PRIVACY-PRESERVING DATA MINING

ONE SEMESTER INDIVIDUAL PROJECT

40 CREDITS

AUTHOR:
JASMIN BECKFORD

SUPERVISOR:
DR. JIANHUA SHAO

MODERATOR:
PADRAIG CORCORAN

ABSTRACT

This project aims to understand the effects of anonymising data on its subsequent effectiveness when subjected to data mining processes. The use of anonymised data to produce decision trees for classification tasks is explored and the resultant accuracy of classifying unseen instances is evaluated. By comparing this with the performance of a decision tree derived from a non-anonymised dataset in the same classification task, the extent to which anonymisation influences the results of data mining can be discovered.

ACKNOWLEDGEMENTS

I would like to take this opportunity to thank my supervisor, Dr. Jianhua Shao, for his time and effort in assisting me with this project. Your guidance has been greatly appreciated.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1 THE CHALLENGE OF PRIVACY

With advancements in the availability and capability of technology, a growing number of organisations and businesses have been able to collect vast amounts of personal data regarding their consumers. This electronic data can be subjected to various data mining techniques in order to provide valuable insight and aid important decision-making processes. Furthermore, it is not uncommon for data to be shared amongst other organisations or researchers within an industry with the view of adopting a collaborative working approach.

As organisations share the personal data of their consumers, it is important that the privacy of the individuals contained within the dataset is maintained. Data providers, such as hospitals and banks, face the challenge of producing anonymised data that can be safely released to a wider audience while still protecting the privacy of their consumers. This encompasses the field of privacy-preserving data mining in which traditional data mining techniques can be carried out on data in a way in which the identities of the concerned individuals remain unknown.

It may be thought that the easiest way to do this is by simply omitting explicit identifiers, such as name or address, which can easily identify an individual. However, research has shown that this does not guarantee protection from re-identification of an individual. An experiment into the 1990 US Census summary data found that 87% of the population of the United States could be uniquely identified using only {5-digit ZIP, gender, date of birth} [1]. Additional methods of anonymising datasets have since been proposed, such as k-anonymity, which reduces the likelihood of an individual being identified. These will be explored further throughout this project.

The greater a dataset is modified to preserve an individual's privacy, the more the granularity of the data is reduced. Thus, the less useful the data becomes. Due to the obfuscation of the values in the modified data, any subsequent data mining is expected to yield less valuable results. The information could become too generic for its intended purpose, meaning that the increased privacy of the data may also simultaneously compromise its effectiveness.

1.2 PROJECT GOALS

Throughout this project, I will explore the trade-off between information loss and privacy and discover by what extent, if any, privacy preservation affects the efficiency of data mining processes. In particular, I will focus on classification as a chosen data mining technique and will evaluate the extent to which decision tree classifiers maintain their accuracy when derived from a privacy-preserved dataset compared to when derived from the same dataset in its original non-privacy-preserved form.

1.3 INTENDED AUDIENCE

This project is aimed at all industries and organisations where data mining is used while the protection of privacy for individuals contained within the data is paramount. Healthcare providers, banks or law enforcement could benefit from the results obtained in this project as it could highlight whether the results of subsequent data mining on any released anonymised data are of significant value.

2. BACKGROUND

2.1 PRIVACY-PRESERVING DATA MINING

Privacy-preserving data mining focuses on protecting the privacy of individuals contained within a dataset while third parties are granted access to submit the data to various data mining techniques. A given dataset is generally classed as privacy preserved if no individuals within the released dataset can be re-identified. In the introduction, it was touched upon that obscuring explicit identifiers from a dataset, such as name and address, is usually not enough to prevent against the re-identification of individuals. This section will further explore the exploitation of such datasets and will discover how their existence has led to the evolution of privacy-preserving data mining as we now know of it today.

2.1.1 RE-IDENTIFICATION BY LINKING

A particular attack technique which has led to tighter definitions on preserving the privacy of datasets is re-identification by linking. This occurs in situations, as discussed above, where not enough information is modified in a released dataset to prevent against identifying a unique individual from the data. To illustrate this issue, consider the Venn diagram displayed in Figure 1. This shows an example of attributes that could be included in two datasets: (1) medical data from a healthcare organisation and (2) data about the people that voted in an election.

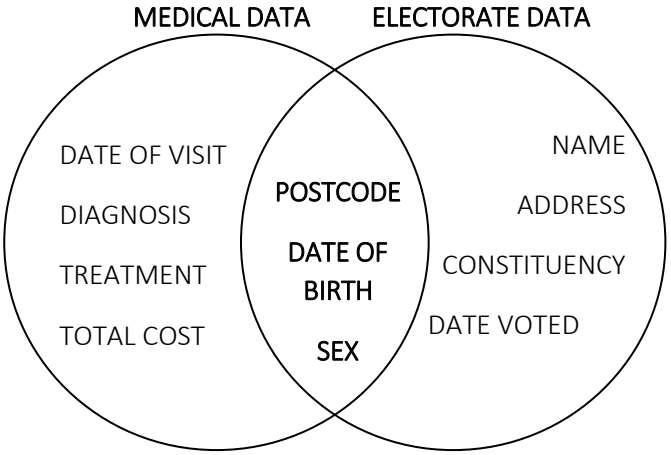

FIGURE 1: A Venn diagram to illustrate a re-identification by linking attack.

The leftmost circle demonstrates attributes that could be included in data collected by healthcare organisations such as a hospital. Imagine that the hospital chooses to share the patient-related data with researchers in order to gain a deeper understanding of medical issues. In order to protect the privacy of its patients, certain person-specific data such as name and address, have been deliberately omitted from the dataset before being shared with third parties. The hospital thinks that it has successfully maintained patient privacy.

The rightmost circle shows attributes that may appear on datasets regarding political elections. For instances like the open register in the UK, it is not uncommon for personal data such as name, address and date of birth to be made publically available for sale to people, companies or organisations. The intersection of the two circles shows that postcode, date of birth and sex appear in both the medical data and the electorate data. This demonstrates that the names and addresses of individuals in the medical data can easily be discovered by linking the two datasets. The data that was released by the hospital is now shown to, in fact, not be preserving patient privacy as unique individuals can be re-identified from the released dataset.

Table 1 and Table 2 further illustrate the scenario by giving example datasets that clearly show how re-identification can occur by linking the attributes shown in grey. Using the publically-available electorate data combined with any background knowledge, it can be found out that in the medical dataset, it was Mr. John Doe that had treatment for appendicitis.

DATE OF BIRTH	SEX	POSTCODE	DATE OF VISIT	DIAGNOSIS	TREATMENT	TOTAL COST
28/03/1978	M	M5 8LS	17/06/2015	Appendicitis	Removal of appendix via surgery	£26,000

TABLE 1: An example of a released medical dataset

NAME	ADDRESS	POSTCODE	SEX	DATE OF BIRTH	CONSTITUENCY	DATE VOTED
Mr. John Doe	109 Wood Street	M5 8LS	M	28/03/1978	Salford and Eccles	07/05/2015

TABLE 2: An example of publically-available electorate data

2.1.2 NOTABLE EXAMPLES OF RE-IDENTIFICATION BY LINKING ATTACKS

1. WILLIAM WELD – GOVERNOR OF MASSACHUSETTS

In 1997, the Governor of Massachusetts, William Weld, was successfully re-identified from a dataset through means of a similar situation as previously described. His medical data was included in a dataset collected by Group Insurance Commission (GIC) regarding health insurance. Knowing that Governor Weld resided in Cambridge, Massachusetts, a computer science student, Latanya Sweeney, purchased a voter registration list for Cambridge containing information such as name, address, ZIP code, sex. Using this Cambridge voter list, Sweeney was able to identify that there were six people in Cambridge that shared Governor Weld's date of birth. Of these six people, three were male and Governor Weld was the only one in his 5-digit ZIP code [2]. With this knowledge, he was successfully identified from the GIC health insurance data which contained confidential information pertaining to his diagnosis and prescriptions [3]. This re-identification by Sweeney was the first of many research works into privacy-preserving data mining.

2. AOL PRIVACY BREACH – USER NO. 4417749

In August 2006, AOL released a list of 20 million web search queries made by users [4]. In an attempt to protect the user's identity, each user was assigned a random ID number which appeared in the released data instead of their name. Amongst this extensive list were a number of searches made by user no. 4417749. Although the name of the user was obfuscated from the data released by AOL, the web searches revealed a number of personally identifiable information. Queries included searches for "landscapers in Lilburn, GA" as well as several queries for people with the surname Arnold. Shortly after the data had been made publically available, The New York Times was able to identify user no. 4417749 as Ms. Thelma Arnold, a 60-year-old woman from Lilburn, Georgia. This clearly highlights how an individual can be uniquely identified from personally identifiable information other than an individual's name and illustrates the growing need for privacy-preserving data.

2.2 K-ANONYMITY

Subsequent to her work on the re-identification of the governor of Massachusetts, Latanya Sweeney proposed k-anonymity as a method to prevent against re-identification by linking attacks. Before exploring further into the field, I would like to highlight the following terminologies.

TERMINOLOGY	DEFINITION	ALSO KNOWN AS
TABLE	A given dataset forms a table which is made up of tuples and attributes.	Data, dataset
TUPLE	A row in a table which constitutes a set of attribute values for a given record.	Row, record
ATTRIBUTE	A column in a table which consists of a set of possible values.	Column, domain, field
QUASI-IDENTIFIER	A set of attributes which, when combined, can be used to uniquely identify an individual.	QI

The formal definition of k-anonymity is shown below. However, in simpler terms, k-anonymity specifies that for a given table, each tuple must not be distinguishable from at least k-1 tuples in the table with respect to a given quasi-identifier.

DEFINITION - K-ANONYMITY

Let RT(A1, ..., An) be a table and QIRT be the quasi-identifier associated with it. RT is said to satisfy k-anonymity if and only if each sequence of values in RT[QIRT] appears with at least k occurrences in RT[QIRT] [1]

For example, consider the quasi-identifiers {Marital Status, Nationality} where k=2, meaning that Marital Status and Nationality provide enough information to uniquely identify an individual in a table. With k equalling 2, this means that for each combination of quasi-identifier values, there must be at least 2 instances of that combination in a given table. Figure 2 illustrates this by depicting a simplified representation of a table which does not adhere to k-anonymity side-by-side with the same table in a k-anonymous form (k=2). Taking the example that Marital Status and Nationality are quasi-identifiers, it can be seen that in the leftmost table, T, every tuple is unique from the other. Thus, meaning that if an individual was known to be included in table T and their marital status and nationality was also known, they could easily be identified in the table. A dataset would usually also contain a sensitive attribute which the individuals would not want to be disclosed such as salary or

T	MARITAL STATUS	NATIONALITY
1	Single	British
2	Married	Italian
3	Married	Spanish
4	Divorced	British
5	Single	Spanish
6	Married	German

T'	MARITAL STATUS	NATIONALITY
1	(Single, Divorced)	British
4	(Single, Divorced)	British
2	Married	(Italian, German)
6	Married	(Italian, German)
3	(Single, Married)	Spanish
5	(Single, Married)	Spanish

FIGURE 2: A non-conforming table versus a k-anonymous table where k = 2

medical diagnosis. Along with the re-identification of an individual in the dataset, the individual's value for the sensitive attribute would also be exposed, thereby removing all privacy for the individual concerned.

Compare this to the rightmost table in Figure 2, T', which adheres to k-anonymity where k = 2. In this table, if an individual is known to be in the table, there are two tuples in the table that could represent them. Providing that the sensitive attributes for each of these two tuples are different, the sensitive attribute for the individual whom we are attempting to re-identify can only be estimated with 50% certainty. It is clear to see that as the value of k increases with regards to k-anonymity, the more difficult it becomes to know the sensitive attribute of a re-identified individual with certainty. Therefore, as k increases, the privacy of the individuals in the table also increases.

2.2.1 GENERALISATION AND SUPPRESSION

There are two main techniques for ensuring k-anonymity in a table: generalisation and suppression. Generalisation involves replacing an attribute value with a broader, less specific value while maintaining its semantic meaning. For example, the values 'Male' or 'Female' for the attribute Sex could be generalised to become 'Person'. Where no generalisation has occurred in a table and the attribute values are as specific as possible, the value for each attribute is said to be in the ground domain [5]. As values become more generalised, the domain of the value increases. This relationship can be represented in a diagram known as a domain generalisation hierarchy. Furthermore, with generalisation, the possible values of a domain are ordered in a hierarchical form which illustrates the relationship between the values at each domain level. This is known as a value generalisation hierarchy.

Figure 3 depicts a domain generalisation hierarchy and a value generalisation hierarchy for the example attribute Postcode. It can be seen how the values become less specific with each generalisation by using asterisks to broaden the possibilities of the values. Starting with a ground domain of specific postcodes in Manchester, the values are generalised to represent a broader area in Manchester and then generalised again to represent any postcode in Manchester. The values at the

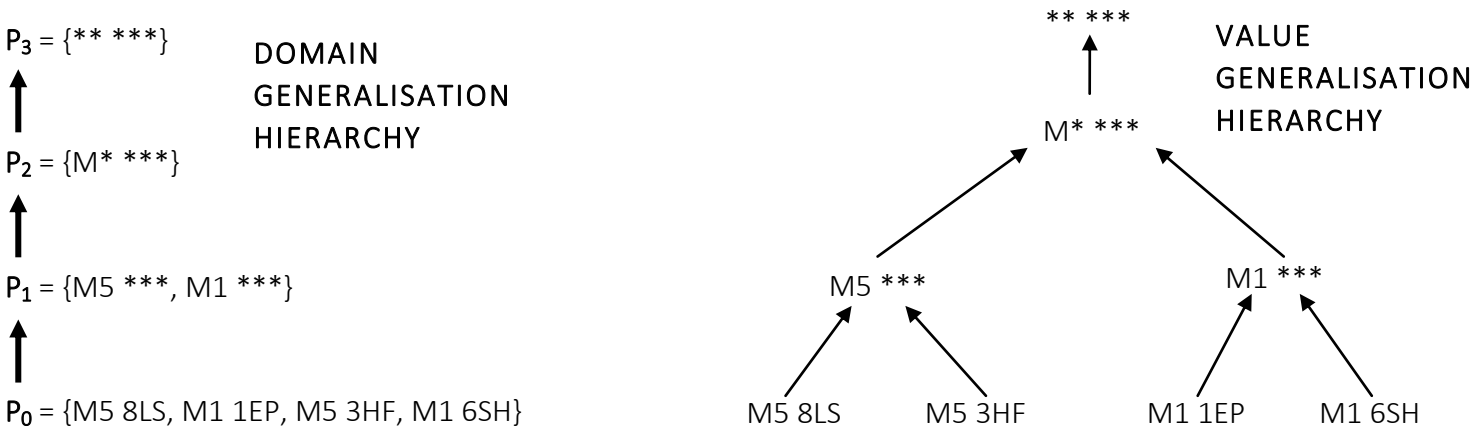

FIGURE 3: An example of a domain generalisation hierarchy and value generalisation hierarchy

top-level of the hierarchies represent suppression. This occurs when a value is removed from the dataset altogether. For example, the suppressed value for the attribute Postcode can represent any postcode in the UK. This is also known as the maximal element since there is no further representation of the value.

Generalisation can assist in enforcing k-anonymity since the quasi-identifiers of different tuples can be modified to represent the same generalised value. For example, two tuples with M5 8LS and M5 3HF for the value of their Postcode attributes respectively can both be modified to possess a value of M5 *** and contribute to adhering to the rules of a 2-anonymous table. As discussed earlier, carrying out privacy-preserving techniques, such as k-anonymity, can affect the usefulness of data with regards to the results of subsequent data mining processes. By observing how generalisation and suppresion function, it can be seen how the results of data mining could become less useful. The more generalisation that occurs in a table, the more generalised the results of data mining processses could also become. It could be the case that the results become so broad that they ultimately provide little or no meaningful information.

2.3 CLASSIFICATION

A popular data mining technique is classification which involves predicting the pre-defined category that an instance will take for a particular attribute. For example, this could include real-world scenarios such as predicting whether an animal is a mammal or a non-mammal or predicting whether an individual should be approved or rejected a bank loan based on other values pertaining to the particular instance.

In order to classify instances, classification uses the concept of splitting the dataset into a training and a test set. The training set encompasses the majority of the dataset and is used to train and derive the classification model which subsequently decides how to classify each instance. This classification model is then used to classify the test set which contains new and unseen instances to

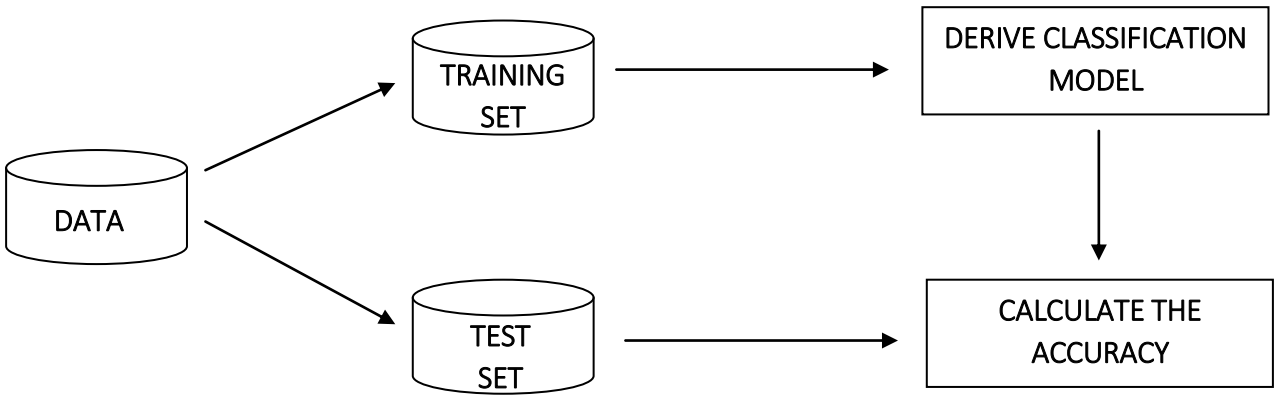

FIGURE 4: A flowchart showing the process of building and testing a classification model

test how accurately the classification model is able to predict values. This process is illustrated in a flowchart in Figure 4.

Table 3 shows an example of a dataset that could be used as a training set in order to classify whether a certain individual should be approved or rejected a bank loan. The attribute that we are trying to classify, Approved/Rejected in this example, is known as the target attribute or the class label. All other attributes are collectively used in order to determine a value for the target attribute based on past instances.

AGE	EMPLOYMENT	CREDIT CARD HOLDER?	DEBT LEVEL	MORTGAGE?	DECISION
54	Full-time	Yes	Low	Yes	Approved
32	Part-time	Yes	Moderate	No	Approved
19	Unemployed	No	High	No	Rejected
28	Full-time	No	Moderate	Yes	Approved
46	Part-time	Yes	Moderate	Yes	Rejected

TABLE 3: An example of a training dataset used to approve or reject bank loans

Table 4 shows a possible test dataset for the same bank loan example where the derived classification model is used to predict whether the individuals in the table should be approved or rejected a bank loan based on the values of their other attributes.

AGE	EMPLOYMENT	CREDIT CARD HOLDER?	DEBT LEVEL	MORTGAGE?	DECISION
49	Full-time	Yes	Low	Yes	?
32	Part-time	No	Moderate	No	?
61	Part-time	Yes	High	Yes	?

TABLE 4: An example of a test dataset to decide whether to approve or reject bank loans

2.3.1 DECISION TREES

Decision trees are a method of classification whereby the tree can be traversed, answering questions about an instance at each stage, until a class label is reached. The decision tree can be depicted as a hierarchical structure consisting of various nodes. The root node of a decision tree is the node at the very top of the structure. From here, different node pathways are followed until the leaf nodes are reached which possess the class value for a particular instance.

Figure 5 shows a possible decision tree for the previous bank loan example that could be used to classify the new test cases in Table 4. Employment is the root node at the top of the tree. Depending on an instance's value for the employment attribute would depend on which route is taken down the tree until a class value is reached. It is important to introduce the idea of decision trees as this concept is used as a means of classification throughout the rest of the project.

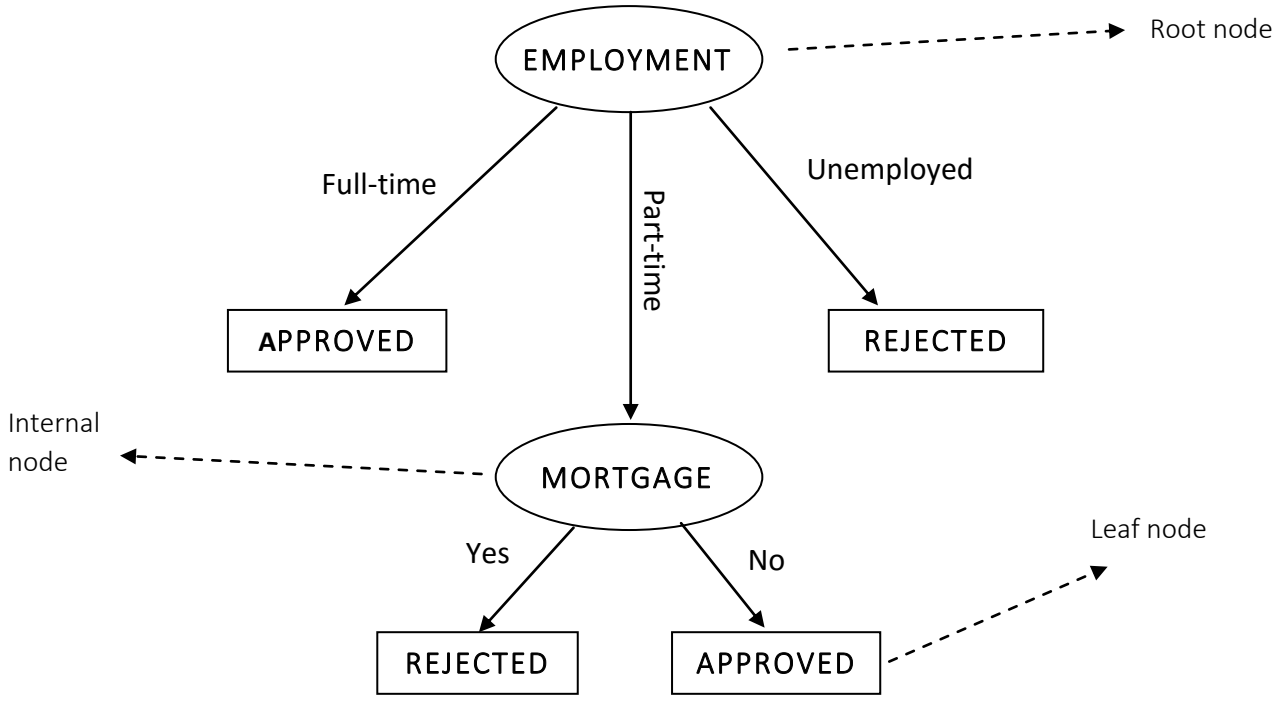

FIGURE 5: A decision tree to illustrate the bank loan classification

2.4 THE IDENTIFIED PROBLEM

As discussed previously, the main problem of privacy-preserving data mining is identified as being the trade-off between privacy and usability. This project will, first and foremost, evaluate the significance of this trade-off and determine whether meaningful results can be produced from the mining of privacy-preserved data. Throughout this project, privacy-preserved values of an attribute

AGE	EMPLOYMENT	NATIONALITY	MARITAL STATUS
54	(Full-time, Part-time)	British	(Cohabiting, Married)
(32-35)	Unemployed	Italian	Divorced

AGE	EMPLOYMENT	NATIONALITY	MARITAL STATUS
54	Employed	British	With Partner
3*	Unemployed	Italian	Divorced

FIGURE 6: Example datasets showing the difference in representation of privacy-preserved values

in a dataset will be represented as a set of possible values, as appears in the topmost table of Figure 6. For example, a value for the Employment attribute could be expressed as (Full-time, Part-time), meaning that the value is one of either 'Full-time' or 'Part-time'. This is a form of generalisation but can be contrasted to the bottommost table in Figure 6 where (Full-time, Part-time) may be displayed as simply 'Employed' and (Cohabiting, Married) can be represented as 'With Partner'.

With this in mind, it can be seen how simple it would be to form a decision tree from the privacy-preserved values as single values, similar to the bottommost table in Figure 6. It becomes more difficult when values are displayed as a set of possible values since there are multiple values to consider. To my knowledge, this has not been explored in the field of privacy-preserving data mining and so this is also a sub-problem that I wish to solve. If done successfully, it can be seen how it could give a higher level of granularity to the results of decision tree classification. Instead of generalising a value to the next level up in a value generalisation hierarchy, a precise set of possible values are deduced thus potentially limiting the possibility of values that an attribute could possess for a particular instance. Moreover, the privacy-preserved values do not necessarily have be to related, meaning that, for the example of the Employment attribute, a possible value set could be (Full-time, Unemployed). Unlike in the typical generalisation approach, this prevents the value from being modified to the maximal element in order to encompass both 'Full-time' and 'Unemployed' as potential values. This means that the dataset can possess a deeper level of granularity while also maintaining the privacy of the individuals to whom the data refers.

2.5 EXISTING SOLUTIONS

There are currently two techniques used to combine k-anonymity with data mining.

1. ANONYMISE-AND-MINE

In this approach, an original table is first anonymised using privacy-preserving techniques such as k-anonymity. The anonymised table is then subjected to various data mining processes. The results of these processes will, by nature, also be anonymised due to the input table being privacy-preserved itself. Decision tree classification would work quite easily in this instance on datasets that adhere to k-anonymity by generalising values into a common single value. Previous work has been conducted on situations such as this whereby datasets anonymised through generalisation have been evaluated for their utility. Inan et al. from the Department of Computer Science at The University of Texas at Dallas Richardson have examined the usefulness of anonymised data for classification of unseen cases. In order to generalise values, they utilised value generalisation hierarchies for given attributes. Furthermore, their work is based on the principle that instead of assuming the probability distribution of the data, statistics are collected during the anonymisation process and these are released alongside the anonymised dataset. By doing this, they found that anonymisation did not, in fact, significantly decrease the classification accuracy [6]. The difference between this existing research and my project lies in the classification method used. Inan et al. used the support vector machine (SVM) method of classification which maps data instances as points in space. During this

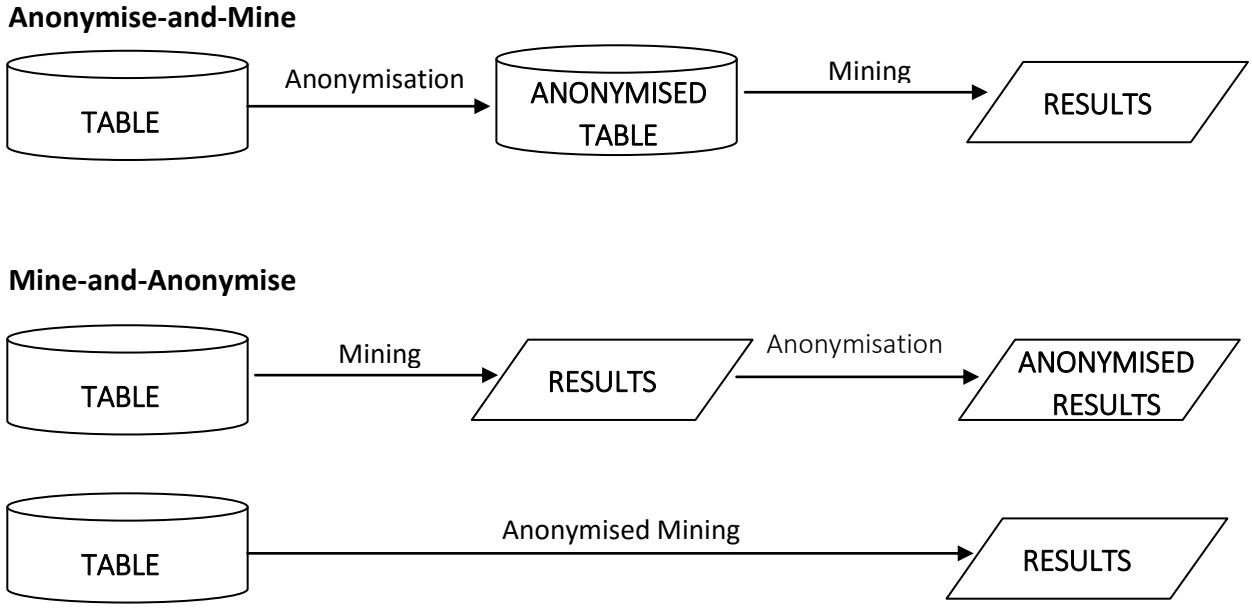

FIGURE 7: Flowchart showing two different approaches to data mining with k-anonymity

project, however, I will be examining decision tree classifiers and their capability of being formed using generalised values as a set of possible values.

The anonymise-and-mine approach has many advantages in the data mining community. One advantage is that data holders, such as hospitals or banks, can subject their original datasets to privacy-preserving methods and then release the anonymised dataset for third parties to analyse. It can be distributed to as many organisations as desired without fear that privacy is sacrificed. These external parties can then conduct their own data mining processes on the anonymised dataset and produce results that are beneficial to them, whatever their data mining goals may be. A disadvantage of the anonymise-and-mine approach is that due to the generalised values in the released dataset, the results of the subsequent data mining processes may be less specific. This could lead to less valuable data mining and may mean that it is not efficient enough for the purposes that the data was intended for.

2. MINE-AND-ANONYMISE

Another approach to combining k-anonymity with data mining is the mine-and-anonymise method. As shown in Figure 7, there are two ways of doing this: either by mining the original table and subsequently anonymising its results or by carrying out anonymised mining in one step. Anonymised mining is a process by which the data mining algorithm used mines the data in a way which satisfies k-anonymity [7]. This is an approach that has been previously investigated with the use of decision tree classifiers in the world of privacy-preserving data mining. However, it requires that the data mining processes be carried out by the data holder themselves in order to protect the privacy of the data. This may not be ideal since different organisations may wish to mine data independently, using specific methods that may be of most benefit to them.

2.6 PROPOSED SOLUTION

In order to determine the extent to which privacy preservation of data truly affects its usefulness, I will implement a comprehensive evaluation method, concluding with quantifiable results which will compare the effects of both privacy-preserved data and non-privacy-preserved data when used to form decision trees to classify data. As part of this, I will also ensure that the chosen decision tree algorithm is able to handle data containing both single values and a set of possible values and compute a functioning decision tree from the given dataset.

Here, I will succinctly summarise the research questions that I set out to answer throughout this project:

1. To what extent, if any, does privacy preservation of data affect its usability with regards to subsequent data mining processes?
2. Is it possible to create a decision tree algorithm that uses data containing privacy-preserved sets of values as input and returns, as output, a functioning decision tree containing only single values?

2.7 FUTURE USES

A large variety of different industries could benefit from the results of this project. Any organisation that collects person-specific data and wishes to share it with external organisations or researchers could find the information found throughout this project to be of use. Before a data holder releases their data, they must ensure that the data is fully privacy preserved in order to protect the identities of the individuals involved. Knowing the extent to which anonymising data could affect the usefulness of data mining processes could be extremely beneficial to data holders.

If privacy preserving greatly decreases the functionality of the data mining processes, data holders may deem it futile to release privacy-preserved data and may instead decide to carry out mine-and-anonymise approaches. If privacy preserving is found to have little or no effect on the functionality of subsequent data mining processes, organisations that are mining the data can be assured knowing that their results will not be affected too greatly.

3. APPROACH

3.1 OVERVIEW OF SOLUTION

The general approach that this project will take is shown in Figure 8.

The project will begin with a dataset designed for classification purposes containing a class attribute for prediction. Using this dataset as training data, a decision tree, D, will be derived which will later be used to classify test data. The dataset will then be anonymised using a chosen privacy-preservation technique to reproduce the original dataset in its anonymised form. The anonymised dataset will then be used as training data to form a second decision tree, D'. Both decision trees, D and D' will be used to classify the same test data in order to predict its class attribute and their classification accuracies will be compared.

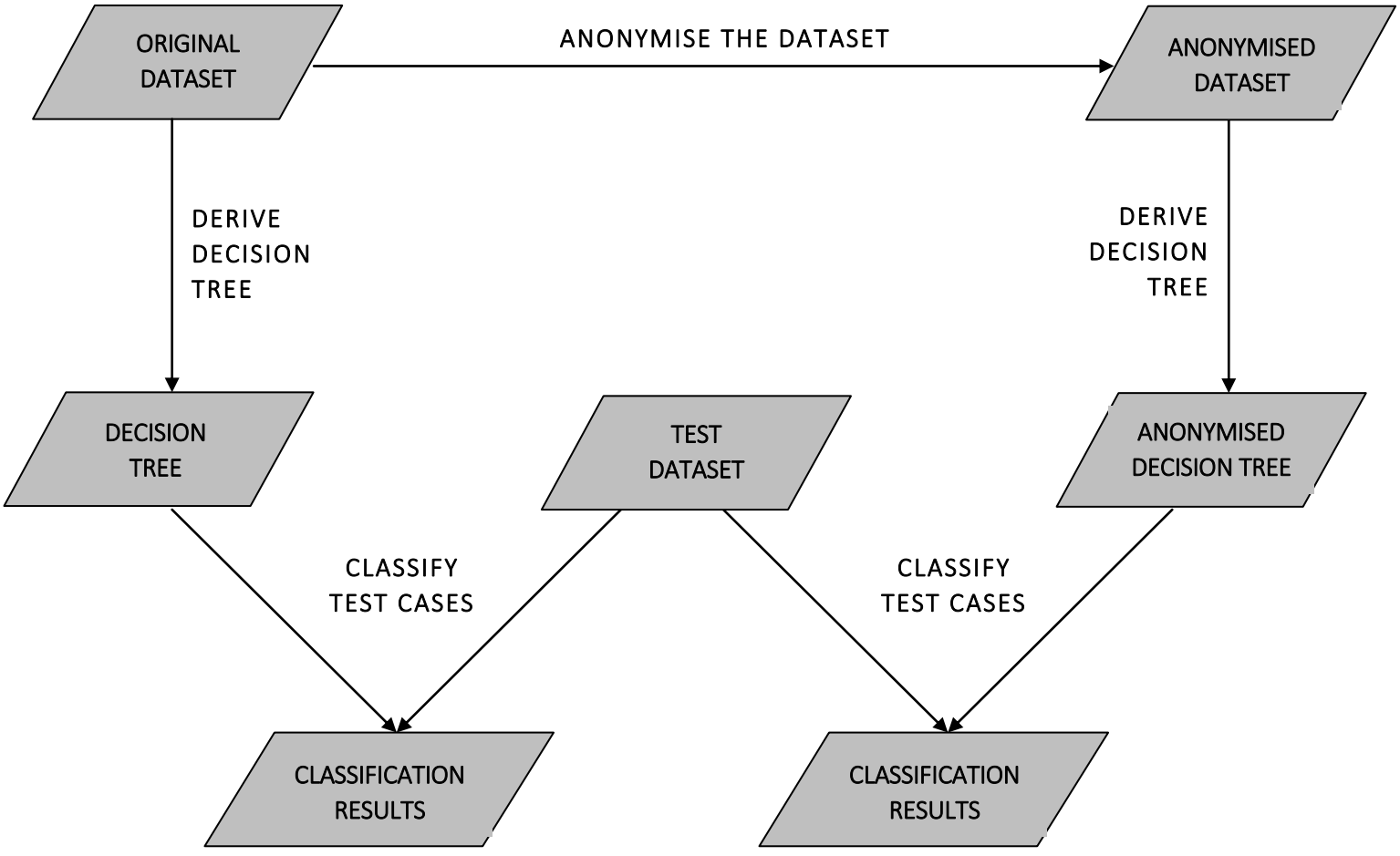

FIGURE 8: A flow diagram showing the approach to the project solution

3.2 DATASET

The data used throughout this project is the Adult dataset from UCI Machine Learning Repository. Its purpose is for classification tasks and it is used widely in work related to k-anonymity. The research work carried out by Inan et al., as mentioned previously, also used the Adult dataset, meaning that the results obtained throughout this project can be compared and contrasted to previous work.

The Adult dataset is also known as "Census Income" dataset as it was extracted from a 1994 Census database [8]. It contains data regarding individuals between the ages of 17 and 90 and is used to predict whether an individual's income is higher or lower than $50,000 a year. The dataset is formed of six numerical fields and eight categorical fields which provides a good variety of attributes to explore when building a decision tree. A full list of the attributes used in Adult dataset along with a description of them can be found in Appendix A.

The Adult dataset is split into training data consisting of 32,561 records and test data consisting of 16,281 records. The class attribute to be predicted is Salary and can take one of two values: <50K and >50K. Throughout this project, a comparison will take place on the ability of decision trees to predict whether an individual has a salary of greater or less than $50,000 when the decision trees are formed from (1) the original Adult dataset and (2) an anonymised Adult dataset.

3.2.1 DATA PREPROCESSING

In the UCI Machine Learning Repository, the Adult dataset appears in a CSV format on the webpage. I copied and pasted the records into a text document and imported it into Microsoft Excel, ensuring to specify both a comma and a space as the delimiters to prevent trailing whitespaces in the values. The Adult dataset, as it comes, has no column headings so I added these manually using the known column headings as specified on the UCI Machine Learning Repository.

In the dataset, missing values are represented as a '?' value. In order to ensure that the data mining that will be carried out is meaningful, I removed all instances with missing values. This reduced the overall number of instances in the Adult dataset from 32,561 to 30,162. Missing values were also removed from the Adult test dataset, leading to a decrease in instances from 16,281 to 15,060. Subsequent to the removal of these values, I saved the dataset as a CSV file which I know to be a very manageable format for use in any later data manipulation and coding.

3.3 PROGRAMMING LANGUAGE

To begin the project, I had to decide on a programming language to use. The requirements for the programming language were that it must be able to handle large datasets and it must provide the functionality in order to construct a decision tree. The options for a potential programming language were narrowed down to Python or Java. Table 5 shows the associated pros and cons of using both languages for this task.

	PYTHON	JAVA
PROS	Multiple libraries which are easy to install	Powerful for use with large-scale projects
	Most familiar language to me and easy to code with	Fast speeds and scalability opportunities
	Used throughout the data science community	
CONS	Not as powerful as other languages such as Java	Not much personal experience in programming with Java

TABLE 5: A table showing the pros and cons of using Python and Java [9].

I decided to use Python as my chosen programming language as I was personally more familiar with it and know that is widely used within the data science community.

3.3.1 PANDAS LIBRARY

The code for this project relies extensively on an open-source Python library called Pandas which is used for data manipulation and data analysis. It provides easy methods for reading data files and storing them in data structures for further manipulation. Some features that I have found useful throughout this project include being able to:

- read and write to CSVs with ease
- query datasets by returning all records containing a particular value for an attribute
- modify values in a dataset based on a given criteria
- slice datasets by attributes

The primary data structure for storing datasets in Pandas is called a DataFrame and will be referred to in the implementation section of this report.

Further information about the Pandas library can be found at http://pandas.pydata.org/.

3.4 PRIVACY PRESERVATION

Many different algorithms have been proposed for enforcing k-anonymity in a dataset such as Datafly and k-Optimise. In this section, a couple of these algorithms will be explored before choosing one which will be implemented throughout this project.

3.4.1 INCOGNITO

Incognito is an algorithm proposed by Kristen LeFevre, David J. DeWitt and Raghu Ramakrishnan that works on the principle of value and domain generalisation hierarchies. The Incognito algorithm is based on a concept called the subset property which states that if a table is k-anonymous with respect to a particular set of quasi-identifiers, it must be also be k-anonymous with respect to the various forms of generalisations of those quasi-identifiers [10]. For example, consider the domain generalisation hierarchy of the attribute Postcode which is shown in Figure 3. Let's also introduce a new attribute, Sex, with a domain generalisation hierarchy consisting of Sex_0 = {Male, Female} and Sex_1 = Person. Imagine that these attributes are featured as the quasi-identifiers in a much larger dataset. Using the subset property, if the dataset is k-anonymous with respect to the multi-attribute generalisation of $\{S_0, P_1\}$, then it is also known that the dataset must be k-anonymous with respect to the generalisation of $\{S_1, P_1\}$.

Incognito produces the 'set of all possible k-anonymous full-domain generalisations' of a table [10]. It does this by creating multi-attribute generalisation lattices for each subset pair of quasi-identifiers. Beginning at the most generalised possibility of pairs of quasi-identifiers, Incognito determines whether a table is k-anonymous with respect to them. If the pairing is found to be k-anonymous at that particular level of generalisation, the subset property is used to also confirm whether further generalisations of the quasi-identifiers are k-anonymous. This means that there is no need to check these further generalisations at the next iteration. This bottom-up breadth-first search of the generalisation lattices is continued.

The aim of Incognito is to find a solution which satisfies k-anonymity at its most minimal generalisation, therefore maintaining the maximum possible usability for the data in subsequent data mining processes.

3.4.2 MONDRIAN

Again proposed by LeFevre, DeWitt and Ramakrishnan, Mondrian is a multidimensional global recoding technique. It works by choosing a dimension (an attribute) and recursively partitioning the dataset via median partitioning until k-anonymity is violated. The quasi-identifiers for all records in a particular partition at the stage previous to when k-anonymity is no longer met are then merged together. The algorithm for Mondrian is shown in Figure 9 [11] and will be explained in more detail throughout this section.

Algorithm 1 *Mondrian*: multidimensional partitioning anonymization

Input: partition P
Output: a set of valid partitions of P

1: **if** no allowable multidimensional cut for P **then**
2: **return** P
3: **else**
4: $dim = choose\_dimension(P)$
5: $fs = frequency\_set(P, dim)$
6: $splitVal = find\_median(fs)$
7: $lhs = \{t \in partition : t.dim \leq splitVal\}$
8: $rhs = \{t \in partition : t.dim > splitVal\}$
9: **return** $partition\_anonymize(rhs) \cup partition\_anonymize(lhs)$
10: **end if**

FIGURE 9: An algorithm for Mondrian

3.4.2.1 MONDRIAN ALGORITHM

The algorithm takes, as input, a partition, P, which in the first instance is the entirety of the dataset. An allowable cut is defined as when further splitting of the partition will result in a left-hand side and a right-hand side of the partition which both satisfy k-anonymity. For example, if k=5, an allowable cut would be a split to a partition resulting in two partitions that both contain at least five records. The steps of the algorithm are shown and explained below.

1. CHOOSE A DIMENSION

An attribute must be chosen which will be used to split the partition. There are many ways to select this dimension although, in literature, the most popular method is to choose the dimension with the widest normalised range of values [12].

2. CALCULATE THE FREQUENCY SET OF THE DIMENSION VALUES

The frequency set for a dimension is an ordered list of the values in a dimension and their associated frequency within the partition. For example, a frequency set for the attribute Race in the Adult dataset could be {Amer-Indian-Eskimo: 286, Asian-Pac-Islander: 895, Black: 2817, Other: 231, White: 25,933} which denotes the frequencies for each value in the dataset. It is this frequency set that will be used to calculate the value on which to split the partition and so the ordering of the values in the frequency set is significant. For numerical fields, the values are displayed in their ascending numerical order. For categorical fields, any ordering can be used but the same ordering must be consistently used for the attribute each time. To make it easier, I chose to use an alphabetical ordering for any categorical values.

3. FIND THE SPLIT VALUE

The split value is the attribute value upon which the partitioning of the dataset will be based. The approach for calculating the split value is to find the value which occurs at the median of the frequency set. Taking the above example of the frequency set {Amer-Indian-Eskimo: 286, Asian-Pac-Islander: 895, Black: 2817, Other: 231, White: 25,933}, the median of the 30,162 records is 15,081 (30, 162 divided by 2). By calculating the sum of the cumulative frequencies in the dataset, shown in Table 6, it can be seen that the 15,081th value occurs within the value 'White'. Thus, given that Race is the chosen dimension, the value 'White' would become the split value for this partition. Throughout this project, when the median of a frequency set has been found to be a decimal number, I have always rounded the number down to the nearest integer.

There are two modes of partitioning for the Mondrian algorithm: strict partitioning and relaxed partitioning. Strict partitioning states that when a partition is split into its left-hand side and right-hand side subsets, they must not contain any overlapping values within the dimension concerned. This means that when the median of a frequency set occurs within a value (i.e. as with the example

VALUE	FREQUENCY	CUMULATIVE FREQUENCY
AMER-INDIAN-ESKIMO	286	286
ASIAN-PAC-ISLANDER	895	1181
BLACK	2817	3998
OTHER	231	4229
WHITE	25933	30162 – median occurs here

TABLE 6: A table showing the calculation of the split value in the Mondrian algorithm.

in Table 6), the value must be taken in its entirety and placed on either the left-hand side or right-hand side of the partition. Relaxed partitioning, however, allows for the left-hand side and right-hand side of a partition to contain the same values. This would mean that for the example in Table 6, the value 'White' could be split into 10,852 instances in the left-hand side of the partition and 15,081 instances on the right-hand side of the partition.

Throughout this project, strict partitioning mode will be used meaning that for the example given, either all 25,933 instances under the value 'White' will be assigned to the left-hand side or all instances will be assigned to the right-hand side.

4. PARTITION THE DATASET

The final step in the algorithm is to split the partition based on the calculated split value and recursively run the algorithm again using the left-hand side and right-hand side of the partition as inputs. The algorithm stops when there are no further allowable cuts for any of the partitions, in which case the quasi-identifier values for the instances contained with each partition are merged.

3.4.2.2 MONDRIAN EXAMPLE

In this section, I will show a short demonstration of how Mondrian works. Consider the non-conforming table from Figure 2. I have added another attribute, Age, to the table and this new table is now depicted in Table 7.

AGE	MARITAL STATUS	NATIONALITY
23	Single	British
49	Married	Italian
37	Married	Spanish
58	Divorced	British
39	Single	Spanish
23	Married	German

TABLE 7: An original non-conforming table before Mondrian has been performed.

Consider that, in this instance, k=2 and the quasi-identifiers are {Age, Nationality}. The first step is to choose a dimension on which to partition the dataset. Imagine that Age has been chosen as the first dimension. The frequency set of the Age attribute would be {23: 2, 37: 1, 39: 1, 49: 1, 58: 1}. The median would be 6 divided by 2, equalling 3. This means that the split value for the dimension Age

LHS

AGE	MARITAL STATUS	NATIONALITY
23	Single	British
37	Married	Spanish
23	Married	German

RHS

AGE	MARITAL STATUS	NATIONALITY
49	Married	Italian
58	Divorced	British
39	Single	Spanish

FIGURE 10: The left-hand side and right-hand side instances of the first recursion call of the Mondrian example

in this case would be the value '37'. The left-hand side of the partition would contain instances with either a value of '23' or a value of '37' for the Age attribute. The right-hand side of the partition would contain instances with a value of '39', '49' or '58' for the Age attribute. Both the left-hand side and the right-hand side partitions are shown in Figure 10.

Since splitting the partition in this way is an allowable cut (i.e. both the left-hand side and right-hand side of the partition do not violate k-anonymity where k=2), the algorithm continues recursively on both sides of the partition.

Firstly, take the left-hand side of the partition (the topmost table in Figure 10). A new dimension is chosen on which to split the partition. Imagine that Nationality was chosen as the next attribute. The frequency set for the Nationality attribute in this case would be {British: 1, German: 1, Spanish: 1}. The median would be 3 divided by 2, equalling 1.5 but the median value is always rounded down to the nearest integer, thus equalling 1. This means that the split value for this partition would be 'British'. The left-hand side of the partition would contain instances with a value of 'British' for the Nationality attribute and the right-hand side of the partition would contain instances with a value of either 'German' or 'Spanish' for the same attribute. Both the left-hand side and right-hand side of the partitions are shown in Figure 11.

LHS

AGE	MARITAL STATUS	NATIONALITY
23	Single	British

RHS

Age	Marital Status	Nationality
37	Married	Spanish
23	Married	German

FIGURE 11: The resulting split of the LHS partition from the first recursive call of the Mondrian example

Note that if the partition was split in this way, the left-hand side of the partition would not satisfy the k-anonymity requirements. Therefore, the splitting of the partition in this way is not an allowable cut and would not be performed. Hence, we revert back to the previous stage shown by the topmost table in Figure 10 and the values of the quasi-identifier attributes for each tuple are merged. Any numeric attributes are displayed as a range and any categorical values are displayed as a set of possible values. The resulting table is shown in Table 8.

Age	Marital Status	Nationality
(23-37)	Single	(British, German, Spanish)
(23-37)	Married	(British, German Spanish)
(23-37)	Married	(British, German, Spanish)

TABLE 8: A table showing a partial k-anonymous table

Next, take a look at the right-hand side partition of the first iteration of Mondrian as shown by the bottommost table in Figure 10. Again, imagine that Nationality has been selected as the attribute on which partition the dataset. The frequency set for the Nationality attribute in this case would be {British: 1, Italian: 1, Spanish: 1}. Thus, making the split value for this partition, again, 'British'. The left-

LHS

AGE	MARITAL STATUS	NATIONALITY
58	Divorced	British

RHS

AGE	MARITAL STATUS	NATIONALITY
49	Married	Italian
39	Single	Spanish

FIGURE 12: The resulting split of the RHS partition from the first recursive call of the Mondrian example

hand side of the partition would contain instances with a value of 'Italian' and the right-hand side of the partition would contain instances with a value of either 'German' or 'Spanish'. Both sides of this partition are shown in Figure 12.

Again, if the partition was split in this way, the LHS shown in Figure 12 would not satisfy k-

AGE	MARITAL STATUS	NATIONALITY
(39-58)	Divorced	(British, Italian, Spanish)
(39-58)	Married	(British, Italian, Spanish)
(39-58)	Single	(British, Italian, Spanish)

TABLE 9: Another table showing a partial k-anonymous table

anonymity. Therefore, the split cannot be performed and we revert back to the previous stage, merging the quasi-identifier values as shown in Table 9.

As there are no further allowable cuts in any of the partitions, the resultant generalised tables are concatenated together to form the completed anonymous table which satisfies k-anonymity when k=2. This completed table is shown in Table 10 below.

AGE	MARITAL STATUS	NATIONALITY
(23-37)	Single	(British, German, Spanish)
(23-37)	Married	(British, German Spanish)
(23-37)	Married	(British, German, Spanish)
(39-58)	Divorced	(British, Italian, Spanish)
(39-58)	Married	(British, Italian, Spanish)
(39-58)	Single	(British, Italian, Spanish)

TABLE 10: The completed k-anonymous table after Mondrian being performed

3.5 DECISION TREE CLASSIFIER

The chosen method of implementing a decision tree for this project is ID3. This is an algorithm created by Ross Quinlan for the purpose of creating decision trees from a given dataset. It operates on a concept known as entropy which calculates how much further information is needed to classify an instance based on the homogeneity of the attribute values in a dataset. This section will explain this concept and also give further insight into the ID3 algorithm and how it works.

3.5.1 ENTROPY

As mentioned, entropy is a measurement of how much information is needed in order to classify an instance. For example, consider the two tables in Figure 13 which represent simplified versions of person-specific data such as the Adult dataset. In Table A, all tuples with a value of 'Female' for the Sex attribute have a salary of less than or equal to $50K. Entropy asks, 'Given this information, how much more information do I need in order to classify a new instance with the same value?' Entropy can be measured on a continuous scale from 0 to 1 where 0 represents a case where no further

Table A

SEX	SALARY
Female	<=50K
Male	>50K
Female	<=50K
Male	>50K

ENTROPY = 0

Table B

SEX	SALARY
Female	<=50K
Male	>50K
Male	>50K
Female	>50K

ENTROPY = 1

FIGURE 13: Two tables to demonstrate the significance of entropy

information is needed and 1 represents a case whereby the maximum information possible is needed. In this way, it can be seen that in order to classify a new instance that is also female based on Table A, we need no further information to be able to predict that the salary is less than or equal to $50K. The entropy value for a situation such as this is 0. Contrast this to Table B where 50% of the females earn a salary over $50K and 50% of the females earn a salary less than or equal to $50K. If we were given a new instance that also possesses the value 'Female' and were required to classify its salary based on Table B, we would need a lot more information about the instance in order to do this since there is an equal probability of it being classified in either class label. In a situation such as this, the entropy value is 1.

The calculation for entropy is as follows:

$$Entropy(S) = -\sum_{i=1}^{k} p(i|S) \log_2 p(i|S)$$

where p(i|S) is the fraction of records in S with class i

Using Table B from Figure 13 as an example, the calculation of the entropy for the 'Female' value is show below:

$$Entropy(Female) = -\frac{1}{2}\log_2\frac{1}{2} - \frac{1}{2}\log_2\frac{1}{2} = 1$$

In order to calculate the entropy for an attribute as a whole, the entropies of the attribute's values are multiplied by the probability of a tuple in the dataset possessing that value and these entropies are then added together. For example, to calculate the entropy for the Sex attribute in Table B, we first need to calculate the entropy for the 'Male' value:

$$Entropy(Male) = -\frac{2}{2}\log_2\frac{2}{2} - \frac{0}{2}\log_2\frac{0}{2} = 0$$

The entropy of the Sex attribute as a whole can be calculated as:

$$Entropy(Sex) = \frac{2}{4} \times Entropy(Female) + \frac{2}{4} \times Entropy(Male)$$

$$= \left(\frac{2}{4} \times 1\right) + \left(\frac{2}{4} \times 0\right)$$

$$= 0.5$$

3.5.2 ID3 ALGORITHM

With a deeper understanding of entropy and its calculations, we can now take a look at the ID3 algorithm itself. This is shown in Figure 14. A walkthrough of the algorithm is detailed in this section with an explanation of its various components.

In order to fully understand the algorithm, consider the new table below, Table 11.

MARITAL STATUS	NATIONALITY	SALARY
Single	British	>50K
Married	Italian	>50K
Married	Spanish	<=50K
Divorced	British	<=50K
Single	Spanish	>50K
Married	German	<=50K

TABLE 11: An example table upon which the ID3 algorithm will be performed

1. CHOOSE A BEST ATTRIBUTE

The first step is to choose the best attribute for which to begin building the tree. In the first iteration of the algorithm, the selected attribute will become the root of the tree. In order to choose a best

Given a dataset D with attributes $A = \{A_1, A_2, \ldots, A_n\}$ and class values $C = \{C_1, C_2, \ldots, C_m\}$

Algorithm DT-Learning (A, D):

Choose a best A_i from A as the root

Partition D into $\{D_1, D_2, \ldots, D_k\}$ by A

Make each D_j a child node of A_i

FOR each D_j **DO**

IF D_j covers C_s well enough **THEN**

Label D_j with C

ELSE DT-Learning(A-$\{A_i\}$, D_j)

FIGURE 14: Pseudocode for the ID3 algorithm [13]

attribute, the entropies for each of the attributes must be calculated. The attribute with the minimum entropy (i.e. the least information required) is chosen as the best attribute. In the example of Table 11, when calculated, the entropy of the attribute 'Marital Status' is 0.459 and the entropy for the attribute 'Nationality' is 0.667. Here, 'Marital Status' would be selected as the best attribute, and therefore the root of the tree, since we require the least information to be able to classify further instances.

2. SPLIT THE DATASET BY BEST ATTRIBUTE VALUES

The next step is to split the dataset into subsets based on the values of the selected best attribute. Continuing the example using Table 11, as 'Marital Status' was selected as the best attribute, the dataset is then split into three datasets for each of its possible values: 'Single', 'Married' and 'Divorced'. Each of these values also becomes a child node for the best attribute, 'Marital Status'. The subsets of the data are shown below in Figure 15.

3. DETERMINE IF THE DATASET COVERS THE CLASS LABEL WELL ENOUGH

To determine this, it is required that all records in the dataset belong to the same class label. For example, the table representing 'Single' in Figure 15 covers the class label well enough as all tuples possess the same value for the Salary attribute. Likewise for the table representing the value of 'Divorced' as there exists only one tuple. Therefore, these datasets are labelled with their associated class value. All values of 'Single' belong to the '>50K' class label and all values of 'Divorced' belong to the '<=50K' class label. Since the table representing the 'Married' value is not yet fully classified (i.e. there are more than one class value), the ID3 algorithm must be ran again using this dataset as input until it is fully classified.

MARITAL STATUS	NATIONALITY	SALARY
Single	British	>50K
Single	Spanish	>50K

MARITAL STATUS	NATIONALITY	SALARY
Married	Italian	>50K
Married	Spanish	<=50K
Married	German	<=50K

MARITAL STATUS	NATIONALITY	SALARY
Divorced	British	<=50K

FIGURE 15: The result of splitting the dataset by best attribute values during the ID3 algorithm

4. REPEAT THE ALGORITHM ON NECESSARY DATASETS UNTIL ALL INSTANCES ARE FULLY CLASSIFIED

Now running the algorithm on all instances containing the 'Married' value for the Marital Status attribute, a new best attribute is selected. Since only Nationality remains, it is selected as the best attribute and the dataset is partitioned into 'Italian', 'Spanish' and 'German' values. It can be seen that there only exist one instance for each value, meaning that the datasets cover the class values well enough and they are labelled with their respective values. The completed decision tree is shown in Figure 16.

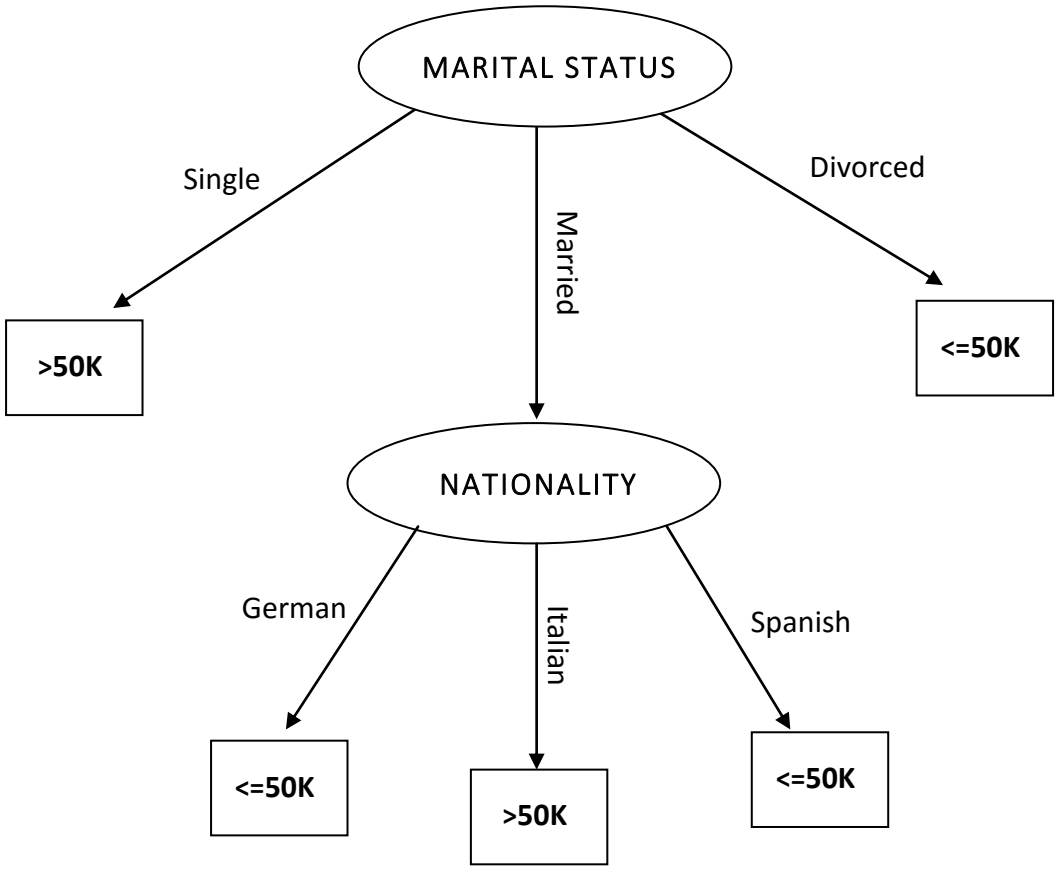

FIGURE 16: A decision tree produced from the ID3 algorithm

3.5.3 VARIATIONS OF THE ID3 ALGORITHM

The ID3 algorithm is not designed to handle continuous attributes. Due to this, it limits the amount of attributes available to build a decision tree where a more efficient decision tree may be able to have been created using the continuous attributes.

J48 is an algorithm that provides an extension to ID3. Unlike the ID3 algorithm, J48 is able to handle continuous attributes and missing values [14]. In this way, it is thought to be an improvement on the ID3 algorithm, leading to greater classification accuracy.

Although other decision tree algorithms exist which may produce greater classification accuracy than ID3, I have chosen to implement ID3 in this project as it is a much simpler algorithm. If the results of this project are significant, it can easily be extended to use more advanced algorithms.

3.5.4 MODIFICATION OF ID3 ALGORITHM

3.5.4.1 CALCULATING ENTROPY

When working with privacy-preserved values, the question is posed of how do we accurately calculate the entropy of attributes? As the values are represented as a set of possible values, there is no way of knowing which exact value the privacy-preserved representation corresponds to. Therefore, it is necessary to devise a method which will take this into account and result in an accurate interpretation of the entropy for a given attribute. Below, three different possible methods for doing so are detailed and evaluated with regards to Table 12, one of which will be selected for implementation.

SEX	SALARY
(M, F)	<=50K
F	<=50K
F	>50K
F	<=50K
M	>50K
(M, F)	<=50K
F	<=50K
(M, F)	>50K

TABLE 12: A example table showing a simplified representation of the Adult dataset

1. INCLUDE PRIVACY-PRESERVED VALUES AS A NEW VALUE LABEL

Consider the privacy-preserved value '(M, F)' representing that the individual is either male or female. Each time it appears in the dataset, instead of attempting to predict whether a particular instance possesses the 'M' or 'F' value, '(M, F)' can be used to represent an entirely new value in itself. In this way, the entropy value of the Sex attribute is not compromised as we are only using the values that we definitely know to compute it instead of guessing whether a privacy-preserved instance is male or female.

Using this method to compute the entropy for the Sex attribute, we obtain the results shown in Figure 17.

$$Sex_F = -\frac{1}{4}\log 2\left(\frac{1}{4}\right) - \frac{3}{4}\log 2\left(\frac{3}{4}\right) = 0.811(3\ sf)$$

$$Sex_M = 0$$

$$Sex_{(M,F)} = -\frac{1}{3}\log 2\left(\frac{1}{3}\right) - \frac{2}{3}\log 2\left(\frac{2}{3}\right) = 0.918(3\ sf)$$

$$E_{Sex} = \frac{4}{8}(Sex_F) + \frac{1}{8}(Sex_M) + \frac{3}{8}\left(Sex_{(M,F)}\right) = 0.75$$

FIGURE 17: The calculation of the Sex attribute's entropy

With this method, another value label would also be introduced in the decision tree meaning that a node containing the Sex attribute could then possibly be split into three different pathways: 'M', 'F', and '(M, F)'. With regards to subsequent classification using the tree, this would pose no problem for datasets that have been privacy preserved in the same way. However, when attempting to use the tree to classify non-anonymous datasets, further work would need to be done since '(M, F)' would not exist as a value in the dataset. This demonstrates how including privacy-preserved values as a new value label solves the issue of accurately representing an attribute's entropy but may lead to problems in the future with subsequent classification.

2. CONVERT THE DATASET INTO SINGLE VALUES

Another method for representing an attribute's entropy is to convert the privacy-preserved values into a dataset containing only single values. For example, taking Table 12, the three '(M, F)' values would be modified to become either an 'M' or an 'F' value. This can be done haphazardly with 'M' or 'F' being selected at random. Once the dataset has been converted, the entropy of the attribute can be calculated in the normal way.

While this presents a solution to the problem and eradicates the problems discussed in the first proposed method, it also brings with it a number of flaws. It could be said that the calculation of the entropy is not accurate when calculated in this way since many assumptions of the values are being made. For example, consider every privacy-preserved value of '(M, F)'. There is a 50% chance that the randomly-selected value will be the value that the particular instance possessed before the anonymisation process. For a large dataset containing many privacy-preserved values, there could be many incorrect assumptions made about the values, thereby skewing the resulting entropies for the attributes.

Furthermore, for a large dataset, it could become computationally intensive to create a new dataset which maps all privacy-preserved values to one of its possible values for every attribute. On reflection, although this method does propose an alternative way to calculate an attribute's entropy, it lacks accuracy and thus could greatly skew the implementation of the decision tree.

3. CALCULATE THE PROBABILITY FOR PRIVACY-PRESERVED VALUES

For this method, the likelihood that a privacy-preserved value possesses each of its possible values is calculated and factored into the process of calculating an attribute's entropy. Using the example from Table 12, a value of '(M, F)' indicates that there is a possibility of 0.5 that the value will be 'M' and a possibility of 0.5 that the value will be 'F'. Likewise, for a privacy-preserved value represented as a set of three possible values, there is a possibility of $\frac{1}{3}$ that each of the values will correspond to the instance's original value. Using this concept, we can say that every privacy-preserved value in Table 12 represents 0.5 of an 'M' value and 0.5 of an 'F' value. When totalling the number of each value in the dataset, we now see that the total number of 'F' values is 5.5 and the total number of 'M' values is 2.5. This totals eight which is the total number of instances in the dataset, meaning that the

$$Sex_F = -\frac{1.5}{5.5}\log 2\left(\frac{1.5}{5.5}\right) - \frac{4}{5.5}\log 2\left(\frac{4}{5.5}\right) = 0.845\ (3\ sf)$$

$$Sex_M = -\frac{1.5}{2.5}\log 2\left(\frac{1.5}{2.5}\right) - \frac{1}{2.5}\log 2\left(\frac{1}{2.5}\right) = 0.971\ (3\ sf)$$

$$E_{Sex} = \frac{5.5}{8}(Sex_F) + \frac{2.5}{8}(Sex_M) = 0.885\ (3sf)$$

FIGURE 18: Another calculation of the Sex attribute's entropy using a different method

original length of the dataset is preserved. Maintaining this concept, the result of calculating the entropy for the Sex attribute is shown in Figure 18.

It can be observed that the information given in the dataset is preserved since the number of 'F' values and the number of 'M' values calculated total the number of tuples in the dataset. Furthermore, in each equation calculating the value's entropy, the number of instances possessing a positive class value plus the number of instances possessing a negative class value equals the total number of instances for that value. In this way, the integrity of the dataset has been maintained. Not only this but also the calculated entropy value for the attribute can be deemed to be more accurate than the previous method since the likelihood that each privacy-preserved value is any of its possible values is taken into account.

Since this final method presents itself as the most accurate method of calculating an attribute's entropy, I will be using this approach during the implementation phase. By calculating a probability for each privacy-preserved value, the resulting entropy poses as a true reflection of how much information is needed for classification to continue.

3.5.4.2 CONVERTING A PRIVACY-PRESERVED VALUE

Since method 1 was not selected in the previous section as a means of calculating an attribute's entropy, we have determined that we do not wish for a privacy-preserved label such as '(M, F)' to appear in the decision tree. In this way, the final decision tree will exist of solely single values which will make the classification process a lot easier.

In order to design a decision tree such as this, it is necessary to map the privacy-preserved values to a single value which can be used throughout the creation of the decision tree. There are three methods of doing this that I considered throughout this project which will be detailed below.

1. RANDOM SELECTION

   Out of the set of possible values, one value will selected at random to become the instance's new value for the given attribute for the remainder of the tree creation. This would mean that each time the algorithm is run, different values could be assigned to each instance and this may be reflected in the final decision tree.

2. HIGHEST FREQUENCY

   Out of a generalised set of possible values, the value that occurs the most throughout the dataset is always selected to become the new value. This method would result in some values that never get selected, meaning that their representation in the decision tree may be skewed.

3. REPLICATE THE DISTRIBUTION IN THE DATASET

   The distribution of the possible values in the original dataset is calculated and this is replicated amongst the assignment of new values. For example, a distribution of 45% males and 55% females in the original dataset would have to also be reflected once the privacy-preserved values are mapped to a single value. This method would result in more work being carried out in order to calculate and replicate the correct distribution. Furthermore, there would be no guarantee that the distribution of values in the original dataset is mapped back to the correct instances. Therefore, the relationship between these values and the other attributes in an instance could also be skewed and this could be reflected in a decision tree which produces lower classification accuracies.

After having evaluated each of the three proposed methods above, I decided to implement method 1 which assigns a value to an instance at random. This could result in a slightly different decision tree being created each time that the algorithm is run on a given dataset. However, I believe that it is the best method to use in order to reflect the most natural relationship between the attributes of an instance since each instance has a fair chance of being mapped back to the values that it had before the anonymisation process.

3.6 MEASURING THE RESULTS

The key question that the results should answer is 'To what extent does privacy-preservation affect the effectiveness of decision tree classification?' In order to answer this, the results of decision tree classification using a decision tree formed from non-anonymised data should be compared with the results of the same classification using a decision tree formed from privacy-preserved data. There are numerous ways in which the results of the classification process can be measured and evaluated. This section will explore the possible options and conclude with the selection of one such approach to be implemented further in the project.

3.6.1 VERIFICATION OF IMPLEMENTATION

In this context, verification refers to confirming the accuracy of the decision tree that has been produced. I will propose two possible methods for doing so.

1. TRAINING DATA

A method that can be used to check whether the decision tree is performing as expected is to use it to classify the data that it was trained with. If a low accuracy is obtained, this could indicate an issue with the efficacy of the decision tree. Likewise, if a very high accuracy is obtained using the training data while a poor accuracy is shown using the test data, it could suggest a problem of model overfitting. This occurs when the decision tree has been implemented too tightly to the training data. As a result, when attempting to classify unseen test cases, the classifier performs poorly as the instances may differ from the training data while still being valid members of a particular class label. By comparing the classification accuracy obtained by the decision tree using the training data with the accuracy obtained using the test data, any issues regarding the effectiveness of the decision tree may be highlighted.

2. WEKA

Weka is a Java-based open-source suite of data mining algorithms that can be applied to a given dataset. In order to verify our decision tree, we can replicate the decision tree implementation and test data classification in Weka and compare the results of this with the results obtained by our

algorithm. This should give a clearer idea of whether the classification results are as expected. Further information about Weka can be found at http://www.cs.waikato.ac.nz/ml/weka.

3.6.2 USING TEST DATA

As mentioned previously, the Adult dataset used throughout this project comes with a test data set in order to calculate the classification accuracy of an implemented decision tree. This will be, first and foremost, the baseline used to measure the accuracy of the decision trees in this project. It provides a simple method of testing the decision tree that has been produced against published data. A problem that could occur when using test data as a means of estimating the effectiveness of a decision tree is that the test data may not be representative of the general population of which the training data was formed from. If the test data consists of many outliers that do not follow the general trend of the population that is present in the training data, it could return a low classification accuracy which could skew the efficacy of the decision tree. However, it is assumed that this would not be the case with regards to test data.

3.6.3 K-FOLD CROSS VALIDATION

K-fold cross validation is an iterative process whereby the original dataset is split into k parts which are known as folds. A common value for k is 10 whereby the dataset is split into ten parts. At each iteration, one fold is labelled as the training data while the remaining folds collectively become the test data. The classification accuracy is tested at each iteration and the recorded accuracies are averaged together in order to produce a resulting value for the classification accuracy.

In the context of this project, cross validation can be used as another means to calculate the classification accuracy of the produced decision tree. The steps for doing so using 10-fold cross validation are shown below:

1. Anonymise the training set (10% of entire dataset) using the Mondrian algorithm
2. Derive a decision tree from the resulting privacy-preserved dataset
3. Use this decision tree to classify the instances of the test set (90% of entire dataset) and record its accuracy

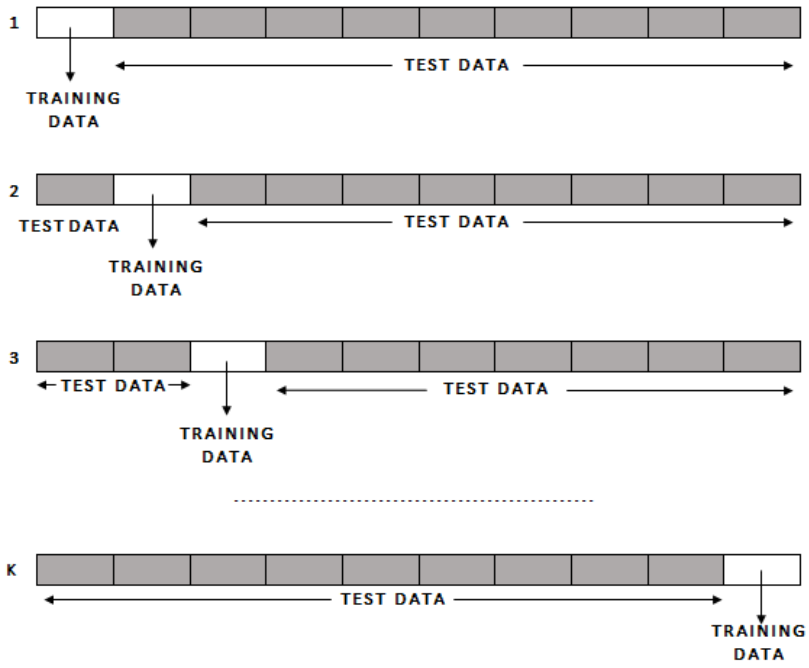

FIGURE 19: A diagram showing the process of k-fold cross validation

4. Repeat steps 1-4 for a total of 10 times and calculate the average of the recorded accuracies

Cross validation provides advantages when there is generally not enough data to be able to partition it into a training set and a test set which are both of ample size. By using cross validation in this situation, the data can be recycled to be used in various different train and test situations where each instance is only ever used once in a given test set.

3.6.4 CHOSEN APPROACH

I have chosen to use the Adult test data in order to measure the classification accuracy of the decision trees in this project. I strongly considered using cross-validation as a measure as I believed that it would have provided slightly better accuracy measures. However, once testing began, I found that the accuracy results produced by the cross validation method were approximately equal to the results produced from using the test data alone. In addition to this, the process of carrying out cross validation was very cumbersome. As the values of k increased, so did the length of time it took to complete cross validation with a fold number of 10. Due to similar results being produced by using the test data, I thought that it would be more efficient to solely use the test data to measure the classification accuracy as it took a considerably shorter length of time.

4. IMPLEMENTATION

This section will detail the implementation of the code in order to create two versions of the ID3 algorithm as well as a Mondrian algorithm for enforcing k-anonymity. It will also explore some of the challenges that were faced throughout this process and the steps that I took to resolve them.

4.1 ID3 ALGORITHM

--------- - **TREE.PY**

The chosen implementation of the ID3 algorithm in this project is loosely based on a version published by Christopher Roach [15]. It can be found in the file *tree.py* and consists of two main sections of code: (1) the *createTree* function and (2) the *selectAttribute* and *calcEntropy* function.

The way in which the decision tree is represented is through the use of a dictionary. I have found this to be an intuitive method of doing so since the dictionaries can be nested to represent subtrees. The dictionary keys represent the attribute and the dictionary values represents either the attribute value if it is a leaf node or otherwise a subtree. For example, the decision tree shown in Figure 16 would be represented as follows: *{'Marital Status': {'Single': '>50K', 'Married': {'Nationality': {'Italian': '>50K', 'Spanish': '<=50K', 'German': '<=50K'}}, 'Divorced': '<=50K'}}*. The decision tree can also be traversed recursively, as shown in section 4.42, in order to classify a particular instance.

```
def selectAttribute(dataframe, attributes, sensitive_attr, pos_val):

        entropies = {}

        for a in attributes:

                if a != sensitive_attr and isNumeric(dataframe, a) == False:
                        entropies[a] = calcEntropy(a, dataframe, sensitive_attr, pos_val)

        return min(entropies, key=entropies.get)
```

FIGURE 20: The selectAttribute function for the ID3 algorithm

4.1.1 SELECT ATTRIBUTE

The *selectAttribute* function, shown in Figure 20, is designed to calculate the entropies of the given attributes and return the attribute with the minimum entropy (i.e. the best attribute). The function only checks the entropies for those attributes that do not contain numeric values nor are the sensitive attribute that is to be classified. This information is obtained from a list at the beginning of the code which specifies the names of the numerical columns and an argument given when the code is ran which specifies the name of the sensitive attribute.

A separate function called *calcEntropy* is used to calculate the entropies of the given attributes using the equation specified in section 3.5.1. By using Python's math module, the logarithmic operator can be used in order to accurately calculate the entropies.

The *selectAttribute* function is then called in the main *createTree* function in order to select the best attribute to continue building the tree.

4.1.2 CREATE TREE

The *createTree* function is the main recursive function that ultimately returns the completed decision tree. It takes, as input, a DataFrame object, a list of attributes and two strings which specify the column name and the positive value of the sensitive attribute respectively. In the Adult dataset, the sensitive attribute would be Salary and the positive value of this attribute would be '>50K'.

To begin *createTree*, the function removes all attributes from the given list that contain numerical values. As previously mentioned, ID3 does not handle continuous values and so they must be removed in order to prevent the algorithm from attempting to select them as the best attribute. The variable, *classificationVals,* stores the possible values for the sensitive attribute.

As *createTree* is a recursive function, it is necessary to implement base cases whereby the function will cease execution of the current iteration if the base cases are met. For this algorithm, the possible base cases are as follows: (1) the length of the DataFrame in the current iteration is equal to zero, (2) there are no further attributes left to classify the instances and (3) all the instances in the given DataFrame possess the same value for the sensitive attribute.

A function named *majority*, is called within the base cases of the *createTree* function. The *majority* function takes in a given DataFrame and returns the value of the sensitive attribute for which there exists the most instances. The possible base cases are explored in further detail below.

1. THE DATAFRAME IS EMPTY

This particular base case occurs when the given DataFrame subset contains no instances and is empty. This may occur if there are no instances that match the specified value for the current best attribute. In this case, the algorithm returns the majority value of the sensitive attribute for the parent set. This is stored in a variable called *parentClass* which is called in the recursive portion of the function. As it is a global variable, the value stored in it from the previous iteration can be maintained and returned in this base case.

2. THERE ARE NO ATTRIBUTES REMAINING

When an attribute has been selected as the best attribute in an iteration of the createTree function, it is removed from the list of attributes for all further iterations until a base case has been reached. This particular base case stores the result of calling the *majority* function on the current DataFrame in a variable called *default*. The value stored in *default* is then returned as a leaf node in the decision tree if this base case is reached. This base case determines when there are no remaining attributes to classify with by calculating how many attributes are remaining after the best attributes have been removed. This is then subtracted by 1 to take into account the sensitive attribute which should be the last attribute remaining. If the value is equal to zero, this represents that there are no further attributes on which to classify the data with.

3. ALL INSTANCES HAVE THE SAME CLASSIFICATION

This base case is reached when all the instances in the current DataFrame object possess the same value for the sensitive attribute. This signifies that the algorithm can be stopped on this iteration since the instances all belong to the same class value and can no longer be further split. At this point, the class value that the instances belong to is returned as a leaf node in the decision tree. '*classificationVals[0]*' can be used to return this value since only one possible value for the sensitive attribute exists in the dataset.

```
subtree = createTree(val_df, [attr for attr in attributes if attr != best], sensitive_attr,
pos_val)

tree[best][val] = subtree
```

FIGURE 21: The recursive call in the ID3 algorithm

The 'else' portion of the *createTree* function contains the main part of the algorithm whereby the recursive call is made. To begin with, a best attribute is selected using the *selectAttribute* function, as explained earlier, and is stored in a variable called *best*. The initial tree is created with the best attribute set as the root node and an empty dictionary set as its subtree. The algorithm then loops through each possible value of the best attribute. For each value, a new DataFrame, *val_df*, is created which contains all instances that possess the current value. This is then where the *parentClass* variable is set in case the next iteration is an empty DataFrame and needs to return the majority value of the sensitive attribute for its parent set.

The recursive call in the ID3 algorithm is shown in Figure 21 along with the line below which shows how the completed tree is formed. The *createTree* function is called on the DataFrame containing the instances that possess the current value. The code '*[attr for attr in attributes if attr != best]*' is a list comprehension which specifies that the attributes for the next iteration will be the same attributes except the attribute that was just used as the best attribute. Again, the sensitive attribute and its positive value are also passed as arguments which remain the same throughout the function.

The code '*tree[best][val]*' adds to the previous line of code '*tree = {best: {}}*' and sets the dictionary value as the already-generated subtree. This completes the algorithm and a decision tree represented in the form of a dictionary is returned.

4.2 MODIFIED ID3 ALGORITHM

--------- - PRIVACYTREE.PY

As previously discussed, it is necessary to manipulate the ID3 algorithm so that it is able to process privacy-preserved values and create a decision tree. I have named the concept of the resulting decision tree a privacy tree and to create it, I took the existing ID3 implementation from *tree.py* and made slight adjustments. These will be explained in more detail throughout this section.

4.2.1 PRIVACY TREE

Here, I will present the overall concept for the creation of a privacy tree before the various components of the algorithm are delved into.

The implementation of a privacy tree works in a similar way to the implementation of the original ID3 algorithm. There are two main differences: (1) the calculation of attribute entropy and (2) the introduction of an additional step whereby each privacy-preserved set of values is assigned a single value. The latter stage is put in place for the purpose of creating a decision tree where privacy-preserved values do not take the form of labels within the final tree.

MASTER_DF

NATIONALITY
(British, German, Spanish)
(British, German Spanish)
(British, German, Spanish)
German
Italian
(British, Italian, Spanish)
(British, Italian, Spanish)
(British, Italian, Spanish)

VAL_DF

NATIONALITY
(British, German, Spanish)
(British, German Spanish)
(British, German, Spanish)
German

PRIVACYRANDOM

NATIONALITY
British
British
Spanish
German

FIGURE 22: The tables produced throughout the process of assigning a privacy-preserved value a single value

Consider Table 10 which shows an anonymised table after the Mondrian algorithm has been performed. I have added a few extra instances which will help to better demonstrate this concept. Imagine that Nationality has been selected as the best attribute in the ID3 algorithm. The leftmost table in Figure 22 illustrates a simplified representation of this whereby we are solely focusing on the Nationality attribute. Consider the situation in which the ID3 algorithm is looping through the possible values of the Nationality attribute and it is currently the turn of the value 'German' to be evaluated. As explained in section 4.1, a new DataFrame called *val_df* is created which contains all instances of the dataset that possess the value 'German'. In the case of the privacy tree, this equates to all instances in the dataset that possess the value 'German' alone or contain the value 'German' in a generalised form. The middle table in Figure 22 illustrates the resultant DataFrame that would be created after this stage.

At this point, it is necessary to convert all privacy-preserved values into a single value. Section 3.5.4.2 explains that the chosen method of doing this is to randomly select a value from the set and assign it this value for the remainder of the tree creation. This process is carried out by the *assignVal* function which be explained in the next section. The rightmost table in Figure 22 shows one possible column

which could occur as a result of this process. I have named such this table *privacyRandom*. The ID3 algorithm then continues as normal, using the newly-assigned single values to classify the instances in the dataset.

4.2.2 ASSIGN VAL

As mentioned the *assignVal* function is responsible for converting all privacy-preserved values under a given attribute into single values. It does this by first checking whether any privacy-preserved values occur in the dataset. If they do, it removes the surrounding brackets and splits up the possible values. It then selects one value at random and converts the privacy-preserved value in the DataFrame to this new value. This function performs the equivalent act of converting the middle table (*val_df*) in Figure 22 to the rightmost table (*privacyRandom).*

4.2.3 CREATE TREE

The *createTree* function for the creation of the privacy tree primarily remains the same apart from a few modified lines of code which I will highlight below.

```
FOR VAL IN GET_VALUES(DATAFRAME, BEST)
```

In order to obtain the possible single values that an attribute in a privacy-preserved dataset could possess, I implemented a new function called *get_values*. The purpose of this function is to split up all privacy-preserved values into their single value components and return all the possible single values that appear in the dataset. For example, the privacy-preserved value '(Male, Female)' would return the list ['Male', 'Female']. The ID3 algorithm will then loop through these possible values in order to create the privacy tree and the addition of this function means that any generalised values will not be evaluated.

```
(DATAFRAME, PRIVACYRANDOM) = ASSIGNVAL(DATAFRAME, VAL_DF, BEST)
```

This line of code stores the DataFrame objects from the *assignVal* function into variables called *dataframe* and *privacyRandom*. Throughout the project, I struggled with this section of code for a while. This line of code was originally written as `PRIVACYRANDOM = ASSIGNVAL(VAL_DF, BEST)` where the *assignVal* function returned a DataFrame object, given the privacy-preserved instances

and the attribute upon which to perform the new assignment of values. The recursive call of the algorithm, at the time, was written to be called using privacyRandom and I was struggling to produce meaningful classification results. I will use Figure 22 to illustrate the problem that this was causing.

The original dataset upon which the ID3 algorithm is being performed is *master_df*. For each possible value of an attribute in *master_df*, a table such as *val_df* is generated. The generalised values are then converted into a randomly-selected single value to generate a table such as *privacyRandom*. In the original code that I had written, for a particular value, this meant that *privacyRandom* was used to generate its subtrees. However, when a base case occurred and the for loop in the algorithm jumped to the next possible value, the dataset reverted back to *master_df* where the values had changed back to their generalised form. This subsequently meant that a privacy-preserved value that had already been assigned a value could be assigned a completely different value in subsequent iterations. This was inconsistent and majorly flawed and led to the decision tree containing a lot more values in the subtrees than it should have, resulting in a higher-than-expected accuracy.

To solve this, I added a third argument to the *assignVal* function which would be the dataset represented in *master_df*. This meant that when modifying an instance's value in *val_df*, I could also ensure that the instance was changed to the same value in *master_df* and that this modified dataset was passed back to the ID3 algorithm to use from this point onwards. By doing this, once a privacy-preserved value had been assigned a single value, it would adopt this same value for the whole duration of the creation of the tree. After a lot of difficulty, this solved the problem and resulted in classification accuracies that precisely reflected the given dataset.

```
SANITISE_DF = PRIVACYRANDOM[PRIVACYRANDOM[BEST] == VAL]
```

Another thing I noticed that was affecting the classification results led to the addition of this line of code. Without this line, on the turn of a value in the for loop, *val_df* was used to store all instances that either possess the current value or contain the current value in its privacy-preserved set of possible values. Subsequently, *privacyRandom* was used to store the dataset containing any newly-assigned values. This was then used to generate the subtrees and derive the final decision tree.

Consider the situation presented in Figure 22 where it is the turn of the value 'German' in the for loop and all generalised values in *val_df* are then randomly assigned to a value other than 'German'. In my original code, the *privacyRandom* dataset, represented by the rightmost table, would then be

used to generate subtrees. However, three out of four of these values no longer belong in this iteration since they don't possess the value 'German'.

It is for this reason that I added the above line of code which is Pandas syntax for creating a new DataFrame object which contains all the instances from *privacyRandom* that possess the current value for the current best attribute. It checks the dataset and makes a new dataset which only includes those values that still belong in the particular iteration. The new DataFrame, *sanitise_df*, is then used in the recursive call of the *createTree* function. As the *assignVal* function also modifies the values of *master_df*, any instances that are removed from an iteration are then picked up when further values are evaluated later on in the algorithm.

The privacy tree algorithm then proceeds to continue in the exact same way as the original ID3 algorithm which results in a decision tree containing only singular values.

4.3 MONDRIAN ALGORITHM

--------- - MONDRIAN.PY

This section will describe the implementation of the Mondrian algorithm. It will be broken up into the four different stages of the algorithm which, on completion, should introduce all necessary aspects of the code. At the start of the code, an empty DataFrame is created and stored in a variable called *result* which adds the anonymised instances after each iteration and ultimately forms the complete anonymised dataset. This is a global variable which means that, when used throughout the *mondrian.py* file, the same dataset instance can be accessed each time despite any recursion.

4.3.1 CHOOSE A DIMENSION

------ (choose_dim)

```
def choose_dim(partition, quasis):

        length = {}

        for q in quasis:
                length[q] = len(get_values(partition, q))

        return max(length, key=length.get)
```

FIGURE 23: The choose_dim function in the Mondrian algorithm

The process of choosing a dimension on which to partition the dataset occurs in the function *choose_dim* which is shown in Figure 23. The method I have implemented of selecting a dimension is to choose the quasi-identifier that has the most values in the dataset. The *choose_dim* function implements this by looping through all the quasi-identifiers and storing the frequency of their values in the dataset into a dictionary. The function then returns the attribute with the largest frequency in the dictionary. If the largest frequency is shared by more than one attribute, the attribute that appears first in the dictionary is chosen.

4.3.2 CALCULATE THE FREQUENCY SET OF DIMENSION VALUES

----- *(frequency_set)*

The function, *frequency_set,* is responsible for generating the frequency set of the values within a dimension. The code for this function is shown in Figure 24. It is within this function that a dimension is selected on which to partition the dataset through the call to *choose_dim*. The selected attribute is stored in a variable called *dim*. The *frequency_set* function operates by creating a dictionary which stores the values of the dimension as dictionary keys and their associated frequencies in the dataset as dictionary values. The resultant dictionary is then ordered either alphabetically for categorical attributes or numerically for continuous attributes. The ordered dictionary, as well as the attribute that was selected as the dimension, is returned by the function.

```
def frequency_set(partition, quasis):

        frequency = {}

        dim = choose_dim(partition, quasis)

        for value in get_values(partition, dim):
                frequency[value] = len(partition[partition[dim] == value])

        ordered_freq = sorted(frequency.items())
        return ordered_freq, dim
```

FIGURE 24: The frequency_set function in the Mondrian algorithm

4.3.3 FIND THE SPLIT VALUE

--- (find_median)

The process of finding the value upon which to split the partition occurs in the function *find_median*. This function begins by initialising the variables *freqSet* and *dim* by calling the previous function *frequency_set* and storing the frequency set and the chosen dimension that it returns to the respective variables. The total number of tuples in the partition as well as the possible values of the selected dimension are then stored in variables called *total* and *values* respectively. The median number is calculated by dividing the total number of instances by two and stored in a variable called *median*.

Next, a counter variable is initialised to zero which will store the cumulative frequencies of the frequency set as it is iterated through. Another counter variable, *val_index*, is initialised to -1. This will be used to store the index of the value in the list, meaning that after being incremented by one, the first value will have a correct index value of zero.

The frequency set is then iterated through by means of a for loop. At each iteration, the frequency of the value is added to the cumulative frequency recorded by the *count* variable. The index of each value is also incremented and stored. When finding the split value, there are two options which could occur. The stages that happen at each option are detailed below.

1. MEDIAN OCCURS AT VALUE BOUNDARY

The first is that the median could be found to be precisely on the boundary of a value. In the code, this would be signified by a value of *count* which exactly equals the value of *median* for the calculation of the cumulative frequency. A situation such as this is shown in Figure 25. In this case, instances containing Value 1 or Value 2 would become the left-hand side of the partition and instances containing Value 3, Value 4 and Value 5 would make up the right-hand side of the partition. Throughout this project, since strict partitioning is being used, I have decided that the

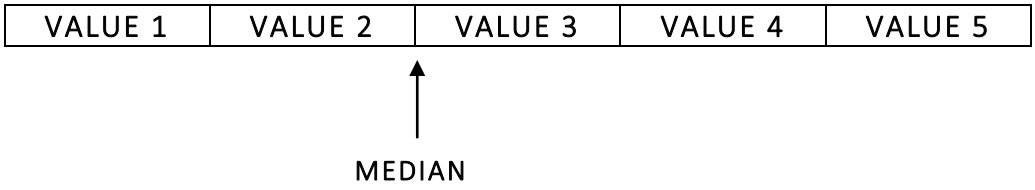

FIGURE 25: A diagram to illustrate the median occurring on the boundary of a value

split value will symbolise that all previous values in the frequency set and the split value are included in the left-hand side of the partition, whilst the right-hand side of the partition consists of all other values. That said, the split value in this instance would be Value 2 since it is included on the left-hand side of the partition.

In my code, the variable *split_val* is used to store the split value of a given frequency set. For cases such as those shown in Figure 25 where the median occurs at the boundary of the value, *split_val* is set to be the current value whose frequency has most recently been added to the cumulative frequency. If this happens, the *find_median* function returns the split value, the current dimension, the list of values and the value index of the split value in the frequency set. The for loop is then broken out of to cease execution of any further code and the running of the function is complete.

2. MEDIAN OCCURS WITHIN VALUE

The second option that could occur is that the median is found to exist within a value. This is illustrated by Figure 26. Since strict partitioning is used, in this instance, there are two options: either the instances containing this value will be added to the left-hand side of the partition or the instances will be added to the right-hand side of the partition. To calculate which of these options will occur, we need to simulate the two possibilities and choose the option which gives the most equal distribution amongst the left-hand side and the right-hand side of the partition. To demonstrate this, recall the frequency set used in section 3.4.2: {Amer-Indian-Eskimo: 286, Asian-Pac-Islander: 895, Black: 2817, Other: 231, White: 25,933}. The median value occurs at the 15,081th instance which falls within the value 'White'. This gives us two different scenarios. The first is that the value 'White' is included in the left-hand side of the partition, along with all the other values. This would give a ratio of left-hand side partition to right-hand side partition of 30,162:0. In this case, the split value would be 'White'. Alternatively, the value 'White' could be included in the right-hand side of the partition which would, instead, give a ratio of 4229:25933. In this case, the split value would be the previous value which is 'Other'. It can be seen that in the latter occasion, the distribution between the left-hand side partition and right-hand side partition is more equal. Thus,

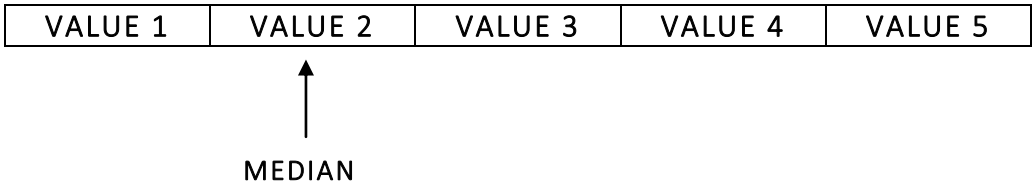

FIGURE 26: A diagram to illustrate the median occurring within a value

the split value is chosen to be the value 'Other'.

To implement this in the code, it first features an if statement to catch when the value of the variable *count* exceeds the value of the variable *median*. This means that the median lies in the value that was most recently added to the cumulative frequency. The previous value is then calculated and stored in the variable *prev_val* by using the value index of the current value and subtracting one. The variable *split_val* is also assigned to be the current value for the time being. As previously mentioned, the distribution of the two possibilities will need to be calculated and either the value contained in *prev_val* or the value contained in *split_val* will be selected as the split value. These distributions are then calculated and depending on which has the most equal distribution (i.e. the smallest difference between the left-hand side and right-hand side partitions), the function returns the correct split value for the chosen distribution. Along with this, the dimension that is currently being used to split the partition as well as the list of values and the associated value index of the split value are also returned.

4.3.4 PARTITION THE DATASET

----- (*partition)*

The function, *partition*, is a recursive function which makes up the majority of the Mondrian algorithm. The function begins with two base cases that, if processed, stop further execution of the algorithm signifying that k-anonymity has been satisfied for a particular partition. These base cases are detailed below. Appendix B highlights the various possible scenarios that could occur within this function and details how these scenarios are handled.

1. ALL THE QUASI-IDENTIFIER VALUES ARE THE SAME

This first base case is executed if the partition contains instances which all possess the same values for their quasi-identifiers. In this situation, there is no need to compare the length of the partition with the value of k since it is assumed that the recursive portion of this function will have caught any violations of k-anonymity and resolved them. Therefore, any partition entering the *partition* function is known to already be satisfying k-anonymity. In the instance of this base case, as long as all instances in the partition possess the same quasi-identifier values, they can be immediately added to the *result* variable as it is assumed that the length of the partition will be equal or greater than the

AGE	MARITAL STATUS	NATIONALITY
23	Married	Italian
49	Married	Italian
37	Married	Italian

TABLE 13: An example of a partition where all instances possess the same quasi-identifier values

value of k. An example of such a partition is show in Table 13 where the quasi-identifiers are {Marital Status, Nationality} and k=2.

2. THE QUASI-IDENTIFIER VALUES ARE DIFFERENT AND THE LENGTH OF THE PARTITION IS EQUAL TO K

This base case is executed if the quasi-identifier values in a given partition are not identical and the length of the partition is equal to the value of k. For this to occur, we assume two possible options: (1) the length of the entire dataset is equal to k or (2) the dataset has previously been passed into the recursive function call and the partition currently being processed has been created as a result of that. In either situation, since the length of the partition is equal to k, it is clear that k-anonymity is being satisfied. Moreover, we can conclusively say that the partition will not be able to be further partitioned while maintaining k-anonymity. This is because the partition is already at its minimal length for achieving k-anonymity. Any further partitioning will result in a partition whose length is less than the value of k and thus k-anonymity would be violated. With that said, the only option remaining for this partition is to merge its quasi-identifier values[1] and add the result of this to the dataset stored in the *result* variable. An example of a partition in which this base case would apply in shown in Figure 27 where k=2 and the quasi-identifiers are {Marital Status, Nationality}.

AGE	MARITAL STATUS	NATIONALITY
23	Married	Italian
49	Single	British

MERGE QUASI-IDENTIFER VALUES

AGE	MARITAL STATUS	NATIONALITY
23	(Married, Single)	(British, Italian, Spanish)
49	(Married, Single)	(British, Italian, Spanish)

FIGURE 27: An example of a partition where all instances possess the same quasi-identifier values

[1] Any merging of values is carried out by the *merge_vals* function. When given a partition on which to operate and a list of quasi-identifier attributes, this function returns a partition where all quasi-identifier values have been merged. Numeric values are modified into a range of values (e.g. 13-20) and categorical values are modified into a set of possible values. For any categorical values, the order in which they appear in a set of possible values is always alphabetical to ensure consistency in semantically-similar values throughout the data. The resulting dataset can then be said to satisfy k-anonymity.

Following the above two base cases in the code comes the main recursive part of the *partition* function. This is carried out if the quasi-identifier values in a partition are not all identical and the length of the partition is greater than K. Therefore, it is necessary to check whether a partition can be further partitioned without violating k-anonymity and if so, the partition is split.

To begin this section of the code, *find_median* is called which in turn contains a call to the *frequency_set* function and the *choose_dim* function. This means that by calling *find_median* in the partition algorithm, a dimension is selected upon which to split the partition, as well as its split value for that dimension. These are stored in the variables *dim* and *split_val* respectively. In addition to this, the possible values of the dimension are stored as well as the index in this list of values that is represented by the split value. Using this index, the list of values is sliced into two separate lists: the values belonging to the left-hand side partition and the values belonging to the right-hand side partition. These lists are then used to generate a DataFrame for the left-hand side of the partition containing those instances that possess any of the values in the left-hand side list and the respective DataFrame for the right-hand side of the partition. These DataFrames are stored in variables called *lhs* and *rhs*.

After this point, the original code that I had written included a simple if statement that specified that if the length of either the left-hand side or the right-hand side partition was less than the value of k (i.e. this was not an allowable cut), the quasi-identifier values of both partitions should be merged and added to the result. On the contrary, if the lengths of these partitions were equal to or greater than k (i.e. an allowable cut), further partitioning should occur by means of the recursive function call. With this piece of code, I was finding that the result of the Mondrian algorithm on the dataset was not optimised to its fullest potential. I considered the situation in which the selected dimension on which to partition the dataset resulted in no further allowable cuts but another dimension did result in allowable cuts and therefore allowed further partitioning on the dataset. This would consequently lead to less generalised values where possible and therefore could signify that the usability of the dataset in further data mining processes may be higher. This was definitely something that I wanted to include in my algorithm and so I implemented the following piece of code.

After splitting a partition into its respective left and right-hand sides, the function makes a copy of the quasi-identifier attributes. I then implemented a while loop which executes while the length of either the left-hand side partition or the right-hand side partition is less than the value of K. Thus, meaning that this is not an allowable cut as k-anonymity is violated. The purpose of the while loop is to choose another dimension from the quasi-identifier and check whether this new dimension

```
while len(lhs) < k or len(rhs) < k:

        allowable = False

        if len(temp_quasis) == 1:

                merged = merge_vals(dataframe, quasis)
                result = result.append(merged)
                break

        temp_quasis.remove(dim)

        (split_val, dim, values, index) = find_median(dataframe, temp_quasis)

        lhs_vals = values[:index+1]
        rhs_vals = values[index+1:]

        lhs = dataframe[dataframe[dim].isin(lhs_vals)]
        rhs = dataframe[dataframe[dim].isin(rhs_vals)]

        allowable = True
```

FIGURE 28: A section of code from the *partition* function which evaluates all possible dimensions for allowable cuts before merging the quasi-identifier values.

results in an allowable cut. If so, the partition can be split using this dimension instead. If not, another dimension is selected and the process is repeated until all dimensions have been checked and neither of them results in an allowable cut. This while loop is shown in Figure 28 and will be explained below.

At the start of the *partition* function, a Boolean marker is set to True and stored in the variable *allowable*. This signifies that there is an allowable cut. If the while loop is executed, the Boolean variable, *allowable*, is set to False as k-anonymity is not currently satisfied. The while loop then contains an if statement which acts as a base case and an opportunity to terminate the while loop if all quasi-identifier attributes have been checked and neither of them produces an allowable cut. In this case, both the left-hand side and the right-hand side partition are concatenated back together to form the original partition in the iteration and their quasi-identifier values are merged. The outcome of this is then added to the *result* variable and will appear in the anonymised dataset. The while loop is then broken out of and the function is complete.

We will now evaluate the while loop in the case of when the if statement has not been executed. The Boolean variable, *allowable,* is still set to False. The current dimension that is stored in the variable *dim* is removed from the duplicate quasi-identifier list. This is because by the sheer fact that the while loop was executed in the first place, we know that this dimension does not produce an

allowable cut. Therefore, the duplicate quasi-identifier list now only contains those attributes that could potentially lead to an allowable cut and a satisfaction of k-anonymity.

Out of these remaining dimensions, a new dimension is selected upon which to attempt to split the partition. Following the same procedure as before, the partition is split into a left-hand side partition and a right-hand side partition. The variable *lhs* and *rhs* are reassigned to store these newly-created partitions created by using the new dimension. The variable *allowable* is then set to True and two possible situations could occur. The first is that one or both of the datasets contained in *lhs* and *rhs* have lengths which are less than the value of k. This means that the new dimension that was selected does not, in fact, satisfy k-anonymity and the while loop is repeated, setting the variable *allowable* back to False and selecting a new dimension on which to split the partition. The second situation that could occur is that the lengths of both of the datasets stored in *lhs* and *rhs* are greater than k and the while loop ceases execution. In this case, we have found a dimension that does produce an allowable cut and the Boolean variable *allowable* keeps its value of True.

The final part of the function contains the recursive call. A concluding if statement specifies that if the value of *allowable* is True (i.e. there is an allowable cut), both the left-hand side and right-hand side partitions should be recursively used in the call to *partition,* along with the original list of quasi-identifier values and the value of k which remains the same throughout. At the end of the function, after execution of all if, elif and else statements, the global *variable* result is returned containing the completed anonymised dataset.

4.4 CLASSIFICATION ALGORITHM

---------- **CLASSIFICATION.PY**

This algorithm exists as a means of classifying new instances within a dataset into their predicted class labels. It uses a given dataset as well as a decision tree upon which the classification is based; both of which are specified by their respective file paths. The algorithm consists of three main functions which will be explained below.

4.4.1 CALCULATING THE CLASSIFICATION ACCURACY

----- (*accuracy)*

The function, *accuracy*, takes as input a decision tree in the form of a dictionary, the test data for which classification will occur and the attribute for which a value will be predicted. To begin, the function initialises a counter variable called *correct* which is used to record the number of correct predictions. A for loop is used to iterate through each row in the test data and record its predicted value, as selected by the decision tree, and its actual value, as displayed in the dataset. The variable *correct* is incremented by one if the row's predicted value is equal to its actual value. Subsequent to the completion of the for loop when all rows in the test data have been predicted, the number of correct predictions is converted into a percentage and returned by the function. The prediction of the value of a row's sensitive attribute is carried out by the function *classify*. The operation of this function will be explained in the section below.

4.4.2 CLASSIFYING AN INSTANCE

----- (*classify)*

The *classify* function is used to predict the class label for a given instance using the decision tree. It is a recursive function which starts with two base cases. The first base case checks whether a tree exists and returns 'No tree' if not. The second base case specifies that if the tree given is not a dictionary, the tree should be returned. A situation such as this would occur after several iterations

```
def classify(tree, row):

        if not tree:
                return 'No tree'
        if not isinstance(tree, dict):
                return tree

        attribute = list(tree.keys())[0]
        values = list(tree.values())[0]

        instance_value = row[attribute]

        if instance_value not in values:
                return majority(attribute, instance_value, 'Salary')

        return classify(values[instance_value], row)
```

FIGURE 29: The *classify* function in *classification.py*

AGE	MARITAL STATUS	NATIONALITY	SALARY
23	Married	Italian	?

TABLE 14: A simplified representation of a table containing a row to be classified

of the function where through traversing the tree, a leaf node has been reached. In this situation, the class label that the leaf node possesses is returned as the predicted class label for the given row.

Continuing to follow the code, a few variables are now assigned. A variable called *attribute* stores the current parent node while a variable called *values* stores the children of the parent node which would also be in the form of a dictionary. Consider the decision tree example that was given in Figure 16. The resulting decision tree is represented as *{'Marital Status': {'Single': '>50K', 'Married': {'Nationality': {'Italian': '>50K', 'Spanish': '<=50K', 'German': '<=50K'}}, 'Divorced': '<=50K'}}*. Using this example, in the first iteration, the value of *attribute* would be 'Marital Status' and the value of *values* would equal *{'Marital Status': {'Single': '>50K', 'Married': {'Nationality': {'Italian': '>50K', 'Spanish': '<=50K', 'German': '<=50K'}}, 'Divorced': '<=50K'}* where the child nodes are 'Single', 'Married' and 'Divorced'.

The *classify* function continues by obtaining the value of the parent attribute for the instance that is currently being classified. This is stored in a variable called *instance_value*. For example, using the above decision tree to classify the instance shown in Table 14, as previously explained, Marital Status is the parent node in the first iteration, meaning that *instance_value* for this row would be 'Married'. The particular value stored in *instance_value* is then checked to see whether it appears as a possibility in the tree under the current parent node. In our example, the value 'Married' would be checked for existence in the subtree *{'Single': '>50K', 'Married': {'Nationality': {'Italian': '>50K', 'Spanish': '<=50K', 'German': '<=50K'}}, 'Divorced': '<=50K'}*. If the value does not exist within the subtree, the class label that has the highest frequency in the dataset amongst the particular value is returned as the class label. On the other hand, if the value does exist within the subtree, as it does in the case of our example, a recursive call to the *classify* function is made using the subtree of this value which exists in the decision tree. The algorithm continues until a leaf node is found and returned as the predicted class label.

The process of fully classifying the instance in Table 15 is shown in Appendix C which, when read left to right, follows the progression of the classification algorithm. At each iteration, the subtree of the instance value is used as the tree in the next iteration.

5. RESULTS AND EVALUATION

The question that I set out to answer throughout this project was 'To what extent, if any, does privacy preservation of data affect its usability with regards to subsequent data mining processes?' In this section, I will detail the methodology used to answer this question as well as the results that I obtained.

5.1 METHODOLOGY

This section will explain the process that I took in order to be able to answer the question that I set out to solve.

5.1.1 EXPERIMENTAL PROCESS

In order to evaluate the effectiveness of privacy-preserved data when used in data mining processes, I needed to compare the results obtained from carrying out data mining on a non-anonymised dataset with the results of carrying out data mining on the same dataset in an anonymised form. The code detailed in the implementation section of this report allowed me to do this and the procedure of doing so will now be explained.

As mentioned in section 3 of this report, I used the Adult dataset from the UCI Machine Learning Repository as the dataset for which to conduct this investigation on. Firstly, I ran the ID3 algorithm implemented in *tree.py* using the Adult dataset as input. With the outputted decision tree, I used *classification.py* to classify the instances in the Adult test dataset. This provided a classification accuracy which represented the effectiveness of classification on a standard, non-anonymised dataset. This was used as the benchmark for all further classifications.

Following this, I anonymised the Adult dataset several times using various quasi-identifiers and varying values of k. To do this, I used *mondrian.py* to anonymise the dataset, ensuring to specify different parameters each time. For each anonymised version of the dataset, I inputted it into *privacytree.py* three times to produce three privacy trees. The reason I decided to carry out this step three times is since due to the randomisation process of selecting a privacy-preserved value, a different privacy tree is generated each time *privacytree.py* is run. For this reason, I decided to produce three privacy trees from the dataset and calculate the average of their resultant accuracies

when classifying the instances in the Adult test dataset. This will be used to represent the overall classification accuracy for that particular anonymised dataset.

It will then be compared with the classification accuracy obtained when using the non-anonymised dataset. Furthermore, the accuracies obtained from anonymising the Adult dataset using varying values of k and varying quasi-identifiers can also be compared against each other to see what effect, if any, they also have on the effectiveness of the given classification task.

5.1.2 VARIABLES

As mentioned, I will be conducting the experimental process of calculating classification accuracies multiple times, using different variables each time. The variables that I am going to modify throughout this investigation are (1) the value of k and (2) the number of quasi-identifiers. I would like to see if changing these parameters within the anonymisation process leads to a change in the effectiveness of the classification task.

Table 15 shows the different anonymisation parameters that will be set each time. In total, there will be eleven different anonymised datasets produced from the Adult dataset. These datasets will then be subjected to *privacytree.py* three times each and their resultant privacy trees will be classified using *classification.py*.

DATASET NUMBER	K	NUMBER OF QIS
1	4	6
2	8	6
3	16	6
4	32	6
5	64	6
6	64	7
7	64	8
8	64	9
9	64	10
10	64	11
11	64	12

TABLE 15: A table showing the different anonymised datasets that will be produced along with their associated value of k and the number of quasi-identifiers.

In their paper, Using Anonymised Data for Classification, Inan et al. used the default quasi-identifiers for anonymising the Adult dataset of *{Age, Work Class, Education, Hours Per Week, Capital Gain ,Work Class}*. In order for a comparison to be made between this work and my own, I will choose to use these quasi-identifiers as the default parameters when varying the value of k. This

is the case for the anonymised datasets 1 through to 5 in Table 15. The value of k doubles with each dataset, starting at k=4 and terminating at k=64.

For the anonymised datasets 6 to 11, k is set to a constant value of 64 while the number of quasi-identifiers is varied, starting from 7 quasi-identifiers and terminating at a dataset consisting of 12 quasi-identifiers. My justification for setting k to 64 when testing these classification accuracies is that, as will be shown further in this section, a k value of 64 produces the most significant results and so any differences between the classification accuracies when using a varying number of quasi-identifiers will be highlighted more clearly. In some ways, it could be said that not only will the number of quasi-identifiers have an effect on the resultant classification accuracies but also the particular quasi-identifiers that have been chosen could also have an effect. For example, although both {Relationship, Race, Sex} and {Occupation, Marital Status, Work Class} consist of three quasi-identifiers, the classification accuracies produced when using datasets formed from these quasi-identifiers may be different. In this way, any differences in classification accuracies due to varying the number of quasi-identifiers should be viewed with caution as it is not clear whether these differences occur due to the number of quasi-identifiers or, instead, the choice of those particular quasi-identifiers.

With that said, I have decided that the most intuitive method of testing this would be to start with the same default quasi-identifiers that were used to test the effects of varying the value of k. This means that the minimum number of quasi-identifiers tested will be 6. As the number of quasi-identifiers increases with each anonymised dataset, I decided that the next quasi-identifier added to the list will be the one with the most possible values in the dataset, starting with all the categorical attributes and then moving to the numerical attributes. I came to the decision of not including the Fnlwgt attribute in this list because there were too many possible values and also not including the Education-num attribute since it is merely displaying the same information as the Education attribute. Therefore, the maximum number of quasi-identifiers in this investigation will be 12.

By selecting the quasi-identifiers in this way, I am hoping to see whether choosing quasi-identifiers with more possible values results in a lower classification accuracy than selecting quasi-identifiers with less possible values. My reasoning behind this is that with more possible values, there will be a lesser chance of the privacy-preserved values being mapped back to their original value in the creation of the privacy tree and therefore I would like to test whether this would be reflected in the resultant classification accuracy. Appendix D shows the different attributes selected as quasi-identifiers for each number of quasi-identifiers used.

5.2 RESULTS

The first result that was obtained was the classification accuracy when deriving a decision tree from the non-anonymised dataset and using this decision tree to classify the instances in the Adult test dataset. The classification accuracy for this was 80.96%. This was used as a benchmark for all further classifications whereby the parameters in the anonymisation process were changed. A summary of this is shown in Table 16 which is represented by zero quasi-identifiers and a k value of 1. The results obtained when varying the value of k and varying the number of quasi-identifiers are displayed in the next sections. A more detailed list of these results can be found in Appendix E.

NUMBER OF QUASI-IDENTIFIERS	VALUE OF K	AVERAGE CLASSIFICATION ACCURACY (%, 2 DP)
0	1	80.96

TABLE 16: A table showing the classification accuracy obtained using a decision tree derived from the non-anonymised Adult dataset

5.2.1 VARYING THE VALUE OF K

The results obtained from varying the value of k in the anonymisation process are shown in Table 17.

VALUE OF K	AVERAGE CLASSIFICATION ACCURACY (%, 2 DP)
4	80.87
8	80.80
16	80.46
32	79.87
64	79.28

TABLE 17: A table showing the classification accuracies obtained using a decision tree derived from anonymised datasets with varying values of k

To test the effect of varying the value of k on the classification accuracy, I used a default number of six quasi-identifiers *{Age, Work Class, Education, Hours Per Week, Capital Gain ,Work Class}* and tested different values of k ranging from 4 to 64. As shown in Table 17, as predicted, an increase in the value of k led to a decrease in the classification accuracy. This trend is also displayed in the graph shown in Figure 30 where the data labels represent the value of k and the red line represents the accuracy obtained when using the non-anonymised dataset (80.96%). It can be seen that with a k value of 4, the classification accuracy is only decreased by 0.09% when compared to the classification accuracy of the non-anonymised dataset. This is because at low levels of k, a lot of the values in the

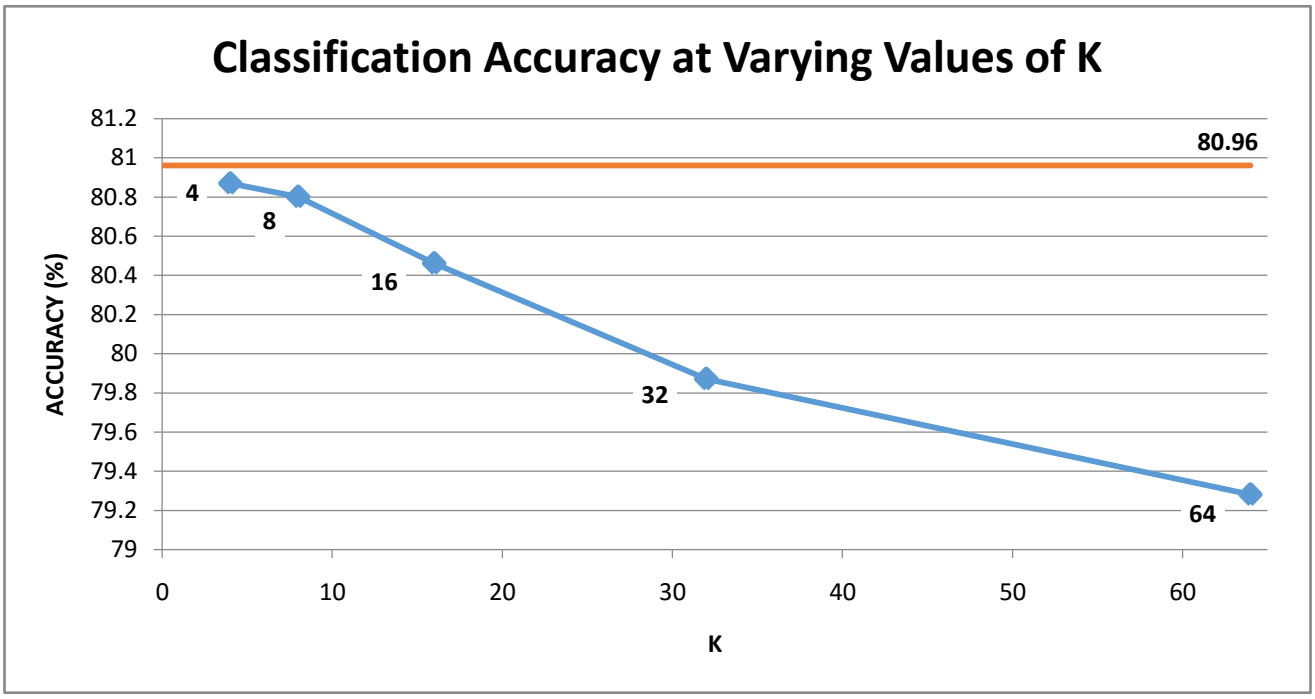

FIGURE 30: A graph showing the classification accuracies obtained using varying values of k

dataset are not generalised since the quasi-identifier values occur in the dataset enough times to satisfy k-anonymity as it is. For example, when looking at the Adult dataset, the value combination of *{Age=45, Work Class=Private, Education=HS-grad, Capital Gain=0, Hours Per Week=40, Native Country=United-States}* appeared 77 times. This would mean that the instances in this particular combination would not be generalised until the value of k was set to 78 or higher.

Furthermore, for those values in the dataset that are generalised, they are usually only paired with one or two other values to form the privacy-preserved set of possible values. In this way, the likelihood that a privacy-preserved value will be mapped back to its original value in the creation of the privacy tree is still quite high. Therefore, the semantic relationships between the attributes are preserved which is also reflected in the resultant decision tree, leading to a classification accuracy which is similar to the accuracy obtained through the use of the non-anonymised dataset. Although the values of k I have experimented with only go up to 64, I believe that this is sufficient as the differences in classification accuracy between the various values of k can still be analysed and compared.

When k is at its maximum value of 64, it can be seen that the classification accuracy is 79.28% which represents a decrease of 1.68% when compared to the classification accuracy obtained through using the non-anonymised dataset. After considering all factors, it could be said that an accuracy decrease of 1.68% is a minimal cost for ensuring that data is privacy-preserved. However, in the test data that has been used 1.68% equates to 253 instances, meaning that anonymising the dataset with a k value of 64 results in an addition of 253 incorrectly-classified instances compared to a non-anonymised dataset. As mentioned, using a higher value of k will further decrease the classification accuracy until

there can be no further reductions since there will become a point where no further generalisations can be made to the dataset.

There are two factors to consider here: (1) the level of anonymity that the data holder requires and (2) the level of classification accuracy that the data miner requires. If the data holder must ensure that the dataset is highly anonymised, the classification accuracy that will be achieved by the data miner will suffer, thus leading to less significant data mining results. Likewise, if the data miner requires high levels of classification accuracy due to the nature of the prediction task, the extent to which the dataset is anonymised will be compromised. This is the trade-off that this project has set out to evaluate, the majority of which can be drawn back to whether a data miner is able to suffice with, in this example, a decrease in accuracy of 0.09% or greater.

5.2.2 VARYING THE NUMBER OF QUASI-IDENTIFIERS

The results obtained from varying the value of k in the anonymisation process are shown in Table 18.

NUMBER OF QUASI-IDENTIFIERS	AVERAGE CLASSIFICATION ACCURACY (%, 2 DP)
6	79.28
7	75.77
8	75.63
9	71.09
10	70.84
11	70.46
12	69.60

TABLE 18: A table showing the classification accuracies obtained using a decision tree derived from anonymised datasets with varying values of k

To test the effect of varying the number of quasi-identifiers on the classification accuracy, I used a default k value of 64 and varied the number of quasi-identifiers in each anonymised dataset from 6 to 12. From looking at Table 18, it can be seen that as the number of quasi-identifiers used in the anonymisation process increases, the classification accuracy decreases. This is also reflected in Figure 31 where, again, the red line represents the accuracy obtained when using a non-anonymised dataset. This occurs since the more quasi-identifiers that are specified, the rarer the value combinations become in the dataset. This means that the combinations of values appear less frequently in the dataset, meaning that more values are generalised in order to enforce k-anonymity. This leads to more instances that have privacy-preserved values and more values included in an attribute's set of possible values. Both of these result in a lesser likelihood that in the creation of a privacy tree, a

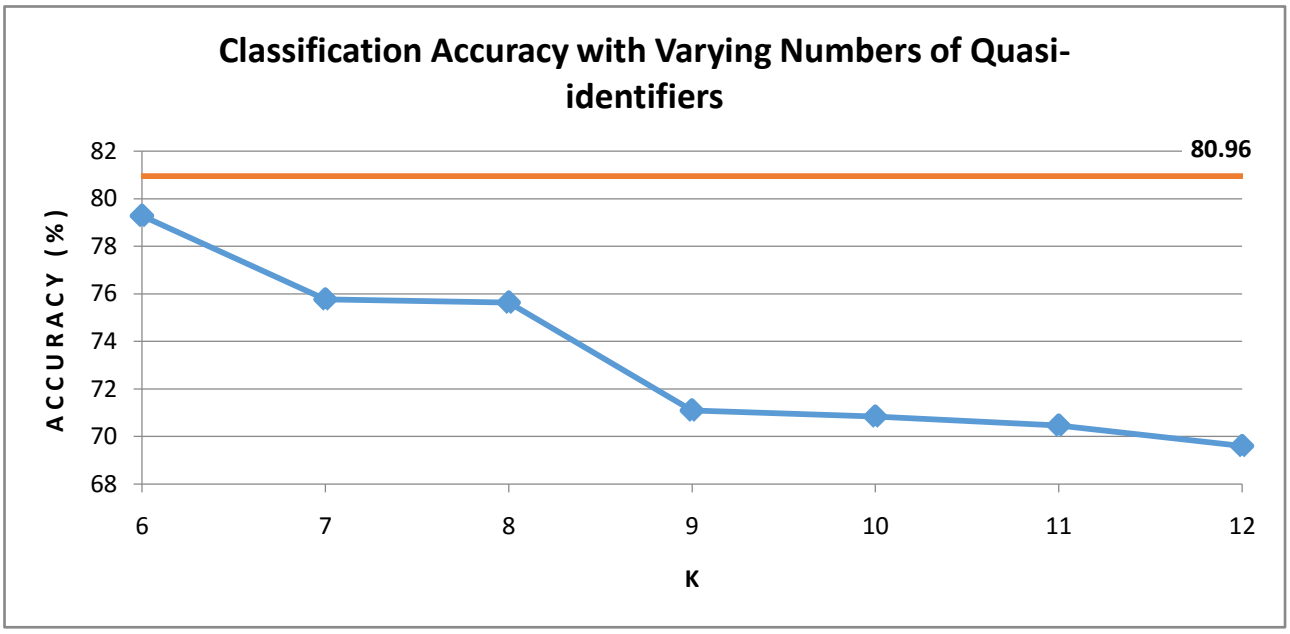

FIGURE 31: A graph showing the classification accuracies obtained using a varying number of quasi-identifiers.

privacy-preserved value will be mapped back to the value that it originally was in the non-anonymised dataset. This, in turn, leads to lower classification accuracies since the semantic relationships between the attribute values may have been slightly distorted.

It could be thought that the amount by which the classification accuracy decreases with each addition of a new quasi-identifier is lower than expected. Through the analysis of the decision trees produced from the anonymised datasets at each stage, I have discovered a tendency which may explain this. With the addition of each new quasi-identifier, the decision tree tends to favour an attribute that has not been privacy-preserved to use as the root of the tree. Table 19 shows the number of quasi-identifiers used for each anonymised dataset along with the attribute that was added to the quasi-identifier list each time and the attribute that was selected as the root of the decision tree.

The default quasi-identifier combination consisting of six attributes shows that the attribute with minimum entropy in the dataset at this point is Relationship, meaning that it is selected as the root

NUMBER OF QUASI-IDENTIFIERS	QUASI-IDENTIFER ADDED	ROOT ATTRIBUTE OF DECISION TREE
6	N/A	Relationship
7	Occupation	Relationship
8	Marital Status	Relationship
9	Relationship	Sex
10	Race	Sex
11	Sex	Native Country
12	Capital Loss	Native Country

TABLE 19: A table showing the attribute which is selected as the root of the tree for each new addition of a quasi-identifier

of the tree. This means that the Relationship attribute tells us the most information in the dataset with regards to classifying further unseen instances. Relationship is then also selected as the root of the decision tree for those datasets with 7 and 8 quasi-identifiers as the values of the Relationship attribute have still not been generalised at this point. Due to this, it is still the attribute that provides the decision tree with the most amount of information. It is interesting to see what happens when the Relationship attribute becomes generalised which occurs when 9 quasi-identifiers are used in the anonymisation process. At this stage, the decision tree no longer uses Relationship as its root and instead opts for Sex. The Relationship attribute no longer holds the minimum entropy as it no longer provides the most information to the decision tree. This is because its values have been generalised and the fact that a privacy-preserved value could, in fact, take any one of the values in its generalisation is taken into account when calculating the entropy as shown in section 3.5.4.1.

Sex continues to be selected as the root of the decision tree for those datasets with 9 and 10 quasi-identifiers until Sex is added as a quasi-identifier itself in which case Native Country is selected instead. Native Country is, in fact, an attribute which is generalised in all of the datasets. However, after Sex has been generalised, it still provides the most information to the decision tree despite being generalised itself. In this way, the decision tree compensates for the privacy preservation of attributes by selecting an attribute which has not been privacy preserved to use as its root. Due to this, the decision tree is able to maintain reasonably satisfactory classification accuracies despite an increasing number of quasi-identifiers in the dataset. Conclusively, it can be seen in Figure 31 that there is a negative correlation between the number of quasi-identifiers used and the classification accuracy obtained.

From looking at the graph, it appears that there are not big differences in classification accuracy between some consecutive numbers of quasi-identifiers. For example, the difference between using 7 quasi-identifiers and 8 quasi-identifiers is a decrease in classification accuracy of only 0.14%. It is unclear why some consecutive numbers produce such a small decrease in accuracy while others lead to a larger decrease. It is clearly related to the characteristics of the quasi-identifier that was added but this has not been extensively tested. For this reason, I cannot say whether there is any meaningful correlation between the particular quasi-identifiers selected for anonymisation and the classification accuracies obtained.

5.2.3 COMPARISON WITH WEKA

	WEKA
	CLASSIFICATION ACCURACY (%, 2 DP)
WITH CONTINUOUS ATTRIBUTES	91.23
WITHOUT CONTINUOUS ATTRIBUTES	85.96

TABLE 20: A table showing the results of using Weka to derive a decision tree and classify the test data

The accuracy of the decision tree created using the non-anonymised Adult dataset can be compared with the result of inputting the same dataset into Weka. Weka can then derive its own decision tree and use this to classify the instances in the Adult test dataset. Weka uses the J48 algorithm which is discussed in section 3.5.3. It is expected to give higher classification accuracies due to the inclusion of continuous attributes in the algorithm as well as its ability to prune the decision tree in order to reduce those sections of the tree which provide little benefit and to prevent against overfitting.

Table 20 shows the results of using Weka to train the decision tree and classify the instances in the Adult test dataset. The first result was obtained when inputting the Adult dataset as it is including its continuous attributes. This achieves an accuracy of 91.23% which, as predicted, is higher than the accuracy of 80.96% which was achieved through the use of my ID3 algorithm. This is because J48 can handle continuous attributes and includes them in the decision tree.

Subsequently, I removed all continuous attributes from the Adult dataset and tested the results again using Weka on this modified dataset. This resulted in a classification accuracy of 85.96% which is closer to the 80.96% accuracy that my ID3 algorithm achieved. Again, J48 also conducts tree pruning which my ID3 algorithm does not which may explain the higher accuracy. However, overall the accuracies are quite similar which affirms the belief that the ID3 algorithm that I implemented is effective.

5.2.4 COMPARISON WITH OTHER LITERATURE

In order to validate the results I have obtained, I compared them with the results obtained by Inan et al. as described in the paper, Using Anonymised Data for Classification. In order to conduct their investigation, Inan et al. used the Datafly anonymisation method which was proposed by Latanya Sweeney. Datafly is a computer system based on databases that is designed to allow a user to specify parameters such as the anonymity level in order to produce an anonymised dataset [16]. Datafly automatically replaces identifiable attributes with a generalised form or fictitious values are added to replace them instead. As discussed in section 2.5, Inan et al. also used SVM as a classification method in order to classify their test instances. When comparing results, it must be taken into account that a different anonymisation method and classification method has been used which may influence the accuracies obtained. However, although the Mondrian algorithm and decision tree learning methods were not adopted in their paper, Inan et al. did use the Adult dataset in their investigation as I also did.

When using the SVM classification method to train the anonymised dataset and test on the original dataset, the results in the paper show that an accuracy of approximately 84.5% was achieved when using the original dataset [17]. This is shown in Figure 3a in the paper. Moreover, when the value of k was set to 8, the average accuracy decreased to approximately 82.5% and decreased again to approximately 81% when the k was set to 64. This reflects a decrease in classification accuracy of 2% between using the non-anonymised dataset and using an anonymised dataset where k=64. Comparing this to the results that I obtained where the difference between the same two datasets was a decrease in accuracy of 1.68%, I would say that this is fairly similar. Although Inan et al. did achieve slightly higher classification accuracies overall, this could be attributed to the different anonymisation and classification methods that they used.

When calculating the results of the effect of the number of quasi-identifiers in an anonymised dataset, Inan et al. used a default value of k=32 for their anonymisations. It can be shown in Figure 3b of the paper that the dataset containing six quasi-identifiers achieved an accuracy of just below 80%. When compared to the results I obtained using an anonymised dataset with six quasi-identifiers where k=32, the results are the same. I achieved a classification accuracy of 79.87% for the same parameters as shown in Table 17. I am unable to compare the results obtained for any other number of quasi-identifiers since it is not specified in Inan et al.'s paper the combination of quasi-identifiers that was used for each experiment.

6. FUTURE WORK

This section will discuss various concepts that I believe can be further researched in the future in order to expand the current scope that I have presented in this project.

1. DIFFERENT METHODS OF GENERALISING VALUES

It has been previously discussed that my chosen method of generalisation of values in the anonymised datasets was to represent the values as a set of possible values. I would like to further explore ways of representing these privacy-preserved values such as by using domain generalisation hierarchies to replace a value with a broader term which maintains any semantic connotations. Due to the complexity of the task, I was not able to implement it in this project however I believe it would be interesting to see the classification accuracies that are obtained when using this method of generalisation. By doing this, a comparison can be made between the accuracies obtained and those that are shown within this project.

2. IMPLEMENTING A DIFFERENT CLASSIFICATION ALGORITHM

In this project, the ID3 algorithm was implemented as a means of classification because it was simple and could be used as a baseline for classification using different algorithms. Future work could include the research of using more advanced classification methods such as J48 which handles numerical attributes. Comparisons can be drawn from the classification accuracies using the two different algorithms and it can be seen if the anonymisation parameters such as the value of k and the number of quasi-identifiers have an effect on J48 in the same way as it does with the ID3 algorithm.

3. FURTHER EXPLORING THE RELATIONSHIP BETWEEN QUASI-IDENTIFIERS SELECTED FOR ANONYMISATION AND CLASSIFICATION ACCURACY

I would like to further explore if the particular quasi-identifiers used in the anonymisation process have any effect on the associated classification accuracies obtained. Due to time constraints, I was unable to conduct this fully in this project since a variety of quasi-identifier combinations would need to be tested in order to check for any trends in the accuracies produced. This is something that

could be researched as a supplementary piece to this project as I believe that it could produce interesting results.

7. CONCLUSIONS

At the start of this project, I set out to discover whether the privacy-preservation of data affected the usability of such data in data mining processes with regards to its effectiveness. I aimed to be able to quantify the extent, if any, that data classification tasks were affected when the dataset that was used to form the classification model was anonymised. The results that I have obtained during this project have demonstrated that, overall, classification accuracy is not significantly affected by the anonymisation of data.

With regards to using k-anonymity as a method of anonymisation, the results show that although there is a decrease in the classification accuracy when using higher values of k to anonymise a dataset, the difference in accuracy can be considered quite miniscule. Using the Adult dataset with a value of k equal to 64, the decrease in accuracy can be quantified as 0.09% when compared to the accuracy obtained from using the original, non-anonymised dataset. For most organisations, it would not be necessary to use such a large value of k such as 64. It would most likely be sufficient to anonymise the data to a much lesser extent while still complying with regulations in order to reduce the likelihood of re-identification attacks. Due to this, depending on the dataset used, it is likely that the reduction in classification accuracy could be much lower than 0.09%. Therefore, the information gained from any subsequent data mining will still maintain a large amount of usability despite the anonymisation methods placed on the data.

In terms of the number of quasi-identifiers used to anonymise a dataset, the results obtained in this project demonstrate that the larger the number of quasi-identifiers, the lower the classification accuracy. However, even with a longer list of quasi-identifiers, a good level of classification accuracy can be obtained since the decision tree modifies the attributes that are used as the root of the tree and thus the accuracy does not decrease to the levels that I would originally have expected.

To conclude, I would say that this project has demonstrated that it is possible to obtain both privacy within a dataset and limited information loss. It has indicated that the trade-off between the two desirable qualities is not as considerable as was first thought. This has significant implications for many organisations that choose to share their data with third parties but also must protect the privacy of the individuals to which the data refers. It will allow them to have confidence that the individuals' privacy is sufficiently protected whilst being able to offer valuable data to third parties and researchers whereby meaningful information can still be obtained. This could lead to much

advancement of insights in industries such as healthcare, law enforcement or banking and could change the perception of the current weaknesses in the world of privacy-preserving data mining.

8. REFLECTION ON LEARNING

On reflection of the way in which I have worked during this project, I am extremely pleased with the outcome and the process in which I took to produce it. I have enjoyed delving into privacy-preserving data mining and learning a whole new topic that I had no knowledge of before. I am proud that I have been able to absorb a variety of new concepts and apply them in order to produce meaningful results and successfully address the question that I originally set out to answer.

I encountered many problems during this project regarding the understanding of complex ideas or the programming of my code. However, I am pleased with my approach to solving those problems as I didn't give up. By calmly dissecting the information or taking a break and resuming the work when I had a clearer state of mind, I was successfully able to overcome those obstacles and produce a piece of work that I was proud of.

Meeting my supervisor, Dr. Jianhua Shao, each week was a good way for me to stay on track with my project and ensure that I had done sufficient work each week to make good progress. It helped me to set goals for the week and to have a clear idea of the direction in which my project was heading and the necessary tasks that I had outstanding. In hindsight, I should have referred back to my initial work plan more often in order to guide my progress through the project. However, I found my supervisor meetings to be sufficient in keeping me on track and I managed to complete all the work in ample time without feeling rushed in the final stages of the project.

Overall, the project definitely did come with its difficulties but, regardless, I am proud of the work that I have achieved and have enjoyed learning about the chosen topic. It has given me a greater appreciation of the study of computer science as it demonstrates just how varied the field can be. It has really opened my eyes up to the type of work that can be conducted within the remit of computer science and has allowed me to discover an area that I would never have otherwise had the opportunity to explore.

9. APPENDICES

APPENDIX A – A list of attributes contained in the Adult dataset

ATTRIBUTE	CONTINUOUS	CATEGORICAL	DESCRIPTION	VALUES
AGE	✔		The age of an individual.	Continuous
WORK CLASS		✔	The work class that an individual belongs to.	Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked.
FNLWGT	✔		The number of people in the general population that an individual is believed to represent.	Continuous
EDUCATION		✔	The highest level of education an individual has achieved.	Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool.
EDUCATION-NUM	✔		Gives the same information as Education but in a numerical form.	Continuous
MARITAL-STATUS		✔	The marital status of an individual.	Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse.
OCCUPATION		✔	The occupation of an individual.	Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-speciality, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces.
RELATIONSHIP		✔		Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried.
RACE		✔	The race of an individual.	White, Asian-Pac-Islander, Amer-Indian-

				Eskimo, Other, Black.
SEX		✔	The sex of an individual.	Female, Male.
CAPITAL-GAIN	✔		Capital gains recorded.	Continuous
CAPITAL-LOSS	✔		Capital losses recorded.	Continuous
HOURS PER WEEK	✔		The number of hours an individual works per work.	Continuous
NATIVE COUNTRY		✔	The native country of an individual.	United-States, Cambodia, England, Puerto-Rico, Canada, Germany, Outlying-US(Guam-USVI-etc), India, Japan, Greece, South, China, Cuba, Iran, Honduras, Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland, France, Dominican-Republic, Laos, Ecuador, Taiwan, Haiti, Columbia, Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador, Trinadad&Tobago, Peru, Hong, Holand-Netherlands.

APPENDIX B – A table showing the possible scenarios of the *partition* function in *Mondrian.py* and their associated outcomes.

POSSIBLE SCENARIO	LENGTH OF PARTITION	OUTCOME
Quasi-identifier values are NOT identical	Length of partition is greater than K	Recursively partition the dataset further. If no allowable cut exists, this will be flagged and the quasi-identifier values will be merged and added to *result*.
	Length of partition is equal to K	No further partitions will be allowed as the partition is already at its minimal length. The quasi-identifier values are merged and added to *result*.
	Length of partition is less than K	This situation will never occur since the function never lets the partition get this small. In the recursive function, if a further partition will result in a partition which is less than K, the quasi-identifier values are merged.
Quasi-identifier values are identical	Length of partition is greater than K	Partition is added to *result* as k-anonymity is satisfied.
	Length of partition is equal to K	Partition is added to *result* as k-anonymity is satisfied
	Length of partition is less than K	This situation will never occur since the function never lets the partition get this small.

APPENDIX C – A table showing the progression of the *classify* function in *classification.py* using the example shown in section4.4.2.

ITERATION	TREE	ATTRIBUTE	VALUES	INSTANCE VALUE	SUBTREE
1	{'Marital Status': {'Single': '>50K', 'Married': {'Nationality': {'Italian': '>50K', 'Spanish': '<=50K', 'German': '<=50K'}}, 'Divorced': '<=50K'}	Marital Status	{'Single': '>50K', 'Married': {'Nationality': {'Italian': '>50K', 'Spanish': '<=50K', 'German': '<=50K'}}, 'Divorced': '<=50K'}	Married	{'Nationality': {'Italian': '>50K', 'Spanish': '<=50K', 'German': '<=50K'}}
2	{'Nationality': {'Italian': '>50K', 'Spanish': '<=50K', 'German': '<=50K'}}	Nationality	{'Italian': '>50K', 'Spanish': '<=50K', 'German': '<=50K'}	Italian	>50K

APPENDIX D – A table showing the attributes selected as quasi-identifiers for each number of quasi-identifiers.

		NUMBER OF QUASI-IDENTIFIERS						
		Default (6)	7	8	9	10	11	12
ATTRIBUTES	Age	✔	✔	✔	✔	✔	✔	✔
	Work Class	✔	✔	✔	✔	✔	✔	✔
	Education	✔	✔	✔	✔	✔	✔	✔
	Marital Status			✔	✔	✔	✔	✔
	Occupation		✔	✔	✔	✔	✔	✔
	Relationship				✔	✔	✔	✔
	Race					✔	✔	✔
	Sex						✔	✔
	Capital Gain	✔	✔	✔	✔	✔	✔	✔
	Capital Loss							✔
	Hours Per Week	✔	✔	✔	✔	✔	✔	✔
	Native Country	✔	✔	✔	✔	✔	✔	✔

APPENDIX E – A table showing the full results obtained from varying the value of K and varying the number of quasi-identifiers in the anonymisation process.

NO. OF QUASI-IDENTIFIERS	QUASI-IDENTIFIERS	K	ACCURACY (%)	AVERAGE ACCURACY (%, 2 DP)
6	[Age, Work Class, Education, Capital Gain, Hours Per Week, Native Country]	4	80.79	80.87
			80.97	
			80.84	
		8	80.81	80.80
			80.97	
			80.61	
		16	80.47	80.46
			80.61	
			80.3	
		32	79.7	79.87
			79.67	
			80.23	
		64	79.29	79.28
			79.29	
			79.27	
7	[Age, Work Class, Education, Occupation, Capital Gain, Hours Per Week, Native Country]	64	75.84	75.77
			75.84	
			75.63	

8	[Age, Work Class, Education, **Marital Status**, Occupation, Capital Gain, Hours Per Week, Native Country]	64	76.4 74.93 75.56	75.63
9	[Age, Work Class, Education, Marital Status, Occupation, **Relationship**, Capital Gain, Hours Per Week, Native Country]	64	70.75 70.42 72.09	71.09
10	[Age, Work Class, Education, Marital Status, Occupation, Relationship, **Race**, Capital Gain, Hours Per Week, Native Country]	64	71.85 70.55 70.12	70.84
11	[Age, Work Class, Education, Marital Status, Occupation, Relationship, Race, **Sex**, Capital Gain, Hours Per Week, Native Country]	64	70.23 70.88 70.28	70.46
12	[Age, Work Class, Education, Marital Status, Occupation, Relationship, Race, Sex, Capital Gain, **Capital Loss**, Hours Per Week, Native Country]	64	68.26 70.01 70.42	69.6

10. REFERENCES

[1] L. Sweeney *Simple Demographics Often Identify People Uniquely*. Carnegie Mellon University, Data Privacy Working Paper 3. Pittsburgh 2000, p2.

[2] Sweeney, L. (2002). k-anonymity: A Model For Protecting Privacy. International Journal on Uncertainty, Fuzziness and Knowledge-based Systems, 10(5), p3.

[3] Barth-Jones, D. (2012). The "Re-identification" of Governor William Weld's Medical Information: A Critical Re-examination of Health Data Identification Risks and Privacy Protections, Then and Now, p.2.

[4] Barbaro, M. and Zeller Jr., T. (2006). A Face Is Exposed for AOL Searcher No. 4417749. [online] nytimes.com. Available at: http://www.nytimes.com/2006/08/09/technology/09aol.html [Accessed 4 May 2017].

[5] Sweeney, L. (2002). Achieving k-anonymity privacy protection using generalization and suppression. International Journal on Uncertainty, Fuzziness and Knowledge-based Systems, 10(5), p3.

[6] Inan, A., Kantarcioglu, M. and Bertino, E. (n.d.). Using Anonymized Data For Classification, p11.

[7] Aggarwal, C. and Yu, P. (2008). Privacy-Preserving Data Mining: Models and Algorithms. 1st ed. Springer Science+Business Media, p.122.

[8] Archive.ics.uci.edu. (n.d.). UCI Machine Learning Repository: Adult Data Set. [online] Available at: https://archive.ics.uci.edu/ml/datasets/adult [Accessed 4 May 2017].

[9] Keenan, T. (n.d.). R vs. Java vs. Python: Which Is Right for Your Project?. [online] Upwork. Available at: https://www.upwork.com/hiring/data/r-vs-java-vs-python-which-is-best/ [Accessed 4 May 2017].

[10] LeFevre, K., DeWitt, D. and Ramakrishnan, R. (2005). Incognito: Efficient Full-Domain K-Anonymity, p4-p5.

[11] Podgursky, B. (2011). Practical k-anonymity on Large Datasets. Faculty of the Graduate School of Vanderbilt University.

[12] LeFevre, K., DeWitt, D. and Ramakrishnan, R. (n.d.). Mondrian Multidimensional K-Anonymity, p6.

[13] Shao, J. (2016). Data Mining Lecture 4.

[14] Kaur, G. and Amit Chhabra, A. (2014). Improved J48 Classification Algorithm for the Prediction of Diabetes. International Journal of Computer Applications (0975 – 8887), 98(22), p13.

[15] Roach, C. (2006). Building Decision Trees in Python - O'Reilly Media. [online] Onlamp.com. Available at: http://www.onlamp.com/pub/a/python/2006/02/09/ai_decision_trees.html?page=3 [Accessed 4 May 2017].

[16] Sweeney, L. (n.d.). Datafly: a System for Providing Anonymity in Medical Data, p6.

[17] [6] Inan, A., Kantarcioglu, M. and Bertino, E. (n.d.). Using Anonymized Data For Classification, p10.