axis is the percentage of noise from 0% to 30%.
Table 5.6: A matrix of correlations for each combination of two measures, for all two noise types. We include the $p$ value of each correlation in brackets (i.e., the probability of observing a result at least as extreme as the one reported by chance, assuming there is zero correlation).

<table>
<thead>
<tr>
<th>Measure</th>
<th>PLD</th>
<th>PSD</th>
<th>Prediction Accuracy</th>
<th>AUC</th>
<th>F-measure</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>UN</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pattern Accuracy</td>
<td>-0.77 (0.00)</td>
<td>-0.83 (0.00)</td>
<td>0.36 (0.00)</td>
<td>0.34 (0.00)</td>
<td>0.26 (0.00)</td>
</tr>
<tr>
<td>PLD</td>
<td>0.58 (0.00)</td>
<td>-0.32 (0.00)</td>
<td>-0.23 (0.01)</td>
<td>-0.09 (0.33)</td>
<td></td>
</tr>
<tr>
<td>PSD</td>
<td>-0.21 (0.02)</td>
<td>-0.35 (0.00)</td>
<td>-0.28 (0.00)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prediction Accuracy</td>
<td>0.77 (0.00)</td>
<td></td>
<td>0.47 (0.00)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>GN</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pattern Accuracy</td>
<td>-0.86 (0.00)</td>
<td>-0.77 (0.00)</td>
<td>0.19 (0.04)</td>
<td>0.20 (0.02)</td>
<td>0.21 (0.02)</td>
</tr>
<tr>
<td>PLD</td>
<td>0.44 (0.00)</td>
<td>-0.04 (0.67)</td>
<td>-0.08 (0.35)</td>
<td>-0.09 (0.34)</td>
<td></td>
</tr>
<tr>
<td>PSD</td>
<td>-0.13 (0.14)</td>
<td>-0.27 (0.00)</td>
<td>-0.26 (0.00)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prediction Accuracy</td>
<td>0.72 (0.00)</td>
<td></td>
<td>0.50 (0.00)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>AUC</strong></td>
<td></td>
<td></td>
<td></td>
<td>0.91 (0.00)</td>
<td></td>
</tr>
</tbody>
</table>

5.8.3 Correlations Between Utility Measures

The differences in trends seen in Figure 5.4 are quantified by the correlations between the measures, seen in Table 5.6. The correlations are calculated using Pearson’s correlation coefficient [Pearson, 1901] as described in Section 5.7.3, and prediction accuracy, AUC and F-measure are included as well. Unlike Figure 5.4, the correlations are calculated using the nine datasets with binary labels for the benefit of AUC and F-measure.\(^{10}\)

One observation we can make about Table 5.6 is that despite all the measures using the same data, they do not always agree with each other. Just because prediction accuracy decreases does not mean that F-measure also decreases, for example. Another

\(^{10}\)The correlations among PLD, PSD, pattern accuracy and prediction accuracy when using the datasets shown in Figure 5.4 are similar to those shown in Table 5.6.
observation is that prediction accuracy, F-measure and AUC have very weak correlations with any of our implementations of our proposed framework. This is interesting, and confirms our suspicions that just because a good classifier (that is, a classifier that achieves good results) can be made from noisy data, does not mean that the patterns in the noisy data have the same properties as the original patterns, or even that the original patterns are in the noisy data at all. For example if a user observed a particular amount of prediction accuracy loss after anonymizing $x$ to $z$, there is no way to tell how much the support of the patterns in $x$ might have changed.

5.8.4 Real-World Differentially Private Data

In this experiment, we apply a real-world technique proposed in Xiao et al. [2010] that creates a differentially private version $z$ of some dataset $x$. Here we use the Banknotes dataset as $x$. The details of the technique can be found in the original paper, suffice to say that new records are created based on the original records. The amount of noise incurred during this process depends on the “privacy budget”; the less budget there is to spend, the more noise that needs to be added to preserve privacy.

The results of this experiment can be seen in Figure 5.5. Four utility measures are presented: pattern accuracy, prediction accuracy, PLD, and PSD. Several observations can be made when looking at the results. Firstly, all four measures report that the quality of $z$ worsens as the privacy budget decreases, as expected. However, we can see that aside from the initial drop between the original data (which has a functional privacy budget of infinity) and a privacy budget of 0.8, prediction accuracy does not detect any loss of quality until the budget drops to below 0.02. Pattern accuracy, on the other hand, detects a difference in quality when the budget drops to 0.08. Interestingly, pattern accuracy reports that almost all of the records in $z$ can be correctly predicted by the original patterns until the budget is 0.08. Of course, neither of these results are “wrong” – they are measuring different things. The results reported by prediction accuracy tell us that with budgets between 0.8 and 0.02, a CART decision tree can be created without any significant difference in their ability to predict the label of future records. However even with a budget of 0.8, CART cannot produce a classifier with anywhere near as good as the one produced from $x$, even though pattern accuracy tells us that $z$ obeys the original patterns. This observation may encourage the data owner running these experiments in the real world to manually view the CART tree to see where the patterns are changing, and to test with other classifiers to see if the problem persists.
Chapter 5. *Measuring Pattern Retention after Anonymization*

Banknotes

![Graph showing pattern accuracy and prediction accuracy](image)

![Graph showing PLD and PSD](image)

**Figure 5.5:** The mean results of pattern accuracy, prediction accuracy, PLD and PSD as the privacy budget decreases. We use the differentially-private technique proposed by Xiao et al. [2010] to generate new data based on the Banknotes dataset.

Similar observations can be made with PLD and PSD. The average Chi-squared distance between the labels in $x$ and $z$ that obey the patterns found in $x$ steadily rises as the privacy budget decreases. The average support of each of the original patterns, on the other hand, stays the same until the budget decreases to 0.02. A budget of 0.02 is the same budget when the CART classifier started to deteriorate, which may be enough evidence for the data owner to investigate exactly which patterns are changing in support, and whether or not they are the patterns responsible for worsening the classifier.

Utility measures such as the four presented in Figure 5.5 (and hopefully many more) provide the data owner with the information they need to make informed decisions about the data. Based on the four measures provided, if the owner had to decide what data
was of acceptable quality to release to the public, which trying to maximize privacy preservation, they may decide to use a budget of 0.02.

5.9 Discussion

None of our proposed measures can tell a user if a good classifier can be made from $z$. They are not trying to! If a user wishes to learn that, they can use machine learning algorithms on $z$ and see if the resulting classifier has good performance, using measures such as prediction accuracy. Doing so, however, will not tell them if those machine learning algorithms found the same patterns that existed in $x$. That is where our proposed framework – and our implementations of that framework – come in.

Pattern accuracy, PSD and PLD should not be interpreted as exhaustively measuring all aspects of pattern retention. Rather, they are examples of quantifying specific effects a privacy-preservation technique can have on data. It is the responsibility of the data scientist performing the anonymization of $x$ to assess what properties of a dataset are relevant or important, and then to measure how those properties might have changed after anonymization. Pattern accuracy measures the overall retention of the original patterns; PSD measures changes in pattern support, per pattern; PLD measures changes in label distribution per pattern; other measures might focus on quantifying changes in pattern conciseness or peculiarity or any number of other properties that might make patterns interesting to a user.

Prediction accuracy is currently heavily relied upon in privacy-preservation research. While the measure itself is very useful, it should not be viewed as an all-encompassing measure of the quality of anonymized data, but rather as another example of quantifying a specific property – the ability for accurate classifiers to be built using a variety of machine-learning algorithms.

Measuring properties of $Ψ_x$ in $z$ is straightforward, both conceptually and computationally, and can be easily used in conjunction with prediction accuracy and other measures. It enables the user to quantify aspects of $M$ that previously could only be assessed with experience or intuition.
Part III

Decision Trees
Truth resists simplicity.

JOHN GREEN
Chapter 6

Differentially-Private Greedy Trees

In Section 3.3 we explored the work that has been done in the past in creating decision trees under the heavy requirements of differential privacy. In this chapter, and the following two chapters, we build on the work of the past and propose new strategies for improving the utility of differentially-private decision trees.

In this chapter we balance the desire for prediction accuracy with the desire for knowledge discovery. This differs from our approach in the next two chapters, and is a trade-off that we discuss in Chapter 9. Here, we propose a decision forest algorithm based on CART [Breiman et al., 1984]; a greedy decision tree algorithm built using the process described in Section 2.3.1 using the Gini index. We focus on discrete attributes in this chapter, but note that methods exist to “discretize” continuous attributes [Kotsiantis and Kanellopoulos, 2006, Rahman and Islam, 2016].

We will be using the Gini index $G$ described in Section 2.3.1, which we repeat here for easier referencing:

$$G(x_i, A) = -\sum_{v \in A} |x_i^v| \left(1 - \sum_{c \in C} \left(\frac{|x_i^{v,c}|}{|x_i^v|}\right)^2\right)$$  \hspace{1cm} (6.1)

where $|x_i^v|$ is the number of records in $i$ with value $v \in A$, and similarly for $|x_i^{v,c}|$ with class label $c \in C$.

---

The work in this chapter has been published in the following paper: Sam Fletcher and Md Zahidul Islam. A Differentially Private Decision Forest. In 13th Australasian Data Mining Conference, pages 1–10, Sydney, Australia, 2015.
Our aim is to build $\tau$ decision trees ($0 < \tau < \infty$) by only submitting $\epsilon$-differentially private queries $f$ to $x$, and without exceeding our total budget $\beta$. The decision trees should be of acceptably high quality so that meaningful knowledge can be discovered. Figure 2.4 in Chapter 2 outlines how our algorithm interfaces with the sensitive data via differentially-private mechanisms.

### 6.1 Our Contributions

Our contributions can be summarized as the following:

- We present a differentially-private decision forest algorithm (referred to as DPDF).
- We output all the rules in the forest, including their confidence and support.
  - We also output the confidence and support of every subset of the rules (i.e. increasingly more general versions of the final rules, by removing the antecedents one at a time).
- We propose and prove a theorem for the worst-case local sensitivity\(^1\) of the Gini index.
- We demonstrate that the prediction accuracy of our DPDF is close to the accuracy of a standard Random Forest classifier [Breiman, 2001a], even for low $\epsilon$ values.

In Section 6.2 we propose our differentially private decision forest algorithm and discuss the important components of it. In Section 6.3 we discuss another technique that is similar to ours, FS [Friedman and Schuster, 2010], and how it differs from our technique. In Section 6.4 we empirically compare our technique to FS, using Random Forest [Breiman, 2001a] as a benchmark of prediction accuracy on various datasets.

### 6.2 A Differentially Private Decision Forest

In order to make a decision tree (and from there a decision forest), we need to consider what information we require about $x$. In order to make the decision tree differentially private, we then need to query $f$ in a way that returns to us all the information we need, without distorting the answers too much.

We break down the tree-building process into the following queries:

\(^1\)Sensitivity is an important component of differential privacy, described in Section 2.5.
\[ f_1(x_i) = \{|x_i^c|; \forall c \in C\} \]  \hspace{1cm} (6.2)

and

\[ f_2(x_i) = \arg \max_{A \in \mathcal{A}} (G(x_i, A)) \] , \hspace{1cm} (6.3)

These two queries are all that is required to recursively build the tree described in Section 2.3.1.

To make these queries differentially private, we add uncertainty to the output of each query in the following way:

\[ M_1(x_i) = \{|x_i^c| + \text{Lap}(\frac{1}{\epsilon}); \forall c \in C\} \] \hspace{1cm} (6.4)

and

\[ M_2(x_i) = A \in \mathcal{A} : \Pr(\arg \max_{A \in \mathcal{A}} (G(x_i, A))) \propto \exp \left( \frac{\epsilon \times G(x_i, A)}{2\Delta(G)} \right) . \] \hspace{1cm} (6.5)

In words, \( M_1(x_i) \) outputs a histogram of the frequencies of each class value \( c \) in \( x_i \), with Laplace noise added to each class count using Definition 2.4. The scale of the Laplace distribution is equal to \( 1/\epsilon \) for several reasons: when outputting a count of records, the sensitivity of the count is always \( \Delta = 1 \) [Dwork et al., 2006]; and when outputting a histogram, the sensitivity remains equal to one because each bin of the histogram is disjoint – adding or removing one record can only affect one bin in the histogram (see Definition 2.3).

In words, \( M_2(x_i) \) outputs the attribute with the best Gini index result with a certain probability, as described in Definition 2.5. The probability of any attribute \( A \in \mathcal{A} \) being outputted is dependent on the Gini index of \( A \), the sensitivity of the Gini index, and \( \epsilon \). Since \( \Delta(G) \) and \( \epsilon \) are unchanging for all \( A \), we can see that an attribute with a good Gini index is exponentially more likely to be outputted than an attribute with a poor Gini index. High \( \Delta(G) \) and low \( \epsilon \) both reduce this likelihood; we discuss how we reduce \( \Delta(G) \) to a much lower value than that suggested by Definition 2.6 in Section 2.5.

The next question is how to divide the total privacy budget \( \beta \) amongst \( M_1(x_i) \) and \( M_2(x_i) \) for all \( i \). Recall that a \( x_i \) exists for every \( v \in \mathcal{A} \), for every \( A \) used to split the previous \( x_i \) in the recursive algorithm described in Section 2.3.1, until each recursion chain has met one of the termination conditions listed in Section 2.3. Using composition (Definition 2.2), we know that the \( \epsilon \) we use for each query will be summed together, until it totals \( \beta \) and our connection to the dataset is severed by the data owner. We also know that for each depth level \( d \), each \( x_i \) is disjoint, and \( \sum_i n_i = n \) for each \( d \) (i.e. all
records will belong to one and only one \( x_i \) for each \( d \)). We can therefore treat each \( x_i \) on the same depth level in a similar way to how we treat bins in a histogram, and that adding or removing one record can only affect one \( x_i \). This allows us to apply parallel composition (Definition 2.3) and ask \( M_1 \) and \( M_2 \) in a way that returns the output for each \( x_i \), and only subtract \( \epsilon \) from \( \beta \) once for \( M_1 \) and once for \( M_2 \).

Given the above, each query we use is given the following \( \epsilon \) budget:

\[
\epsilon = \frac{\beta}{2\delta - 1} \tag{6.6}
\]

where \( 0 < \delta < m \) is the maximum depth we allow \( d \) to reach. We multiply \( \delta \) by two because we are asking two queries per depth level: \( M_1 \) and \( M_2 \). We subtract one because upon reaching the end of a recursion chain (a leaf in the tree), we only need to ask \( M_1 \) (to get the final distribution of class values).

We also expand our algorithm to be capable of producing multiple trees. Decision forests are known to often produce higher prediction accuracy than a single decision tree [Breiman, 2001a, Islam and Giggins, 2011]. The reason for this depends on the decision forest algorithm used, but is essentially because each tree will select different attributes \( A \) at different points in the tree, leading to different rules that might have better accuracy. For our algorithm, we guarantee that each tree will be different by requiring the first attribute chosen (the root) to be different for each tree. The noisy output of \( M_2 \) also provides potential for different attributes to be chosen at other points in the trees.

When calculating the prediction accuracy of forest \( F \) in this chapter, we use a simple voting technique of taking the weighted average of the predicted class values of record \( r \), where the weight of each tree \( T \)'s prediction is defined by the confidence of the most common class value in the leaf that record \( r \) fits into. This is repeated for each record \( r \) in the testing data \( w \). Any other voting technique can easily be used though, as long as it can be calculated using only the distribution of class values in \( x_i \) (unless some of budget \( \beta \) remains for more queries).

After defining the number of trees \( \tau \) to build, the first time each tree (after the first tree) asks \( M_2 \), the attributes chosen as roots of the previous trees are removed. There will therefore be \( \tau \) different attributes used as the root attributes of the \( \tau \) trees. The trees are completely independent beyond that.

We adjust the budget \( \epsilon \) given to each query to account for \( \tau \):

\[
\epsilon = \frac{\beta}{\tau(2\delta - 1)} \tag{6.7}
\]
where $1 < \tau < \infty$. We provide the full algorithm for DPDF later in Algorithm 6.1.

### 6.2.1 The Local Sensitivity of the Gini Index

The sensitivity seen in Definition 2.6 is sometimes referred to as the “global sensitivity” of query $f$, due to it making no assumptions about the data and simply returning the worst possible difference between any two datasets $x$ and $y$ that differ by one record. By using the output of query $M_1(x_i)$, we learn information about $x_i$ that allows us to greatly reduce the sensitivity of $M_2(D_i)$. We provide a theorem for the local sensitivity of $G(x_i, A)$ (and therefore of $M_2(x_i)$), and its proof, below:

**Theorem 6.1 (Local Sensitivity of the Gini Index).** The sensitivity of the Gini index $\Delta(G(x_i, A))$ when applied to data with known size $|x_i|$ is

\[
\Delta(G(x_i, A)) = 1 - \left( \frac{|x_i|}{|x_i| + 1} \right)^2 - \left( \frac{1}{|x_i| + 1} \right)^2. \tag{6.8}
\]

It is independent of $A$ and $C$ and therefore we can abbreviate $\Delta(G(x_i, A))$ to $\Delta(G(x_i))$.

**Proof.** From Equation 2.6 we see that in order to maximize $\Delta(G(x_i, A))$ we must produce a maximum and a minimum $G(x_i, A)$ such that both outputs are possible by only changing one record in $x_i$. Using the same notation outlined in Section 2.7, a dataset that differs from $x_i$ by one record is referred to as $y_i$. Using Equation 6.1, we reduce the problem to

\[
\max \sum_{c \in C} \left( \frac{|x_{v,c}^i|}{|x_i^i|} \right)^2 - \sum_{c \in C} \left( \frac{|y_{v,c}^i|}{|y_i^i|} \right)^2 \quad \tag{6.9}
\]

and will then extrapolate to all $v \in A$.

For $x_i$, we can write $\sum_{c \in C} \left( \frac{|x_{v,c}^i|}{|x_i^i|} \right)^2$ in a more general way:

\[
\sum_{h} \left( \frac{j_h}{k} \right)^2 : j, k \in \mathbb{N} \tag{6.10}
\]

where $\sum_j j_h = k$. For $y$, if we assume that we are considering $v : r_A = v$, where $r$ is in $y$ but not in $x$, and that $j_1 = c : r_C = c$ we can write $\sum_{c \in C} \left( \frac{|y_{v,c}^i|}{|y_i^i|} \right)^2$ as

\[
\left( \frac{j_1 + 1}{k + 1} \right)^2 + \sum_{h=2} \left( \frac{j_h}{k + 1} \right)^2. \tag{6.11}
\]
That is, the class count $j_1$ and the total $k$ were increased by one because $r$ was added to $y$. It is possible that $j_1 = 0$ for $x$, which we write as

$$(k + 1)^2 - k^2 > \left( (j_1 + 1)^2 + \sum_{h=2}^{k} (j_h)^2 \right) - \sum_{h=1}^{k} (j_h)^2 > \left( 1 + \sum_{h=2}^{k} (j_h)^2 \right) - \sum_{h=1}^{k} (j_h)^2$$

which means the denominator is guaranteed to increase by more than the numerator (and thus result in a smaller number) when adding $r$ to $x_i$, and Equation 6.12 will always be smaller than Equation 6.11. Thus we maximize Equation 6.9 by using Equation 6.10 (for $x_i$) and Equation 6.12 (for $y_i$).

By taking advantage of the fact that

$$\sum_{h=1}^{k} \left( \frac{j_h}{k} \right)^2 > \sum_{h=1}^{k} \left( \frac{\hat{j}_h}{k} \right)^2$$

and

$$\left( \frac{k}{k} \right)^2 > \left( \frac{j_1}{k} \right)^2 + \left( \frac{j_2}{k} \right)^2$$

where $\sum_{h} j_h = k$, we can see that the worst-case scenario (i.e. where Equation 6.9 is maximized) is when $|x_{i,v,c}| = |x_{v}^i|$ (i.e. all records in $x_{i,v}^i$ have the same class value $c$) and $y_i = x_i \cup r$ where $r \neq c$. That is,

$$\max \left( \sum_{v \in C} \left( \frac{|x_{i,v,c}|}{|x_{i,v}^i|} \right)^2 \right) = \left( \frac{|x_{i,v}^i|}{|x_{i,v}^i|} \right)^2$$

and

$$\min \left( \sum_{v \in C} \left( \frac{|y_{i,v,c}|}{|y_{i,v}^i|} \right)^2 \right) = \left( \frac{|x_{i,v}^i|}{|x_{i,v}^i| + 1} \right)^2 + \left( \frac{1}{|x_{i,v}^i| + 1} \right)^2,$$

meaning that Equation 6.9 is equal to

$$\left| \left( \frac{|x_{i,v}^i|}{|x_{i,v}^i|} \right)^2 - \left( \frac{|x_{i,v}^i|}{|x_{i,v}^i| + 1} \right)^2 + \left( \frac{1}{|x_{i,v}^i| + 1} \right)^2 \right|.$$

If Equation 6.18 is the maximum difference for $v \in A$, then it is also the maximum difference $\forall v \in A$, meaning the weighted average performed in the Gini index (Equation 6.1)
can be simplified:

\[-\sum_{v \in A} \frac{|x^v_i|}{|x_i|} \left( 1 - \sum_{c \in C} \left( \frac{|x^{v,c}_i|}{|x^v_i|} \right)^2 \right) = -\left( 1 - \sum_{c \in C} \left( \frac{|x^c_i|}{|x_i|} \right)^2 \right).\]  \hspace{1cm} (6.19)

From Equation 6.16 and Equation 6.17, we know that Equation 6.9 (and therefore Equation 6.19) is optimal when all records in \(x_i\) have the same class value. Therefore when considering Equation 6.19 for \(x_i\), we get

\[- \left( 1 - \left( \frac{|x_i|}{|x_i|} \right)^2 \right) = -(1 - 1) = 0 ,\]  \hspace{1cm} (6.20)

and when considering Equation 6.19 for \(y_i\), we get

\[- \left( 1 - \left( \frac{|x_i|}{|x_i| + 1} \right)^2 - \left( \frac{1}{|x_i| + 1} \right)^2 \right).\]  \hspace{1cm} (6.21)

Combining the solutions for \(x_i\) and \(y_i\) (Equation 6.20 and Equation 6.21), we arrive at

\[\Delta(G(x_i, A)) = \max_{x_i, y_i} |G(x_i, A) - G(y_i, A)| = |0 - \left( 1 - \left( \frac{|x_i|}{|x_i| + 1} \right)^2 - \left( \frac{1}{|x_i| + 1} \right)^2 \right)| ,\]  \hspace{1cm} (6.22)

noting that the above proof holds for the alternate case where one record is removed from a dataset by considering \(|x| = |y| - 1.\)

\(\square\)

Using Theorem 6.1 we can calculate the global sensitivity of the Gini index, where the size of \(x_i\) is not known (recalling that the sensitivity is when \(\Delta(G)\) is maximal):

\[\Delta(G) = \max_{0 < |x_i| < \infty} \left( 1 - \left( \frac{|x_i|}{|x_i| + 1} \right)^2 - \left( \frac{1}{|x_i| + 1} \right)^2 \right)\]
\[= 1 - \left( \frac{1}{1+1} \right)^2 - \left( \frac{1}{1+1} \right)^2\]
\[= 0.5 .\]

If \(|x_i|\) is known, even a modest size of \(|x_i| = 100\) heavily reduces the sensitivity of the Gini index:

\[\Delta(G(x_i)) = 1 - \left( \frac{100}{100 + 1} \right)^2 - \left( \frac{1}{100 + 1} \right)^2\]
\[= 0.0196 .\]
This drastically reduces the amount of noise added to the outputs of queries using the Exponential mechanism (Definition 2.5) where the Gini index (Equation 6.1) is the utility function $u$.

For DPDF, we can calculate $|x_i|$ using the sum of the class value frequencies we learned with query $M_1(x_i)$:

$$|x_i| = \sum_{c \in M_1(x_i)} |x_i^c|$$ (6.23)

Using this logic, we define a minimum number of records in $x_i$ before the recursion chain is terminated (as explained in Section 2.3). For our experiments we set the minimum size of $x_i$ to 100, thus making the upper limit of the sensitivity of $M_2$ equal to $\Delta(G(x_i)) = 0.0196$.

### 6.2.2 Pruning the Tree

Aside from calculating $|x_i|$, there is another advantage to using query $M_1(x_i)$ at every $x_i$ node, and not just in the leaf nodes where we need the class value distribution for predicting $r_C; \forall r \in w$ (where $w$ is the testing data). By knowing the class distribution in every $x_i$ node in every $root \rightarrow leaf$ recursion chain, it allows us to compare the leaf nodes to their parent nodes (the node above them in the chain) and assess their quality. By “quality”, we mean “ability to correctly classify records $r$ in $w$”. If a parent node has higher quality than it’s average child node (assuming that all the child nodes are leaf nodes), we perform what is known as “pruning”.

Pruning is a component of many decision tree (and forest) algorithms, where leaf nodes are removed if they do not increase the prediction capabilities of the tree [Breiman et al., 1984, Quinlan, 1993]. The longer the $root \rightarrow leaf$ chains are, the more complicated they are, and the more they divide the records into smaller subsets, potentially over-fitting the tree to the training data $x$ at the expense of prediction accuracy on the testing data $w$. If a leaf node is not actively helping the tree, it is more beneficial to remove the leaf. Some pruning techniques achieve pruning by using a validation set $w' : w' \cap w = \emptyset$ and $w' \cap x = \emptyset$, such as CART’s minimal cost complexity pruning [Breiman et al., 1984] and reduced error pruning [Quinlan, 1993]. However these techniques reduce the size of the training dataset $x$, which would increase the amount of relative noise added by $M_1$ and $M_2$. Since this is something we want to avoid, we instead perform pruning using the information we have already gathered during the tree-building process, specifically with $M_1$.

Since all values $v \in A$ need to be handled, instead of removing individual bad leaf nodes, we measure the average quality of all leaf nodes $\forall v \in A$ and compare that to their parent
node (which they all have in common). If
\[
G(x_i) \geq \sum_{v \in A} \frac{|x^v_i|}{|x_i|} G(x^v_i)
\] (6.24)

where \(x^v_i \forall v \in A\) are leaf nodes, we remove all of \(x_i\)’s leaf nodes, causing \(x_i\) to become a leaf node instead. In the above equation, \(A\) is the attribute used to split \(x_i\). Also recall that the only information required to calculate the Gini index \(G(x_i)\) is \(M_1(x_i)\).

The reason that Equation 6.24 can possibly be true is that our tree building algorithm always splits a node \(x_i\) until a termination condition is met, even if all possible attributes to choose from have lower Gini index results than \(x_i\). Even if some attributes are better, the noisy output of \(M_2(x_i)\) might cause a poor attribute to be chosen. By allowing these situations to potentially occur, we help DPDF avoid getting stuck in a local optima – even if a child node \(Y\) has worse Gini index than it’s parent node \(X\), the child node \(Z\) of \(Y\) could still beat \(X\)’s Gini index! Including this pruning step in our algorithm then checks if either of these situations occurs, and retracts the tree to the global optima within the space explored by \(T\).

6.2.3 Outputting the Rules and Subrules

Not only does possessing the output of \(M_1(x_i); \forall i\) allow us to perform pruning, but it also allows us to have many more rules than would be possible if we only had the class distribution of the leaf nodes. By a “rule”, we mean the attributes chosen along a \(root \rightarrow leaf\) chain, leading to the prediction of a certain class value \(c\) with a certain confidence. Depending on what sort of knowledge the user is searching for, these rules can be extremely valuable, allowing the user to see humanly-readable patterns in the data. The alternative is that tree \(T\) (or forest \(F\)) merely acts as a “black box” classifier, where records \(r \in w\) are inputted and a predicted class value is outputted, with no information on why.

Not only does DPDF output the \(root \rightarrow leaf\) rules, but also all \(root \rightarrow node\) subsets of the \(root \rightarrow leaf\) rules. This is only possible because we know what the predicted class value is for every node \(x_i\), and the confidence of that prediction. Given that the user has a very strict privacy budget \(\beta\) with which to learn about dataset \(x\), the more information we can gain from our queries – and the more we can recycle that information for multiple purposes – the better.
Algorithm 6.1 The proposed Differentially Private Decision Forest (DPDF)

1: procedure DPDF\( (x, C, A, \epsilon, \delta, \tau) \)
2: \( \epsilon = \frac{d}{1 - d^2} \)
3: \( A_{\text{root}} = \{ \} \quad \triangleright \ A_{\text{root}} \text{ will be used as a global variable} \)
4: \( F = \{ \} \)
5: for \( t = 1, \ldots, \tau \) do
6: \( T = \text{BuildTree}\( (x, C, A, \epsilon, \delta, 1, \text{True}) \) \quad \triangleright \text{The forest is composed of trees} \)
7: \( T = \text{PruneTree}\( T \) \)
8: \( F = F \cup T \)
9: end for
10: return \( F \) \quad \triangleright \text{The forest is composed of trees} \)
11: end procedure

12: procedure BuildTree\( (x, C, A, \epsilon, \delta, d, \text{root}) \)
13: \( T = \{ \} \) \quad \triangleright \text{The start of a tree, or a subtree} \)
14: \( H = \{ |x|^\epsilon + \text{Lap}\left( \frac{1}{2} \right); \forall c \in C \} \quad \triangleright \text{i.e. } M_1(x), \text{the noisy counts of each class value in } x \)
15: \( |x| = \sum_{i \in H} |x|^\epsilon \)
16: if \( d \leq \delta \) and \( |x| \geq 100 \quad \text{and} \quad |x|^\epsilon < 1; \forall |x|^\epsilon \in H \text{ and } |A| > 0 \) then
17: \( \Delta(G(x)) = 1 - (\frac{|x|^\epsilon}{|x|^\epsilon + 1})^2 \quad \triangleright \text{The worst-case local sensitivity} \)
18: \( A = \text{SplittingAttribute}\( x, C, A, \epsilon, \Delta(G(x)), \text{root} \) \)
19: \( A = A - \{ A \} \quad \triangleright \text{We cannot split on an attribute twice in a } \text{root} \rightarrow \text{leaf} \text{ chain} \)
20: for all \( v \in A \text{ do} \quad \triangleright \text{Recall that } A \text{ is public knowledge} \)
21: \( x^\epsilon = \{ r : r_A = v, \forall r \in x \} \quad \triangleright \text{Note that } x^\epsilon \text{ must remain on the server to preserve privacy} \)
22: \( T = T \cup \text{BuildTree}\( x^\epsilon, C, A, \epsilon, \delta, d + 1, \text{False} \) \quad \triangleright \text{Attach a subtree to the tree} \)
23: end for
24: end if
25: return \( \{ T, H \} \quad \triangleright \text{The tree is composed of nodes, each composed of a subtree and a class histogram} \)
26: end procedure

27: procedure SplittingAttribute\( (x, C, A, \epsilon, \Delta(G(x)), \text{root} \) \)
28: if \( \text{root} \text{ then} \)
29: \( A = A - A_{\text{root}} \quad \triangleright \text{Two trees cannot have the same root attribute} \)
30: end if
31: \( A = \text{Using the Exponential mechanism, return } A \in A : P_T( \arg \max_{A \in A}(G(x, A)) ) \propto \exp(\frac{\epsilon x}{\Delta(G(x, A))}) \quad \triangleright \text{i.e. } M_2(x) \)
32: if \( \text{root} \text{ then} \)
33: \( A_{\text{root}} = A_{\text{root}} \cup \{ A \} \quad \triangleright \text{Recall that } A_{\text{root}} \text{ is global} \)
34: end if
35: return \( A \)
36: end procedure

37: procedure PruneTree\( (T) \)
38: for all Parent Nodes \( P \in T \text{ do} \quad \triangleright \text{A Parent Node is any node with Child Nodes} \)
39: \( \text{Child Nodes } Q^P = \{ \text{All Child Nodes of } P \} \)
40: if All Nodes \( i \in Q^P \text{ are Leaf Nodes} \) then \( \triangleright \text{A Leaf Node is a Node with 0 Child Nodes} \)
41: \( |Q^P| = \text{The sum of all class counts in } Q^P \) \quad \triangleright \text{Each Node in } H^P \text{ has a histogram} \)
42: \( |P| = \sum_{i \in Q^P} |Q^P| \)
43: if \( \sum_{i \in Q^P} \frac{|Q^P|}{|P|} \times G(Q^P) < G(P) \) then \( \triangleright \text{We only need } H \text{ to calculate the Gini index} \)
44: \( \text{Remove } Q^P \text{ from } P \quad \triangleright \text{ } P \text{ is now a leaf node} \)
45: \( \text{PruneTree}(T \text{ with updated } P) \)
46: end if
47: end if
48: end for
49: return \( T \)
50: end procedure
6.3 Related Work

The most closely-related differentially private decision tree algorithm to our proposal is FS by Friedman and Schuster [2010], discussed in detail in Chapter 3.

FS uses a similar algorithm to our method, with some very important exceptions:

- Instead of getting the distribution of class values in each $x_i$, $M_1$ just returns a noisy count of $|x_i|$.
  - This places limitations on FS’s ability to prune the tree [Friedman and Schuster, 2010].
  - It also does not allow for the first termination condition we use at each $x_i$ subset (i.e. node): “All records in $x_i$ have the same class value $c$”, listed in Section 2.3. This increases the computation time of the algorithm by a small amount, both because it causes the algorithm to spend time making redundant nodes, and also because it increases the amount of pruning that needs to be done.

- They provide an extension to their algorithm to handle continuous attributes, however only at heavy cost to the budget $\beta$. Their results suggest a more feasible solution will need to be developed to handle continuous attributes with realistic $\beta$ values.

- They do not discuss extending their algorithm beyond a single tree, while our algorithm builds $\tau$ distinct trees.

- They divide the privacy budget less efficiently, with $\epsilon = \frac{\beta}{2\delta}$, which makes a non-trivial difference at common values of $\delta$.

- Most importantly of all, they use the global sensitivity of the Gini index ($\Delta(G) = 0.5$), adding a huge amount of unnecessary noise to their tree and reducing the prediction accuracy of the tree, as seen in Section 6.4.

We demonstrate in Section 6.4 that our improvements over their algorithm have a substantial positive effect on the prediction accuracy of the classifier.

6.4 Experiments and Results

Using six datasets from the UCI Machine Learning Repository [Bache and Lichman, 2013], we compare the prediction accuracy of our proposed algorithm, DPDF, to FS
Table 6.1: The size of $\epsilon$ per query, depending on the technique used and the total privacy budget $\beta$. In the example shown, the depth of the trees is $\delta = 5$.

<table>
<thead>
<tr>
<th>Privacy Budget $\beta$</th>
<th>0.1</th>
<th>0.25</th>
<th>0.5</th>
<th>1.0</th>
<th>2.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>FS</td>
<td>0.010</td>
<td>0.025</td>
<td>0.050</td>
<td>0.100</td>
<td>0.200</td>
</tr>
<tr>
<td>DPDF, $\tau = 1$</td>
<td>0.011</td>
<td>0.028</td>
<td>0.056</td>
<td>0.111</td>
<td>0.222</td>
</tr>
<tr>
<td>DPDF, $\tau = 4$</td>
<td>0.003</td>
<td>0.007</td>
<td>0.014</td>
<td>0.028</td>
<td>0.056</td>
</tr>
</tbody>
</table>

[Friedman and Schuster, 2010]. We implement the main version of FS [Friedman and Schuster, 2010] (that uses the Exponential mechanism), and avoid datasets with continuous attributes so that DiffID3 is not disadvantaged by its expensive usage of the privacy budget $\beta$ for continuous attributes. As a benchmark to demonstrate the potential the chosen datasets have for classification, we provide the prediction accuracy results of the popular Random Forest algorithm developed by Breiman [2001a]. We build the Random Forest using all of the default parameter settings in sci-kit learn 0.15.2 [Pedregosa et al., 2011]. All reported prediction accuracies for all algorithms are the average prediction accuracy results of performing 10 iterations of stratified 10-fold cross-validation. This involves randomly dividing each dataset into 10 equal partitions in a way that keeps the class distribution in each partition as close to the whole dataset as possible. Nine of the partitions are then combined to make $x$ and are used to build the classifier (whether it is DPDF, FS or Random Forest). The final partition is used as the testing data $w$. This is repeated with all 10 combinations of nine partitions, so that each partition is used as $w$ once. Ten iterations of this cross-validation process are performed, with the partitions being randomly generated each time. This means that 100 prediction accuracy results are produced, and the average of these are what we report in all our figures. The standard deviation of these results for our proposed algorithm is, on average, 2.881 percentage points.

We test two parameter settings for our DPDF: where $\tau = 1$ and where $\tau = 4$. For all experiments with DPDF or FS, $\delta = 5$. We use 5 values of $\beta$ for our experiments: 0.1, 0.25, 0.5, 1.0 and 2.0. Note that no privacy preservation of any kind is applied to Random Forest.

The datasets used are all publicly available and have the following names in the UCI Machine Learning Repository: “Nursery” (Figure 6.1), “Tic-Tac-Toe Endgame” (Figure 6.2), “Connect4” (Figure 6.3), “Car Evaluation” (Figure 6.4), “Chess (King-Rook vs. King)” (Figure 6.5), and “Mushroom” (Figure 6.6). The number of records in each dataset ranges from 958 to 67557; the number of attributes ranges from six to 42; the
number of class values ranges from two to 18. Details of the datasets can be found in Appendix A.

For most datasets, our algorithm halves the difference between FS’s prediction accuracy and Random Forest’s prediction accuracy. The biggest improvement over DiffID3 is seen with the Nursery dataset, where DPDF with $\tau = 1$ beats FS by almost 25 percentage points, and comes within 7 percent percentage points of Random Forest. With the Tic-Tac-Toe and Connect4 datasets, we can see some overlap between FS and DPDF when
Figure 6.3: A comparison of our technique (DPDF) to FS using the Connect4 dataset, with Random Forest included as a benchmark. We test two parameter settings for DPDF: $\tau = 1$ and $\tau = 4$.

Figure 6.4: A comparison of our technique (DPDF) to FS using the Car dataset, with Random Forest included as a benchmark. We test two parameter settings for DPDF: $\tau = 1$ and $\tau = 4$.

$\tau = 1$, however in both cases, DPDF always beats FS when $\tau = 4$, indicating the benefit of having multiple trees. Interestingly, sometimes $\tau = 1$ beats $\tau = 4$; this may be due to the lower $\epsilon$ value available to each query $M$ when $\tau = 4$, leading to more noisy outputs. In Table 6.1, we demonstrate the difference between the size of $\epsilon$ for FS and DPDF when $\tau = 1$ and $\tau = 4$, for each of the five $\beta$ values we are testing.

As expected, the prediction accuracy of FS and DPDF (for both $\tau = 1$ and $\tau = 4$) generally increases as the budget $\beta$ increases. In a few situations this does not happen,
Figure 6.5: A comparison of our technique (DPDF) to FS using the Chess dataset, with Random Forest included as a benchmark. We test two parameter settings for DPDF: $\tau = 1$ and $\tau = 4$. Note that there are 18 class values, so randomly guessing would give a prediction accuracy of 5.55%; hence the low prediction accuracy.

Figure 6.6: A comparison of our technique (DPDF) to FS using the Mushroom dataset, with Random Forest included as a benchmark. We test two parameter settings for DPDF: $\tau = 1$ and $\tau = 4$.

notably for the Connect4 dataset. However in this dataset, Random Forest produces a worse prediction accuracy than all the differentially private algorithms at all $\beta$ values, suggesting that the noisy outputs of FS and DPDF helped the tree-building process. This may be due to the trees in Random Forest getting stuck in local optima, or simply that decision trees are not a good choice for data mining the Connect4 dataset.

Overall, it appears that for most datasets (except Tic-Tac-Toe, perhaps because it is by
far the smallest dataset and thus has the noisiest outputs), DPDF produces a decision forest of acceptable quality for most data mining needs. This is true even when $\beta$ is very low; as low as $\beta = 0.1$, where each query $M$ has $\epsilon = 0.003$ when $\tau = 4$. The user also has the complete list of rules and sub-rules from $F$, including the class distribution of each rule. This means the user can easily remove any rules or sub-rules with low confidence, leaving them with a shorter list of high quality rules.

### 6.5 Summary

The success of DPDF at even very low $\beta$ values provides the user with some valuable options – instead of using their entire allocated privacy budget $\beta$ on a single run of DPDF, they can instead use only a fraction of it. The rest of the budget could be spent on anything the user wishes. This might include some preliminary queries on $x$ in order to tune the $\delta$ and $\tau$ parameters, which would be an interesting direction for future research. It could include multiple runs of DPDF, each with different parameters. It could include completely different data mining algorithms, such as clustering (as long as it is differentially private). The strong mathematical properties of differential privacy allow the data owner to guarantee the participants in the dataset that their presence in the dataset is almost completely undetectable, no matter how a user decides to divide their $\beta$ budget. Our novel theorem on the local sensitivity of the Gini index will be useful to any future classification algorithms that wish to perform differentially private data mining. We provide the code required to implement DPDF online at http://samfletcher.work/code/ and http://csusap.csu.edu.au/~zislam/, so that data scientists and fellow researchers may take advantage of it.

As a continuation of Table 3.1 (summarizing the properties of different differentially-private decision tree algorithms), Table 6.2 summarizes the algorithm we presented in this chapter. A combined table that includes all of our algorithms is presented and discussed in Chapter 9.

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Budget per Query</th>
<th>Tree Type</th>
<th>Tree Depth $d$</th>
<th>Forest Size $\tau$</th>
<th>Handles Cont. Attributes</th>
<th>Prediction Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapter 6</td>
<td>$\frac{\beta}{\tau(2\tau-1)}$</td>
<td>Greedy</td>
<td>5</td>
<td>4</td>
<td>No</td>
<td>Low</td>
</tr>
</tbody>
</table>
Chapter 7

Differentially-Private Random Trees

As the title implies, this chapter differs from the previous chapter in that instead of building greedy trees, we build random trees. This has the effect of reducing the number of queries we need to submit to the dataset, and in turn increasing prediction accuracy. Asking less queries also means that the amount of knowledge we can discover (such as useful patterns) is reduced. Whether this trade-off is worthwhile will depend on the needs of the user.

Using differential privacy, we propose a data mining algorithm that builds an ensemble of randomized decision trees (i.e a forest), queries the data in a differentially private manner, and outputs a classifier capable of high accuracy even with very high privacy requirements. We approach the problem by phrasing it in terms of the Signal-to-Noise Ratio of our queries, and using signal averaging to reduce the noise in the queries.

7.1 Our Contributions

Our novel contributions can be summarized as the following:

The work in this chapter has been published in the following paper: Sam Fletcher and Md Zahidul Islam. A Differentially-Private Random Decision Forest using Reliable Signal-to-Noise Ratios. In 28th Australasian Joint Conference on Artificial Intelligence, Canberra, Australia, 2015, Lecture Notes in Computer Science (LNCS), 9457:192–203
Chapter 7. Differentially-Private Random Trees

• We re-phrase the problem of making a differentially private data mining algorithm in a novel way, by using Signal-to-Noise Ratio theory to assess the noise added to differentially private queries (Section 7.3).

• We present a differentially-private randomized decision forest algorithm (referred to as DP-RF) in which the structure of the decision trees is decided before querying the dataset $x$ at all (Section 7.4).

• Our algorithm automatically tunes all parameters, with the only inputs being: access to the secure dataset $x$, the domains of the attributes $\mathcal{A}$, and the privacy budget $\beta$ (Section 7.4.3, Section 7.4.4 and Section 7.4.5).

• We take full advantage of the benefits of randomly built decision trees, while identifying the assumptions usually made about decision trees that no longer hold, and providing solutions (Section 7.4.6).

We also provide code for our algorithm online at http://samfletcher.work/code/ and http://csusap.csu.edu.au/~zislam/.

7.2 Previous Work

Several differentially-private decision tree algorithms have been proposed in recent years [Friedman and Schuster, 2010, Jagannathan et al., 2012, Rana et al., 2016]. Of these, one of them took a similar approach to our work in this chapter and in Chapter 8, and used random decision trees to construct a decision forest [Jagannathan et al., 2012]. The others have all been greedy approaches, using a splitting criteria in each node. Using a splitting criteria comes with a disadvantage: the user must query the data to do so. This is an expenditure of the privacy budget that random decision trees avoid entirely. Random decision trees have been shown to be competitive against greedy decision trees even in non-private scenarios [Fan et al., 2003, Geurts et al., 2006], and especially appropriate in our private scenario. Empirically, this intuition has been validated [Jagannathan et al., 2012], with better prediction accuracy being achieved over their greedy counterparts [Friedman and Schuster, 2010, Rana et al., 2016].

The work most closely related to ours is the differentially private random decision forest proposed by Jagannathan et al. [2012] in 2012. The main differences between our proposed algorithm and theirs stem from our re-framing of the scenario in terms of Signal-to-Noise Ratios. Our algorithm automatically tunes the required parameters, while the
work in Jagannathan et al. [2012] requires the user to manually set parameters, or otherwise uses heuristics. Their heuristics are based on the combinatorial reasoning used in Fan et al. [2003]’s non-private random decision tree algorithm.

7.3 Signal-to-Noise Ratio

Signal-to-Noise Ratio is a comparison between some sort of measurement (signal) and the background noise accompanying that measurement [Van Drongelen, 2006]. The ratio of the “proper” or “real” signal to the noise is the Signal-to-Noise Ratio, or SNR. The mathematics underpinning the SNR concept is applicable in any scenario where the following assumptions are met: the signal and the noise are uncorrelated; the signal would be the same if it was measured again (ignoring the noise); and the noise is random, with a mean of 0 and a constant variance for repeated measurements of the signal. Differential privacy meets these assumptions: noise added with the Laplace Mechanism is only dependent on $\epsilon$ and $\Delta$, neither of which are affected by the signal; the number of records meeting a query’s criteria will not change if the same dataset is queried again; and the Laplace distribution has a mean of 0 and a known, constant variance of $2(\Delta/\epsilon)^2$.

Expressing the SNR mathematically is quite simple:

**Definition 7.1** (Signal-to-Noise Ratio). The Signal-to-Noise Ratio of a measurement can be expressed as $rac{\text{signal}}{\text{noise}}$, where the signal and noise are expressed in the same units. An alternate way of writing this is:

$$SNR = \frac{\mu}{\sigma}$$

(7.1)

where $\mu$ is the signal mean or expected value and $\sigma$ is the standard deviation of the noise.

We take advantage of Definition 7.1 in Section 7.4.

7.3.1 Signal Averaging

Once a problem has been phrased in terms of the Signal-to-Noise ratio, there is additional property we are able to take advantage of: signal averaging. Signal averaging provides an explicit definition of the intuition that if noise can increase or decrease a signal with equal probability, then summing multiple signals in a set $S$ together will result in a total that is less noisy:

$$SNR = \frac{\mu}{\sigma} = \frac{\sum_{x \in S} \mu_x}{\sqrt{|S|\sigma^2}}.$$  

(7.2)
7.4 Our Differentially Private Random Decision Forest

It has been demonstrated in the past that randomized decision trees can have surprisingly high performance, and often have much lower computational complexity compared to their less random counterparts [Fan et al., 2003, Geurts et al., 2006]. They gain their computational efficiency from the fact that they do not need to use the training data \( x \) in order to build a tree. This is a valuable advantage when trying to achieve differential privacy, as the less times we need to query the training data the better. The overall design philosophy of a differentially private machine learning algorithm is to be as efficient at spending the privacy budget \( \beta \) as possible. Below, we present our novel algorithm for building a differentially private random forest (DP-RF), and achieve better accuracy than any differentially private decision tree/forest algorithm proposed before it, even with very low privacy budgets.

7.4.1 Overview of Our Algorithm, DP-RF

Our algorithm, described in detail in the sections below, can be summarized as the following steps. Algorithm 7.1 presents our proposal as pseudocode.

1. Based off the size of the privacy budget \( \beta \), the size of the dataset \( x \) and the domain sizes of the attributes \( \mathcal{A} \), automatically tune the following parameters:
   - \( \tau \), the number of trees in the decision forest, which in turn dictates the \( \epsilon \) spent per tree (Section 7.4.4, and Line 3 of Algorithm 7.1).
   - \( \theta \), the minimum support threshold for nodes in the trees (Section 7.4.3, and Line 5 of Algorithm 7.1).

2. Build a randomized decision forest using \( \tau \) and \( \theta \), and no user-inputted parameters (Section 7.4.3, and Lines 15–25 of Algorithm 7.1).

3. Query the dataset using the Laplace Mechanism to learn the class counts in each leaf (Section 7.4.2, and Line 32 of Algorithm 7.1).

4. Prune away nodes with SNR \(< 1\), using signal averaging for non-leaf nodes (Section 7.4.5, and Line 40 of Algorithm 7.1).

5. Find the node with the highest confidence in each path from the root node to a leaf node, in each tree (Section 7.4.6, and Line 51 of Algorithm 7.1).

6. Predict the class value of future records by voting on the most confident predictions made by each tree (Section 7.4.7).
Algorithm 7.1 The proposed Differentially Private Random Decision Forest.

1: procedure BuildForest(Privacy budget \( \beta \), dataset \( x \), number of trees \( \tau \), set of attributes \( A \), class \( C \))
2: \( F \leftarrow \{\} \)
3: \( \tau \leftarrow \text{The number of trees to build, according to Section 7.4.4.} \)
4: \( \epsilon \leftarrow \beta / \tau \)
5: \( \theta \leftarrow |C|\sqrt{2}/\epsilon \), defined in Equation 7.3.
6: for \( t = 1, \ldots, \tau \) do
7: \( T \leftarrow \text{BuildTree}(\theta, |x|, A) \)
8: \( F \leftarrow F \cup T \)
9: end for
10: \( F \leftarrow \text{CountLabels}(\epsilon, x, C, F) \)
11: \( F \leftarrow \text{Prune}(\epsilon, C, F) \)
12: \( F \leftarrow \text{SelectConfident}(F) \)
13: return \( F \)
14: end procedure

15: procedure BuildTree(Minimum support threshold \( \theta \), estimated support \( n \), attributes \( A \))
16: \( T \leftarrow \{\} \)
17: if \( |A| > 0 \) and \( n > \theta \) then \( \triangleright \text{Termination criteria.} \)
18: Uniformly randomly select an attribute \( A \) from \( A \) to split the current node.
19: \( A \leftarrow A \setminus \{A\} \) \( \triangleright \text{Attributes are only chosen once in a root-leaf path.} \)
20: \( n \leftarrow n/|A| \) \( \triangleright \text{Update the estimated support of the child nodes.} \)
21: for all \( v \in A \) do
22: \( T \leftarrow T \cup \text{BuildTree}(\theta, n, A) \) \( \triangleright \text{All child nodes.} \)
23: end for
24: end if
25: return \( T \)
26: end procedure

27: procedure CountLabels(Budget \( \epsilon \), dataset \( x \), class label \( C \), forest \( F \))
28: for all Trees \( T \) in forest \( F \) do
29: Filter all records in \( x \) into tree \( T \), until each record belongs to one leaf node.
30: for all Leaf nodes \( L \) in tree \( T \) do
31: Count frequency of each class label \( c \in C \) in leaf node \( L \).
32: Add \( \text{Lap}(1/\epsilon) \) to each class count.
33: end for
34: end for
35: return \( F \)
36: end procedure

37: procedure Prune(Budget \( \epsilon \), class label \( C \), forest \( F \))
38: for Trees \( T \) in forest \( F \) do
39: for all Nodes \( N \) in tree \( T \) do
40: if \( \text{SNR} < 1 \), as defined by Equation 7.8 then
41: Remove \( N \) (and any child nodes) from \( T \).
42: end if
43: end for
44: end for
45: return \( F \)
46: end procedure

37: procedure SelectConfident( Forest \( F \))
38: for Trees \( T \) in forest \( F \) do
39: for all Leaf nodes \( L \) in tree \( T \) do
40: \( P \leftarrow L + \text{Ancestor nodes of} \ L \) \( \triangleright \text{Parent node, Grandparent node, etc.} \)
41: Mark the node in \( P \) with the highest confidence as the node to use when predicting the label of future records that obey the filters of \( P \).
42: end for
43: end for
44: return \( F \)
45: end procedure
7.4.2 Querying the Leafs of a Tree

By following any chain of directed edges from a decision tree’s root to a leaf, we have what is called a “decision rule”. Each decision rule is a collection of attribute values, defined by the attributes from $A$ that each node splits on, with each directed edge having one value $v$ from the attribute $A$ in the node it came from. Every record $r$ in the training data $x$ possesses a value $v; \forall A \in A$ that make it match one and only one decision rule in the decision tree. We therefore say that each record “fits into” or “belongs to” the leaf at the end of the chain of nodes its values match.

Parallel composition (see Definition 2.3) means that for a decision tree – where every record appears in one and only one leaf – we can perform an $\epsilon$-differentially private query $f$ on every leaf and use a total of $\epsilon$ out of the privacy budget $\beta$ [Jagannathan et al., 2012].

This is precisely what we will be doing: performing a single query $f$ on each leaf of a decision tree. That single query will be, “How many records have each class value?”. In other words, we will get a histogram of the class counts in each leaf. The sensitivity $\Delta$ of this query $f$ is $\Delta(f) = 1$, because the bins in a histogram are disjoint – the removal or addition of a single record can affect at most one bin of the histogram. Using the Laplace mechanism (Definition 2.4) we can achieve $\epsilon$-differential privacy by adding $\text{Lap}(1/\epsilon)$ to each class count, in each leaf of a decision tree.

7.4.3 Building a Random Tree

The building of a random decision tree is straightforward, and does not require the training data $x$ at all. The process is explained in detail in Section 2.3.2, but to summarize: an attribute $A$ is chosen as the root node of the tree, with a directed edge (i.e. a branch of the tree) being created for each value $v \in A$. This process is recursively performed for the nodes at the end of each directed edge. The chosen attribute $A$ at each node is selected from the set of attributes not chosen previously in the recursion chain, so that for any chain of nodes (i.e. path) from the root node to the bottom of the tree, no attribute appears twice.

The recursion ends for a given chain in the tree when either of the following termination criteria are met: (1.) there are no attributes left that have not been used previously in the recursion chain; or (2.) The estimated support of the current node is so low that the estimated SNR is below 1 (i.e the noise outweighs the signal).

Criteria 2 uses the Signal-to-Noise Ratio, described in Section 7.3, and refers to the “estimated support” of the node. We define “estimated support” as the number of
records that are estimated to match the attribute values \( v \in A \); \( \forall A \) on the directed edges that led to the current node. The number of records is predicted by using the assumption that the subset of records \( x_i \) in a given node will be divided equally amongst all the directed edges leaving that node. In other words, starting from the root node and working our way down the tree, we make a rough estimate of the support of each node by dividing it by the domain size of each attribute used earlier in the chain.

This has the effect of not restricting the entire tree to the same maximum depth, unlike the differentially-private random decision tree algorithm proposed by Jagannathan et al. [2012]. Assuming that each value of an attribute has an equal portion of the records in \( x \) is clearly not an accurate assumption, but it does not need to be for our purposes. While rough, we mostly need the order of magnitude of a node’s support, and we apply pruning later (see Section 7.4.5) to clean up the rough estimates. By estimating the support of nodes without using the training data, we avoid having to spend any of the privacy budget \( \beta \) on queries.

In the case of differential privacy, there is an additional reason to define a minimum support threshold: the larger the support, the less likely it is that noise will disrupt the signal. If we apply a query \( f \) to a leaf with very small support, the noise could easily completely overwhelm the signal.

In order for the SNR to be above 1, the estimated support of a node must be larger than the minimum support threshold. We define the minimum support threshold as:

\[
\theta = |C| \times \sqrt{2} \times \frac{1}{\epsilon}.
\]  

(7.3)

This is derived from the Laplace Distribution’s variance, \( \sigma^2 = 2(\Delta/\epsilon)^2 \) (see Definition 2.4), and the Signal-to-Noise Ratio, \( \mu/\sigma \). Because the sensitivity of the counting query is 1, we can substitute that in for \( \Delta \), and by taking the square root of the variance we are left with the standard deviation, as seen in the formula for the SNR (Definition 7.1). Combining these observations leaves us with:

\[
\frac{\mu}{\sigma} = \frac{\mu}{\sqrt{2} \times (1/\epsilon)^2} = \frac{\epsilon \mu}{\sqrt{2}}.
\]  

(7.4)

A simple rearrangement lets us see that the signal \( \mu \) (i.e. the support of a node) must be larger than \( \mu > \sqrt{2}/\epsilon \) in order for \( \mu/\sigma > 1 \). Note that this assumes that \( \text{Lap}(1/\epsilon) \) is only being added to the support of a node once. However, it as actually being added to each node a number of times equal to the size of the class attribute: \( |C| \). We therefore
need to increase the noise component of the SNR accordingly:

\[ \mu > \frac{|C|\sqrt{2}}{\epsilon} \]  

(7.5)

and thus we arrive at (Equation 7.3): the estimated support \( \mu \) must be larger than \( \theta \).

After recursively building upon each node until the estimated support of each node is less than the minimum support threshold \( \theta \), we can then learn how many records in the training data \( x \) fit into each leaf. Learning about the records in each leaf requires querying the dataset, and spending \( \epsilon \) amount of the privacy budget \( \beta \). This process is described in full in Section 7.4.2.

Thus we have built a random decision tree. We then repeat this tree-building process a number of times equal to \( \tau \) to end up with a forest of random decision trees. The parameter \( \tau \) is automatically defined by our algorithm, and is described below in Section 7.4.4. \( \tau \) defines the fraction of the privacy budget that each decision tree can spend. When differentially-privately querying the leafs of one decision tree, \( \epsilon \) is defined as:

\[ \epsilon = \frac{\beta}{\tau} \]  

(7.6)

### 7.4.4 Defining the Number of Trees

To promote as diverse a collection of random trees as possible, we can make the root node of each tree unique by making sure a different attribute is chosen each time. Diversity is well known to be an advantageous property of decision forests [Breiman, 2001a] – hence why randomly built trees are valuable even when there are no privacy restrictions involved [Fan et al., 2003, Geurts et al., 2006]. Extending this idea, we can define the number of trees as being equal to the number of attributes – thus the dataset will be partitioned based off the values of every attribute, giving us a chance to learn how predictive each attribute can be on their own. Any more trees and we would start having redundant root nodes, and thus be getting less value for the privacy budget spent on it than we did for the earlier trees.

Limiting the number of trees is only necessary because we have a privacy budget we must adhere to. As explained in Section 7.4.2, we can query the dataset about the (disjoint) leafs in a tree using the same portion of the privacy budget. However we cannot use the same portion of the privacy budget for multiple trees – each tree is a re-partitioning of the same data, and so they are not disjoint. This means that the more trees we have, the more we have to divide the privacy budget among them due to Definition 2.2 (see (Equation 7.6)).
It is therefore possible that if there is a large number of attributes, the budget may be spread too thin to output useful query results. In our case, “too thin” can be interpreted as “the minimum support threshold is too high to build a tree”. We address this by reducing the number of trees to a point where the minimum support threshold becomes acceptable. We do so by imagining a scenario in which the dataset $x$ is split into a number of partitions $x_i$, equal to the average domain size of the attributes $\mathcal{A}$—we now have the “stump” of a decision tree. We then repeat this, dividing each node on the next level of the tree by the average domain size of the attributes. We now have a tree with a depth of 3: the dataset is divided up twice by attributes (with average domain sizes), with the last level of the tree being leafs. We consider this to be the minimally acceptable tree size. For this minimally acceptable tree size to be possible (at least in the average case), the minimum support threshold must be as follows:

$$\theta < \frac{\mid D \mid}{b^2},$$

(7.7)

where $\theta$ is defined by (Equation 7.3) and $b$ is the average “branching factor” of the attributes. The branching factor of an attribute refers to how many child nodes will be created if a node is split using that attribute. This is the same as the domain size of the attribute if it uses discrete values, and is equal to $b = 2$ if the attribute is continuous. Recall that there is no maximum depth defined in our algorithm: some parts of the decision tree might have a depth shorter than 3, while other parts of the same tree might have a much larger depth. Thus we define $\tau$ as the largest number, up to the number of attributes $\mid \mathcal{A} \mid$, that satisfies (Equation 7.7) and (Equation 7.3).

### 7.4.5 Pruning to a Reliable Depth

Once the forest of random decision trees has been built, the next step in our algorithm is to prune each decision tree. This is somewhat similar to the pruning done in other decision tree algorithms [Han et al., 2006], including the pruning done in JPW [Jagannathan et al., 2012]. The only pruning done in JPW is to remove leafs with no records in them. We propose a more sophisticated approach, but first a remark on their approach. It is important to note that even if there are 0 records in a leaf, Laplace noise still needs to be added to each class count. While of course a count cannot go below 0, a count that was originally 0 can be raised above 0 by the noise. This drastically reduces the amount of pruning that is done in JPW, and keeps leafs that most likely have a Signal-to-Noise Ratio of 0 (i.e. they are 100% noise). Our proposed approach solves this problem.

Earlier, in Section 7.4.3, we estimated the support of each node in order to estimate its Signal-to-Noise Ratio and decide whether to branch from the node further. Now that we
have queried the server and have the (noisy) class counts of each leaf, we no longer have to use such rough estimates for each node’s support – we can re-check which nodes have acceptable SNRs, and which do not. We can also take advantage of signal averaging, described in Section 7.3.1. The idea is simple: by summing together the class counts of all the leafs that have the same parent node, we know the class counts of the parent node. This can be repeated up the tree until we know the class counts of every node in the tree. Not only that, but the class counts of the parent node are actually less noisy than those of the child nodes. We can see this in (Equation 7.2), as well as the intuition behind it in Section 7.3.1. The noise is reduced by a factor equal to the square root of the number of leaf nodes being summed together.

Using this knowledge, we now not only have more accurate SNRs for the leafs than we did in Section 7.4.3, but we also have increasingly more accurate SNRs for the nodes above the leafs, the further we traverse up the tree. Our pruning is simple: we remove any nodes with SNR < 1. Because of the reduction in noise in nodes using the sum of class counts in its children, it becomes easier and easier for nodes higher up the tree to have SNR > 1. When re-written for this scenario, (Equation 7.2) becomes:

$$\text{SNR} = \frac{\epsilon \sum_{L}^{\text{Leafs}} \left( \sum_{c}^{C} L_c \right)}{|C| \sqrt{2 \times |\text{Leafs}|}},$$

(7.8)

where Leaf is the set of all leaf nodes that can be reached by traversing down from the current node (not all leafs in the entire tree), and $L_c$ is the class count for $c \in C$ in leaf $L$. Observe that (Equation 7.8) reduces down to (Equation 7.4) when considering one leaf and one class value – they both originate from the SNR theory.

### 7.4.6 Finding the Most Confident Prediction in Each Rule

Once unreliable leafs have been pruned away, we are left with only nodes where the true counts of the class values outweigh the noise. Our aim now is to use these nodes to predict the class values of future records. For a given future record $r$ and a given decision tree, we can find the leaf that $r$ belongs to, and know the decision rule that it obeys.

Usually in non-randomly built decision trees, a node is only split into more nodes if an attribute can be found that increases the confidence of the decision rule’s prediction of $r$’s class value [Breiman et al., 1984]. If we define the predicted class value of $r$ as the majority class value in the leaf of the decision rule, the “confidence” of the prediction is then the fraction of the records in the leaf that have the majority class value. However
in the case of a randomly built decision tree, there is nothing preventing a node above the leaf from having a higher confidence than the leaf.

We therefore find the most confident prediction in each decision rule, and then use that prediction for any future records that obey that rule.

### 7.4.7 Voting

When predicting the class value of a future record $r$, the process of finding the most confident prediction in the decision rule that $r$ obeys is repeated for each decision tree in the forest. $r$ obeys one decision rule per tree, and our algorithm selects one node out of each decision rule with the highest confidence, as described in Section 7.4.6. It is unlikely that all the predictions are predicting the same class value, so a voting mechanism is required to decide which class value is the decision forest’s final prediction.

Our algorithm votes using the following process: (1.) The final prediction is the class value with the highest confidence; if two different class values are predicted with equal confidence, then (2.) sum the confidences of each class value from the nodes selected in Section 7.4.6 (including the non-majority class values) and (3.) the final prediction is the class value with the highest summed confidence.

### 7.5 Experiments

We test our proposed algorithm (DP-RF) using 9 datasets from the UCI Machine Learning Repository [Bache and Lichman, 2013]. The details of the datasets can be found in Appendix A. We use 10-fold stratified cross-validation repeated 30 times to calculate the average prediction accuracy of our algorithm, using various privacy budgets. We test the following privacy budgets: $\beta = 0.01, 0.05, 0.1, 0.25, 0.5, 1.0, 2.0$. We compare our technique to JPW, proposed by Jagannathan et al. [2012] (described in Chapter 3). For continuous attributes, we discretize them by splitting their domain into 5 bins of equal range. We use the default parameters recommended in Jagannathan et al. [2012] for JPW. Note that high values of $\beta$ (especially values $\geq 1$) are unlikely in real-world situations, and are presented to demonstrate the trends.

Our results are presented in Figure 7.1, Figure 7.2 and Figure 7.3. For all datasets, our algorithm has higher prediction accuracy than JPW on average. Out of the 63 measurements (7 values for $\beta$ per dataset) our algorithm out-performs JPW in 55 cases, sometimes only under-performing by $< 1\%$ (i.e. 0.01 on the $y$ axis). At its best our
Figure 7.1: Comparing the prediction accuracy of our technique (DP-RF) to JPW [Jagannathan et al., 2012] with different privacy budgets, using three datasets from [Bache and Lichman, 2013].
Figure 7.2: Comparing the prediction accuracy of our technique (DP-RF) to JPW [Jagannathan et al., 2012] with different privacy budgets, using three more datasets from [Bache and Lichman, 2013].
Figure 7.3: Comparing the prediction accuracy of our technique (DP-RF) to JPW [Jagannathan et al., 2012] with different privacy budgets, using three more datasets from [Bache and Lichman, 2013].
Table 7.1: The number of trees built for each dataset used in our experiments.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Number of Attributes</th>
<th>Number of Trees</th>
<th>( \beta = 1.0 )</th>
<th>( \beta = 0.1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adult</td>
<td>14</td>
<td>14</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>Banknotes</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>GammaTele</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>Mushroom</td>
<td>22</td>
<td>22</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>Nursery</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>RedWine</td>
<td>11</td>
<td>7</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Vehicle</td>
<td>18</td>
<td>6</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>WBC</td>
<td>9</td>
<td>9</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Yeast</td>
<td>8</td>
<td>5</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

The algorithm can beat JPW by 45%; at its worst it loses by 7%. The standard deviation of these results for our proposed algorithm is, on average, 1.905 percentage points.

To demonstrate our definition for the number of trees (presented in Section 7.4.4), Table 7.1 shows the number of trees built for each dataset used in our experiments when \( \beta = 1.0 \) and \( \beta = 0.1 \). We can see that for larger privacy budgets, the number of trees can equal the number of attributes, but for smaller budgets this number must be reduced. This can be counteracted by using more data; that is, increasing the numerator in Equation 7.7.

### 7.6 Summary

By re-framing the problem of building a differentially private decision forest in terms of the Signal-to-Noise Ratios, we are able to propose intuitive methods for automating the tuning of necessary parameters. We are also able to guarantee that any predictions made about future records are made using class counts that not only have high confidence, but also outweigh any noise that might have been added to them. Our results prove the success of this approach, and pave the way for extending the application of Signal-to-Noise Ratio theory to other implementations of differential privacy.

As a continuation of Table 3.1 (summarizing the properties of different differentially-private decision tree algorithms), Table 7.2 summarizes the algorithm we presented in
this chapter. A combined table that includes all of our algorithms is presented and discussed in Chapter 9.

Table 7.2: The main properties of the differentially-private decision tree algorithm presented in this chapter.

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Budget per Query</th>
<th>Tree Type</th>
<th>Tree Depth $d$</th>
<th>Forest Size $\tau$</th>
<th>Handles Cont. Attributes</th>
<th>Prediction Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapter 7</td>
<td>$\beta/\tau$</td>
<td>Random</td>
<td>Dynamic</td>
<td>$\leq m$</td>
<td>No</td>
<td>Medium</td>
</tr>
</tbody>
</table>
Chapter 8

Harnessing Smooth Sensitivity

This chapter continues the trend of Chapter 7 by further reducing the possibility of discovering knowledge in $x$, but gains substantially increased model performance in exchange. We do so by proposing another random decision forest algorithm, that this time harnesses the concept of “smooth sensitivity”. Defined formally in Section 2.5.1, smooth sensitivity looks at the actual dataset $x$ to calculate the sensitivity of the queries, rather than just assuming the worst-case scenario.

Out of all the possible classification algorithms [Han et al., 2006] that could be used to learn from data, decision trees are a good choice for minimizing the drawbacks of providing differential privacy. The main factors that dictate the impact of differential privacy on a classifier are:

1. the size of the privacy budget;
2. the number of queries that are required to build the classifier; and
3. the sensitivity of those queries to small changes in the data.

The size of the budget is outside the control of the user; all a classifier can aim to do is produce high-quality predictions with as small a budget as possible. The algorithm we propose in this chapter manages to build a good classifier with an even smaller budget than the algorithms we proposed in Chapter 6 and Chapter 7. Given a reasonably large dataset, we achieve high accuracy with a budget as small as $\epsilon = 0.1$.

Chapter 8. *Harnessing Smooth Sensitivity*

The number of queries that are required to build the classifier dictates how much the budget needs to be divided up. Unless the queries are applied to disjoint subsets of the data, they compose and each costs a portion of the budget (see Section 2.5). Just as Chapter 7’s algorithm used less queries than Chapter 6’s, this chapter’s algorithm uses even less than Chapter 7’s. In fact we use the absolute minimum: one query, repeated across many disjoint subsets of the data, each able to use the entire privacy budget.

Third and finally, the impact of differential privacy is dictated by the sensitivity of the queries. Our work is the first to apply the concept of smooth sensitivity to decision trees. We demonstrate that our single query is often orders of magnitude less sensitive than the queries used by other differentially private decision trees, and equally sensitive in the worst-case scenario.

Additionally, the observations we make and the theorems we prove are not limited to decision trees, but can be used in any differentially-private scenario that aims to output discrete answers to queries about the most or least frequent item in a set.

### 8.1 Our Contribution

In this chapter, we propose a differentially-private decision forest algorithm that makes very efficient use of the privacy budget $\epsilon$ to output a classifier with high prediction accuracy. We achieve this by proposing a query in Section 8.4.1 that outputs the most frequent label in some subset $x_i$ of the data with high probability, and using this query in each leaf node of all the trees in a forest. We prove that this query has low sensitivity, making it reliable even without a large privacy budget. This proof is generalized to the non-binary case, and tested on several datasets that have more than two class labels. It is also generalized to any scenario where a differentially private query aims to output the most (or least) frequent item in a set.

We also extend the work done by Fan et al. [2003], where combinatorial reasoning is used to calculate the optimal depth of the decision trees. This work was only applicable when all the features were discrete; we extend it by proving the optimal tree depth needed for continuous features in Section 8.4.3.

The reliability and accuracy of our differentially-private decision forest is further improved using theoretical and empirical observations in Section 8.4.2 and Section 8.4.4. We provide empirical results throughout the chapter, demonstrating the real-world effect of our theory. The methodology of our experiments is laid out in Section 8.2.
We finish by demonstrating that our algorithm achieves significantly better prediction accuracy than other similar algorithms, both statistically and practically, in Section 8.5. We show that even when competing against a decision forest that uses a weaker definition of differential privacy [Rana et al., 2016], our algorithm performs very well. This disputes any arguments that traditional $(\epsilon, 0)$-differential privacy is too strict to produce high quality results. We contextualize our results with each dataset with the results obtained by Brieman’s Random Forest algorithm. This algorithm is very much non-private; that is, it makes no attempt whatsoever to protect the privacy of the people in the dataset. This of course means that it can produce more accurate models than a private model can ever achieve, and the aim of our work (and others like it) is instead to accomplish the much more difficult task of balancing two fundamentally adversarial concepts – knowledge discovery and individual privacy – to produce a high quality model. We discuss the ramifications of our findings in Section 8.6. A full implementation of our algorithm is available online at http://samfletcher.work/code/ and http://csusap.csu.edu.au/~zislam/.

### 8.2 Preliminaries: Experiment Methodology

We present empirical results throughout the chapter, demonstrating how the theory of our algorithm performs in practice. All our experiments are repeated ten times, with each test using ten-fold cross validation, for a total of 100 results that are then averaged. We include one standard deviation when presenting the average result. The majority of our experiments use a privacy budget of $\epsilon = 1$. We also perform some experiments with $\epsilon = 0.01, 0.1, 0.2$. In real-world scenarios, the privacy budget given to a user very much depends on that specific case; while a value of $\epsilon = 0.01$ is sometimes suggested [Dwork, 2008, Dwork and Roth, 2013], values as high as $\epsilon = 8.6$ have been used in large, public projects [Machanavajjhala et al., 2008]. We also demonstrate in Section 8.5.1 how the larger the size $n$ of a dataset, the smaller $\epsilon$ can be without injecting too much noise into the resulting trees; a phenomena well understood when implementing differential privacy [Dwork, 2008, 2011, Dwork and Roth, 2013].

In our experiments involving real-world data, the data was collected from the UCI Machine Learning Repository [Bache and Lichman, 2013]. Details of these datasets are presented in Table 8.4; descriptions and URLs for each can be found in Appendix A. Some of the real-world datasets we use have more than two class labels; PenWritten has ten class labels, and WallSensor, Nursery and Claves have four. One of the advantages of our findings is that they are generalized to the multi-label (i.e., non-binary) case, and so we include that case in our testing. In experiments involving synthetic
Table 8.1: Parameters for the synthetic datasets we use throughout the chapter.

<table>
<thead>
<tr>
<th>Dataset Name</th>
<th>n_informative</th>
<th>n_random</th>
</tr>
</thead>
<tbody>
<tr>
<td>SynthA</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>SynthB</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>SynthC</td>
<td>15</td>
<td>0</td>
</tr>
<tr>
<td>SynthD</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>SynthE</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>SynthF</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>SynthG</td>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>

data, the data are generated with the sci-kit learn package in Python [Pedregosa et al., 2011]. The parameters that differentiate each synthetic dataset are defined by the sklearn.datasets.make_classification function. We generate \( n = 30,000 \) records for each dataset, with different numbers of continuous features \( m \), and use a balanced binary class feature. The parameters using non-default settings are defined as follows:

**n_informative:** “The number of informative features. Each class is composed of a number of Gaussian clusters each located around the vertices of a hypercube in a subspace of dimension \( n_{\text{informative}} \). For each cluster, informative features are drawn independently from \( N(0,1) \) and then randomly linearly combined [into records] within each cluster in order to add covariance.” [Pedregosa et al., 2011]

**n_random:** The number of useless features, generated randomly.

The above parameters sum to the total number of features \( m \) generated for a dataset. Table 8.1 provides the parameter values we use for the seven synthetic datasets used in our experiments.

In some experiments we compare our differentially private algorithm to “non-private” algorithms. By “non-private” we mean classification algorithms that do not attempt to protect privacy in any way – they simply try to maximize prediction accuracy as much as possible. We present results for two non-private algorithms in this chapter: Breiman’s Random Forest algorithm [Breiman, 2001a], and a version of our proposed algorithm with the privacy protection removed. More specifically, the non-private version of our proposed algorithm is exactly the same as the private version except for the following: it uses a privacy budget of \( \epsilon = \infty \) (i.e., it always outputs the correct answer); and it filters all the data through every tree instead of a subset of data (explained further in Section 8.4.2). Note that our aim is not to beat these non-private techniques (which is
all but impossible), but simply to use them as a reference point for how high prediction accuracy can realistically get for the datasets we use.

### 8.3 Related Work

All of the differentially-private decision tree algorithms discussed in Section 3.3, Chapter 5 and Chapter 6 achieve differential privacy by adding Laplace noise [Dwork et al., 2006] to the counts of the labels in the nodes. While this approach makes good use of parallel composition (Definition 2.3), it scales poorly with multiple labels, since noise needs to be independently added to each label count.

Additionally, none of these techniques can take advantage of smooth sensitivity, and add much more noise than is necessary to build a differentially-privacy classifier. In fact, the amount of Laplace noise that needs to be added to a frequency query (such as label counts) cannot be reduced by using smooth sensitivity instead of global sensitivity; adding or removing one record can always change a count by 1, even when considering a specific dataset $x$. What this means is that submitting a frequency query to the dataset is an inherently expensive query, and should be avoided in favor of less expensive queries if possible. We demonstrate that this is indeed possible by devising a less sensitivity query that outputs a similar answer, resulting in less noise overall.

Aside from Rana et al. [2016], none of the decision tree algorithms discussed in Section 3.3 tested their algorithms with continuous features. However Friedman and Schuster [2010] includes a technically correct but costly (in terms of privacy) extension for handling continuous features, and Jagannathan et al. [2012] states that their algorithm can be trivially extended to handle continuous attributes by uniformly randomly selecting a split point from the continuous feature’s domain. Randomly selecting a split point is the same approach used by non-private random decision trees, and is the same approach we implement for our proposed algorithm. We compare our algorithm to Friedman and Schuster [2010], Jagannathan et al. [2012] and Rana et al. [2016] in our experiments, including the extensions for continuous features.

### 8.4 Our Decision Forest Algorithm

We propose a decision forest algorithm that is tailored to the differential privacy scenario. The complete algorithm can be seen in Algorithm 8.1 and Algorithm 8.2. We summarize
Algorithm 8.1 The proposed Differentially Private Random Decision Forest with Smooth Sensitivity.

1: procedure BUILDFOREST(Privacy budget $\epsilon$, dataset $x$, number of trees $\tau$, set of continuous features $S$, set of discrete features $R$, class label $C$)
2:     $F \leftarrow \{\}$
3:     $d \leftarrow$ The optimal tree depth according to our Theorem 8.2.
4:     for $t = 1, \ldots, \tau$ do
5:         $T \leftarrow$ BUILDTREE($d$, 0, $S$, $R$)
6:         $F \leftarrow F \cup T$
7:     end for
8:     $F \leftarrow$ GETMAJORITYLABELS($\epsilon$, $x$, $C$, $F$)
9:     return $F$
10: end procedure

our algorithm with the following steps:

**Step 1** Calculate the optimal tree depth (Section 8.4.3 and Line 3 of Algorithm 8.1).

**Step 2** Decide how many trees $\tau$ to build (Section 8.4.4 and user input in Algorithm 8.1).

**Step 3** Build a forest (Lines 4–7 of Algorithm 8.1) of $\tau$ random decision trees, without needing to query the data (Section 2.3.2 and the BUILDTREE procedure in Algorithm 8.2).

**Step 4** Query the leaf nodes and output the majority class labels (Section 8.4.1, Line 25 of Algorithm 8.2).

**Output** The decision forest model has finished being built. It can now be used to classify the labels of unseen data (Section 2.3.2).

The construction of the random decision trees is the same as the conventional approach discussed in Section 2.3.2; attributes are randomly chosen for each node. The BUILDTREE procedure in Algorithm 8.2 outlines the recursive tree-building process. The novel parts of our algorithm are the following: how we output the majority label of each leaf node (Section 8.4.1); our efficient utilization of the privacy budget (Section 8.4.2); our proposed tree depth, extending the non-private work of Fan et al. [2003] to handle numerical features (Section 8.4.3); and the number of trees we build (Section 8.4.4).

Observe that in Step 4, the query constructed for each leaf node includes the rules (i.e., tests) used by the nodes above the leaf node leading back to the root node. This is how a differentially-private algorithm can achieve the “filtering” process described in Section 2.3.2; since we can only access the data via queries, we cannot hold the whole dataset in memory for the root node, and partition it down through the tree. Instead, each leaf node’s query narrows the parameters of the what training records are included
Algorithm 8.2 A continuation of the algorithm presented in Algorithm 8.1.

1: **procedure** `BuildTree`(Maximum tree depth $d$, current depth $d'$, continuous features $S$, discrete features $R$)
2: $T \leftarrow \{\}$
3: if $d' < d$ then  ▶ Termination criteria.
4: Uniformly randomly select a feature $g$ from $S \cup R$ to split the current node.
5: if $g \in S$ then
6: Uniformly randomly select a split point $p$ within the current domain of $g$.
7: $T \leftarrow T \cup \text{BuildTree}(d, d' + 1, S, R)$  ▶ Left child node.
8: Original domain of $g$ ← Update the lower bound to $p$.
9: $T \leftarrow T \cup \text{BuildTree}(d, d' + 1, S, R)$  ▶ Right child node.
10: Original domain of $g$ ← Update the upper bound to $p$.
11: else $g \in R$
12: $B \leftarrow B - g$  ▶ Discrete features are only chosen once in a root-leaf path.
13: for all $g_i \in g$ do
14: $T \leftarrow T \cup \text{BuildTree}(d, d' + 1, S, R)$  ▶ All child nodes.
15: end for
16: end if
17: end if
18: return $T$
19: **end procedure**

20: **procedure** `GetMajorityLabels`(Budget $\epsilon$, dataset $x$, class label $C$, forest $F$)
21: for $t = 1, \ldots, \tau$ do
22: Query $f(x_t) \leftarrow$ Scope query $f$ to $x_t$ where $x_t$ is a disjoint subset of $x = \{x_1, \ldots, x_\tau\}$
23: for all Leaf nodes $L = \{L_i, \forall i\}$ in tree $F_t$, $F = \{F_1, \ldots, F_\tau\}$ do
24: $f(x_t \cap L_i) \leftarrow$ Scope $f(x_t)$ with all the tests in the root-to-leaf path leading to leaf node $L_i$.
25: Majority label of $L_i \leftarrow c \in C$ outputted by the Exponential mechanism using $f(x_t \cap L_i)$ and $\epsilon$, as well as the scoring function and smooth sensitivity proposed in Theorem 8.1.
26: end for
27: end for
28: return $F$
29: **end procedure**

so that they match the tests in the root-to-leaf path. This is an engineering consideration more than anything (it is less computationally efficient for example), with no effect on the algorithm’s results, but is worth mentioning nonetheless.

Differential privacy is achieved by our algorithm only outputting the following: the structure of the trees (which does not use the data); and the most frequent label in each leaf node, which is done using the Exponential mechanism. The user can then use the outputted labels from the leaf nodes in whatever way they wish; differential privacy is immune from post-processing, and differentially-private outputs can never incur additional privacy costs [Dwork and Roth, 2013]. More specifically, the user is
free to use majority voting; they can predict the label of a new record using the most common predicted label from all the trees in the ensemble.

8.4.1 Outputting the Majority Label

When outputting details about leaf nodes, rather than trying to return approximately-correct class frequencies like in Jagannathan et al. [2012] and Chapter 7, we instead observe that this is more information than is necessary in order to make a highly predictive classifier. To predict the class label of future records, we only need to know the majority class label in each leaf node (regardless of the number of times that label occurred). By not “wasting” some of the privacy budget on information we do not need, we propose querying the data with the Exponential mechanism to only output the (discrete) class label that is most frequent.

Definition 2.5 describes how the Exponential mechanism is capable of returning the discrete output of the most frequent class label in a leaf $x$. Our query takes the same form as seen in the definition, where the query $f$ will output the most frequently occurring label in $x$ with high probability. The precise probability is dependent on the scoring function $u$ and the privacy budget $\epsilon$. We propose the following novel scoring function:

Theorem 8.1. Given the leaf $x$ of a decision tree, the most frequent label can be differentially-privately queried using the Exponential mechanism [McSherry and Talwar, 2007]. The scoring function used for this query can be the piecewise linear function:

$$u(c, x) = \begin{cases} 1 & c = \arg \max_{i \in C} n_i \\ 0 & \text{otherwise} \end{cases},$$

where $n_c$ is the number of occurrences of $c$ in $x$. Each class label $c \in C$ will have a score: 1 if the label is the most frequently occurring label in the leaf; 0 otherwise.

The smooth sensitivity of $u(c, x)$ is

$$S^*(u, x) = e^{-j\epsilon}$$

where $\epsilon$ is the privacy budget of the query and $j$ equals the difference between the most frequent and the second-most frequent labels in $x$, $n_{c_1} - n_{c_2}$.

Proof. The global sensitivity of the scoring function $u$ seen in Equation 8.1 is 1 because, when considering any possible neighbors $x$ and $y$, adding or removing any record has
the potential to change which label(s) occurs most frequently:

$$GS(u) = \max_{x,y;||x-y||_1 \geq 1} \max_{c \in C} ||u(c,x) - u(c,y)||_1 = 1 . \quad (8.3)$$

A label will change from a score of 0 to 1 when it appears equally as frequently as the original most frequent label (at this point, 2 labels will be reporting a score of 1). A label’s score will change from 1 to 0 when another label occurs more frequently. In either case, the most that the scoring function $u$ can change by is $GS(u) = 1$.

The local sensitivity of $u$ differs from the global sensitivity by taking into account a specific $x$, rather than considering the worst theoretical outcome over all possible $x$’s. From Definition 2.7 and Definition 2.8 we get

$$S^k(x) = \max_{y;||x-y||_1 \leq k} \max_{c \in C} ||u(c,x) - u(c,y)||_1 . \quad (8.4)$$

Because we are considering the actual frequencies of each label in $x$, we are interested in how many records would need to be added or removed before a different label achieves a score $u$ of 1. This occurs when enough records with a different label have been added to equal the most frequent label in $x$, or when enough records have been removed from $x$ to drop the most frequent label to the same frequency as the next most common label. In either case, this first will occur at a distance $j$ from $x$, where $j$ is the difference between the most frequent and second-most frequent labels in $x$. Until then, for $k < j$, $S^k(x) = 0$.

When $k = j$, it is possible to have two labels with a score of $u = 1$. When $k > j$, it is possible to have one or more labels with a score of $u = 1$. Regardless, for $k \geq j$, the local sensitivity is $S^k(x) = 1$. To summarize, the local sensitivity of our scoring function $u$ is 0 until $y$ is sufficiently far away from $x$, at which point the sensitivity becomes 1.

We can represent this as

$$S^k(x) = \begin{cases} 0 & k < j \\ 1 & k \geq j \end{cases} : j = n_{c_1} - n_{c_2}, k \in \mathbb{N} , \quad (8.5)$$

where $n_{c_1}$ and $n_{c_2}$ are the frequencies of the most common and second-most common labels in $x$, respectively.

To finish implementing Definition 2.8, we input $S^k(x)$ from (Equation 8.5) into the smooth sensitivity (Equation 2.9). We now find the value of $k$ for which $e^{-k\epsilon} S^k(x)$ is maximized. For $k < j$, we have

$$e^{-k\epsilon} S^k(x) = 0 . \quad (8.6)$$
Table 8.2: Example smooth sensitivities of our scoring function $u$ using Theorem 8.1, when $\epsilon = 0.01, 0.1, 1.0$. Note how $j$ and $\epsilon$ have an equal impact on the result – increasing $j$ tenfold is the same as increasing $\epsilon$ tenfold.

<table>
<thead>
<tr>
<th>$j$</th>
<th>$S^*(u, x)$ when $\epsilon = 0.01$</th>
<th>$S^*(u, x)$ when $\epsilon = 0.1$</th>
<th>$S^*(u, x)$ when $\epsilon = 1.0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.00000</td>
<td>1.00000</td>
<td>1.00000</td>
</tr>
<tr>
<td>1</td>
<td>0.99004</td>
<td>0.90483</td>
<td>0.36788</td>
</tr>
<tr>
<td>5</td>
<td>0.95122</td>
<td>0.60653</td>
<td>0.00674</td>
</tr>
<tr>
<td>10</td>
<td>0.90483</td>
<td>0.36788</td>
<td>0.00005</td>
</tr>
<tr>
<td>50</td>
<td>0.60653</td>
<td>0.00674</td>
<td>$1.93 \times 10^{-22}$</td>
</tr>
<tr>
<td>100</td>
<td>0.36788</td>
<td>0.00005</td>
<td>$3.72 \times 10^{-44}$</td>
</tr>
<tr>
<td>500</td>
<td>0.00674</td>
<td>$1.93 \times 10^{-22}$</td>
<td>$7.12 \times 10^{-218}$</td>
</tr>
</tbody>
</table>

Two other possible scenarios exist: $k = j$ and $k > j$. Because $S^k(x)$ is never larger than 1, and $e^{-k\epsilon}$ becomes smaller as $k$ gets larger, we can deduce that $e^{-k\epsilon}S^k(x)$ is largest when $k = j$:

$$S^*(u, x) = \max_{k=0,1,...,n} e^{-k\epsilon}S^k(x) = e^{-j\epsilon}. \tag{8.7}$$

The above proof holds in any scenario where the most frequent item in a set is to be outputted. It also holds if the least frequent item is the desired output.

Using our proposed smooth sensitivity instead of the global sensitivity of 1, we can guarantee that equal or less noise is added to all queries. In some cases, the noise can be substantially lower. The smooth sensitivity of a query on data $x$ to return the most frequent label is dependent on only $j$ and $\epsilon$. We present the worst-case scenario (i.e., when $j = 0$) as well as several other scenarios in Table 8.2, using $\epsilon = 0.01$, $\epsilon = 0.1$ and $\epsilon = 1.0$.

In Figure 8.1, we present the practical effect of our smooth sensitivity. We present the prediction accuracy of our differentially-private random decision forest on seven synthetic datasets (described in Section 8.5). Two scenarios are demonstrated in the figure; when the Exponential mechanism queries in the leaf nodes use smooth sensitivity, and when they use global sensitivity. All our other parameters, described in the following sections, use the default settings.

Note the substantial improvement in prediction accuracy when using our smooth sensitivity – up to 26 percentage points in some cases, and never worse. This matches what we
expect from the theory, where even in the worse case scenario, the smooth sensitivity is equal to the global sensitivity. Figure 8.1 presents the results for $\epsilon = 1$, where Table 8.2 tells us that even with a modest $j$ of 10, the exponent of the Exponential mechanism is improved 20,000-fold. For $\epsilon = 0.1$, the improvement would still be three-fold.

### 8.4.2 Using Disjoint Data

When building multiple trees (say $\tau$ trees), there are two fundamentally different ways we can use our privacy budget $\epsilon$. One is to use composition (Definition 2.2), where all the records in $x$ ($n = |x|$) are used in every tree and we divide $\epsilon$ evenly amongst the trees, $\epsilon' = \epsilon / \tau$. The other way is to use parallel composition (Definition 2.3), dividing $x$ into disjoint subsets\(^2\) evenly amongst the trees ($n = |x| / \tau$) and using the entire $\epsilon$ budget in each tree.

When deciding which method to use, let us consider the factors that affect how noisy the output of our query proposed in Theorem 8.1 is. There are two such factors: $\epsilon$; and

---

\(^2\)When we refer to disjoint subsets of data, we are referring to a collection of records that are sampled from the total dataset without replacement. Each record can only appear in one disjoint subset.
Chapter 8. *Harnessing Smooth Sensitivity*

the sensitivity $S^*(u, x)$, which is itself dependent on only $\epsilon$ and $j$. When $n$ is larger, each leaf node will contain more records on average. Since we assume all the records in $x$ are sampled from the same population, the multinomial distribution will be approximately the same for both $n = |x|$ and $n = |x|/\tau$ number of records. Thus the relative frequency of both the number of records, and the ratio of class labels, will be the same in each leaf node, and $j$ will change accordingly. We can write this as $E[j] \propto n$.

One ramification of using all of $x$ in each tree is that the smaller $\epsilon/\tau$ privacy budget affects both the numerator and denominator of the Exponential mechanism, unlike $j$ which only affects the denominator. This means that $\epsilon$ has a larger impact on the result than $j$, and good-scoring labels have a higher chance of being outputted by the Exponential mechanism when disjoint subsets of $x$ are used:

$$\exp\left(\frac{\epsilon \times u(z, x)}{2 \exp(-j/\tau \times \epsilon)}\right) > \exp\left(\frac{\epsilon/\tau \times u(z, x)}{2 \exp(-j \times \epsilon/\tau)}\right)$$

when $E[j] \propto n$. This effect is most prominent with smaller datasets where $j$ is small, before the exponential nature of the smooth sensitivity overpowers the numerator by more than two or three orders of magnitude.

Another factor to consider when choosing between the composition and parallel composition theorems is the fact that the correlation between $j$ and $n$ is not one-to-one. If it was, that would imply that all leaf nodes that are empty (i.e., have no records in them) when $n = |x|/\tau$ will remain empty when $n = |x|$, when instead the reality is that the sample size was simply not big enough for any records to be in some of the leaf nodes. Therefore we expect that for some leaf nodes, neither their support nor their class label proportions (and in turn, $j$) will increase linearly with $n$. Of course, some previously empty leaf nodes now have records in them, and the most frequent label outputted by the Exponential mechanism in these leaf nodes will no longer be purely random. The privacy budget is only $\epsilon/\tau$ in this scenario though, leading to the difference between “purely random” and “almost purely random” being trivially small for some of these leaves. We empirically test the correlation between $j$ and $n$, as well as the number of empty leaf nodes, in Section 8.4.2.1.

Due to the above factors, we propose using parallel composition, and using disjoint data in each tree with the full privacy budget. 8.2a empirically demonstrates the improved prediction accuracy we see due to this decision. The only parameter changed for the comparison is whether or not discrete subsets of data are used in each of the 100 trees (and if not, $\epsilon$ is divided instead). All other parameters use the default settings, described in the other sections. This decision has the added benefit of substantially decreasing the
Figure 8.2: The (a) average prediction accuracy and (b) average smooth sensitivity of our proposed algorithm, with and without using disjoint data in each decision tree. All other parameters remain constant, using the default settings described in the other sections. The budget is $\epsilon = 1$ and the number of trees is $\tau = 100$. Recall that lower sensitivity is better.

computation time of our algorithm, from $O(n \log n)$ to

$$O\left(\frac{n}{\tau} \log \frac{n}{\tau}\right),$$

where $n$ is the number of records in the dataset.

We see improvements of up to 18 percentage points when disjoint data is used, and no losses by more than a fraction of one standard deviation. Aside from the large improvements in prediction accuracy, we can make another observation from 8.2b; one that might be surprising at first. The average\(^3\) smooth sensitivity (that is, $\exp(-j\epsilon)$) is substantially better when disjoint data is used, when initial intuition might tell us that $j$ and $\epsilon$ should be offsetting one another somewhat equally. Instead, we find that using all of the data in every tree does not lead to an increase in the average $j$ that is as large as the decrease in the privacy budget when going from $\epsilon$ to $\epsilon/\tau$. We explore this phenomena further in Section 8.4.2.1.

\(^3\)Note that we don’t include empty leaf nodes (which have a smooth sensitivity of 1) when calculating the average smooth sensitivity, since 70 to 99.9% of the leaf nodes are usually empty.
8.4.2.1 Empty Leaf Nodes

For almost any (non-uniform) distribution of feature values, records will start clumping together in certain nodes, with other nodes receiving very few records. This clumping becomes exacerbated for each non-uniformly distributed feature tested in each level of a tree. As a tree grows larger, the chances of some leaves having zero records in them also grows larger. We refer to leaf nodes with zero records in them as “empty”. We use a simple example to demonstrate:

**Example 8.1.** Let us imagine we have a dataset made of $m$ features and $n$ records, where each feature has three discrete values that follow the normal distribution: value $v_1$ contains 68% of the records; value $v_2$ contains $95\% - 68\% = 27\%$; and value $v_3$ has the remaining $100\% - 95\% = 5\%$ of the records. Let us further assume that all $m$ features are independent, to simplify the simulation. If we were to build a tree of depth $m/2$ with these $m$ features, we would have a tree with $3^{m/2}$ leaf nodes. Out of these leaf nodes, a single leaf would contain $0.68^{m/2} \times n$ records, and $(m/2) - 1$ other leaf nodes would contain proportions of records that were multiplications of 0.68 and 0.27 (such as $0.68 \times 0.68 \times 0.27 \times \ldots$). Conversely, there would be a leaf with $0.05^{m/2} \times n$ records in it; using a conservative $m = 10$, this would require a dataset of size $n = 3,200,000$ for there to be even one record in this leaf.

In our experiments, using both synthetic and real-world data, over 85% of the leaf nodes in any non-trivial tree are usually empty. By virtue of future data being from the same distribution as the training data, however, these empty leaf nodes are unlikely to be visited by future records. Any records that do finish at an empty leaf node will be predicted to have a class label that is randomly chosen with uniform probability (due to all labels having a score of 0 in the Exponential mechanism). We consider this to be less damaging than the same scenario in Jagannathan’s implementation of a differentially private random decision tree [Jagannathan et al., 2012], where labels with a frequency of zero still have Laplace noise added to them. In an empty leaf node in Jagannathan’s implementation, where every label reports a purely random frequency (because the true frequencies are zero), highly confident predictions could be falsely created.

Our empirical results are presented in Table 8.3. Note how when the data is divided among more trees, the number of empty leaf nodes always increases, as we would expect. This supports the observation we made with 8.2b: $j$ does not increase at the same ratio that $n$ does, because many of the extra records are going to previously empty leaf nodes. While this means these previously empty leaf nodes are no longer outputting a most frequent label with pure randomness, 8.2b shows us that this reduction in randomness
Table 8.3: The percentage of leaf nodes with no records in them, when building random trees with eight real-world datasets and seven synthetic datasets. We present the results for one tree and 30 trees, when using disjoint data in each tree, and a privacy budget of $\epsilon = 1$. We include one standard deviation for each result. The depth of the trees is defined by a novel theorem that we present in Section 8.4.3.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Total No. of Records</th>
<th>Tree Depth</th>
<th>% of Empty Leaves (1 tree)</th>
<th>% of Empty Leaves (30 trees)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SynthA</td>
<td>30000</td>
<td>5</td>
<td>49.8 ± 12.8</td>
<td>68.5 ± 2.0</td>
</tr>
<tr>
<td>SynthB</td>
<td>30000</td>
<td>8</td>
<td>79.1 ± 6.3</td>
<td>90.3 ± 0.7</td>
</tr>
<tr>
<td>SynthC</td>
<td>30000</td>
<td>12</td>
<td>95.0 ± 1.5</td>
<td>98.5 ± 0.1</td>
</tr>
<tr>
<td>SynthD</td>
<td>30000</td>
<td>12</td>
<td>95.7 ± 1.3</td>
<td>98.5 ± 0.1</td>
</tr>
<tr>
<td>SynthE</td>
<td>30000</td>
<td>12</td>
<td>63.3 ± 1.8</td>
<td>73.7 ± 0.1</td>
</tr>
<tr>
<td>SynthF</td>
<td>30000</td>
<td>8</td>
<td>79.2 ± 6.2</td>
<td>89.9 ± 0.7</td>
</tr>
<tr>
<td>SynthG</td>
<td>30000</td>
<td>15</td>
<td>98.7 ± 0.4</td>
<td>99.7 ± 0.0</td>
</tr>
<tr>
<td>WallSensor</td>
<td>5456</td>
<td>4</td>
<td>48.6 ± 12.3</td>
<td>65.1 ± 2.1</td>
</tr>
<tr>
<td>PenWritten</td>
<td>10992</td>
<td>12</td>
<td>92.4 ± 1.2</td>
<td>97.7 ± 0.1</td>
</tr>
<tr>
<td>GammaTele</td>
<td>19014</td>
<td>8</td>
<td>86.2 ± 3.8</td>
<td>93.5 ± 0.5</td>
</tr>
<tr>
<td>Adult</td>
<td>30162</td>
<td>9</td>
<td>99.8 ± 0.0</td>
<td>99.9 ± 0.0</td>
</tr>
<tr>
<td>Mushroom</td>
<td>5644</td>
<td>11</td>
<td>99.9 ± 0.0</td>
<td>99.9 ± 0.0</td>
</tr>
<tr>
<td>Claves</td>
<td>10800</td>
<td>8</td>
<td>0.0 ± 0.0</td>
<td>28.7 ± 0.4</td>
</tr>
<tr>
<td>Nursery</td>
<td>12960</td>
<td>4</td>
<td>0.2 ± 0.5</td>
<td>7.2 ± 0.8</td>
</tr>
</tbody>
</table>

does not negate the increase in randomness in all leaf nodes from dividing the privacy budget into $\epsilon/\tau$.

8.4.3 Tree Depth

An existing theory about the depth of a random tree was introduced in Section 2.3.2, where the ideal depth is half the number of features $m/2$. However this assumes that the features can only be selected once in any root-to-leaf path (by virtue of being discrete features, where all values are separated by the first node that selects the feature). Continuous features, on the other hand, can be randomly chosen any number of times (since there are many more split points remaining that could separate the records). The reasoning behind wanting to test $m/2$ unique features in each path still stands, but the probability of this happening at a depth of exactly $m/2$ is much lower if all (or some) of the features are continuous. We propose a new tree depth using the analysis below.
Table 8.4: The depths calculated using Theorem 8.2 for the synthetic and real datasets used in our experiments.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Continuous Features</th>
<th>Discrete Features</th>
<th>Depth</th>
</tr>
</thead>
<tbody>
<tr>
<td>SynthA</td>
<td>5</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>SynthB</td>
<td>10</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>SynthC</td>
<td>15</td>
<td>0</td>
<td>12</td>
</tr>
<tr>
<td>SynthD</td>
<td>15</td>
<td>0</td>
<td>12</td>
</tr>
<tr>
<td>SynthE</td>
<td>15</td>
<td>0</td>
<td>12</td>
</tr>
<tr>
<td>SynthF</td>
<td>10</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>SynthG</td>
<td>20</td>
<td>0</td>
<td>15</td>
</tr>
<tr>
<td>WallSensor</td>
<td>4</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>PenWritten</td>
<td>16</td>
<td>0</td>
<td>12</td>
</tr>
<tr>
<td>GammaTele</td>
<td>10</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>Adult</td>
<td>6</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>Mushroom</td>
<td>0</td>
<td>22</td>
<td>11</td>
</tr>
<tr>
<td>Claves</td>
<td>0</td>
<td>16</td>
<td>8</td>
</tr>
<tr>
<td>Nursery</td>
<td>0</td>
<td>8</td>
<td>4</td>
</tr>
</tbody>
</table>

Theorem 8.2. The expected number $X$ of continuous features $s$ not tested, on any particular root-to-leaf path of depth $d$, is equal to

$$E[X|d] = s \left( \frac{s-1}{s} \right)^d,$$

where each tested features is uniformly randomly selected with replacement. Using the same combinatorial reasoning used in Fan et al. [2003] and described in Section Section 2.3.2, the optimal tree depth is therefore

$$d = \arg \min_{d:X<s/2} E[X|d].$$

Proof. Our proof is the same as the proof for estimating the number of empty bins in the “Balls and Bins” problem (a companion of the famous “Birthday Paradox” and “Coupon Collector Problem” [Flajolet and Sedgewick, 2009]). Each continuous feature is a “bin”, and each node in a root-to-leaf path in a tree is a “ball”. Each time we randomly select
a feature (i.e., throw a ball), it has an equal chance of being (i.e., landing in) any of the 
scontinuous features (i.e., bins).

Let the random variable $X$ equal the number of features never selected in a root-to-leaf path. For each feature $i$, $X_i$ equals 1 if $i$ is never selected, and 0 otherwise. Since

$$E[X] = E[\sum_{i} X_i] = \sum_{i} E[X_i],$$

we only need to know $E[X_i]$ to know the expected number of features that are never selected. For any feature $i$, the probability that it will not be selected in a node is equal to the probability that any of the other $s - 1$ features will be selected, which equals
\[ \frac{s-1}{s} \]. If we repeat this for all \( d \) nodes, where each selection is independent of the others, the probability that \( i \) is never selected is equal to \( (\frac{s-1}{s})^d \). Thus \( \mathbb{E}[X_i] = (\frac{s-1}{s})^d \), and \( \mathbb{E}[X] = s(\frac{s-1}{s})^d \).

If a dataset has \( r \) discrete features in addition to the \( s \) continuous features, we add \( r/2 \) to the depth \( d \) defined by Theorem 8.2 as per the combinatorial reasoning of Fan et al. [2003]. Table 8.4 shows the tree depths for the datasets we use in our experiments, calculated using Theorem 8.2.

Figure 8.3 provides empirical evidence of the prediction accuracy gained when using our proposed depth, rather than simply \( m/2 \). We also present the prediction accuracy when a depth of \( m \) is used, to demonstrate that increasing the depth arbitrarily does not necessarily increase the accuracy (it actually decreases the accuracy for all but one dataset). Our proposed depth provides higher prediction accuracy than \( m/2 \) for all datasets, by at least 2 percentage points and as much as 30 percentage points. For most datasets, we also see less variance in the standard deviation when using our proposed depth over \( m/2 \), as expected from the analysis done by Fan et al. [2003].

### 8.4.4 Number of Trees

Another factor in the implementation of a differentially private decision forest with random decision trees is the number of trees to build. Figure 8.4 and Figure 8.5 present empirical results for a range of forest sizes. Figure 8.4 shows the average prediction accuracy results for our seven synthetic datasets, for both \( \epsilon = 0.2 \) and \( \epsilon = 1 \).

From these results, one observation is that having more trees is not necessarily better. Indeed, for \( \epsilon = 0.2 \) especially, we can see that there appears to be a “sweet spot” at 30 to 100 trees where prediction accuracy is highest. As \( \epsilon \) increases, this sweet spot increases to 100 to 300 trees. Another observation we can make is that for 1 to 10 trees, the prediction accuracy results vary by a lot more compared to when there are more trees, seen by the larger standard deviations. This is something we expect to see, given the high randomness in the construction of the trees. If we only build one random tree, there is a much higher chance of us getting very lucky or very unlucky when predicting future labels than there is if we build many random trees. With many trees, we can use the predictions made by each tree as votes and select the most voted class label as our prediction. The variance is also reduced due to the disjoint data used in each tree; using bootstrap (i.e. selected with replacement) samples is strongly advised for even non-private trees [Breiman, 2001a, Geurts et al., 2006]. While we cannot use sampling with replacement in our algorithm due to the privacy costs, sampling without replacement...
Chapter 8. *Harnessing Smooth Sensitivity*

### Figure 8.4

The average prediction accuracy of our proposed algorithm when building different numbers of trees, with privacy budget (a) $\epsilon = 0.2$ and (b) $\epsilon = 1.0$. We recommend building 100 trees when using our algorithm, seen in blue.
Figure 8.5: The percentage of non-empty leaf nodes that had their majority labels changed by the Exponential mechanism, when building different numbers of trees with a budget of $\epsilon = 1$. 
(which is what disjoint subsets achieves) reduces over-reliance on individual records, and thus variance, in the same way [Breiman, 2001a]. Having more trees also helps average out the noise caused by the Exponential mechanism. Of course, at some point having more votes no longer provides a benefit, and in the case of our differentially private scenario there is the added downside of having to divide up the data into more disjoint subsets – a problem that non-private decision forests are immune from, since they can sample with replacement.

Figure 8.5 portrays a different perspective on the same scenario as Figure 8.4. Figure 8.5 tells us that larger numbers of trees cause more (non-empty) leaf nodes to output a label that differs from the actual most frequent label. In other words, more most frequent label outputs are “flipped” to an incorrect label due to the Exponential mechanism. This is because the disjoint subsets of data are smaller when more trees are generated, which decreases the average size of \( j \) in each of the leaf nodes, which decreases the probability of the Exponential mechanism outputting the label with the highest score. Interestingly though, the proportion of incorrect predictions being caused to preserve privacy remains around 5 to 20%, even with 30,000 records being split among 100 trees. This means that 80% of the predictions are as accurate as possible, given the training data. With 100 trees, each unseen record has 100 votes on what its label is, and at least 80% of the votes (from non-empty leaf nodes) are likely to be correct. Even if the unseen record falls into a lot of empty leaf nodes, 100 votes is a large enough sample that the random votes cast by empty leaf nodes will cancel each other out in most cases. These findings are supported by the prediction accuracy results seen in Figure 8.4. This figure demonstrates that very good prediction accuracy is possible, with many datasets achieving 85% to 90% accuracy in 8.4a when using 100 trees. Given that this is under the strict conditions required to protect privacy, these results are very promising.

For all of the above reasons, and from further testing with additional forest sizes and epsilon values which agreed with the observations seen in Figure 8.4 and Figure 8.5, we recommend building 100 random trees with our algorithm.

8.5 Additional Experiments

Aside from the experiments included throughout Section 8.4, we present some other results here. In Section 8.5.1 we demonstrate that with larger datasets, much smaller privacy budgets are viable, and that larger privacy budgets cause the classifier quality to become asymptotically close to a non-private classifier. In Section 8.5.2 we compare our algorithm to similar algorithms, introduced in Section 8.3.
8.5.1 Scaling with Dataset Size and Privacy Budget

To save on computation time we perform our experiments with small datasets, with 30,000 records in our synthetic datasets and 5,456 to 30,162 records in our real-world datasets. To compensate for this, we use a relatively large privacy budget of $\epsilon = 1$ for most of our experiments. Figure 8.6 demonstrates why we can do this: one of the advantages of differential privacy is that it scales very well, adding less noise the more data there is [Dwork and Roth, 2013]. In other words, a sample of 3,000,000 records with $\epsilon = 0.1$ can achieve comparable results to a sample of 30,000 records when $\epsilon = 1.0$, as seen in Figure 8.6. What privacy budget is actually acceptable for any particular scenario in the real world depends very much on the specifics of the scenario. For example, a large public project was able to use a privacy budget of $\epsilon = 8.6$ [Machanavajjhala et al., 2008].

Differential privacy, and therefore our proposed technique, also scales well with larger privacy budgets, as seen in Figure 8.7. Here we can see that as $\epsilon$ increases, our differentially-private technique gets asymptotically close to a non-private version of our technique (described in Section 8.2). We include Breiman’s Random Forest technique [Breiman, 2001a] to act as a reference point.
Figure 8.7: The average prediction accuracy of our private technique as $\epsilon$ increases for the Adult dataset, approaching the prediction accuracy of a non-private extremely random forest, with all the same parameters as our technique except that no noise is added to the most frequent labels. We also include the accuracy of Breiman [2001a]'s Random Forest as context.

8.5.2 Comparisons with Other Techniques

We implement the following differentially private decision tree algorithms, using all of their recommended parameters: Jagannathan et al.’s [Jagannathan et al., 2012] (henceforth called JPW); Friedman and Schuster’s [Friedman and Schuster, 2010] (henceforth called FS); and Rana et al.’s [Rana et al., 2016] (henceforth called RGV). Since RGV uses a weaker form of differential privacy, we first compare with the algorithms using the same definition as us: JPW and FS. JPW heuristically recommends a forest size of 10 trees [Jagannathan et al., 2012], while FS only builds one tree [Friedman and Schuster, 2010]. For tree depth, JPW use a depth of $d = \min(m/2, \log_b n - 1)$ where $b$ is the average domain size of the features; FS uses a depth of $d = 5$. We then run both of their techniques, as well as ours, on seven synthetic datasets and seven real-world datasets with $\epsilon = 1$. We also run Breiman’s Random Forest technique [Breiman, 2001a] on each of the datasets, to act as a non-private benchmark. While comparing to a technique that completely disregards privacy is obviously unfair, it provides context about the prediction accuracy that is possible under more optimal conditions for each dataset. The results are presented in Figure 8.8.

All of the average prediction accuracy results reported in this section are statistically significant. Using the Wilcoxon signed rank test (since we cannot assume that the results are normally distributed), we find that the differences between our technique and JPW, our technique and FS, and our technique and RGV are statistically significant for all
Figure 8.8: The average prediction accuracy of three differentially-private tree algorithms with $\epsilon = 1$: our proposed technique, JPW [Jagannathan et al., 2012], and FS [Friedman and Schuster, 2010]. We also provide the prediction accuracy of Breiman [2001a]'s (non-private) Random Forest for context, portrayed as an outlined bar.
datasets. The weakest significance is between our technique and FS for the GammaTele dataset, with a p-value of 0.000057.

The last three real-world datasets presented in Figure 8.8 have discrete attributes only. Both of the other papers performed their experiments with only discrete features, and FS greatly prefers them due to continuous features being much more inefficient with the privacy budget [Friedman and Schuster, 2010]. Our only loss out of all datasets comes from one of these discrete datasets, Claves, where JPW achieves 2.5% better prediction accuracy on average.

We beat both techniques in all other cases, both when using discrete data and continuous data. While some improvements are minor (3.5% in the case of the Adult dataset against JPW), others are very large; more than 35% against both JPW and FS for SynthA. We often achieve more than a 10% improvement over JPW for other datasets, and 30% over FS.

Of course, the process of adding noise to query outputs to preserve privacy means that performing better than a technique that makes no effort to preserve privacy is all but impossible. We can see this in Figure 8.8, where Breiman’s Random Forest technique [Breiman, 2001a] always achieves better prediction accuracy. However, our technique performs almost as well for some synthetic datasets, as well as Adult, Mushroom and Nursery.

Rather than relying solely on prediction accuracy to measure the effectiveness of our algorithm, Figure 8.9 and Figure 8.10 present results for AUC (Area Under the ROC Curve) [Hanley and McNeil, 1982] and F1 score (sometimes called F-measure) [van Rijsbergen, 1979] respectively. AUC and F1 score work best on datasets with binary labels, so we limit the figures to just those datasets. When calculating the true and false predictions for the positive and negative labels, we consider the least frequent class label to be the positive case. Figure 8.9 and Figure 8.10 show that our algorithm outperforms JPW in 11 of the 20 results, ties in eight results (if we consider being within one standard deviation of each other a tie) and only loses in one result. This supports the findings of Figure 8.8, where JPW only beat us once, and by less than one standard deviation.

RGV is a little different; they propose a weakened definition of differential privacy, and prove that a large ensemble of trees\textsuperscript{4} can be made weakly differentially private with high utility. Put briefly, their differential privacy definition protects participants from their feature values being leaked, but not from their presence in the dataset being leaked. RGV uses the entire privacy budget in each tree, and determines how many trees can be

\textsuperscript{4}Each tree in RGV is built similarly to FS, with some modifications.
Chapter 8. Harnessing Smooth Sensitivity

Figure 8.9: The average area under the ROC curve of our algorithm, compared to Jagannathan et al. [2012]'s private JPW and Breiman [2001a]'s non-private Random Forest, when $\epsilon = 1$.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Ours</th>
<th>JPW</th>
<th>Breiman’s</th>
</tr>
</thead>
<tbody>
<tr>
<td>SynthA</td>
<td>0.877</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.826</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SynthB</td>
<td>0.924</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.915</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SynthC</td>
<td>0.920</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.888</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SynthD</td>
<td>0.879</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.881</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SynthE</td>
<td>0.907</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.830</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SynthF</td>
<td>0.888</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.946</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SynthG</td>
<td>0.917</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.835</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GammaTele</td>
<td>0.848</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.836</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mushroom</td>
<td>0.992</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.991</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Adult</td>
<td>0.861</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.808</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Even when competing against a less private algorithm, we see in Figure 8.11 that our algorithm still performs well. In fact we beat RGV in five cases, lose in four cases, and tie for the Adult dataset. Our accuracy improvements range from 1.5% to 13.0%, and RGV’s accuracy improvements range from 1.2% to 6.2%. This result is quite impressive given the unfair comparison, and challenges the notion that strict, $(\epsilon, 0)$-differential privacy [Dwork and Roth, 2013] is too strict for high-quality data mining to be possible [Hu et al., 2015, Rana et al., 2016].
8.6 Discussion

This chapter proposes and explores a new method for differentially privately outputting the most (or least) frequent item in a set, using smooth sensitivity. We apply these findings to a random decision forest framework and achieve substantially higher accuracy than the state-of-the-art [Friedman and Schuster, 2010, Jagannathan et al., 2012, Rana et al., 2016]. We also extend the work done by Fan et al. [2003] to calculate the optimal depth for a random decision tree when using continuous features.

Each subsection in Section 8.4 explores a component of our algorithm, leading to several novel conclusions that improve the utility of differentially private random decision forests. Each theoretical conclusion is coupled with empirical results, demonstrating the benefits of the theory when put into practice. Finally, in Section 8.5, combining all the improvements made by the previous subsections, we see that differentially private decision trees can be made from real-world data to create a highly accurate classifier, while simultaneously protecting the privacy of every person in the data. We also demonstrated in Section 8.5.2 that weakening the definition of differential privacy is not necessary in order to achieve good utility.
Figure 8.11: The average prediction accuracy of our technique compared to RGV [Rana et al., 2016] when $\epsilon = 1$. Note that RGV uses a weaker definition of differential privacy.

Section 8.5.1 demonstrates that with enough data, even very small privacy budgets are enough to make an accurate classifier. A user would not even have to use their entire budget on just our classifier, but can instead ask many other queries in addition to outputting our efficient classifier. On the other end of the spectrum, given enough privacy budget, a useful classifier can be built from very little data. Figure 8.6 demonstrates that with a budget of $\epsilon = 1$, 30,000 records is all that is needed to make a classifier with over 85% predictive accuracy.

Our findings in Section 8.4.1 are quite generalizable; now that it has been proven that
the smooth sensitivity of queries that output the most (or least) frequent item in a set is \( e^{-j\epsilon} \), future research (of both ourselves and others) can explore constructing other applications with similar queries. Many machine learning domains can take advantage of queries that output the most frequent item, such as frequent pattern mining [Bhaskar et al., 2010].

We hope that our findings on smooth sensitivity, disjoint data and tree depth with continuous features aid researchers in their future work, and that our high-accuracy classifier aids data scientists in building differentially private applications.

As a continuation of Table 3.1 (summarizing the properties of different differentially-private decision tree algorithms), Table 8.5 summarizes the algorithm we presented in this chapter. A combined table that includes all of our algorithms is presented and discussed in Chapter 9.

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Budget per Query</th>
<th>Tree Type</th>
<th>Tree Depth ( d )</th>
<th>Forest Size ( \tau )</th>
<th>Handles Cont. Attributes</th>
<th>Prediction Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapter 8</td>
<td>( \beta )</td>
<td>Random</td>
<td>Maximized Diversity</td>
<td>100</td>
<td>Yes</td>
<td>High</td>
</tr>
</tbody>
</table>
Part IV

Denouement
Given enough time, hydrogen starts to wonder where it came from, and where it is going.

ANONYMOUS
Chapter 9

Discussion

9.1 Our Decision Trees, in Context

By now, it is clear from our work in the previous three chapters (and the work of others in Section 3.3) that there are many opportunities for creative solutions and optimizations in designing differentially-private decision tree algorithms. Not only are there many moving parts in a decision tree algorithm, each with their own nuances, but there are a variety of differentially-private mechanisms that can preserve the privacy of participants in different ways. In Chapter 6 we explored how differential privacy interacts with splitting functions and pruning. In Chapter 7 we approached pruning from a different angle, in addition to exploring how to optimize the number of trees to build, and how deep they should be. In Chapter 8 we further optimized tree depth, and took advantage of the mathematics of smooth sensitivity to better integrate differential privacy into the tree-building process.

Our three proposed decision tree algorithms improve upon a pre-existing ecosystem of research. We discussed previous algorithms in Section 3.3.9 and included a summary of their properties in Table 3.1. Extending that summary to include our algorithms from Chapter 6, Chapter 7 and Chapter 8 gives us Table 9.1. While this table cannot hope to contain all of the details of each technique, it provides a “big picture” of differentially-private decision tree research, and how our work fits into it.

In the following two sections, we discuss some of the conflicts and trends we see when looking at the big picture.
## Table 9.1: A comparison of the main properties of all differentially-private decision tree algorithms, extending the table presented in Table 3.1.

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Budget per Query</th>
<th>Tree Type</th>
<th>Tree Depth $d$</th>
<th>Forest Size $\tau$</th>
<th>Handles Cont. Attributes</th>
<th>Prediction Accuracy$^1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blum, Dwork, McSherry &amp; Nissim (2005)</td>
<td>$\beta \over 2^{md-1}$</td>
<td>Greedy</td>
<td>$m$</td>
<td>1</td>
<td>No</td>
<td>Very Low$^2$</td>
</tr>
<tr>
<td>Friedman &amp; Schuster (2010)</td>
<td>$\beta \over (2^{d+1})^2$</td>
<td>Greedy</td>
<td>5</td>
<td>1</td>
<td>Poorly</td>
<td>Low</td>
</tr>
<tr>
<td>Mohammed, Barouti, Alhadidi &amp; Chen (2015)</td>
<td>$\beta/d$</td>
<td>Greedy</td>
<td>4</td>
<td>1</td>
<td>No</td>
<td>Medium$^2$</td>
</tr>
<tr>
<td>Patil &amp; Singh (2014)</td>
<td>$\beta \over (2^{d-1})$</td>
<td>Greedy</td>
<td>5</td>
<td>20</td>
<td>No</td>
<td>Very Low</td>
</tr>
<tr>
<td>Chapter 6</td>
<td>$\beta \over (2^{d-1})$</td>
<td>Greedy</td>
<td>5</td>
<td>4</td>
<td>No</td>
<td>Low</td>
</tr>
<tr>
<td>Rana, Gupta &amp; Venkatesh (2016)</td>
<td>$\beta/2d$ or $\beta/3d$</td>
<td>Greedy</td>
<td>Unspecified</td>
<td>500+</td>
<td>Yes</td>
<td>High$^3$</td>
</tr>
<tr>
<td>Jagannathan, Pillaiapakkammatt &amp; Wright (2012)</td>
<td>$\beta/\tau$</td>
<td>Random</td>
<td>Maximised Diversity (Discrete)$^4$</td>
<td>10</td>
<td>Yes</td>
<td>Medium</td>
</tr>
<tr>
<td>Jagannathan, Monteleoni &amp; Pillaiapakkammatt (2015)</td>
<td>$\beta/\tau$</td>
<td>Random</td>
<td>Maximised Diversity (Discrete)</td>
<td>5 then 200</td>
<td>Yes</td>
<td>High</td>
</tr>
<tr>
<td>Chapter 7</td>
<td>$\beta/\tau$</td>
<td>Random</td>
<td>Dynamic</td>
<td>$\leq m$</td>
<td>No</td>
<td>Medium</td>
</tr>
<tr>
<td>Chapter 8</td>
<td>$\beta$</td>
<td>Random</td>
<td>Maximised Diversity</td>
<td>100</td>
<td>Yes</td>
<td>High</td>
</tr>
</tbody>
</table>

$^1$ Prediction accuracy is categorized into quartiles of average relative empirical performance, calculated using the prediction accuracy results presented by each of the authors across a variety of datasets and $\beta$ values.

$^2$ Our best estimation; the authors did not empirically test the algorithm.

$^3$ With a weaker definition of differential privacy.

$^4$ Referring to Fan et al. [2003]’s optimal depth with discrete attributes. This (and other terminology in the table) is fully explained in Section 3.3.9.
9.2 Trading Knowledge for Accuracy

When looking at Table 9.1, one trend that stands out is the gap in prediction accuracy between greedy and random trees. Excluding Rana et al. [2016]'s work with a weaker definition of differential privacy, it has not been possible for a greedy decision tree model to perform as well as most random models. Even our own greedy decision tree algorithm in Chapter 6 fails to predict future labels as well as our work in Chapter 7 and Chapter 8. What Table 9.1 does not compare, however, is how well these models discover knowledge. This is a very difficult concept to quantify, but it is something we tackled in Chapter 4 with the Jaccard index, and use below.

Combining our work from Chapter 4 and Chapter 6, we compare our greedy tree algorithm to FS [Friedman and Schuster, 2010] using the Jaccard index in Figure 9.1. Here, we are measuring how many patterns discovered by our greedy tree are also discovered by the (non-private) CART algorithm [Breiman et al., 1984], and then doing the same with FS. We build one tree with each algorithm\(^1\), extract all the root-to-leaf paths in each tree, convert the paths into elements of a set as described in Chapter 4, and measure their similarity to each other. This process was performed on five datasets and repeated with 10-fold cross-validation, and Figure 9.1 presents the aggregated results. Our algorithm from Chapter 6 successfully finds more of the patterns discovered by CART (where privacy requirements are not perturbing the model) than FS does. Since the patterns discovered by CART are not perturbed, we can trust that they are more representative of real-world phenomena than patterns discovered under privacy restrictions.

\(^1\)The same parameters are used for each tree: maximum depth of \(\delta = 5\); privacy budget of \(\epsilon = 2\); forest size of \(\tau = 1\).
Accompanying Figure 9.1, we also provide the prediction accuracy of the three algorithms in Figure 9.2. Here we confirm that CART achieves higher accuracy, as expected for a non-private algorithm, but also that our algorithm outperforms FS in terms of prediction accuracy. This finding was previously presented in Chapter 6, but we present it here in conjunction with our Jaccard index measure from Chapter 4, demonstrating our greedy tree’s ability to both predict labels and discover knowledge. Not only that, but presenting the results of two different measures and learning different things from each of them demonstrates the usefulness of having workload-specific measures. As we discussed in Chapter 5, no measure can claim to be a “one size fits all” solution for quantifying what the user is interested in, and our Jaccard index measure is no exception. It does, however, perform well as a “one size fits some” solution (as is true of any good measure), and informs us about the quality of the patterns discovered by different algorithms.

In the same way that Figure 9.1 assesses the patterns discovered by Chapter 6’s algorithm and FS, we could perform the same measurement on our algorithms from Chapter 7 and Chapter 8, however the random nature of those algorithms means we would learn nothing by doing so – any patterns that are shared by a random tree and CART are, by definition, a coincidence, and do not tell us anything meaningful. Rather than empirically measuring the possibility for knowledge discovery from our random tree algorithms, we can assess them logically.

In Chapter 7, the algorithm outputs a forest of randomly-built decision trees, where the attributes used to make the root-to-leaf paths are entirely random. All is not lost however, as useful information is outputted in the leaf nodes: the class counts. By having (albeit, noisy) class counts, we learn two properties of each pattern in the decision
Table 9.2: In broad terms, the possibility for knowledge discovery compared to the model accuracy of our three differentially-private decision forest algorithms.

<table>
<thead>
<tr>
<th>Decision Forest</th>
<th>Knowledge Discovery</th>
<th>Prediction Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapter 6</td>
<td>Medium</td>
<td>Low</td>
</tr>
<tr>
<td>Chapter 7</td>
<td>Low</td>
<td>Medium</td>
</tr>
<tr>
<td>Chapter 8</td>
<td>None</td>
<td>High</td>
</tr>
</tbody>
</table>

forest: their support and confidence. As we saw in Chapter 5, these two properties play a key role in the usefulness of a pattern. By filtering the full set of patterns built by our algorithm down to only those patterns that happen to have high support and confidence, we are left with a set of patterns that each accurately classify a high proportion of the training data. This smaller set of patterns is much more credible in the eyes of the user, allowing them to trust the attributes and splitting values that make up those patterns. Essentially, given any sufficiently large set of randomly-built patterns, some of the patterns are bound to coincidentally describe a real pattern, and we can detect those using support and confidence. They are unlikely to be as finely tuned as patterns that are constructed using greedy splitting functions, and may contain redundant attributes, but they are better than nothing.

Meanwhile in Chapter 8, observe that the class counts are not outputted by our algorithm. By stripping away every piece of information that was not directly necessary to reach our objective of maximizing prediction accuracy, we sacrificed the ability to discovery knowledge. The only data-driven information that is outputted by a model built using Chapter 8 is the majority class label in each pattern, but we cannot detect which of these labels has been “flipped”\(^2\) to an incorrect label. Even if a label has not been flipped, we cannot tell what the support or confidence of a patterns is, rendering it useless in terms of knowledge. This is the price that needs to be paid if the user wishes to maximize accuracy, while also preserving privacy.

Table 9.2 summarizes the trade-offs that each of our three algorithms makes, compared to non-private algorithms such as CART and Breiman [2001a]’s Random Forest. The definition of “High” can roughly be considered to be whatever Random Forest can achieve; regarding knowledge discovery, the original paper and later work [Breiman, 2001b] logically and empirically proved the algorithm’s ability to discovery knowledge; regarding prediction accuracy, we demonstrate it empirically with Figure 9.3. This figure’s results are for when \(\epsilon = 1\); remember that given a larger budget, algorithms that

\(^2\)See Section 8.4.4.
query the dataset many more times (such as Chapter 6) will perform better, but larger budgets are also less realistic.

### 9.3 Comparing Computational Complexity

When dealing with privacy preservation, the computational complexity of the privacy-preserving algorithms is less of a concern than meeting the privacy requirements and optimizing utility. Consideration of computational complexity is also less necessary in this scenario given that the process of preserving privacy is usually a non-interactive, “offline” process, where real-time results are not a factor.

Having said that, it can still be interesting to consider the computational efficiency of our three proposed algorithms. At their core, they are decision tree algorithms, and use the same construction procedure as their non-private counterparts. More specifically, our Chapter 6 algorithm has a complexity of $O(mn\delta\tau)$, our Chapter 7 algorithm has a complexity of $O(n\tau)$, and our Chapter 8 algorithm has a complexity of $O(n)$.

Regarding Chapter 6, the algorithm is a greedy algorithm that scans the $m$ attributes of the $n$ records on each of the $\delta$ levels of the tree, for each of the $\tau$ trees. For Chapter 7, the algorithm does not need the data until it counts the class labels in the leaf nodes, which occurs once in each tree. For Chapter 8, the data used in each tree is disjoint, and thus the total $n$ records are only scanned once each.

Note that the privacy-preserving components of these algorithms do not cost any additional computation time – the Laplace mechanism merely adds noise to outputs, and the
Exponential mechanism probabilistically selects an output from a set of outputs that non-private versions of the algorithms would calculate anyway.

9.4 The Future

Looking forward into the future of differentially private trees, several open questions catch our attention. While random trees currently reign supreme when it comes to prediction accuracy (see Figure 3.4 and Figure 8.8), the recent developments with smooth sensitivity (Chapter 8) offer some promising possibilities for greedy trees. We wonder if using smooth sensitivity would lower the noise enough to allow the Exponential mechanism to output the best splitting attribute more reliably at low $\epsilon$ values. Not only would this improve single greedy trees, but it might open the way for ensembles of greedy trees to perform well; as we can see in Table 9.1, it is difficult for even a forest of four trees to perform well, let alone a forest on the same scale as random forests.

The optimal depth of differentially-private greedy trees is another open question. The depth of their random counterparts has been explored much more thoroughly, as we can see in Section 3.3.4. This question becomes even more interesting when considering the cost of the queries in each non-leaf level of the tree; the deeper the tree grows, the more the budget must be divided. This of course presupposes that there is not a creative way of using disjoint data in each level of the tree; such a discovery would open up a wealth of new possibilities.

We can see in Table 9.1 that there is some amount of disagreement about how many trees are needed in a differentially-private random forest. Jagannathan et al. [2012] claim that $\tau = 10$ trees is enough to achieve high accuracy, citing the findings of Fan et al. [2003] as evidence. Conversely, we empirically found in Chapter 8 that on the order of 100 trees was preferable. This matches more closely with the work of Geurts et al. [2006], who used 100 to 1000 random trees. More research in this area may elicit some heuristics for defining $\tau$, or perhaps more rigorous mathematics for finding the optimal $\tau$.

Rana et al. [2016] explored weakening the definition of differential privacy, which led to greatly improved results (discussed in Section 3.3.7). They proposed one strategy for how to weaken the definition, but there is no reason to suspect that there are not other possible definitions. The original privacy definition proposed by Dwork [2006] has been thoroughly investigated and is mathematically rigorous (see Section 2.5), and this high standard should be required of any competing definition. Rana et al. [2016]’s definition (and any future definitions) would thus require more research before they can be considered true alternatives to the original definition when the privacy of real people
is at stake. If such an alternative was developed though, the improvements in data mining quality could be substantial.

In Section 3.3.8 we discussed how small savings can be made with the privacy budget by “cutting corners” where possible. While an elegant algorithm might require $g$ queries, heuristics can allow us to remove $h$ unnecessary queries by either inferring the results from the other $g - h$ queries, or relying on data-independent estimates where accurate results are not absolutely necessary. Increasing the budget per query from $\epsilon = \beta/g$ to $\epsilon = \beta/(g - h)$ is likely worth any increases in algorithmic complexity caused by adding additional steps or heuristics. As differentially-private decision trees are further honed in the future, small improvements such as these will be key to optimizing the results in the real world.

The final open area that catches our attention is the idea of spending unequal quantities of the privacy budget on different queries. So far in the literature, all algorithms have assumed that all $g$ queries will use $\beta/g$ of the budget, implicitly assuming that all the queries are of equal importance to the success of the algorithm, or require equal levels of accuracy to be useful. We suspect that some queries, however, can achieve their purpose in the algorithm with more added noise than other more “fragile” queries. This would be an interesting idea to explore in the future.
Chapter 10

Conclusion

When sensitive data – data that risks the privacy of the people it describes – needs to be modeled, decision trees lend themselves well to the strict requirements of differential privacy. We have surveyed the current literature of differentially-private decision trees, breaking down the state-of-the-art algorithms into their constituent parts. Termination criteria, pruning strategies, building an ensemble of trees, greedy heuristics in non-leaf nodes, and querying the labels in leaf nodes are some of the factors that must be considered when making a high-performing decision tree algorithm. We explored each of these factors in turn, as well as others. We compared and contrasted the design decisions made by the authors, evaluating their effectiveness at discovering predictive knowledge from sensitive data.

We then discussed two styles of measuring comparative utility. One enables you to compare two sets of patterns, measuring how many patterns they have in common. The other is a framework for developing customized comparisons between two datasets. These two styles offer the user ways of measuring previously unmeasurable properties of their anonymized data, and their data mining results.

Three decision tree algorithms were then presented, each appearing at different points along the knowledge-to-accuracy spectrum. More than just self-contained algorithms, their development led to more generalized strategies and mathematics that are applicable beyond the work done in this thesis, hopefully empowering future researchers and data scientists to invent exciting new ways of modeling sensitive data.

One theme that re-occurred throughout our discussion was the ever-present conflict between privacy and utility. Whenever sensitive data is queried, it comes at a cost, and that cost needs to be weighed against the benefits. Those benefits depend on the aim of the algorithm; is it aiming to discover knowledge, build a model with high prediction
accuracy, both, or something else? These aims each lend themselves to slightly different strategies, changing the cost-benefit analysis that the user makes when deciding when their algorithm should query the data. Depending on how a query is asked, its sensitivity can also change drastically. As we saw in Chapter 8, even changing a query slightly from “what are the class counts in this leaf node?” to “what is the majority class label in this leaf node?” allows for a much less noisy answer to be outputted.

Balancing the requirements of differential privacy with a decision tree algorithm that optimizes utility is a delicate process, and one that should be done with great care. It is our hope that the reader has gained a thorough understanding of the nuances involved in designing a differentially-private decision tree algorithm, and we eagerly await the progress that the future holds.
Appendix A

Datasets Used

Below are the full names of all the datasets we use throughout the thesis. We also include a URL to where each dataset can be downloaded, and brief descriptions of each. They are obtained from the UC Irvine Machine Learning Repository, available at https://archive.ics.uci.edu/ml.

**Adult:** Adult Data Set. Predict whether income exceeds $50K/yr based on census data. Also known as “Census Income” dataset.
https://archive.ics.uci.edu/ml/datasets/Adult

**Bank:** Bank Marketing Data Set. The data is related with direct marketing campaigns (phone calls) of a Portuguese banking institution. The classification goal is to predict if the client will subscribe a term deposit.
https://archive.ics.uci.edu/ml/datasets/Bank+Marketing

**Banknotes:** Banknote Authentication Data Set. Data were extracted from images that were taken for the evaluation of an authentication procedure for banknotes.
https://archive.ics.uci.edu/ml/datasets/banknote+authentication

**Car:** Car Evaluation Data Set. Derived from simple hierarchical decision model, this database may be useful for testing constructive induction and structure discovery methods.

**Cardio:** Cardiotocography Data Set. The dataset consists of measurements of fetal heart rate (FHR) and uterine contraction (UC) features on cardiotocograms classified by expert obstetricians.
https://archive.ics.uci.edu/ml/datasets/Cardiotocography
Chess: Chess (King-Rook vs. King-Pawn) Data Set. King+Rook versus King+Pawn on a7.
https://archive.ics.uci.edu/ml/datasets/Chess+(King-Rook+vs.+King-Pawn)

Claves: Firm-Teacher Clave-Direction Classification Data Set. The data are binary attack-point vectors and their clave-direction class(es) according to the partido-alto-based paradigm.
https://archive.ics.uci.edu/ml/datasets/Firm-Teacher_Clave-Direction_Classification

Connect4: Connect-4 Data Set. Contains connect-4 positions.
https://archive.ics.uci.edu/ml/datasets/Connect-4

Credit: Credit Approval Data Set. This data concerns credit card applications.
https://archive.ics.uci.edu/ml/datasets/Credit+Approval

GammaTele: MAGIC Gamma Telescope Data Set. Data are MC generated to simulate registration of high energy gamma particles in an atmospheric Cherenkov telescope.
https://archive.ics.uci.edu/ml/datasets/MAGIC+Gamma+Telescope

Mushroom: Mushroom Data Set. From Audobon Society Field Guide; mushrooms described in terms of physical characteristics; classification: poisonous or edible.
https://archive.ics.uci.edu/ml/datasets/Mushroom

Nursery: Nursery Data Set. Nursery Database was derived from a hierarchical decision model originally developed to rank applications for nursery schools.
https://archive.ics.uci.edu/ml/datasets/Nursery

OptDigits: Optical Recognition of Handwritten Digits Data Set. We used pre-processing programs made available by NIST to extract normalized bitmaps of handwritten digits from a pre-printed form of 43 people.

PageBlocks: Page Blocks Classification Data Set. The problem consists of classifying all the blocks of the page layout of a document that has been detected by a segmentation process.
https://archive.ics.uci.edu/ml/datasets/Page+Blocks+Classification

Parkinsons: Parkinsons Telemonitoring Data Set. Oxford Parkinson’s Disease Telemonitoring Dataset.
https://archive.ics.uci.edu/ml/datasets/Parkinsons+Telemonitoring
**Appendix A. Datasets Used**

**PenWritten**: Pen-Based Recognition of Handwritten Digits Data Set. Digit database of 250 samples from 44 writers.

http://archive.ics.uci.edu/ml/datasets/Pen-Based+Recognition+of+Handwritten+Digits

**RedWine**: Wine Quality Data Set. Red vinho verde wine samples, from the north of Portugal. The goal is to model wine quality based on physicochemical tests.

https://archive.ics.uci.edu/ml/datasets/Wine+Quality

**Shuttle**: Statlog (Shuttle) Data Set. The shuttle dataset contains 9 attributes all of which are numerical. Approximately 80% of the data belongs to class 1.

https://archive.ics.uci.edu/ml/datasets/Statlog+(Shuttle)

**Spambase**: Spambase Data Set. Classifying Email as Spam or Non-Spam.

https://archive.ics.uci.edu/ml/datasets/Spambase

**TicTacToe**: Tic-Tac-Toe Endgame Data Set. Binary classification task on possible configurations of tic-tac-toe game.

https://archive.ics.uci.edu/ml/datasets/Tic-Tac-Toe+Endgame

**Vehicle**: Statlog (Vehicle Silhouettes) Data Set. 3D objects within a 2D image by application of an ensemble of shape feature extractors to the 2D silhouettes of the objects.

https://archive.ics.uci.edu/ml/datasets/Statlog+(Vehicle+Silhouettes)

**WallSensor**: Wall-Following Robot Navigation Data Data Set. The data were collected as the SCITOS G5 robot navigates through the room following the wall in a clockwise direction, for 4 rounds, using 24 ultrasound sensors arranged circularly around its ‘waist’.


**Wilt**: Wilt Data Set. High-resolution Remote Sensing data set (Quickbird). Small number of training samples of diseased trees, large number for other land cover. Testing data set from stratified random sample of image.

https://archive.ics.uci.edu/ml/datasets/Wilt

**Yeast**: Yeast Data Set. Predicting the Cellular Localization Sites of Proteins.

https://archive.ics.uci.edu/ml/datasets/Yeast
Bibliography


Hong Cheng, Xifeng Yan, Jiawei Han, and Chih-Wei Hsu. Discriminative Frequent Pattern Analysis for Effective Classification. In *23rd International Conference on Data Engineering*, pages 716–725. IEEE, 2007.


Andy Greenberg. Apple’s “Differential Privacy” is about collecting your data – but not your data, 2016. URL


Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. In ACM SIGMOD Record, volume 29, pages 1–12, Dallas, Texas, 2000. ACM.

Jiawei Han, Micheline Kamber, and Jian Pei. Data Mining: Concepts and Techniques. Morgan Kaufmann Publishers, San Diego, 2006.

Jiawei Han, Hong Cheng, Dong Xin, and Xifeng Yan. Frequent pattern mining: current status and future directions. Data Mining and Knowledge Discovery, 15(1):55–86, 2007.


P. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*. Addison-Wesley, Boston, USA, 1 edition, 2005.


