THE CONSTRUCTION AND EXPLOITATION OF ATTRIBUTE-VALUE TAXONOMIES IN DATA MINING

Hong-Yan Yi

A thesis submitted to the University of East Anglia School of Computing Sciences in fulfilment of the requirements for the degree of Doctor of Philosophy.

February 2012

Statement of Originality

Unless otherwise noted or reference in the text, the work described in this thesis is, to the best of my knowledge and belief, original and my own work. It has not been submitted, either in whole or in part, for any degree at this or any other academic or professional institution.

Dedication

To my parents and my beloved husband

Abstract

With the widespread computerization in science, business, and government, the efficient and effective discovery of interesting information and knowledge from large databases becomes essential. Knowledge Discovery in Databases (KDD) or Data Mining plays a key role in data analysis and has been found to be beneficial in many fields. Much previous research and many applications have focused on the discovery of knowledge from the raw data, which means the discovered patterns or knowledge are limited to the primitive level and restricted to the provided data source. It is often desirable to discover knowledge at multiple conceptual levels, from specific to general, which will provide a compact and easy interpretive understanding for the decision makers. Against this background, this thesis aims to construct and exploit Attribute-Value Taxonomies (AVT) for compact and accurate classifier learning.

For taxonomy construction, we first introduce the concept of an ontology, a scheme of knowledge representation which is domain shareable and reusable. An algorithm is developed to implement the extraction of taxonomies from an existing ontology. Apart from obtaining the taxonomies from the pre-existing knowledge, we also consider a way of automatic generation. Some typical clustering algorithms are chosen to build the tree hierarchies for both nominal and numeric attributes based on the distribution of classes that co-occur with the values. Although this automated approach cannot guarantee each generated taxonomy has the same semantic meanings as manually defined ones, these taxonomies reveal the statistical distribution characteristic of the data, and can be easily transformed to human-understandable forms.

In order to generate much simpler and readable trees and smaller, but more useful, rule sets, we propose methods of using Attribute-Value Taxonomies (AVT) in the decision tree and association rule classifier learning procedure. We illustrate our approach by using the C5 tree induction algorithm, and Apriori association rule algorithm using Receiver Operating Characteristic (ROC) analysis, respectively. We test our approach on two real world data sets from the UCI repository. The experimental results show that the AVT-guided learning algorithm enables us to learn a classifier that is compact but still maintains reasonable accuracy.

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Prof. Vic Rayward-Smith, for his extraordinary support, guidance, and caring through out my PhD journey. This thesis would not have existed without his encouragement and great patience at the hardest times.

I would also like to thank the School of Computing Sciences at the University of East Anglia for providing an excellent research environment and the support of a research studentship to undertake this work.

My thanks also go to Dr. Wenjia Wang for his helpful suggestions during his reviewing my thesis.

Last but definitely not the least, special thanks go to my parents, my husband and my other family members who have been so supportive since the beginning of my studies. I would not have been able to achieve this work without them.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and Motivation

In the last three decades, because of the rapid advancement of computer hardware and computing technology, terabytes of data are generated, stored, and updated every day in scientific, business, government, and other organisational databases. However, as the volume of data increases, the proportion of it that people understand decreases. As Stoll and Schubert [83] have pointed out, "Data is not information, information is not knowledge, knowledge is not understanding, understanding is not wisdom". The growing amount of data is not very useful until we can find the right tool to extract interesting information and knowledge from it. This is more crucial for data analysts, or decision makers who want to make most use of the existing data to make better decisions or take beneficial actions.

Since the early 1990s, Knowledge Discovery in Databases (KDD) has become a dynamic, fast-expanding field to fulfill this need. KDD has strong links with a variety of research areas including machine learning, pattern recognition, databases, statistics, artificial intelligence, knowledge acquisition for expert systems, data visualisation and data warehousing. This powerful process has been applied to a

great number of real world problems, such as targeting customer types with high credit risk or cross-sale potential, and predicting good candidates for a surgical procedure. Success in these and many other diverse areas has spawned an increasing interest in applying knowledge discovery to many other domains.

Our first motivation relates to the ability of learning associated information at different levels of a concept hierarchy or taxonomy.

Much previous research and many applications have focused on the discovery of knowledge from the raw data [5,6,95,108,127], which means that the discovered patterns or knowledge are limited to the primitive level and restricted to the provided data source. Finding rules at a single concept level, even it is a high level, may still not result in discovering desired knowledge, since such knowledge may already be well-known. In practice, it is often desirable to discover knowledge at multiple conceptual levels, from specific to general, which will provide a compact and easy interpretive understanding for the decision makers.

There have been some efforts made to mine multiple level association rules [27, 50, 62]. To extract multi-level association rules, concept hierarchies or item taxonomies are needed. A concept hierarchy is modelled by a directed acyclic graph (DAG) whose nodes represent items and arcs represent "is-a" relations between two items. Concept hierarchies/taxonomies represent the relationships of generalization and specification between the items, and classify them at several levels of abstraction. In the transaction databases, these concept hierarchies are available or easily established, since either the lower-level concepts or the higher-level concepts exist in the same database. For example, particular brands of milk or bread, e.g. "Dairyland" and "Hovis" (brand names), are bottom level instances, whilst "Milk" and "Bread" are higher level concepts.

The use of concept hierarchies can benefit data mining. This is because the discovery of interesting knowledge at multiple abstraction levels broadens the scope

of knowledge discovery and helps users progressively focus on relatively interesting "hot spots" and deepen the data mining process. However, there are still some issues that may be of concern:

- The concept hierarchies "hidden" in the databases may not be complex enough. Generally, the hierarchies of transaction concepts have just two or three levels. For example, the hierarchy of "Bread" only contains high level nodes, such as "white", "brown", "wheat", etc., and low level nodes, such as "Hovis", "Kingsmill", "Warburtons", etc. Thus, the discovered rules may be either too specific or too general.

- For real world data, unlike transaction data, it may be difficult to form concept hierarchies, since the data may only occur at a single concept level. However, for nominal fields or attributes, if the number of distinctive values are big enough, we can treat these distinctive values as individual concepts, and build a hierarchy for them, which is called an attribute-value taxonomy in this thesis.

- There are some predefined concept hierarchies or taxonomies available. It is worthwhile to exploit such prior knowledge and construct more suitable taxonomies for the specific applications.

Our second motivation is triggered by the increasing demand of designing and constructing ontologies for information sharing between organizations, domain experts and scientific communities.

When our research started, ontology was becoming a hot topic in the area of bioinformatics, semantic web, etc. Ontologies provide a rich repository of semantic information and relations. An ontology might encompass a number of taxonomies, with each taxonomy organizing a subject in a particular way. From the technical point of view, a taxonomy is often referred as a "tree", while an ontology is often

more of a "forest". Moreover, due to the shareable and reusable character of an ontology, it can help to facilitate information integration and data mining from heterogeneous data sets.

There are two research directions for studying ontologies in this thesis:

1. Manually design and present ontologies using ontology language based on the background knowledge. Maintaining ontology is a long term work which requires improving and updating its contents so that it adequately captures all possible activities across domains.

2. Automatically extract taxonomies from the existing ontology. This is more feasible and realistic when seeking a way of combining ontologies with data mining process.

1.2 The Aims and Research Design

The overall aim of this research is to develop a methodology that exploits an attribute-value taxonomy for compact and accurate classifier learning. In this thesis, two main issues are addressed:

1. How to construct the attribute-value taxonomy (abbreviated as AVT).

2. How to exploit AVT for learning simpler decision trees and smaller sets of association rules without sacrificing too much accuracy.

1.2.1 Attribute-Value Taxonomy Construction

In order to construct an attribute-value taxonomy, we attempted two different approaches. The first one is to automatically extract taxonomies from a predefined ontology, assuming that there is an existing ontology available for the specific domain application. The second one is to automatically generate an AVT by using

some typical clustering algorithms. Apart from producing dendrograms for nominal attribute values using hierarchical clustering algorithms, we also explored an approach to generate such taxonomies by using partitional algorithms. In general, partitional algorithms will only split the data into a specific number of disjoint clusters. We propose a bottom-up iterative approach to construct the attribute-value taxonomies.

1.2.2 Exploitation of AVT for Classifier Learning

Once the AVT is constructed for an attribute, the representation space of this attribute will no longer be limited to the concepts at a primitive level. The whole space will be greatly extended with concepts at different abstraction levels. We use the term, *Cut*, to present a possible concept tuple, which contains the concepts at different levels, and covers the space without any intersections among the included concepts. In other words, for any non-primitive level concept in a concept tuple, neither its descendant nor its ancestor will occur in this tuple. We are interested in AVTs that are useful for generating accurate and compact classifiers. However, due to the large number of possible cuts, it is not computationally feasible to apply all cuts to build decision tree classifier and learn association rules. In this thesis, the method of using *gain ratio* to rank and select the top five cuts is proposed. We illustrate this methodology with two real world data sets and the experimental results reveal the expected improvements can be achieved.

1.2.3 Data Sources

Since we aim to exploit AVT for generating more readable and interpretative decision tree and rules, we are more interested in applying our approach to nominal attributes. There are two necessary criteria for the database that we select to illustrate our methodology.

1. The nominal attributes are dominant, not only among the data set attributes, but also among the decision nodes of generated decision trees.

2. The number of the distinctive values of the nominal attribute is large enough to form at least a two level concept hierarchy, which is also called attribute-value taxonomy.

In this thesis, two real world data sets, obtained from the UCI data repository [12] and meeting the two criteria, are used in our experiments for performance evaluation.

1. The *Adult* data set contains information on various individuals, totally 6 numeric and 8 nominal attributes, such as education, marital status, occupation, etc. The income information is simply recorded as either more than \$50K or less than \$50K. The distribution of instances is that 75.22% of people (34,014 instances) earn less that \$50K per year, and the rest 11,208 instances (24.78%) belong to the other class. The *Adult* data is randomly split into training and test data, accounting for 2/3 (30,162 instances) and 1/3 (15,060 instances) respectively, once all missing and unknown data are removed. See appendix B.1 for details.

2. The *Mushroom* data set contains 22 nominal attributes. The target class attribute establishes whether an instance is edible or poisonous. There are 8,124 instances in the data set; 51.8% of mushrooms (4,208 instances) are edible and 48.2% (3,916 instances) are poisonous. The *Mushroom* data are also randomly split into training and test data, accounting for 90% and 10% respectively. See appendix B.2 for details.

1.3 Thesis Overview

The remainder of this thesis is organized as follows.

In Chapter 2, a brief introduction to KDD and related techniques used in the thesis are given. Preliminary concepts of taxonomy and attribute-value taxonomy are also provided. A brief survey of the related work on learning decision trees and rule-based classifiers by using AVT is given.

In Chapter 3, we review some important issues about ontologies, such as definition, types, representation languages and existing applications. We present an algorithm for automatically extracting taxonomies from a predefined ontology and illustrate the technique with some examples.

In Chapter 4, two typical hierarchical clustering algorithms and two partitional algorithms are introduced and used to perform the automatic generation of an attribute-value taxonomy. Two real world data sets are chosen for performance evaluation.

In Chapter 5, we describe an approach that is able to exploit user supplied attribute-value taxonomies (AVTs) to learn simpler decision tree with reasonable classification accuracy. Since the complex taxonomies may generate a large number of cuts, this will easily cause the increase of computing resource, and thus affect the performance of tree induction algorithm. Some preprocessing efforts need to be done to properly select the number of top ranked cuts for AVTs according to some constrains. We present experiments on two data sets for performance comparison with a standard decision tree learning algorithm.

In Chapter 6, we develop an approach to integrate the AVTs with rule learning into a framework to train a rule based classifier. The work starts with training a rule based classifier using the ROC analysis method. Then we apply the attribute-value taxonomies to the refinement of the learned rule set. The use of AVTs is based on two different measurements, *Gain Ratio* and the number of learned rules,

which can be used to select the possible optimal cuts over the selected attribute-value taxonomies.

We conclude the thesis work in Chapter 7. A summary and the conclusions from the study are given. Some interesting future research problems are also addressed.

Chapter 2

Preliminary Concepts and Related Work

This chapter gives a brief introduction to KDD and related techniques we will use in the thesis. The concept of taxonomy and attribute value taxonomy are also introduced. We finally examine the related approaches in the literature on taxonomy construction and learning classifiers from data by using attribute value taxonomies.

2.1 Overview of KDD and Data Mining

2.1.1 Definitions

In general, *Knowledge Discovery in Databases* (KDD) has been considered as the overall process of discovering useful knowledge from data. In this context, a widely accepted definition of KDD, extracted from [37] is:

"*Knowledge Discovery in Databases* is the automatic non-trivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data."

As a part or a stage of the KDD process, the definition of Data Mining given in [37] is:

"*Data Mining* is a step in the KDD process consisting of particular data mining algorithms that, under some acceptable computational efficiency limitations, produce a particular enumeration of patterns over the data."

2.1.2 KDD Process

The KDD Process is a highly iterative, user involved, multi-step process. According to Fayyad, as shown in figure 2.1, the overall KDD process can be separated into five steps: **Selection**, **Pre-processing**, **Transformation**, **Data Mining** and **Interpretation**. These five steps are passed through iteratively. Every step can be seen as a work-through phase. Such a phase requires the supervision of a user and can lead to multiple intermediate results. The "best" of these human evaluated results is used for the next iteration, the others should be documented. The brief descriptions of these steps are given as follows:

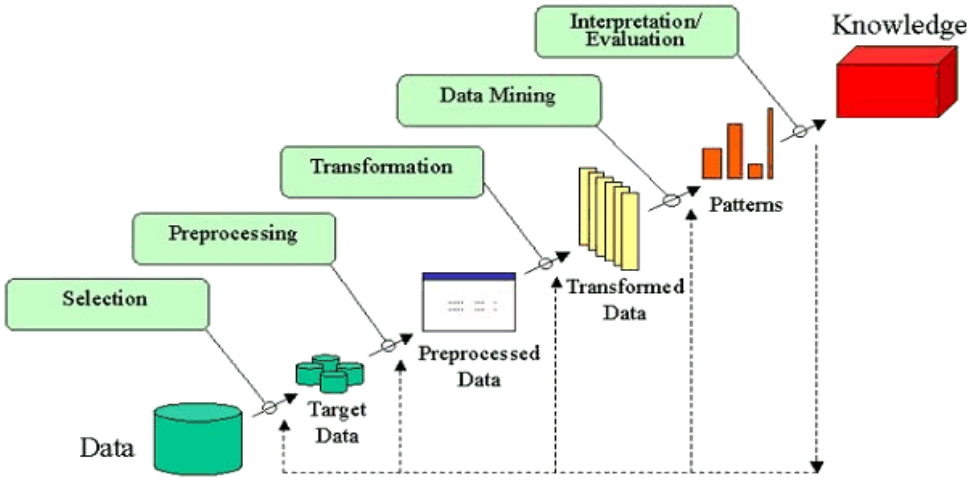


Figure 2.1: Steps of the KDD process (Fayyad et al. 1996)

Step 1 - Data Selection:

The goal of this phase is the extraction from a larger data store of only the

data that is relevant to the data mining analysis. This data extraction helps to streamline and speed up the process.

Step 2 - Data Preprocessing:

This phase of KDD is concerned with data cleansing and preparation tasks that are necessary to ensure correct results. Eliminating missing values in the data, ensuring that coded values have a uniform meaning and ensuring that no spurious data values exist are typical actions that occur during this phase. 3) Data Transformation This phase of the lifecycle is aimed at converting the data into a two-dimensional table and eliminating unwanted or highly correlated fields so the results are valid.

Step 3 - Data Transformation:

This phase of the lifecycle is aimed at converting the data into a two-dimensional table and eliminating unwanted or highly correlated fields so the results are valid.

Step 4 - Data Mining:

The goal of the data mining phase is to analyse the data by an appropriate set of algorithms in order to discover meaningful patterns and rules and produce predictive models. Various qualities of the knowledge discovered considered relevant to the particular application are measured (e.g. accuracy, interest, etc.). The whole process is repeated until knowledge of adequate quality is obtained. Part of the iteration may involve going over previous steps of the KDD process, and consulting with the users or domain experts.

Step 5 - Interpretation and Evaluation :

Once the knowledge extraction has taken place and a set of results of acceptable quality is obtained, the detected pattern needs to be interpreted to determine whether or not it is interesting. This would normally require some

involvement on the part of the user/domain expert. In some cases, the interpretation of certain qualities (e.g. novelty, interest) will be delayed to this step, so iteration to previous steps for another attempt may be necessary.

2.1.3 Data Mining Tasks

The two "high-level" primary goals of data mining are *prediction* and *description*, referred to [37].

1. **Prediction** involves using some variables or fields in the database to predict unknown or future values of other variables of interest.

2. **Description** focuses on finding human-interpretable patterns describing the data.

The following four primary data mining tasks are usually used for prediction and description:

- **Classification** is learning a function that maps (classifies) a data item into one of several predefined classes. Common algorithms include decision tree learning, nearest neighbour, naive Bayesian classification, neural networks and support vector machines.

- **Clustering** is a common descriptive task where one seeks to identify a finite set of categories or clusters to describe the data.

- **Dependency Modelling (Association rule learning)** consists of finding a model which describes significant dependencies between variables.

- **Regression** is learning a function which maps a data item to a real-valued prediction variable.

In this thesis, our work involves the first three tasks. The following two sections provide a basic introduction to the algorithms of classification and association rule learning we will use in later chapters. A detailed introduce to some related techniques of clustering will be given in chapter 4.

2.2 Decision Tree Introduction

A decision tree is a tree structured classifier, where each node is either a *decision node* that represents a test on one input attribute, with one branch for each possible test result, or a *leaf node* that indicates a value of a target class, i.e. a class the objects are to be classified.

Given an object in a tuple of input attribute values, it is classified by following a path from the root down to a leaf node, which is formed by branches corresponding to the results of the tests applied to the object when visiting the decision nodes.

There are two types of decision trees: one is *classification tree* if the target class is categorical; the other is *regression tree* if the target class is continuous. In this thesis, we restrict our study to classification trees [141].

2.2.1 Decision Tree Building

Building a decision tree is all a matter of choosing which attribute to test at each non-leaf node in the tree. There are exponentially many decision trees that can be constructed from the input attributes, in which some trees are more accurate than others. Given time consumption, finding a reasonably accurate decision tree, instead of the optimal one, is generally more feasible. To fulfil this purpose, most tree induction algorithms employ a top-down, greedy approach to induce trees. A basic recursive procedure for decision tree building is described as follows.

Let $D = \{x_1, x_2, ..., x_n\}$ be the set of training data, and $C = \{C_1, C_2, ..., C_k\}$

be the set of target class to be classified. The attribute set is denoted by $A = \{A_1, A_2, ..., A_m\}$.

1. If all the instances in D belong to the same class C_t $(1 \leq t \leq k)$, then create a leaf node C_t and stop.

2. If D contains instances belong to more than one class, then search for a best possible test, T, on attribute A_i $(1 \leq i \leq m)$ according to some splitting measure, and create a decision node.

3. Split the current decision node, i.e. partition D into subsets, using that test.

4. Recursively apply the above three steps to each subset of D.

Here, the choice of "best" test is what make the algorithm greedy, and it normally maximises the chosen splitting criterion at each decision node, so it is only a locally optimal choice. Some popular splitting measures will be introduced in the next section.

When a training set is rather small or there is some "noise" in the data (where "noise" means instances that are misclassified or where some attribute values are wrong), pursuing high classification accuracy may produce a complex decision tree that over-fits the training data. As a result, the test data or unseen data may not be well classified. To solve this problem, tree pruning is a necessary procedure to avoid building an overfitting tree. This will be discussed in section 2.2.3.

2.2.2 Splitting Measures

There are many measures that can be used to select the best split. These measures are often based upon the impurity of the nodes. The greater the difference of impurity between the parent nodes (before splitting) and their child nodes (after splitting), the better the test condition performs. For a classification tree, the

impurity is defined in terms of the class distribution of the records before and after splitting.

Let S denote the subset of data that would be considered by a decision node. The given target class partitions S into $\{S_1, S_2, \ldots, S_n\}$, where all the records in S_i have the same categorical value. Now, say a test, T, on an attribute with k values partitions S into k subsets, $\{S'_1, S'_2, \ldots, S'_k\}$, which may be associated with k descendent decision nodes. Then three commonly used impurity measures and the corresponding gains used as splitting criterions are defined as follows:

- **Information/Entropy Gain**:

$$Gain_{Info}(S,T) = Entropy(S) - \sum_{j=1}^{k} \frac{|S'_j|}{|S|}\ Entropy(S'_j), \tag{2.1}$$

$$Entropy(S) = -\sum_{i=1}^{n} \frac{|S_i|}{|S|} \log_2 \frac{|S_i|}{|S|}. \tag{2.2}$$

- **Gini Index**:

$$Gain_{Gini}(S,T) = Gini(S) - \sum_{j=1}^{k} \frac{|S'_j|}{|S|}\ Gini(S'_j), \tag{2.3}$$

$$Gini(S) = 1 - \sum_{i=1}^{n} \frac{|S_i|^2}{|S|^2}. \tag{2.4}$$

- **Classification Error**:

$$Gain_{Error}(S,T) = Error(S) - \sum_{j=1}^{k} \frac{|S'_j|}{|S|}\ Error(S'_j), \tag{2.5}$$

$$Error(S) = 1 - max\{\frac{|S_1|}{|S|}, \frac{|S_2|}{|S|}, \ldots, \frac{|S_n|}{|S|}\}. \tag{2.6}$$

No matter which impurity measure is chosen, the impurity of the parent node

is always a constant for all test conditions, so maximising the gain is equivalent to minimising the weighted average impurity of the child nodes.

2.2.3 Tree Pruning Methods

When building a decision tree, a tree pruning step can not only be performed to simplify the tree by reducing its size, but also help handle the overfitting problem. Two techniques, pre-pruning and post-pruning, are used to do the pruning.

- **Pre-pruning** works by setting a stop condition to terminate the further split during the tree construction. For example, if the observed gain in some impurity measure, or the number of records at some decision nodes falls below a predefined threshold, then the algorithm will stop expanding their child nodes.

- **Post-pruning** works by pruning some branches from the fully grown tree. Pruning can be done by replacing, for example, a subtree with a new leaf node whose class label is determined by the majority class of records occurred in the subtree, or the most frequently used branch, i.e. the branch has more training records than others, of the subtree. The tree pruning step terminates when no further improvement is observed [132].

In practice, post-pruning is more successful, since pre-pruning may lead to premature termination of the tree growth if the threshold is not set appropriately. However, additional computations are needed in order to grow the full tree for post-pruning, which may be wasted when some subtrees will be pruned finally.

2.2.4 Evaluating Decision Tree

There are many approaches to decision tree construction, which lead to quite different trees being produced. Generally, we will be interested in how accurately an

algorithm can predict or how concise the generated decision tree is. The following three measures can be used to evaluate the quality of the tree.

- **Classification Accuracy**: the percentage of those instances in the data for which, given the values of the input attributes, the tree classifies them to the correct target class.
- **Tree Size**: the total number of nodes in the tree, while some others, e.g. Murthy [104], define it as the number of leaf nodes.
- **Maximum Depth**: the number of tree levels from the root to the farthest leaf node.

Obviously, the higher the classification and prediction accuracy, the better quality the tree has. Apart from accuracy, simpler tree with less tree nodes or depth may imply better interpretability and computational efficiency.

2.3 Rule Learning Introduction

A rule learning system aims to construct a set of if-then rules. An if-then rule has the form: IF $<$*Conditions*$>$ THEN $<$*Class*$>$. Conditions contains one or more attribute tests, usually of the form "$A_i = v_{ij}$" for nominal attributes, and "$A_i < v$" or "$A_i \geq v$" for continuous attributes. Here, v is a threshold value that does not need to correspond to a value of the attribute observed in examples.

Mining association rules is an important technique for discovering meaningful patterns in databases. Formally, the problem can be formulated as follows [4]. Let $A = \{A_1, A_2, \cdots, A_m\}$ be a set of m binary attributes called *items*. Let $D = \{x_1, x_2, \cdots, x_n\}$ be a set of records called the *database*. Each record in D has a unique ID and contains a subset of the items in A [59]. An association rule is a rule of the form $Y \leftarrow X$, where $X, Y \subseteq A$, and X, Y are two disjoint sets of

items. It means that if all the items in X are found in a record then it is likely that the items in Y are also contained in the record. The sets of items X and Y are respectively called the *antecedent* and *consequent* of the rule [59]. To select interesting rules from the set of all possible rules, constraints on various measures of significance and strength can be used. The best-known constraints are minimum thresholds on support and confidence [5].

$$supp(Y \leftarrow X) = supp(X \cup Y) = \frac{C_{XY}}{M} \tag{2.7}$$

$$conf(Y \leftarrow X) = \frac{supp(X \cup Y)}{supp(X)} = \frac{C_{XY}}{C_X} \tag{2.8}$$

where C_{XY} is the number of records which contain all the items in X and Y, C_X is the number of record containing the items in X, and M is the number of records in the database.

Support, in Equation 2.7, is defined as the fraction of records in the database which contain all items in a specific rule [4]. *Confidence*, in Equation 2.8, is an estimate of the conditional probability $P(Y|X)$.

An association mining problem can be decomposed into two sub-problems:

- Find all combinations of items in a set of records that occur with a specified minimum frequency. These combinations are called **frequent itemsets**.
- Calculate rules that express the probable co-occurrence of items within frequent itemsets.

Apriori is the widely used algorithm for calculating the probability of an item being present in a frequent itemset, given that another item or items are present. **Apriori** discovers patterns with frequency above the minimum support threshold, which express probabilistic relationships between items in frequent itemsets. For

example, a rule derived from frequent itemsets containing A, B, and C might state that if A and B are included in a record, then C is likely to also be included.

2.4 Introduction to Taxonomy

Etymologically speaking, *taxonomy* comes from the Greek term *"taxis"* (meaning arrangement or division) and *"nomos"* (meaning law). Modern taxonomy originated in the mid-1700s when a Swedish botanist, Carl Linnaeus, named with the term taxonomy the classification of living beings into hierarchical groups, ordered from the most generic to the most specific (kingdom, type, order, gender, and species).

By the beginning of the 1990s, taxonomy was being used in many fields of knowledge, such as psychology, social sciences and information technology, to name almost all the access systems to the information that attempt to establish coincidences between the terminology of the user, and that of the system.

2.4.1 Definitions

There are various definitions of taxonomy defined by people from different areas. We list following two typical definitions.

- A taxonomy is a scheme that partitions a body of knowledge and defines the relationships among the pieces [107]. It is used for classifying and understanding the body of knowledge. Such body of knowledge may refer to almost anything, i.e., animate objects, inanimate objects, places, or events.

- A taxonomy is typically a controlled vocabulary with a hierarchical structure, with the understanding that there are different definitions of a hierarchy. Terms within a taxonomy have relations to other terms within the taxonomy.

These are typically: parent/broader term, child/narrower term, or often both if the term is at mid-level within a hierarchy.

There are some other names for taxonomy in the literature, for example, concept hierarchy or is-a hierarchy, structured attribute.

2.4.2 Attribute Value Taxonomy

In a typical inductive learning scenario, instances to be classified are represented as ordered tuples of attribute-values. However, attribute values can be grouped together to reflect assumed or actual similarities among the values in a domain of interest or in the context of a specific application. Such a hierarchical grouping of attribute values yields an attribute value taxonomy (AVT).

As taxonomies are often displayed as a tree structure, the terms within a taxonomy are called "nodes". It can be formally defined as follows.

A *taxonomy*, T, a finite set of concept C, is a tree structured hierarchy in the form of a poset (partially ordered set) $(T, \prec)$, where $\prec$ is the partial order that presents "is-a" relationship on C.

Hence if $A = \{A_1, A_2, ..., A_m\}$ represent a set of nominal attributes, and V_i is the value set of A_i. Then we can construct a set of attribute-value taxonomies, $T = \{T_1, T_2, ..., T_m\}$, where each T_i has leaf node set equal V_i.

2.4.3 Reasons to use Attribute Value Taxonomy

- The availability of AVTs presents the opportunity to learn classification rules that are expressed in terms of abstract attribute values leading to simpler, easier-to-comprehend rules that are expressed in terms of hierarchically related values.

- Exploiting information provided by AVTs can potentially perform regularization, like the shrinkage [96] technique used by statisticians when estimating from small samples, to minimise overfitting when learning from relatively small data sets.

2.4.4 Processes for the construction of taxonomies

Taxonomies can be constructed either manually or automatically. In this section, we will only give a brief introduction to manual construction of taxonomies. How to automatically construct attribute-value taxonomies will be discussed in chapter 3 and 4.

Traditionally, the manually constructed taxonomies are totally dependent on the taxonomist's intuition, subjective decision, experience, skill, and perhaps insight. Two distinguished techniques for the development of the structure of taxonomy are the *up to down* technique and the *down to up* technique [26].

- The application of the *up to down* technique involves the initial identification of a limited number of higher categories, and the grouping of the rest of categories in successive levels of subordination down to the most specific levels of categories. This technique can be especially useful with a well understood application domain and is particularly possibility applicable to the construction of taxonomies for the development of browsing systems.

- The application of the *down to up* technique is based on the initial identification of the most specific categories, which are grouped in successive levels of subordination up to higher levels of categories.

2.5 Related Work on Automatically Learning Taxonomy from Data

Some previous work [28, 29, 46, 66, 73, 74, 88, 103, 105, 117, 126, 134] has explored the construction of taxonomies. Some of this research has incorporated clustering techniques. Ganti et al. [46] designed CACTUS, an algorithm that uses intra-attribute summaries to cluster attribute values. Cimiano et al. [28] used clustering methods to learn taxonomies from text information. Kang et al. [74] implemented AVT-learner, a hierarchical agglomerative clustering algorithm to construct AVTs for learning. Yi et al. [142] used a partitional clustering algorithm to build a cluster hierarchy, which is exploited to guide ontology construction.

Some work utilised specific measurement [94, 105] or prior knowledge, such as tagging information [134] and structure information [117, 140]. Murthy et al. [103] constructed a taxonomy using term frequency to generate a natural hierarchy. In [66], a taxonomy is generated automatically using the Heymann algorithm, which determines the generality of terms and iteratively inserts them in a growing taxonomy. Punera et al. [117] explored the construction of a taxonomy by learning n-ary tree based hierarchies of categories with no user-defined parameters, and Wu et al. [140] proposed to learn from data using abstractions over the structured class labels as a generalization of single label and multi label problems. Joo et al. [73] used a genetic algorithm to generate an AVT. Neshati et al. [105] developed a method to use compound similarity measure for taxonomy construction, and Markrechi et al. [94] used the new measure of information theoretic inclusion index, term dependency matrix. Tsui et al. [134] provided a novel approach for generating Taxonomy using tags.

In addition, some methods also make use of an ontology to form a taxonomy. Welty et al. [138] adopted several notions from formal ontology and adapted them

to provide a solid logical framework within which the properties that form a taxonomy can be analysed. This analysis helps make intended meaning more explicit, improving human understanding and reducing the cost of integration. Guarino et al. [57] concentrated on the ontological nature in order to be able to tell whether a single is-a link is ontologically well-founded, and discussed techniques based on the philosophical notions of identity and dependence for information systems design. To reduce the cost of working with large ontologies, Shearer et al. [126] presented a classification algorithm to exploit partial information about subclass relationships.

The construction of taxonomies has also been used in visual information and text processing. Setia et al. [125] proposed to learn a visual taxonomy, given only a set of labels and a set of extracted feature vectors for each image, to enhance the user search experience. Li et al. [88] used relational clustering framework DIVA for document summarisation.

2.6 Related Work on Learning Classifiers Using an Attribute Value Taxonomy

An important goal of inductive learning is to generate accurate and compact classifiers from data. In the machine learning field, some previous work on the problem of learning classifiers from the attribute-value taxonomies has also been explored in the case of decision tree and rule induction.

2.6.1 Tree Induction Using AVT

To handle taxonomies, Quinlan extended C4.5 decision tree by introducing nominal attributes for each level of the hierarchy [119]. Taylor et al. [133] employed an evaluation function in decision tree learning to effectively use ontology within data mining system.

Pereira et al. described distributional clustering for grouping words based on class distributions associated with the words in text classification. Slonim and Tishby [128] described a technique (called the agglomerative information bottleneck method) which extended the distributional clustering approach. Pereira et al. [110] used Jensen-Shannon divergence for measuring distance between document class distributions associated with words and applied it to a text classification task. Baker and McCallum [14] reported improved performance on text classification using a technique similar to distributional clustering and a distance measure, which upon closer examination, can be shown to be equivalent to Jensen-Shannon divergence. Dimitropoulos et al. [35] augmented Internet Autonomous System (AS) Taxonomy to the data set.

Berzal [18] proposed to use multi-way splits for continuous attributes in order to reduce the tree complexity without decreasing classification accuracy. Kang et al. [76] exploited a taxonomy of propositionalised attributes as prior knowledge to generate compact decision trees. Zhang and Honavar designed and implemented AVT-NBL [75] and AVT-DTL [143] for learning AVT-guided Naïve Bayes and Decision tree classifiers.

To the best of our knowledge, although some work claimed they have implemented experiments on a broad range of benchmark data sets, the AVTs available for exploitation are rather simple (many of them are just two level taxonomies). It is also not clear what kind of AVTs are used and at which level the tuple of nominal values are used for decision tree learning, since only the accuracy and number of leaf nodes are reported.

2.6.2 Rule Induction Using AVT

Han et al. [25] proposed an approach to find characterization and classification rules at high levels of generality using concept taxonomies. Han and Fu [61] fur-

ther explored hierarchically structured knowledge for learning association rules at multiple levels of abstraction. In [137], Vasile extended RIPPER, a widely used rule-learning algorithm, to exploit knowledge in the form of taxonomies over the values of features used to describe data. Borisova et al. [21] offered a way to solve the construction of taxonomies and decision rules with a function of rival similarity (FRiS-function). Zhang et al. [144] introduced an ontology-driven decision tree learning algorithm to learn classification rules at multiple levels of abstraction. However, these methods do not consider how to combine the rule learning with the construction of attribute value taxonomies.

In the case of processing a data set with many nominal attributes with large number of values, the number of cuts and hence the number of tests to be considered will grow exponentially. To deal with this problem, Almuallim et al. [9] use information gain to choose a node in an AVT for a binary split, and further in [10], multiple split tests were considered, where each test corresponds to a cut through AVT. However these methods still did not consider how to combine the solution of this problem with classifier learning.

To deal with these problems, our work, in this thesis, not only focuses on the construction of an AVT using automatic methods, but also considers how to effectively integrate the construction of an AVT with the machine learning techniques into one framework to improve the performance of a decision system.

Chapter 3

Extracting Taxonomies from Ontologies

The construction of taxonomies in a particular domain is a time and labour consuming task, and often cannot be reused by other domains, which stimulates us to find an automatic way for taxonomy construction and hopefully it can be reusable in domains.

Ontology is a scheme of knowledge representation, originating from philosophy, and has been gradually adopted by the researchers in the area of computer science. Since taxonomies are central components of ontologies, and ontologies are domain shareable and reusable, we propose an algorithm to automatically extract them from existing ontologies in the latter section of this chapter.

3.1 Ontologies Overview

3.1.1 What are ontologies?

Originally ontologies was a term used in the philosophical discipline. Merriam Webster dictionary(1721) provides two definitions (1) a branch of metaphysics

concerned with the nature and relations of being and (2) a particular theory about the nature of being or the kinds of existents. Later it became the topic of academic interest among philosophers, linguists, librarians, artificial intelligence, and most recently, of knowledge representation researchers. Now there are diverse areas in computer science that *Ontologies* can be applied to, for instance, knowledge representation [11, 54, 129], natural language translation [84, 93] or processing [15], database design [24, 135], information retrieval and extraction [55, 97], knowledge management and organization [111], electronic commerce [86], semantic web research [8].

There are many forms of specifications that different people termed ontologies. One widely cited definition of an ontology in the knowledge representation area is Gruber's [53]. He stated that "An ontology is an explicit specification of a conceptualization." Here the conceptualization means the objects, concepts, and other entities that are assumed to exist in some area of interest and the relationships that hold among them [48]. Or more simply, "An ontology defines a common vocabulary for researchers who need to share information in a domain. It includes machine-interpretable definitions of basic concepts in the domain and relations among them" [106].

McGuinness [98] refined the various definitions to a linear ontology spectrum showed in figure 3.1 below, which is very helpful for us to understand the real meaning of the ontologies.

From figure 3.1, we can visualize the evolution of ontology from the simple form (left points) with few functions to the formal and more complex form (right points). The following explanation for this ontology spectrum is a rewritten description following McGuinness.

The points to the left of the oblique line cannot be called an ontology, but they have some original properties of ontologies.

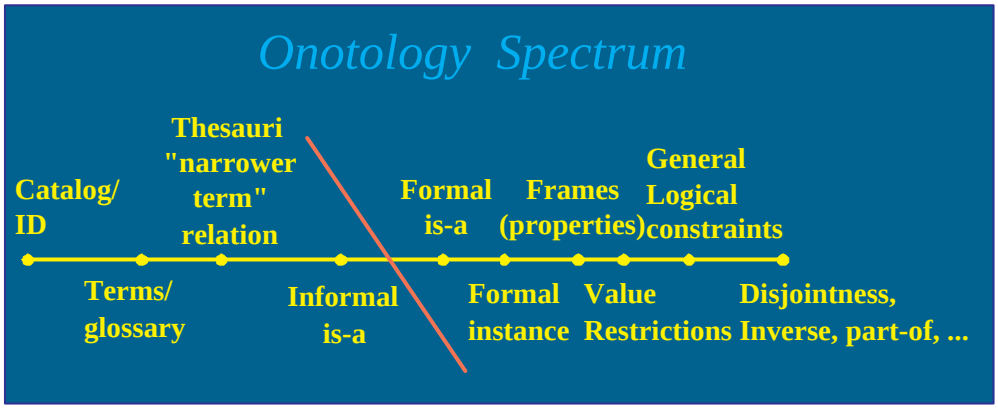


Figure 3.1: Ontology Spectrum (McGuinness 2002)

- The first point on the left side, represented by a controlled vocabulary, which is a finite list of terms, may be one of the simplest notations and is a prototype of a possible ontology. Catalogues, as an example, can provide an unambiguous interpretation of terms.

- A glossary presented as a list of terms with meanings may be another potential ontology specification. But the meanings in the glossary are typically specified using natural language statements, which are not unambiguous and thus could not be easily processed by machine.

- Thesauri, the third point from left, provide some additional semantics in their relations between terms, such as synonym relationships, but do not provide an explicit hierarchy.

- Early web specifications of term hierarchies, such as Yahoo's, provide a basic notion of generalization and specialization. However, Yahoo's hierarchy is not a strict subclass or "is-a" [22], while it supports the case that an instance of a more specific class is also an instance of the more general class [98], which may be described as an informal "is-a" relationship.

The points to the right of the oblique line meet at least the basic conditions required by a simple ontology defined by McGuinness [98], and show how ontologies

can become more and more expressive.

- The first two ontologies include strict subclass hierarchies which are necessary for exploitation of inheritance, i.e. if A is a superclass of B, then if an object x is an instance of B it necessarily follows that x is an instance of A.

- Frames or properties mean the ontology could include property information for its concepts or classes. For example, "isMadeFrom" and "Price" are two common properties for consumer products.

- The ontology also can include value restrictions on a property. For instance, the "Price" property might be restricted to a lie within a given range.

- More expressive ontologies may also include logic constraints between terms and more detailed relationships such as disjointness, inverse, part-whole, etc.

3.1.2 Why develop Ontologies?

Ontology borrows ideas from object-oriented software engineering, which allow people to understand, share and reuse domain knowledge more quickly and more conveniently. Once an ontology has been defined, it can be reused, inherited, or modified by other researchers from the same or different domain. Also, some common operations, or tasks, or configurations, i.e. so-called operational knowledge, can be separated from domain knowledge and become independent, so that they can be used by different ontologies, avoiding having to be repeatedly defined by each of them. For example, the "made-to-order" [123] operation could be shared by all components ontologies for industry products. Moreover, analyzing domain knowledge is possible once a declarative specification of the terms is available. Formal analysis of terms is extremely valuable when both attempting to reuse existing ontologies and extending them [99].

In short, the following benefits, described by Noy [106], can be obtained when using ontologies.

- Sharing common understanding of the structure of information among people or software agents
- Enabling reuse of domain knowledge
- Making domain assumptions explicit
- Separating domain knowledge from the operational knowledge
- Enabling the analysis of domain knowledge

3.1.3 Components of an Ontology

In this thesis, we will follow the criterion of simple ontologies defined by McGuinness [98] when building an ontology as a case study in the later section, and we also use Gómez-Pérez's [49] description about the components[1] of an ontology as follows, adding the explanations and some examples to each component .

- **Concepts** are organized in tree-like taxonomies with tree nodes $C_1, C_2, \ldots, C_n$. Each C_i can be a object in the real world or a concept which corresponds to a set of objects.

  e.g. The concept "*line*" corresponds to all lines in the real world.

- **Relations** are implicit or explicit relations among concepts, i.e. the relations are inferred from the taxonomy or independent of the taxonomy.

  A relation, R, of arity k, can be denoted as a set of tuples of the form

  $$R(C_{\lambda 1}, C_{\lambda 2}, \ldots, C_{\lambda k}), \text{ where } \lambda i \in \{1, ..., n\},\ 1 \leq i \leq k.$$

  e.g. *Subclass-of*(Concept1, Concept2) is a binary implicit relation inferred

[1]Note: According to McGuinness's definition, "Functions", "Instances", and "Axioms" are not mandatory for a simple ontology.

from the ontology;

Connected-to(Component1, Component2) could be an explicit relation defined using an explicit statement within the ontology.

- **Functions** are properties that relate some concepts to other concepts, and are denoted by

  $$f\colon C_{\lambda 1} \times C_{\lambda 2} \times \ldots \times C_{\lambda k} \mapsto C_n, \quad \text{where } \lambda_i \neq n,\ 1 \leq i \leq k.$$

  e.g. *Mother-of*: Person $\mapsto$ Female,

  Price of a used car: Model $\times$ Year $\times$ Kilometers $\mapsto$ Price.

- **Instances** relate objects of the real world as a concept.

  e.g. *Instance-of*: Margaret Thatcher $\mapsto$ Prime Minister,

  Instance-of: Margaret Thatcher $\mapsto$ Female.

- **Axioms** are sentences describing the concepts with logic expressions which are always true.

  e.g. *Parallel(line1, line2)* $\Rightarrow$ *Disjoint(line1, line2)*

  Here, "*Parallel*" is a relation between two instances of the concept "*Line*", and "*Disjoint*" is the property that always exists for two parallel lines.

3.1.4 Types of Ontologies

Nowadays, an increasing number of ontologies have been developed by different organizations and are available on the web. They are classified into many different kinds, such as *Top-Level ontologies*, *Meta or Core ontologies*, etc., see [49,56]. Here we follow the most commonly used classification in [131, 136]. According to their different generality levels, ontologies can be distinguished into following four main types:

- *Generic ontologies* describe very general and basic concepts across domains, e.g. state, event, process, action, component, etc., which are independent

of a particular domain. A generic ontology is also referred to as a *Core ontology* [136] or as a *Top-Level ontology* [56].

- *Domain ontologies* capture the knowledge valid for a particular type of domain, and are reusable within the given domain, e.g. electronic, biological, medical, or mechanical.

- *Representation ontologies* provide a representational framework without stating what should be represented and do not refer to any particular domain. Domain ontologies and generic ontologies are described using the primitives provided by representation ontologies. An example in this category is the Frame Ontology used in Ontolingua [53], which defines concepts such as frames, slots and slot constraints allowing expressing knowledge in an object-oriented or frame-based way.

- *Application ontologies* contain all the definitions that are needed to model the knowledge required for a particular application. Typically, application ontologies are a mix of concepts that are taken from domain ontologies and generic ontologies. Application ontologies are not reusable themselves.

3.1.5 Languages and Development Environment for Ontology

Ontologies can be built in different ways. Here we list some most representative languages and environments used when formalizing ontologies.

Ontology Languages

Ontologies must be expressed in a concrete notation, i.e. encoded in some language. There are a number of languages being proposed for ontology representation, see [34, 38] for an overview. The following languages are the most popular:

- **CycL** [87] is a declarative and expressive language, similar to first-order predicate calculus with extensions. CycL is used to represent the knowledge stored in Lenat's Cyc Artificial Intelligence project.

- **KIF** [47] + **Ontolingua** [36] KIF (Knowledge Interchange Format) was developed to solve the problem of heterogeneity of languages for knowledge representation. It has declarative semantics and provides for the definition of objects, functions and relations. It is based on the first-order predicate calculus, with a prefix notation. Ontolingua is a language based on KIF and on the Frame Ontology [53]. Ontolingua allows ontolgoies to be built by using KIF expressions, or Frame Ontology vocabulary, or both of them.

- **DAML+OIL** [33] (DARPA (Defense Advanced Research Projects Agency of US) Agent Markup Language and Ontology Inference Layer) is a markup language for web ontology. It attempts to merge the best of existing web languages such as XML (eXtensive Markup Language) [1] and RDF (Resource Description Framework) [3], description logics, and frame reasoning systems. It has been evolved into the OWL (Web Ontology Language) standard [2].

Development Environment

In some sense, the development of ontologies is similar to object-oriented software engineering, which also has its own life cycle [49]. During the life cycle, ontologies need to be analyzed and specified, implemented, modified, evaluated and regularly maintained, so some ontology tools will be needed. There are several ontology toolkits available commercially and some now available for free. We list some popular tools as follows.

- ***Ontolingua Server*** [36] is exclusively developed for building ontologies in the ontolingua language by the Knowledge Systems Laboratory at Stanford

University. It provides a collaborative environment to browse, create, edit, modify, and use ontologies for geographically distributed groups.

- ***Protégé-2000*** [52] is an ontology editor and a knowledge-base editor, developed at the Stanford Medical Informatics department. It is also an open-source, Java tool that provides an extensible architecture for the creation of customized knowledge-based applications.

- ***OilEd*** [16] is an ontology editor developed at the University of Manchester. It allows the users to construct and manipulate DAML+OIL and OWL, and use the reasoner to classify and check the consistency of ontologies.

3.1.6 Applications that exploit Ontologies

Various ontologies have been created or used by the people from different domains during the last few decades. For example, Gene Ontology (GO) [2] is the most developed and widely used ontology in the biological domain; ONIONS [45] is an integration of medical terminologies; OntoWordNet [44] is for Linguistic and Cognitive Engineering. For the design criteria and principles of ontology development please see [49] for details.

In this section, we will give some examples of the ontology-based applications within the domain of computer science, especially those related to knowledge presentation and data mining.

Ontologies used in Text Mining

Text mining or Knowledge Discovery in Texts (KDT) is aiming at the previously unknown information extracted from the different written resources, usually plain textual documents instead of well structured databases in KDD. There are also

[2] http://www.geneontology.org/

some attempts to make use of additional structural information such as HTML or XML documents in text mining, see [89].

Text clustering and text classification are two main tasks in the text mining community. The main idea of text clustering is to place, or cluster, documents with the most words in common into the same group; this is also called unsupervised categorization. The task of text classification, or supervised categorization, is to assign documents to a list of predefined target categories. So far, existing text clustering and classification systems are restricted to detecting patterns in the used terminology only, ignoring the conceptual patterns. For example, terms like "gold" and "silver" can not be generalized to the common hypernym "precious metal". With the help of ontology, researchers can start looking for the benefits that can be achieved by integrating the explicit conceptual terms found in ontologies into text mining. Andreas Hotho [67] claims that the set of terms are enriched and many measures for text clustering results improved when adding general concepts or replacing terms by concepts from the domain specific ontology. Also for text classification, the classical document representation can be enhanced and consistent improvement of the results can be achieved through concepts extracted from background knowledge provided by ontology structure, see [20].

Ontologies for building Semantic Web

Most of existing web contents are designed to be read and understood by humans, not machines. For example, it is not easy to make a computer accurately find the cheapest hotel and reserve it for you without any human interference. The Semantic Web [17] is an extension of the current web in which information is given in a well-defined, computer-processable format to enable computers and people to work in cooperation. A popular application of the Semantic Web is Friend of a

Friend (FOAF) [3], which creates machine-readable homepages describing people's personal information and relationships, for example, your name, address, interests, friends, etc. Holding all these data in the form of FOAF, a computer can then automatically search and merge some interesting information according to your complex requests, such as "Show me the pictures of someone who is interested in skiing and lives near me" or "Show me recent articles written by people at this meeting".

XML and RDF are the two standards for establishing semantic interoperability on the Web. XML addresses document structure; while RDF facilitates interoperation because it provides a data model. Ontology becomes the crucial part in a semantic web since it has been widely accepted as the most advanced knowledge representation model. OWL Web Ontology Language [2] is specially designed for this purpose. It adds more vocabulary for describing properties and classes than RDF. For example, "disjointness", "exactly one" relations for classes; equality, symmetry characteristics for properties defined in RDF.

OntoBroker [39] is a typical project which uses ontologies to annotate and wrap web documents, represents ontologies, and formulates an ontology-based answering service to perform intelligent access. The tool set of Ontobroker allows us to access information and knowledge from the web and to infer new knowledge with an inference engine based on techniques from logic programming.

[3] http://www.foaf-project.org/

3.2 Extracting Taxonomies from Ontologies

3.2.1 Motivation

During data preprocessing in data mining, attribute construction aims at improving the concept representation space. It can be done in several ways, such as combining attributes to generate new attributes beyond those provided in the input data, removing less relevant or irrelevant attributes, and/or abstracting values of given attributes, see e.g. [91, 92]. This last method is also called concept generalization.

Using attribute-value taxonomies can be a good way to implement the abstraction of attribute values, since a new set of attribute values can be constructed by a cut (see Definition 5.1) on the taxonomy tree. But such a taxonomy, manually defined by domain experts, is often viewed as a time consuming task, and inevitably is sometimes done rather haphazardly, especially for data containing many attributes each with many values. Moreover, domain experts from different domains may define different taxonomies involving the same attribute.

Ontologies provide a well-structured knowledge hierarchy for both domain experts and those undertaking knowledge representation, which can also be shared or reused among different domains. Ontologies are often constructed from one or more underlying taxonomies. Above these taxonomies, ontologies also represent rich associative or logical relationships that hold amongst the concepts or taxonomies, based on the background knowledge. For a certain attribute or concept, the taxonomy rooted in it may not be available straightforwardly from the ontology. It appears sensible to think of ways of extracting some interesting taxonomies from a predefined ontology automatically.

3.2.2 Language and Type of Ontology Used in this Thesis

We will base our study on DAML+OIL, choosing it for its expressive power of representation and reasoning ability. In particular, the properties provided by DAML+OIL, like *subClassOf*, *disjointWith*, *sameClassAs*, and *sameIndividualAs*, are the relations that we need when building the concept taxonomies covering heterogeneous databases or from various sources.

As a part of our early research, we created a *Student ontology*, using the DAML+OIL ontology language, and designed various examples for each property of DAML+OIL. This part of work is shown in appendix A.

Table 3.1 summarizes the properties, which may form axioms, supported by the DAML+OIL. In this table, 11 properties are defined, and the first two are imported from RDF(S) [3]. We specify their domain and range in the middle column, that is, the classes or properties between which these axioms can hold. Examples are given by using the standard description logic (DL) syntax in the right column.

Table 3.1: DAML+OIL Axiom Elements

Property Name	Domain&Range	Example
rdfs:subClassOf	Class	$Postgraduate \sqsubseteq Student$
rdfs:subPropertyOf	ObjectProperty	$hasSon \sqsubseteq hasChild$
daml:sameClassAs	Class	$Man \equiv Human \cap Male$
daml:samePropertyAs	ObjectProperty	$hasSupervisor \equiv hasAdvisor$
daml:disjointWith	Class	$Male \sqsubseteq \neg Female$
daml:inverseOf	ObjectProperty	$hasChild \equiv hasParent^{-}$
daml:sameIndividualAs	Thing	$\{Football\} \equiv \{Soccer\}$
daml:differentIndividualFrom	Thing	$\{George\ Bush\} \sqsubseteq \neg\{Barack\ Obama\}$
daml:TransitiveProperty	ObjectProperty	$hasDescendant$
daml:UniqueProperty	ObjectProperty	$hasFather$
daml:UnambiguousProperty	ObjectProperty	$isFather$

DAML+OIL provides modelling primitives by using a combination of frames and first-order logic to produce a frame-based ontology. Some people name such an ontology as a Knowledge Representation (KR) ontology [49, 129, 136], equivalent to the type *Representation ontology*. In this thesis, KR ontology and Domain ontology are used.

3.2.3 Implementing the Extraction of Taxonomies from Ontologies

In this section, the output taxonomies will follow this convention, having only the *whole-part* (presented by "*subClassOf*" in DAML+OIL ontology) or *is-a* parent-child relationship. Moreover, each term in these taxonomies will only have one parent, although some taxonomies may allow poly-hierarchy[4].

Actually, such poly-hierarchy occurs in ontologies with which we are dealing. A reusable KR ontology may contain concepts which are taken from various sources. Some of them are synonymous, some are distinct, and some may form different subclass partitions of the same superclass concept. Such combinations can result in a poly-hierarchy of the concepts.

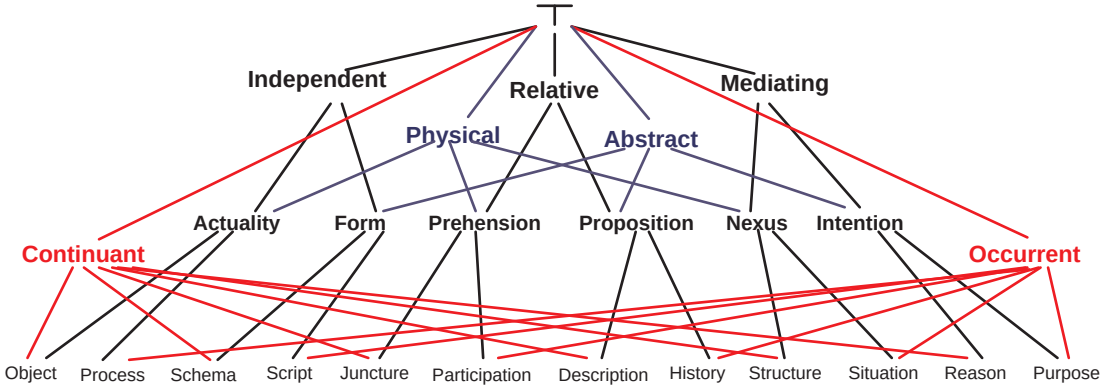


Figure 3.2: Hierarchy of Top-Level Categories (Sowa 2000)

Figure 3.2 shows an example of a poly hierarchy. It is a philosophical top-level categories of a KR ontology, developed by Sowa, and a synthesis of different logicians' discoveries referred to in [129]. Here we will not discuss the philosophical meaning of each category but only take it as a typical example of the basic structure of a simple real ontology.

[4]Poly-hierarchy means that a term can have multiple parents, and although the term may appear in multiple places, it is the same term.

In figure 3.2, the root $\top$ denotes the universal type, and is the primitive that satisfies the axioms:

- There exists something: $(\exists x)\top(x)$.
- Everything is an instance of $\top$:$(\forall x)\top(x)$.
- Every type is a subtype of $\top$: $(\forall t : Type)\ t \leq \top$.

{Independent, Relative, Mediating}, {Physical, Abstract}, and {Continuant, Occurrent} are the three top partitions at the first level below the root $\top$. The other categories are then derived as combinations of these three basic partitions. Table 3.2 displays the relationships between the twelve leaf node categories and these three top partitions through a matrix.

Table 3.2: Matrix of the Twelve Leaf Node Categories

	Physical		**Abstract**	
	Continuant	**Occurrent**	**Continuant**	**Occurrent**
Independent	Object	Process	Schema	Script
Relative	Juncture	Participation	Description	History
Mediating	Structure	Situation	Reason	Purpose

Given such a poly-hierarchy, we are aiming to extract several simple taxonomies in which each node (except the root node) has only one parent node.

In practice, an ontology is often formally represented by using an ontology language, e.g. DAML+OIL, without providing an explicit hierarchical structure as shown in figure 3.2. To implement the extraction, some preprocessing work can be done to construct the poly-hierarchy. Two different ways of construction are described below according to the complexity of the ontology.

Preprocessing an Ontology Hierarchy

Given an ontology, e.g. written in DAML+OIL, its term taxonomy can be constructed as follows.

1. For the small ontologies, say containing less than 20-30 terms/concepts, the poly-hierarchy can be drawn manually when looking through the ontology annotations.

2. For the large (over 30 terms, say) and complex ontologies, automatically searching and storing the terms seems more efficient. Here we will use an example from appendix A to illustrate the automatic way of taxonomy construction.

```
<daml:Class rdf:ID="Postgraduate">
   <rdfs:subClassOf rdf:resource="#Student"/>
   <rdfs:comment>It represents all the postgraduate students.
   </rdfs:comment>
</daml:Class>
```

Above DAML+OIL annotation is one of class definitions in a *Student Ontolgoy*. In DAML+OIL, only the tags starting with "*daml:Class*" and "*rdfs:subClassOf rdf:resource=*"[5] are the source of the terms.

Simply exploiting the text searching and matching algorithm, the double quoted terms appearing in the *Class* and *subClassOf* tags can be easily picked up, e.g. *Postgraduate* and *Student* in the above example. The extracted terms will be stored in a list, and the parent-child relationships can be established by creating a series of relation lists. Each relation list contains the newly found child term followed by its parent(s), e.g. {Postgraduate, Student} is the list generated from above class definition block.

Method and Algorithm for Taxonomic Extraction

The main idea of taxonomic extraction is that we transform the ontology hierarchy into a directed acyclic graph, which we will call the Directed Acyclic Taxonomy Graph (DATG) since it has some constraints; we will then exploit some graph

[5]Note: The tag "<*rdfs:subClassOf*>" is used to describe a class-specific range restriction, not the parent class.

properties to extract the required taxonomies from it. In our DATG, the *subClassOf* relation guides the direction of the edges.

DATG Definition: the DATG is a directed acyclic graph which satisfies the following constraints:

1. There is only one vertex which is designated as the root in the DATG, i.e. the root is the source vertex with zero indegree.

2. DATG has leaf vertices, i.e. the vertices with zero outdegree.

Figure 3.3 shows a sample of a DATG, in which the root is vertex 1 and the leaf vertices are $\{7, 8, 9, 10, 11, 12\}$.

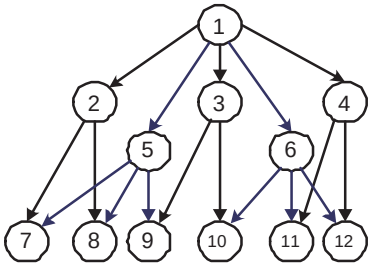


Figure 3.3: A Sample of DATG

DATG Generation: Transforming an ontology hierarchy to a DATG can be simply done by adding the direction to each edge of the hierarchy to make it point towards child vertices and away from parent vertices. The semantic meaning of those edges is the "*subClassOf*" relation.

Since the ontology hierarchy is a poly-hierarchical tree structured taxonomy, the transformed graph generally meets the requirements of being a DATG.

If the terms in ontology have already been stored in lists during the preprocessing, this procedure can be skipped.

Taxonomies Extraction:

We are looking for all the possible sub-DATGs from the DATG, where each vertex has at most one incoming edge and has zero or more outgoing edges. These sub-DATGs must contain the root vertex and all the leaf vertices of the original DATG, since they will be exploited for attribute-value abstraction where the input attribute values are all the leaf nodes of the ontology hierarchy. Furthermore, if any internal node is included in a sub-DATG, then all its child nodes should also be presented in that sub-DATG, so that we can make sure all the top partitions, like the example shown in figure 3.2, can be individually extracted. We name such subgraphs as Fully Leaved and Rooted Subgraphs (FLRSs).

FLRS Definition: Formally, a FLRS, $G' = (V', E')$, is the subgraph of a DATG, $G = (V, E)$, where V is the set of vertices and E is the set of directed edges, and $V' \subseteq V$, $E' \subseteq E$, s.t.

1. $\forall\, v' \in V'$, the indegree of v', denoted as $deg^+(v')$, satisfies $deg^+(v') = \{0, 1\}$.

2. $\forall\, e' \in E'$, e' links two distinct vertices u' and v' that $u', v' \in V'$. We then denote e' by $e' : u' \rightarrow v'$.

3. V' contains the root vertex of V, and no other vertex has $deg^+(v) = 0$.

4. V' contains all the leaf vertices, i.e. $\forall\, v \in V$, if $deg^-(v) = 0$, then $v \in V'$.

5. $\forall\, v' \in V'$, the outdegree of v' is denoted as $deg^-(v')$, if $deg^+(v') > 0$ and $deg^-(v') > 0$, i.e. v' is an internal vertex of G, then $\forall\, v' \rightarrow v'' \in E \Rightarrow v' \rightarrow v'' \in E'$.

FLRS Generation: The pseudo-code in figure 3.4 and 3.5 presents the algorithm for generating FLRSs from a DATG with n vertices. For ease of implementation, we label the vertices with consecutive integers $\{1, 2, ..., n\}$. Here 1 is assigned to the root vertex by default, and the rest vertices are labelled in a top

down and left to right order.

Searching FLRS is a recursive procedure. The algorithm starts bottom-up searching from each leaf vertex, to its parent vertex, recursively until reaching the root vertex. For each input vertex, the search aims to find its parent vertex and all its sibling vertices, as well as the vertices derived from them, and store them as the vertices of a FLRS candidate. The indegrees of vertices are used to control the iteration in this algorithm: each time when a FLRS is generated, the indegrees of all the vertices in that FLRS will be decreased by 1, and the iteration will be terminated when the indegrees of all the vertices are equal to 0. After each iteration, the edges linking any two vertices (except the root vertex) in the newly generated FLRS will be removed. Sometimes when the indegrees of all the leaf vertices are equal to 0, while there are some internal vertices left without being included by any FLRS, the search procedure will start from these vertices until a new FLRS is generated. In this way, we make sure every edge is traversed at least once during the set of FLRSs' generation, although some of them will be discarded for not meeting the requirement of being a FLRS.

Input: a DATG, $G = (V, E)$, $V = \{v_1, v_2, ..., v_n\}$, $v_i \in \{1, ..., n\}$, $E = \{e_1, e_2, ...\}$.
Output: a set of FLRSs, $G' = \{g'_1, g'_2, ...\}$, $g'_i = (V'_i, E'_i)$, $V'_i \subseteq V$, $E'_i \subseteq E$.

Algorithm:

1. Create the incoming edge list for each vertex: $\{inL_1, inL_2, ..., inL_n\}$, where inL_i stores all the parent nodes of v_i;
2. Create the outgoing edge list for each vertex: $\{outL_1, outL_2, ..., outL_n\}$, where $outL_i$ stores all the child nodes of v_i;
3. Calculate the indegree and outdegree for each vertex: $\{In_1, In_2, ..., In_n\}$ *and* $\{Out_1, Out_2, ..., Out_n\}$;
4. **searchFLRS**

```
k = 1;  // k is the number of FLRS, behave the kth FLRS g'_k is derived by G
while ∃ In_i ∈ {In_1, ..., In_n}, In_i ≠ 0 do
    if ∀ v_i ∈ leaf vertices, In_{v_i} = 0 or inL_{v_i} is null
    then for i = 1 to n do
            if In_i ≠ 0
            then t = i;  // t is a temporary variable to control the while loop
                while t ≠ 1 and t ∉ V'_k and inL_l is not null do
                    t = searchSibling(t);
                end while
            end if
    else for each leaf vertex v_i in V do
            l = v_i;  // l is a temporary variable to control the while loop
            while l ≠ 1 and l ∉ V'_k and inL_l is not null do
                l = searchSibling(l);
            end while
        end for
    end if
    add root vertex (default is 1) to V'_k;
    ∀ v' ∈ V'_k, if In_{v'} ≠ 0 then In_{v'} = In_{v'} − 1;
    remove all the non-leaf vertices (except root vertex) in V'_k from the incoming edge lists;
    if V'_k doesn't contain all the leaf vertices of V then discard g'_k;
    else k = k + 1;
    end if
end while
end searchFLRS
```

Figure 3.4: Algorithm for Generating FLRSs from DATG (a)

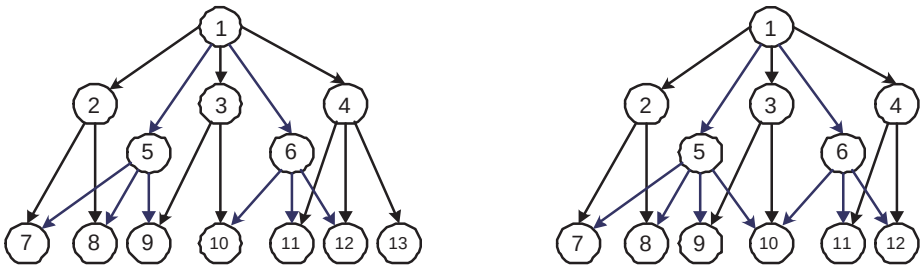
```
// For a given vertex, search for its sibling vertices and store them to the vertices set of the
   target FLRS.
searchSibling(v_in)
vertex = 0;  // vertex is the index of the vertices in inL_{v_in}
while vertex is not equal to the size of inL_{v_in} do
   v' = inL_{v_in}.get(vertex);
   if v' = null
   then add v_in to V'_k;
        v' = 1;
        break;
   else if v' = 1  //reach the root vertex
        then add v_in to V'_k;
             break;
        else if outL_{v'} ⊈ V'_k
             then add all the vertices in outL_{v'} to V'_k;
                  for each vertex, v''_i, in outL_{v'} do
                      if v''_i is not a leaf vertex
                      then searchLeafVertex(v''_i);
                      end if
                  end for
                  break;
             else vertex = vertex + 1;
                  v' = 1;
             end if
        end if
   end if
end while
return v';
end searchSibling

// For a given vertex, top-down search for all the vertices (until reach to leaf vertices) derived
   from it, and store them to the vertices set of the target FLRS.
searchLeafVertex(v_in)
isLeaf = False;
while isLeaf = False do
   for each vertex, v'_in, in outL_{v_in}
      add v'_in to V'_k if v'_in ∉ V'_k;
      if v'_in is not a leaf vertex
      then searchLeafVertex(v'_in);
      end if
   end for
   isLeaf = True;
end while
end searchLeafVertex
```

Figure 3.5: Algorithm for Generating FLRSs from DATG (b)

3.2.4 Illustrations of the Algorithm

1. We applied the algorithm on the sample of DATG shown in figure 3.3. Two FLRSs, whose vertices set are $\{1, 2, 3, 4, 7, 8, 9, 10, 11, 12\}$ and $\{1, 5, 6, 7, 8, 9, 10, 11, 12\}$, are obtained. So the corresponding taxonomies can be directly inferred from these subgraphs.

2. Similarly, we also tried some extreme situations to test the algorithm. Figure 3.6 gives two extreme examples of DATG.



(a) DATG 1 (b) DATG 2

Figure 3.6: Some Extreme DATGs

For DATG 1, only one FLRS can be extracted, whose vertices set is $\{1, 2, 3, 4, 7, 8, 9, 10, 11, 12, 13\}$. The other possible FLRS, $\{1, 5, 6, 7, 8, 9, 10, 11, 12\}$, is discarded since it does not contain all the leaf vertices.

For DATG 2, two FLRSs can be separately extracted in different runs, since the algorithm forbids any two FLRSs generated from one DATG share the same internal vertices at the second level from the top, i.e. the direct children vertices of the root vertex must be divided into different FLRSs without overlapping. The vertices set is either $\{1, 2, 3, 4, 7, 8, 9, 10, 11, 12\}$ or $\{1, 5, 4, 7, 8, 9, 10, 11, 12\}$. Vertex 6 is not included in any of the FLRSs, since its child vertex 10 will dissatisfy the first requirement of being a FLRS. Actually, if we consider the semantic meanings in ontology hierarchy, vertice 5 should be the parent vertex of vertice 2 and 3, or the synonym of their parent vertex, so this case can be prevented. In practice,

DATG 2 will be presented in another form, see figure 3.7, then only one FLRS can be extracted, whose vertices are $\{1, 5, 4, 2, 3, 7, 8, 9, 10, 11, 12\}$.

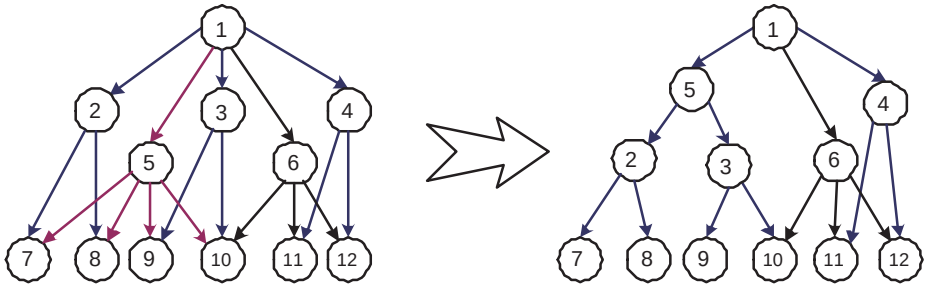


Figure 3.7: Reconstruction of DATG 2

3. When applying the approach to the hierarchy of top-level categories in figure 3.2, we firstly code the name of each node from top to bottom, as shown in figure below. After running the algorithm, three taxonomies can be extracted, shown in figure 3.8 $\sim$ 3.10, respectively.

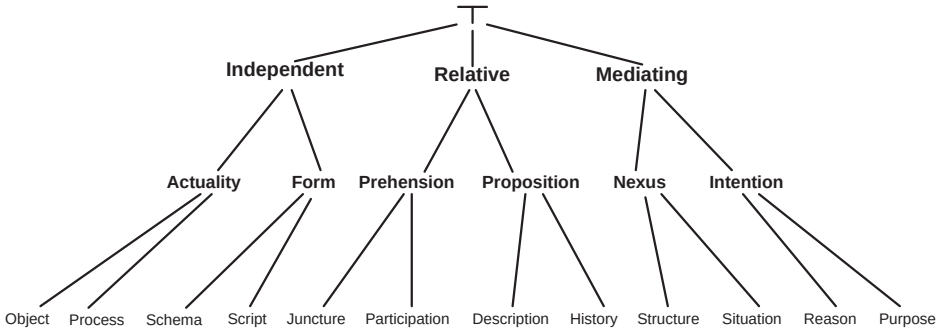


Figure 3.8: Extracted Taxonomy 1 from Figure 3.2

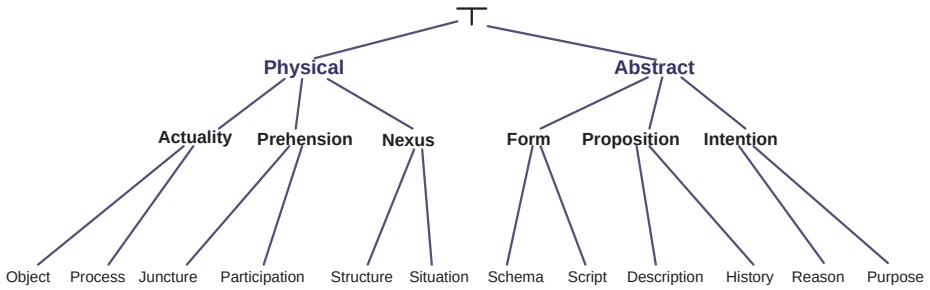


Figure 3.9: Extracted Taxonomy 2 from Figure 3.2

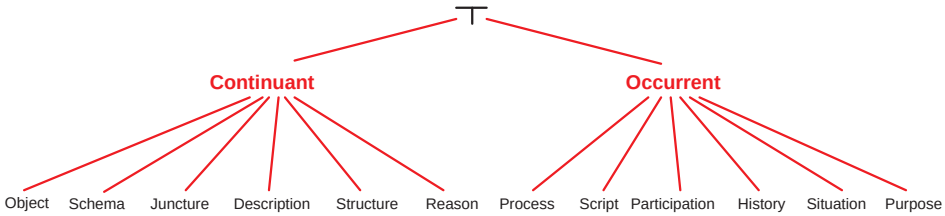


Figure 3.10: Extracted Taxonomy 3 from Figure 3.2

3.3 Summary

The main purpose of this chapter is to give a study of ontology and explore the feasibility of exploiting some unique properties for taxonomy construction and even for information integration and data mining from semantically heterogeneous data sets.

The most significant characteristic of ontologies is that they group concepts together in terms of taxonomies, along with rich associative or logical relationships that hold amongst the concepts. In section 3.1 we reviewed some important issues about ontologies, such as definition, types, representation languages, existing applications, etc.

Our work can be divided into two directions. The first direction is ontology building. As a part of our early research, we created a *Student ontology*, using the DAML+OIL ontology language, and designed various examples for each property of DAML+OIL. This part of work is shown in appendix A.

The second direction is to extract taxonomies from predefined ontologies. Considering there are so many domain ontologies being defined, and these can be reused between domains, it seems worthwhile to make use of them to generate the required taxonomies. In section 3.2, we propose an algorithm of automatically extracting taxonomies from ontology and illustrate our idea with some examples.

Chapter 4

Constructing AVTs using Clustering Algorithms

In data mining, some clustering algorithms can be used to automatically generate a tree structured hierarchy for the data set. In this chapter, we start by reviewing some typical clustering algorithms. Then we study the behaviour of these algorithms and consider their efficacy in constructing relevant semantically interpretable taxonomies. We also propose an iterative approach to build the attribute-value taxonomies by using some partitional algorithms. Two types of attribute value, nominal and numeric, are considered in order to construct a suitable taxonomy for each case. Two real world data sets are chosen for performance evaluation.

Since we will be applying clustering to a single field of the data set, only one dimensional clustering is needed, although clustering algorithms can work effectively on multi-dimensional data.

In general, clustering algorithms can be classified into two categories: (1) hierarchical and (2) partitional. For the given set of data objects, hierarchical algorithms aim to find a series of nested clusters of data so as to form a tree diagram or dendrogram; partitioning algorithms will only split the data into a specified number of disjoint clusters.

4.1 Hierarchical Algorithms

Hierarchical algorithms can also be classified into either *agglomerative* clustering or *divisive* clustering. *Agglomerative* is a bottom-up approach which starts with each data item in a cluster and successively merges the closest clusters into larger ones, until the largest cluster containing all the data is achieved or some termination conditions hold. *Divisive*, on the contrary, is a top-down approach which starts with all the data objects in one single cluster and successively splits it into smaller clusters, until each object is in one cluster or some termination conditions are met.

4.1.1 Distance Measure

The first key step in a hierarchical clustering is to select a distance measure as the criterion for forming the clusters.

Given a data set $D = \{x_1, x_2, ..., x_n\}$, where each $x_i = (x_{i1}, x_{i2}, ..., x_{id})$ is d-dimensional, the distances between data objects can be computed by many measures (see e.g. [72]) including the following common measures.

- **Euclidean distance**: $\mathbf{dis}(x_i, x_j) = \sqrt{\sum_{\mathbf{k=1}}^{\mathbf{d}}(x_{ik} - x_{jk})^2}$.
- **Squared Euclidean distance**: $\mathbf{dis^2}(x_i, x_j) = \sum_{\mathbf{k=1}}^{\mathbf{d}}(x_{ik} - x_{jk})^2$.
- **City-block (Manhattan) distance**: $\mathbf{dis}(x_i, x_j) = \sum_{\mathbf{k=1}}^{\mathbf{d}} |x_{ik} - x_{jk}|$.

For one dimensional data, the *Euclidean distance* and the *Manhattan distance* are identical.

For a data set, D, once a distance measure is selected, for any two subsets, S and T, we can extend **dis** to following functions.

- $\mathbf{dis}(\mathbf{S}, \mathbf{T}) = \mathbf{min}\{\mathbf{dis}(\mathbf{x}, \mathbf{y}) : \mathbf{x} \in \mathbf{S},\ \mathbf{y} \in \mathbf{T}\}$, which will find the smallest distance between two clusters, also called **Single-link** (or nearest neighbour) [51].

- $\mathbf{dis}(\mathbf{S}, \mathbf{T}) = \mathbf{max}\{\mathbf{dis}(\mathbf{x}, \mathbf{y}) : \mathbf{x} \in \mathbf{S},\ \mathbf{y} \in \mathbf{T}\}$, which will find the largest distance between two clusters, also called **Complete-link** (or furthest neighbour) [13].

- $\mathbf{dis}(\mathbf{S}, \mathbf{T}) = \frac{\mathbf{1}}{\mathbf{|S||T|}} \sum_{\mathbf{x} \in \mathbf{S}} \sum_{\mathbf{y} \in \mathbf{T}} \mathbf{dis}(\mathbf{x}, \mathbf{y})$, which will find the average distance between two clusters, also called **Average-link**.

- $\mathbf{dis}(\mathbf{S}, \mathbf{T}) = \mathbf{dis}(\hat{\mathbf{c_S}}, \hat{\mathbf{c_T}})$, where $\hat{c_S}$ and $\hat{c_T}$ are the centroids of S and T respectively.

- $\mathbf{dis}(\mathbf{S}, \mathbf{T}) = \mathbf{dis}(\hat{\mathbf{m}_S}, \hat{\mathbf{m}_T})$, where $\hat{m_S}$ and $\hat{m_T}$ are the medoids of S and T respectively.

If $S \in 2^D$, then **Centroid** of S is defined by $\hat{\mathbf{c}}(\mathbf{S}) \in \mathbf{D}$, s.t. $\sum_{\mathbf{x} \in \mathbf{S}}\ \mathbf{dis^2}(\mathbf{x}, \hat{\mathbf{c}}(\mathbf{S}))$ is minimised. For numeric data, if the *Euclidean distance* is chosen, then the centroid will be the mean of S, i.e. $\hat{\mathbf{c}}(\mathbf{S}) = \frac{\mathbf{1}}{\mathbf{|S|}} \sum_{\mathbf{x} \in \mathbf{S}} \mathbf{x}$.

The **Medoid** of S is defined by $\hat{\mathbf{m}}(\mathbf{S}) \in \mathbf{S}$, s.t. $\sum_{\mathbf{x} \in \mathbf{S}}\ \mathbf{dis}(\mathbf{x}, \hat{\mathbf{m}}(\mathbf{S}))$ is minimised, so the medoid must be an element of S.

The **Diameter** of S is defined by $\mathbf{diam}(\mathbf{S}) = \mathbf{max}\{\mathbf{dis}(\mathbf{x}, \mathbf{y}) : \mathbf{x}, \mathbf{y} \in \mathbf{S}\}$.

4.1.2 Agglomerative Clustering

There are a number of agglomerative clustering algorithms (see e.g. [58, 77, 81]), but they mainly differ on which distance measure between clusters is chosen.

Given a data set $D = \{x_1, x_2, ..., x_n\}$, the procedure of agglomerative clustering is as follows.

1. Place each object in its own cluster, construct a list of singleton clusters $L = C_1, C_2, ..., C_n$, and calculate the distance between any two clusters by using one of the distance measures so as to yield a distance matrix;

2. For each pair of clusters in L, find the two closest clusters (say C_i and C_j) whose distance is under a certain threshold;

3. Update the cluster list by removing the pair of clusters C_i and C_j from L, and adding a new merged cluster $C_i \bigcup C_j$ to L;

4. Update the distance matrix to reflect the merging operation;

5. Go to step 2 until only one cluster remains, or until no pair of clusters exist with distance between them greater than the threshold.

4.1.3 Divisive Clustering

Divisive algorithms differ in the way they split the clusters. In this section, the commonly used DIANA (DIvisia ANAlysis) [81] method is described. Given a data set $D = \{x_1, x_2, ..., x_n\}$, the procedure of DIANA can be described as follows.

1. Place all the data objects in one cluster $C_0 = \{x_1, x_2, ..., x_n\}$, and calculate the distance between any pair of objects in C_0 to yield a distance matrix. The algorithm works with a set of clusters, initially just $S = \{C_0\}$;

2. Let C be the cluster with the largest diameter in S, split it into C_1 and C_2, $C_1 = C$, $C_2 = \emptyset$; find an object x_j in C_1, which has a maximum average distance to the others, and then move x_j to C_2;

3. For each remaining object in C_1, compute both its average distance to the other objects in C_1 and the average distance to object(s) in C_2. If the former is greater than the latter, then move this object to C_2;

4. Replace C in S by C_1 and C_2;

5. Go to step 2, until all the clusters contain only one object, or some terminating criterion is met.

Other divisive algorithms that minimise the larger of the two cluster diameters are described in [122] and [68]. Some algorithms minimise the sum of the two cluster diameters instead (see e.g. [63]).

4.2 Partitional Algorithms

In partitional clustering, it is often computationally infeasible to try all the possible splits, so greedy heuristics are commonly used in the form of iterative optimization. However, when the number of points is small, then an exact algorithm can be considered, e.g. Fisher's algorithm [40, 64]. This section will focus on introducing two algorithms — k-means and Fisher's algorithm.

Both algorithms require real-valued input data, and can be considered as a method for numeric attribute value discretisation. Experimental comparisons are discussed in a later section.

4.2.1 K-means Clustering

K-means is the most widely used greedy heuristic partitional clustering technique (see e.g. [41], [100], [70], [71], [124], [130]), and is an iterative centroid-based divisive algorithm.

Given a data set $D = \{x_1, x_2, ..., x_n\}$, the procedure of k-means clustering is as follows.

1. Select k initial points to act as cluster centroids, $c_1, c_2, ..., c_k$;
2. Assign each object x_i in D to cluster C_i whose centroid c_i is the nearest to x_i;
3. For each cluster, recompute its centroid based on the elements assigned to that cluster;
4. Go to step 2 until the centroids no longer move.

To generate good quality clusters, the k-means algorithm aims to minimise the within-cluster sum-of-square fitness measure

$$\sum_{i=1}^{k} \sum_{x,y \in C_i} dis^2(x, y). \tag{4.1}$$

If using Euclidean distance as defined in section 4.1.1, then the following between-cluster measure will be simultaneously maximised.

$$\sum_{i=1}^{k} \sum_{x \in C_i, y \notin C_i} dis^2(x, y) \tag{4.2}$$

The reason is that the sum of measure 4.1 and 4.2 is a constant, which is

$$\sum_{x,y \in D} dis^2(x, y). \tag{4.3}$$

In 1D clustering, this within-cluster fitness measure can be expressed as

$$\sum_{i=1}^{k} \sum_{x,y \in C_i} (x-y)^2. \tag{4.4}$$

The following theorem, proved by Al-Harbi [7], also shows the similarity between the within-cluster sum-of-square and the centroid measure.

Theorem 4.1 Given a cluster, C, in the Euclidean space R^n, a clustering that minimises $\sum_{\mathbf{x} \in \mathbf{C}} \mathbf{dis^2}(\mathbf{x}, \hat{\mathbf{c}})$, where $\hat{c}$ is the centroid of C, will also minimise the within-cluster sum-of-square measure $\sum_{\mathbf{x},\mathbf{y} \in \mathbf{C}} \mathbf{dis^2}(\mathbf{x}, \mathbf{y})$ and vice versa.

Once the k clusters have been found, they can be replaced by their centroids or medoids. These values can then themselves be clustered. Repeating this process results in a tree hierarchy of clusters. A case study will be presented later.

4.2.2 Fisher's Algorithm

Working on an ordered data set, or continuous real values, Fisher's clustering algorithm [40, 64] seeks an optimal partition with respect to a given measure, providing the measure satisfies the certain constraint that if x, y are both in a cluster and the data z, $x < z < y$, then z is necessarily also in the same cluster.

Algorithm Description

Given a set of ordered points $D = \{x_1, x_2, \ldots, x_n\}$, with $x_1 < x_2 \ldots < x_n$ on the real line, we seek a partition of the points into K clusters $\{C_1, C_2, \ldots, C_K\}$, where

C_1 comprises points $x_1 < x_2 \ldots < x_{n_1}$,

C_2 comprises points $x_{n_1+1} < x_{n_1+2} < \ldots < x_{n_2}$,

$\vdots$

C_K comprises points $x_{n_{K-1}+1} < x_{n_{K-1}+2} < \ldots < x_{n_K}$, and $x_{n_K} = x_n$.

Thus, such a clustering is uniquely determined by the values $x_{n_1}, x_{n_2}, \ldots, x_{n_{k-1}}$. Any clustering of the points of this form will be called an *interval-clustering*. For certain quality measures on clusters, an optimal interval clustering will always be an optimal clustering.

For example, consider a within-cluster fitness measure for the K interval clusters $C = \{C_1, C_2, \ldots, C_K\}$ of data set D

$$Fit(D, K) = \sum_{i=1}^{K} d(C_i), \tag{4.5}$$

where, $d(C_i)$ is a measure of the value of the cluster C_i and

$$d(C_i) = \sum_{x_j \in C_i} (x_j - \mu_i)^2, \quad \text{where} \quad \mu_i = \sum_{x_j \in C_i} \frac{x_j}{|C_i|}. \tag{4.6}$$

The optimal clustering is the partition which minimises $Fit(D, K)$, and this must necessarily be an interval clustering.

Theorem 4.2 Providing the fitness measure is of form (4.6), the minimum of $Fit(D, K)$ is always obtained from an interval clustering.

Proof. Assume $C = \{C_1, C_2, \ldots, C_K\}$ is a non-interval clustering on n points $\{x_1, \ldots, x_n\}$, which minimises $Fit(D, K)$. Then there exists two clusters, C_i and C_j, s.t. $\exists\ x_{imax} = max\{x_i |\ x_i \in C_i\}\ x_{jmin} = min\{x_j |\ x_j \in C_j\}$, $x_{imax} > x_{jmin}$, but $\mu_i < \mu_j$.

Now swap x_{jmin} and x_{imax}, then we get two new clusters, C_i' and C_j', with $\mu_i' < \mu_i$ and $\mu_j' > \mu_j$. We now show that the new clustering has a smaller value of $Fit(D, K)$ than the old one.

Let $Fit'(D, K)$ denote the fitness measure for the new clustering, then the differences of $Fit(D, K)$, denoted as $\Delta_{Fit(D,K)}$, between the two clusterings can be represented as:

$$
\begin{aligned}
&\Delta_{Fit(D,K)} = Fit'(D,K) - Fit(D,K) \\
&= \sum_{\substack{x_i \in C_i \\ x_i \neq x_{imax}}} (x_i - \mu_i')^2 + (x_{jmin} - \mu_i')^2 + \sum_{\substack{x_j \in C_j \\ x_j \neq x_{jmin}}} (x_j - \mu_j')^2 + (x_{imax} - \mu_j')^2 \\
&\quad - \sum_{\substack{x_i \in C_i \\ x_i \neq x_{imax}}} (x_i - \mu_i)^2 - (x_{imax} - \mu_i)^2 - \sum_{\substack{x_j \in C_j \\ x_j \neq x_{jmin}}} (x_j - \mu_j)^2 - (x_{jmin} - \mu_j)^2 \\
&= \sum_{\substack{x_i \in C_i \\ x_i \neq x_{imax}}} (2x_i - \mu_i' - \mu_i)(\mu_i - \mu_i') + (x_{jmin} - \mu_i')^2 - (x_{imax} - \mu_i)^2 \\
&\quad + \sum_{\substack{x_j \in C_j \\ x_j \neq x_{jmin}}} (2x_j - \mu_j' - \mu_j)(\mu_j - \mu_j') + (x_{imax} - \mu_j')^2 - (x_{jmin} - \mu_j)^2 \\
&= [\sum_{\substack{x_i \in C_i \\ x_i \neq x_{imax}}} 2x_i - \frac{|C_i| - 1}{|C_i|}(\sum_{\substack{x_i \in C_i \\ x_i \neq x_{imax}}} 2x_i + x_{imax} + x_{jmin})] \ \frac{x_{imax} - x_{jmin}}{|C_i|} \\
&\quad + (x_{jmin} + x_{imax} - \frac{\sum_{x_i \in C_i \atop x_i \neq x_{imax}} 2x_i + x_{imax} + x_{jmin}}{|C_i|})(x_{jmin} - x_{imax} + \\
&\quad + \frac{x_{imax} - x_{jmin}}{|C_i|}) + [\sum_{\substack{x_j \in C_j \\ x_j \neq x_{jmin}}} 2x_j - \frac{|C_j| - 1}{|C_j|}(\sum_{\substack{x_j \in C_j \\ x_j \neq x_{jmin}}} 2x_j + x_{imax} + x_{jmin})] \cdot \\
&\quad \frac{x_{jmin} - x_{imax}}{|C_j|} + (x_{imax} + x_{jmin} - \frac{\sum_{x_j \in C_j \atop x_j \neq x_{jmin}} 2x_j + x_{imax} + x_{jmin}}{|C_j|}) \cdot \\
&\quad (x_{imax} - x_{jmin} + \frac{x_{jmin} - x_{imax}}{|C_j|}) \\
&= [\sum_{\substack{x_i \in C_i \\ x_i \neq x_{imax}}} 2x_i - (|C_i| - 1)(x_{imax} + x_{jmin})] \ \frac{x_{imax} - x_{jmin}}{|C_i|^2} \\
&\quad + [\frac{|C_i| - 1}{|C_i|}(x_{jmin} + x_{imax}) - \frac{2}{|C_i|} \sum_{\substack{x_i \in C_i \\ x_i \neq x_{imax}}} x_i] \ \frac{1 - |C_i|}{|C_i|}(x_{imax} - x_{jmin}) \\
&\quad + [\sum_{\substack{x_j \in C_j \\ x_j \neq x_{jmin}}} 2x_j - (|C_j| - 1)(x_{imax} + x_{jmin})] \ \frac{x_{jmin} - x_{imax}}{|C_j|^2} \\
&\quad + [\frac{|C_j| - 1}{|C_j|}(x_{jmin} + x_{imax}) - \frac{2}{|C_j|} \sum_{\substack{x_j \in C_j \\ x_j \neq x_{jmin}}} x_j] \ \frac{|C_j| - 1}{|C_j|}(x_{imax} - x_{jmin})
\end{aligned}
$$

$$
\begin{aligned}
&= (x_{imax} - x_{jmin}) \, [\frac{2}{|C_i|} \sum_{\substack{x_i \in C_i \\ x_i \neq x_{imax}}} x_i - \frac{(|C_i| - 1)}{|C_i|}(x_{imax} + x_{jmin})] \\
&\quad + (x_{imax} - x_{jmin}) \, [\frac{(|C_j| - 1)}{|C_j|}(x_{imax} + x_{jmin}) - \frac{2}{|C_j|} \sum_{\substack{x_j \in C_j \\ x_j \neq x_{jmin}}} x_j] \\
&= (x_{imax} - x_{jmin}) \, [(\mu_i + \mu_i') - (\mu_j' + \mu_j)] \\
&< (x_{imax} - x_{jmin}) \, (2\mu_i - 2\mu_j) \; < \; 0
\end{aligned}
$$

So we have proved that $Fit'(D, K) < Fit(D, K)$. Repeat this swapping if there still exists two points, $x_{imax} \in C_i'$ and $x_{jmin} \in C_j'$, holding $x_{imax} > x_{jmin}$ but $\mu_i' < \mu_j'$, then we can continually obtain another two new clusters with smaller $Fit(D, K)$ value. The two non-interval clusters, C_i and C_j, can be gradually transformed into interval clusters with minimum value of $Fit(D, K)$ in this way.

If we apply the above procedure to any other two non-interval clusters in C, then we finally obtain an interval clustering that can minimise $Fit(D, K)$. Therefore we have also proved that the optimal clustering is an interval clustering.

Algorithm Implementation

Fisher pointed that the optimal K interval clusters can be deduced from the optimal $K-1$ clusters, which means we can successively compute optimal $2, 3, 4, \ldots, K-1$ partitions, and then the optimal K partition. The steps of this dynamic programming procedure are listed below.

1. Create a matrix $dis(j, k)$ which contains the values of the measure $d(C_{jk})$ for every possible interval cluster, $C_{jk} = \{x_j, \ldots, x_k\}$, i.e.

$$
dis(j, k) = \begin{cases} d(C_{jk}) & 1 \leq j < k \leq n, \\ 0 & 1 \leq k \leq j \leq n. \end{cases}
$$

2. Compute the fitness of the optimal 2-partition of any t consecutive points

set $D_t = \{x_1, x_2, ..., x_t\}$, where $2 \leq t \leq n$, and find the minimum by

$$Fit(D_t, 2) = \min_{2 \leq s \leq t} \{dis(1, s-1) + dis(s, t)\}, \text{ where } 2 \leq t \leq n.$$

3. Compute the fitness of the optimal L-interval-partition of any t consecutive points set $D_t = \{x_1, x_2, ..., x_t\}$, where $L \leq t < n$, and $3 \leq L < K$ by using

$$Fit(D_t, L) = \min_{L \leq s \leq t} \{Fit(D_{s-1}, L-1) + dis(s, t)\}, \text{ where } 3 \leq L < K \text{ and } L \leq t \leq n.$$

4. Create a new matrix $f(t, L)$ which stores the fitness computed in the above two steps for all optimal L-partitions $(1 \leq L < K)$ on any t points set $D_t = \{x_1, x_2, ..., x_t\}$, where $(1 \leq t \leq n)$.

$$f(t, L) = \begin{cases} Fit(D_t, L) & 1 < L < K, 1 \leq t \leq n, L < t, \\ dis(1, j) & L = 1, 1 \leq j \leq t, \\ 0 & 1 < L < K, 1 \leq t \leq n, L \geq t. \end{cases}$$

The optimal K-partition can be discovered from the matrix $f(t, L)$ by finding the index l, so that $f(t, K) = f(l, K-1) + dis(l, n)$.

Then the Kth partition is $\{x_l, x_{l+1}, \ldots, x_n\}$, and the $(K-1)$th partition is $\{x_{l^*}, x_{l^*+1}, \ldots, x_{l-1}\}$, where $f(l-1, K-1) = f(l^*-1, K-2) + dis(l^*, l-1)$, and so on.

If we repeatedly apply this algorithm on the output clusters, which are replaced by their centroids or medoids after each run, then the hierarchy of the clusters can be generated level by level. A case study is implemented in a later section.

4.2.3 Measures for Cluster Number Selection

Most partitional clustering algorithms assume that the number of clusters is specified in advance. There are many approaches making an attempt at finding the appropriate number (see e.g. [101], [19], [139]). **Silhouette width** [82] is a very useful measure to show how well the resulting clusters are separated. In this section we will consider an approach which makes use of this measure to get the optimal number of clusters.

Let $C = \{C_1, C_2, ..., C_k\}$ $(1 < k \leq n)$ be a clustering on a data set $D = \{x_1, x_2, ..., x_n\}$. For any object $x \in D$, which is assigned to cluster C_i, the average distance of x to all the other objects in C_i is defined by

$$avg(x) = \begin{cases} \frac{1}{|C_i|-1} \sum_{y \in C_i} dis(x, y) & if\ |C_i| > 1, \\ 0 & if\ |C_i| \leq 1. \end{cases}$$

The average distance of x to all the objects in another cluster C_j $(1 \leq j \leq k, j \neq i)$ is denoted by

$$d(x, C_j) = \frac{1}{|C_j|} \sum_{z \in C_j} dis(x, z). \tag{4.7}$$

Then we call the cluster, C_m, which has the shortest average distance, the *neighbour* of x $(x \in C_i)$, i.e.

$$d(x, C_m) = min\{d(x, C_j) : 1 \leq j \leq k, j \neq i\}. \tag{4.8}$$

Let $avg_{min}(x) = d(x, C_m)$, then the *silhouette* value of object x is defined by

$$S(x) = \frac{avg_{min}(x) - avg(x)}{max\{avg(x), avg_{min}(x)\}}. \tag{4.9}$$

The value of $S(x)$ ranges from -1 to +1, reflecting the following properties:

1. $S(x)$ *is close to -1* — Object x is poorly classified, since its dissimilarity with other objects in its own cluster is much greater than the one in its neighbour/nearest cluster.

2. $S(x)$ *is close to 0* — Object x can be assigned to either C_i or C_m, since its average distance to both clusters are approximately equal.

3. $S(x)$ *is close to +1* — Object x is classified appropriately, since its within-cluster dissimilarity is much smaller than the smallest between-cluster dissimilarity.

After computing the silhouette value for all the objects in D, the overall *average silhouette width* of D can be defined as follows

$$S_{avg}(D) = \frac{1}{|D|} \sum_{x_i \in D} S(x_i). \tag{4.10}$$

$S_{avg}(D)$ can be then used to select the optimal number of clusters. The optimal choice for k is that maximises the average silhouette width of data set D, i.e. $\mathbf{max}\{\mathbf{S_{avg}(D)}\}$.

Figure 4.1 shows an illustration of the average silhouette width plots over all possible k clusters, where k ranges from 2 to the number of objects in the selected database. k-means algorithm is applied on the *Soybean* data set, taken from UCI Repository [12], which contains 47 objects.

In practice, for a large data set, in order to find the optimal number of clusters more efficiently, it is not necessary to calculate the average silhouette width for all the possible k clusters, we could set an upper limit for k instead, i.e. $k \leq k_{max}$, where $k_{max} < |D|$. Although there may be some larger average silhouette widths when k_{max} is increased, those corresponding clusterings are not very interesting.

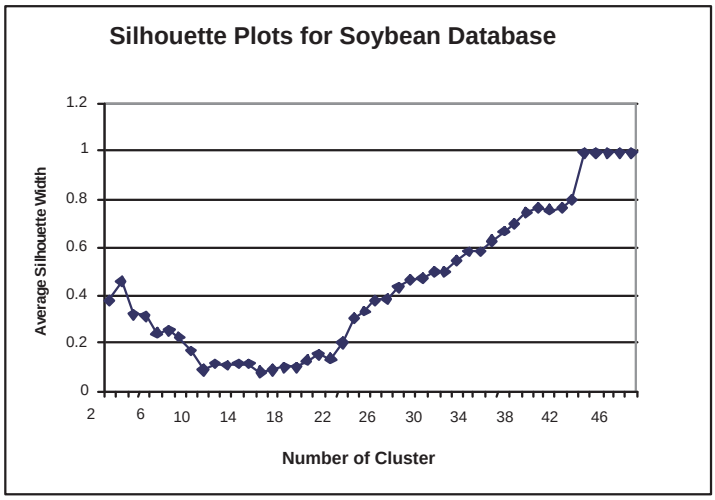


Figure 4.1: Average Silhouette Width Plots for k-means clustering on the *Soybean* Data (the *Soybean* Data, obtained from UCI repository, contains 47 instances with 35 numeric attributes and can be classified into 4 classes.)

The reason is that the larger the k_{max} is, the more single object clusters are produced, and the more running time is consumed. Normally, k_{max} can be set to half the number of the objects. For the case in figure 4.1, if we set k_{max} to 24, then the optimal number of clusters will be 3, where the average silhouette width achieves its peak.

4.3 Exploiting Hierarchical Algorithms for Attribute-Values Taxonomies Construction

In this section, two types of hierarchical algorithm are exploited respectively on both nominal and numeric attribute-values to generate corresponding taxonomies.

4.3.1 Dealing with Nominal Attributes

The domain of nominal attributes could be any arbitrary finite set of non-numeric and unordered values. Typical examples include `sex/gender` attribute with domain {*Male*, *Female*}, and `race/ethnicity` attribute with domain such as {*White*, *Asian*, *Black*, *Indian*, *other*}.

To cluster data with nominal attributes, one common approach is to convert them into numeric attributes, and then apply a clustering algorithm. This is usually done by "exploding" the nominal attribute into a set of new binary attributes, one for each distinct value in the original attribute. For example, the `sex/gender` attribute can be replaced by two attributes, `Male` and `Female`, both with a numeric domain {0, 1}.

Another way of transformation is using of some distinct numerical (real) values to represent nominal values. If again, using `sex/gender` attribute as an example, a numeric domain {1, 0} is a substitute for its nominal domain {*Male*, *Female*}. A more general technique, frequency based analysis, can also be exploited to perform this transformation. For instance, the domain of attribute `race/ethnicity` can be transformed from {*White*, *Asian*, *Black*, *Indian*, *other*} to {0.56, 0.21, 0.12, 0.09, 0.02}, according to their occurrence in the data set.

In this section, we aim to build a taxonomy for nominal values of each specific categorical attribute by using some hierarchical clustering algorithms. The statistical based approach seems more appropriate for this task, because it simplifies the transformation job and each numeric value to be transformed also reveals the statistical information of its original nominal value, such as distribution. Our transformational scheme is to replace each nominal value with its corresponding conditional probability (conditional on the target class membership). Particularly, in order to show the benefits of the attribute construction by using attribute-value taxonomies, the nominal attributes with big domains (say, the number of values

are greater than five) are more interested.

4.3.2 Transformation Scheme

We use the *Adult* data set as an example to illustrate our transformational scheme. Five nominal attributes, Education, Occupation, Workclass, Marital-status, and Native-country, are chosen to do the transformation.

The ">50K" class, denoted by C_h, is chosen as the prediction task, and each value of all selected attributes will be replaced by the conditional probabilities of the person being classified to C_h, given he/she holds this specific value. The training data are used for doing this replacement.

Let $A = \{A_1, A_2, ..., A_m\}$ represent the attributes of *Adult* data set, and $V = \{V_1, V_2, ..., V_l\}$ be the corresponding value set of A. Given $V_{ij} \in V_i$ denotes the jth value of attribute A_i, the conditional probability above can be defined as

$$P(C_h \mid V_{ij}) = \frac{P(C_h, V_{ij})}{P(V_{ij})} = \frac{|C_h V_{ij}|}{|V_{ij}|} \tag{4.11}$$

where $|C_h V_{ij}|$ is the number of instances classified to C_h, whose value of attribute A_i is V_{ij}, and $|V_{ij}|$ is the total number of instances that hold the attribute value V_{ij}.

For example, suppose Marital-status is the fourth attribute in the *Adult* data, and "*Divorced*" is its second value, then $P(C_h \mid V_{42})$ is the probability of the person who can earn more than $50K per year, given he or she is divorced.

4.3.3 Using Hierarchical Algorithms

AGNES (Agglomerative Nesting) [81] and DIANA (DIvisia ANAlysis) [81] algorithms are used in the following experiments. The Euclidean distance measure, which is equivalent in the one dimensional case to the absolute difference, is selected

to calculate the distance between two objects for both cases, and the "complete-link" method is used in AGNES for computing the distance between two clusters.

We apply both the AGNES and DIANA algorithms, using the R package[1], to the matrix of each selected attribute, and the two generated taxonomies will be shown in the same figure, see figure 4.2 to figure 4.9. In these figures, the "Height" in AGNES presents the distance between merging clusters at the successive stages, whilst it is the diameter of the cluster prior to splitting in DIANA.

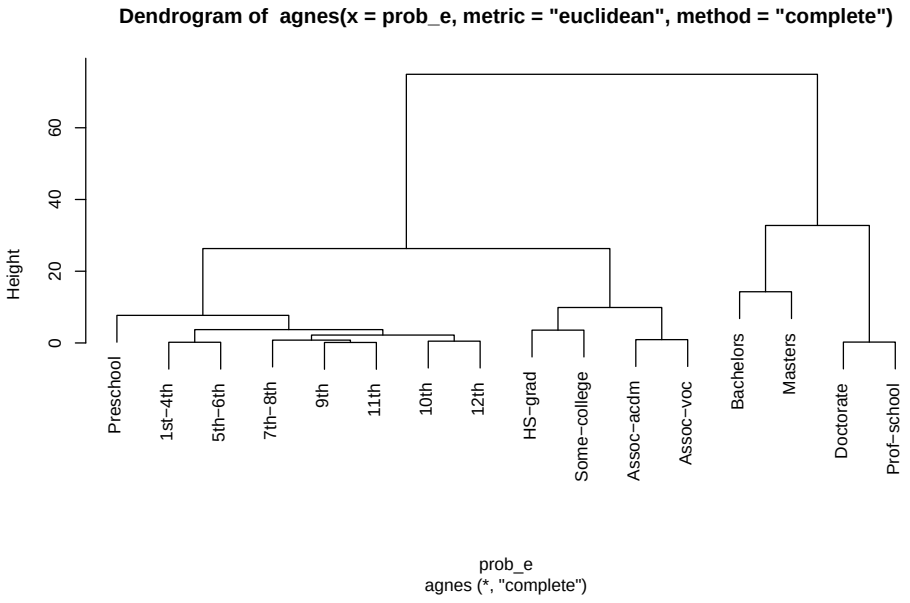
Comparing the two resulted taxonomies of each attribute, we find the similarity between them is very notable. In particular, the two taxonomies of Workclass, Capital-gain, and Capital-loss are identical. If we inspect each pair of taxonomies, most clusters at each level are nearly the same. The AGNES hierarchies differ from DIANA hierarchies on the different order of merging or splitting some small sets of clusters. For example, the two Education taxonomies only differ depending upon the decision as to whether we merge the clusters, {*7th-8th*} and {*9th, 11th*}, or split them.

The same situation occurs in the Marital-status taxonomies. Cluster {*Separated*} is merged with {*Married-spouse-absent*} in AGNES, while they are split and merged with the other two clusters in DIANA.

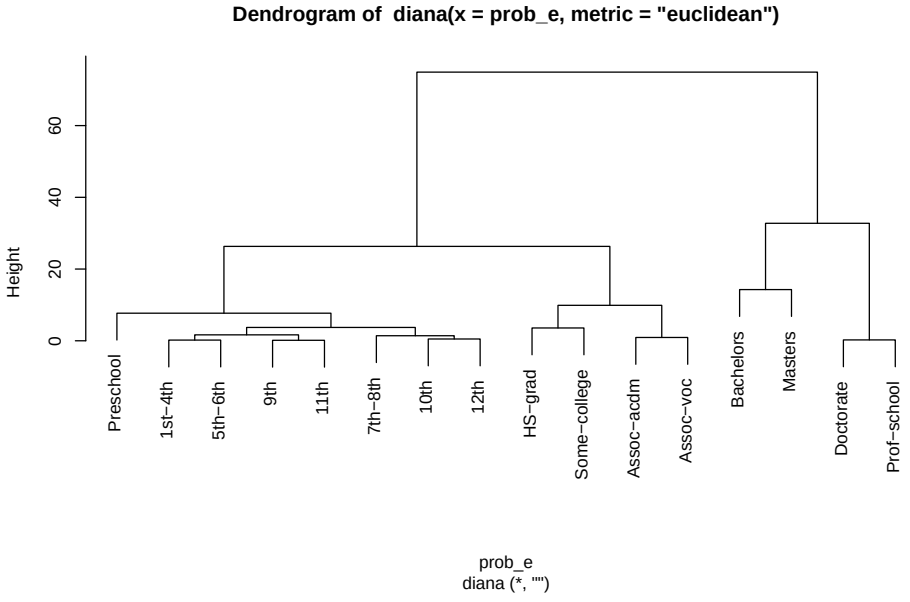
In the Occupation taxonomies, the cluster {{*Farmimg-fishing, Armed-Forces*}, {*Adm-clerical, Machine-op-inspct*}} is either merged with cluster {*Craft-repair, Transport-moving*} or with cluster {{*Handler-cleaners, Other-service*}, *Priv-house-serv*}.

The two taxonomies of Native-country look very different, but actually they have the same two clusters at the top level, e.g. {*Canada, Germany, Italy, England, China, Philippines, Hong, United-States, Greece, Cuba, Hungary, France, Taiwan, Iran, Yugoslavia, India, Japan, Cambodia,*} and {*Puerto-Rico, Trinadad&Tobago,*

[1]R is a language and environment for statistical computing and graphics. R package can be found at http://www.r-project.org/.

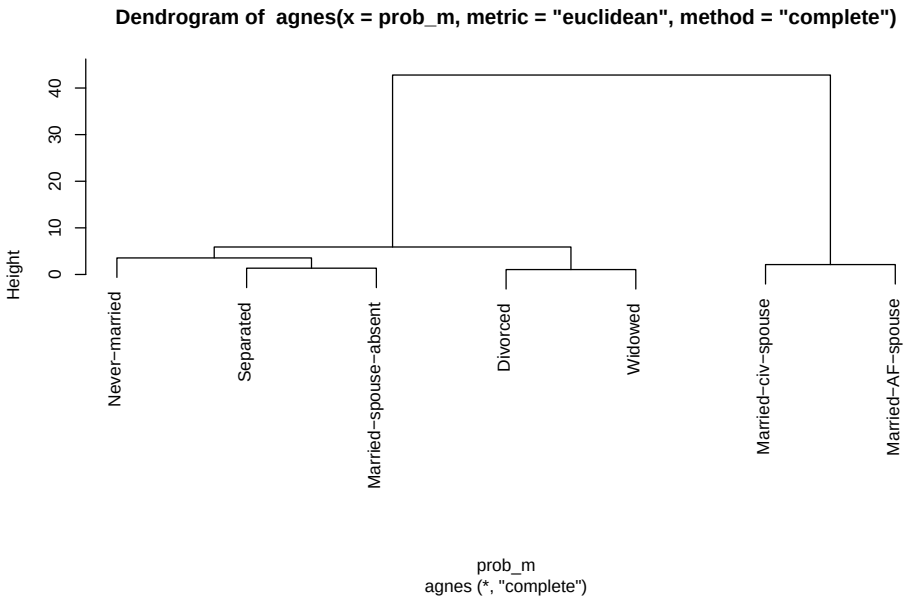


(a) Applying AGNES Algorithm

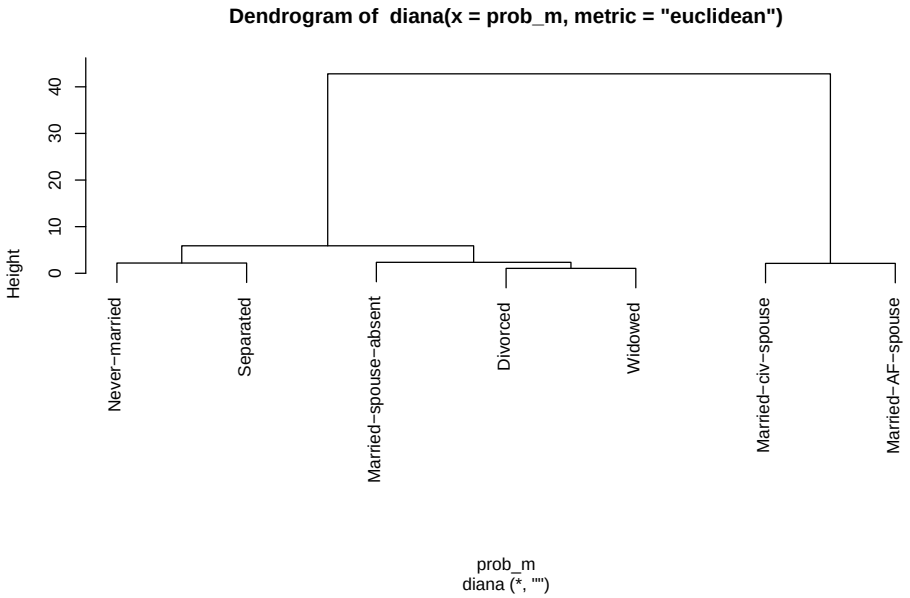


(b) Applying DIANA Algorithm

Figure 4.2: Taxonomies of Education by using Hierarchical Algorithms

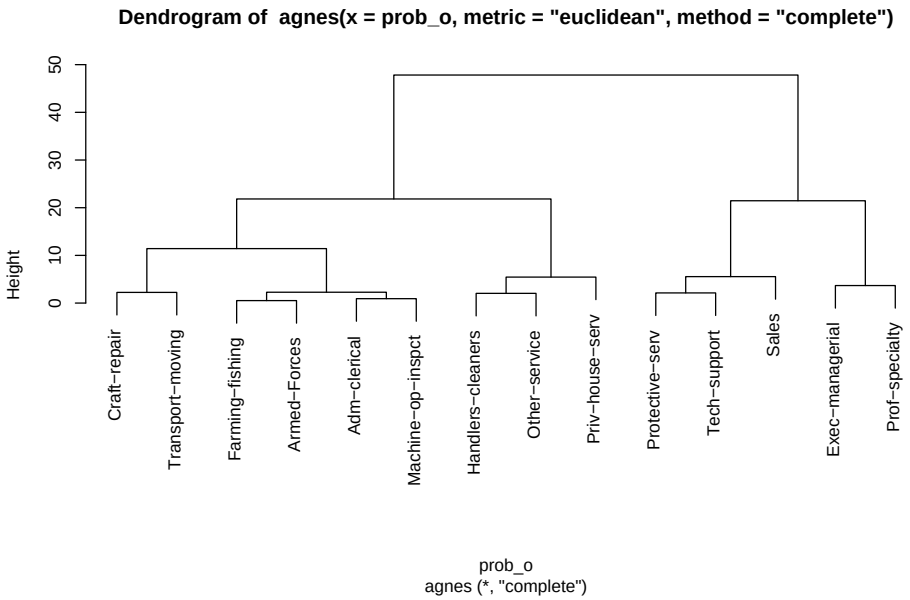


(a) Applying AGNES Algorithm

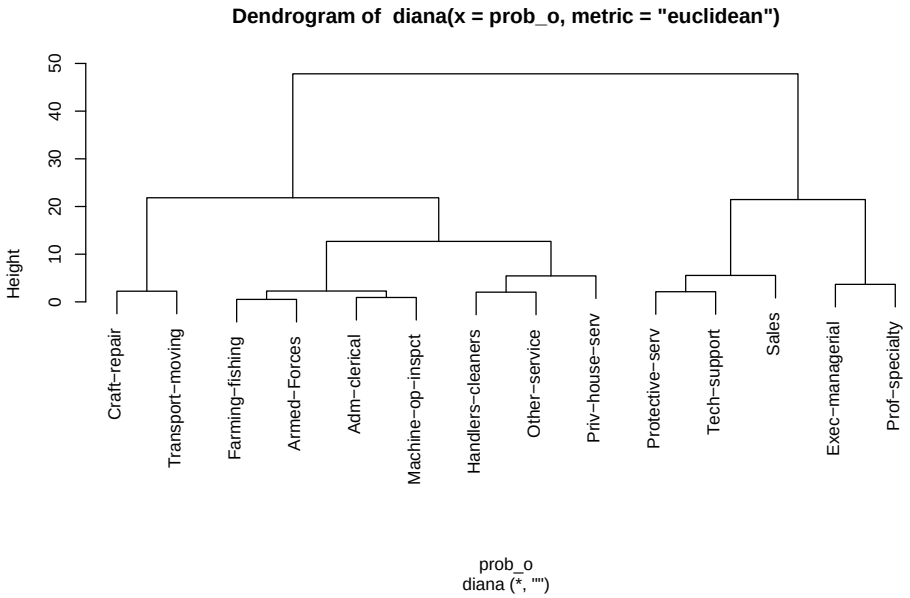


(b) Applying DIANA Algorithm

Figure 4.3: Taxonomies of Marital-status by using Hierarchical Algorithms

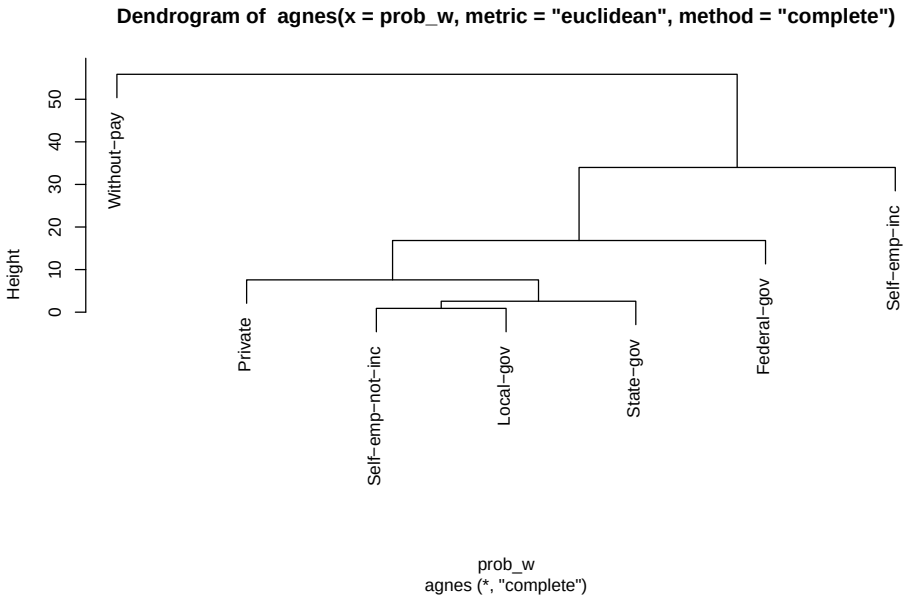


(a) Applying AGNES Algorithm

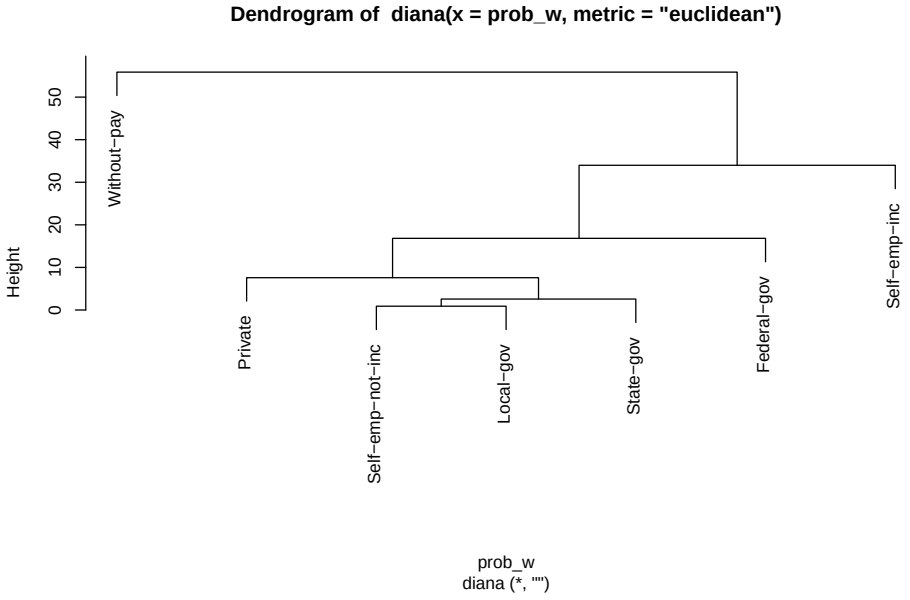


(b) Applying DIANA Algorithm

Figure 4.4: Taxonomies of Occupation by using Hierarchical Algorithms

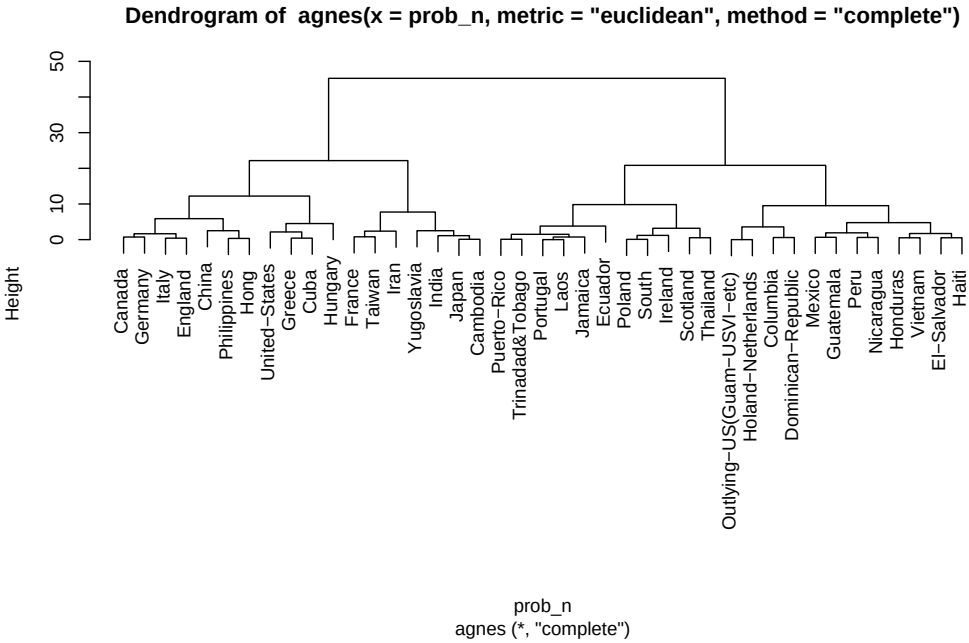


(a) Applying AGNES Algorithm

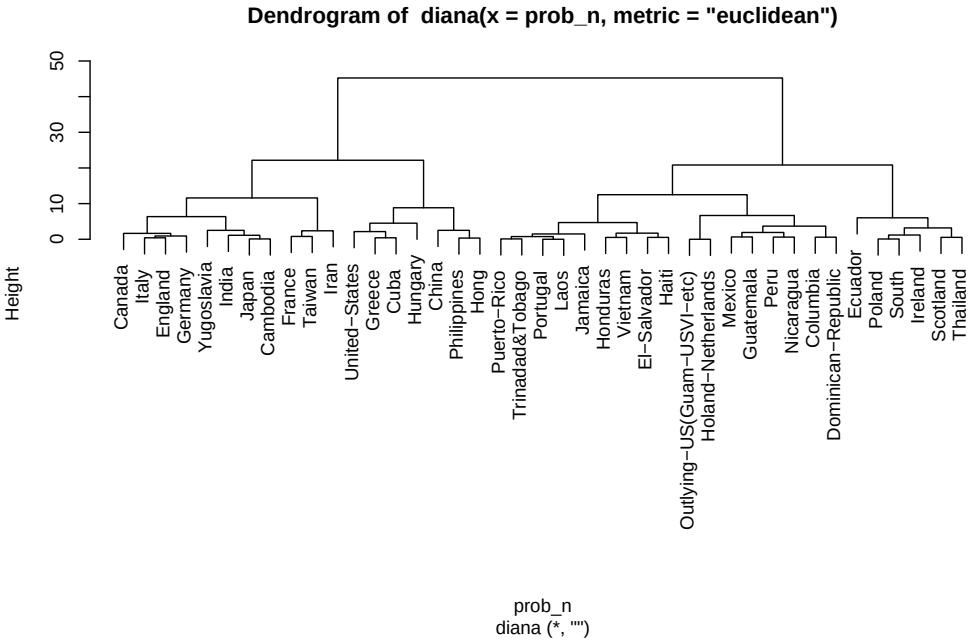


(b) Applying DIANA Algorithm

Figure 4.5: Taxonomies of Workclass by using Hierarchical Algorithms

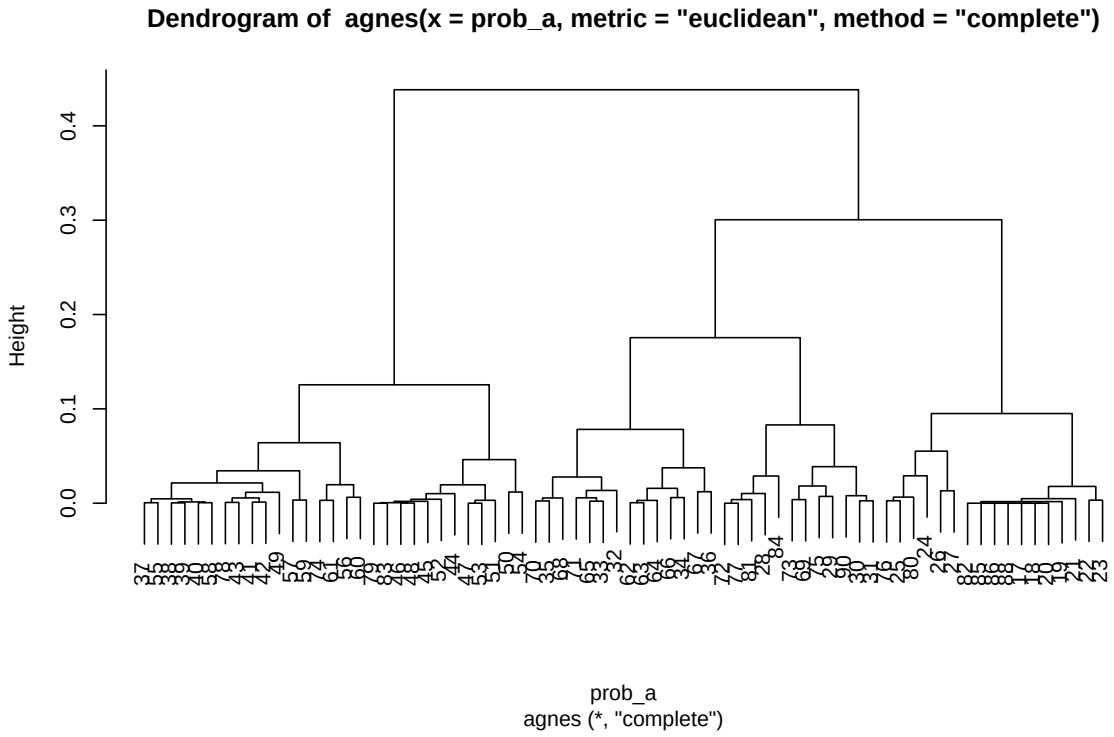


(a) Applying AGNES Algorithm

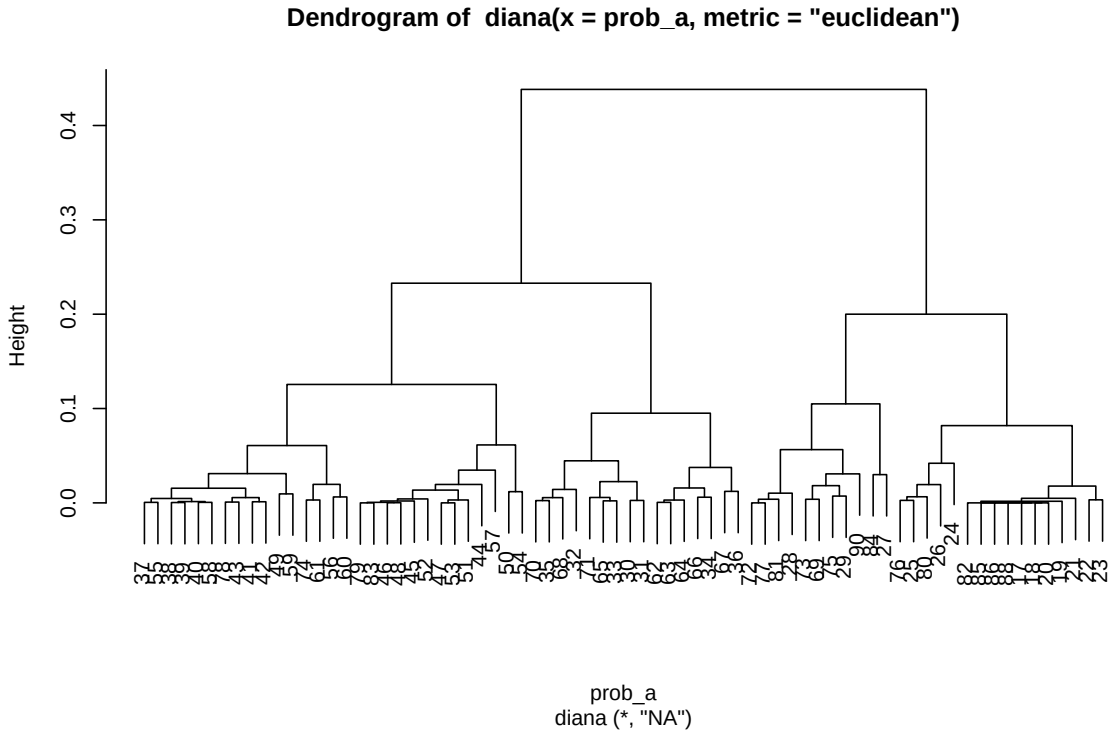


(b) Applying DIANA Algorithm

Figure 4.6: Taxonomies of Native-country by using Hierarchical Algorithms

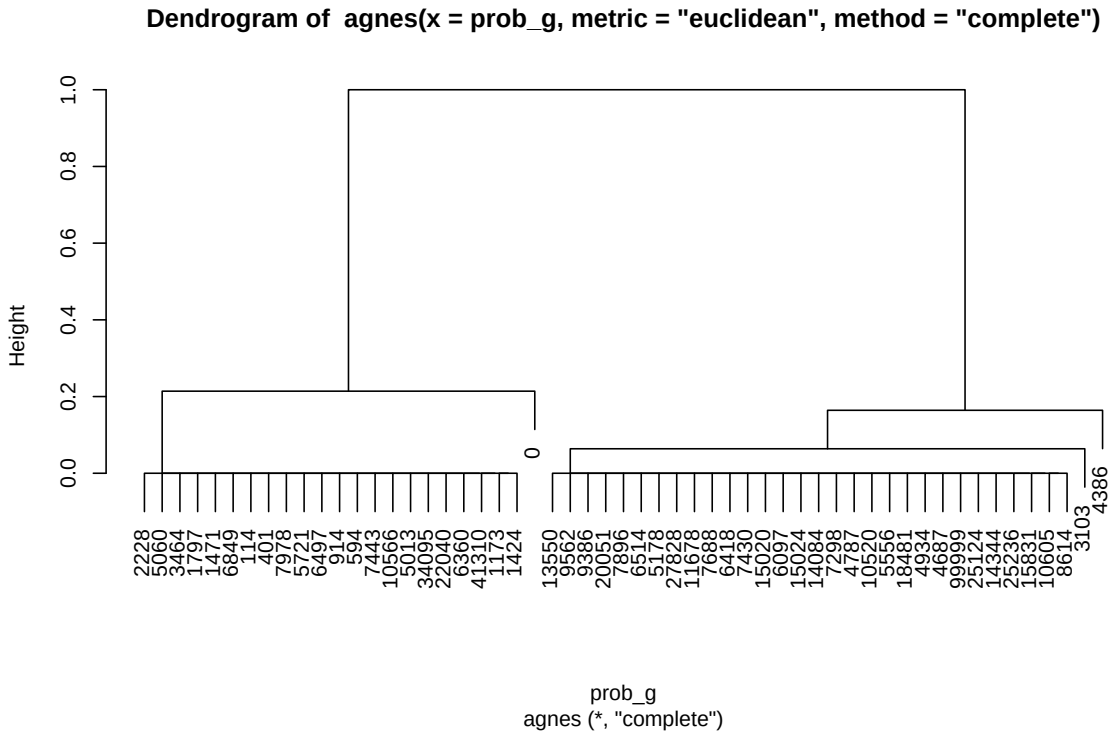


(a) Applying AGNES Algorithm

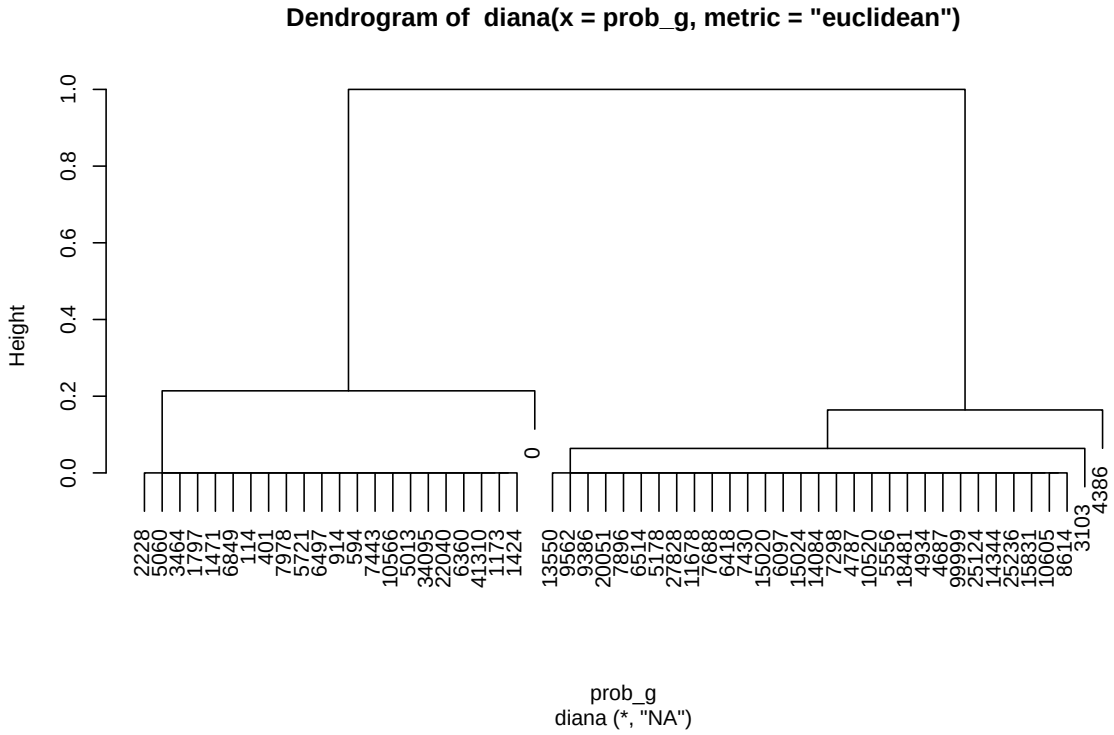


(b) Applying DIANA Algorithm

Figure 4.7: Taxonomies of Age by using Hierarchical Algorithms

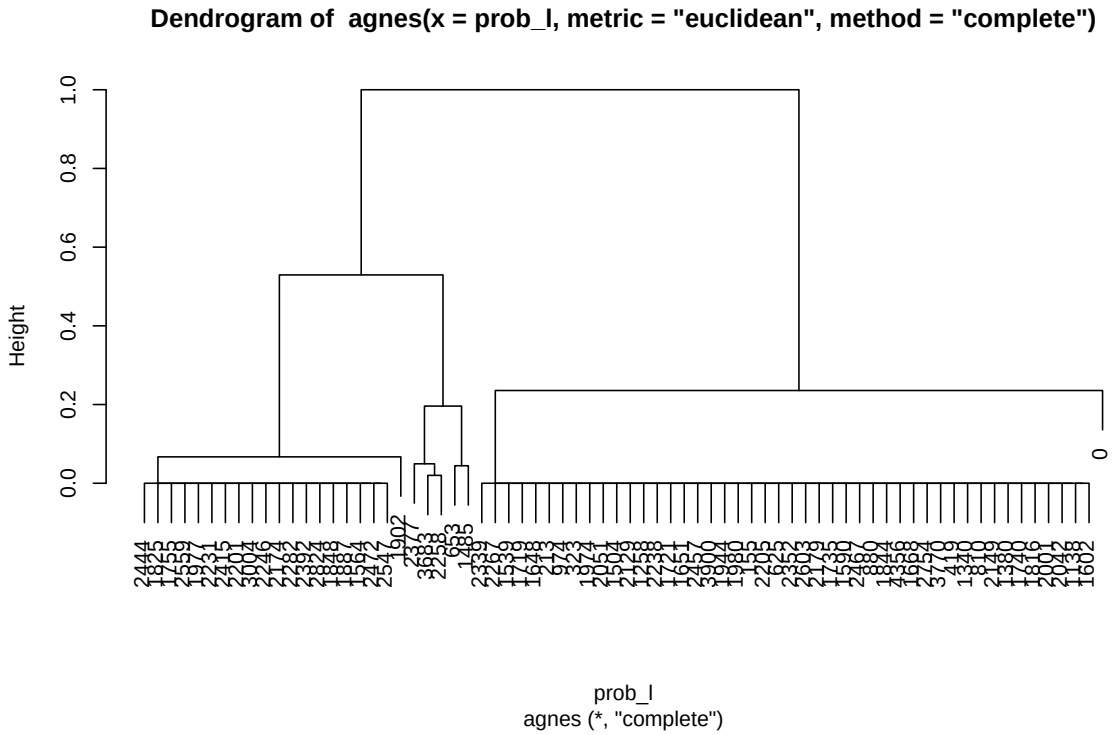


(a) Applying AGNES Algorithm

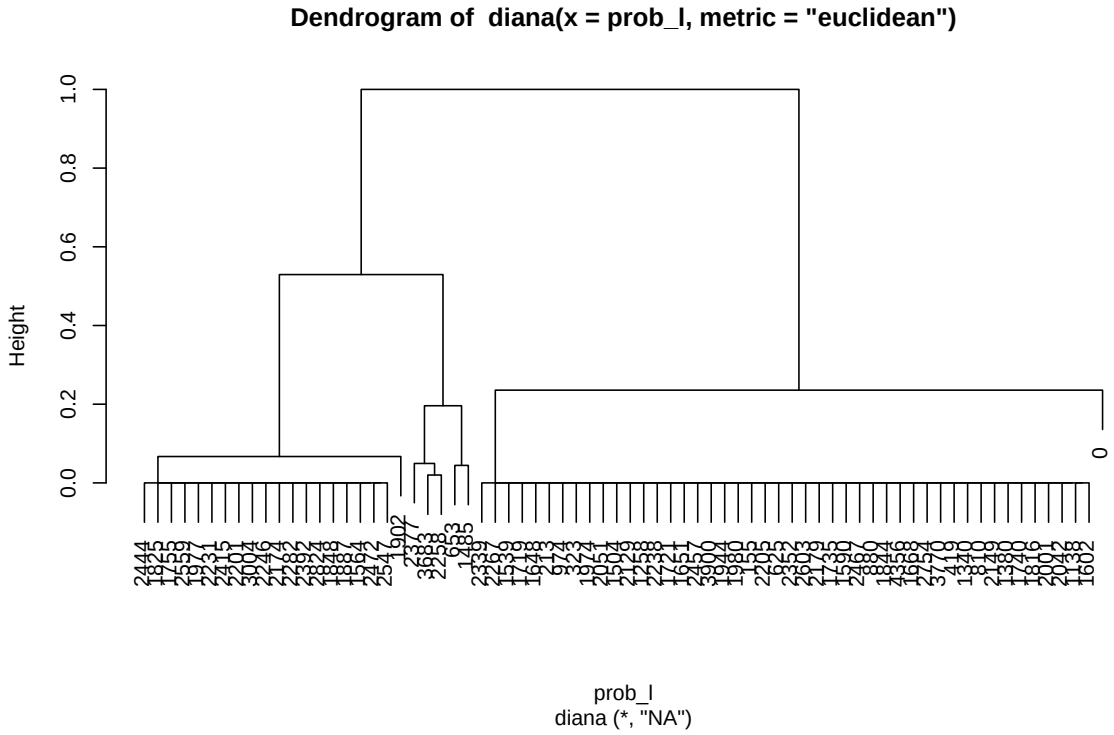


(b) Applying DIANA Algorithm

Figure 4.8: Taxonomies of Capital-gain by using Hierarchical Algorithms



(a) Applying AGNES Algorithm



(b) Applying DIANA Algorithm

Figure 4.9: Taxonomies of Capital-loss by using Hierarchical Algorithms

Portugal, Laos, Jamaica, Ecuador, Poland, South, Ireland, Scotland, Thailand, Outlying-US(Guam-USVI-etc), Holand-Netherlands, Columbia, Dominican-Republic, Mexico, Guatemala, Peru, Nicaragua, Honduras, Vietnam, El-Salvador, Haiti}. The same small set of clusters could also be found in the middle of the taxonomy, e.g. {*Canada, Germany, Italy, England*}, {*China*}, {*Philippines, Hong*}, {*United-States, Greece, Cuba*}, {*Hungary*}, {*France, Taiwan, Iran*}, {*Yugoslavia, India, Japan, Cambodia*}, {*Puerto-Rico, Trinadad&Tobago, Portugal, Laos, Jamaica*}, {*Ecuador*}, {*Poland, South, Ireland*}, {*Scotland, Thailand*}, {*Outlying-US (Guam-USVI-etc), Holand-Netherlands*}, {*Columbia, Dominican-Republic*}, {*Mexico, Guatemala, Peru, Nicaragua*}, and {*Honduras, Vietnam, El-Salvador, Haiti*}. The main differences between the two taxonomies have resulted from the different mergers of these middle level clusters.

As for the three numeric attributes, there are 72, 118, and 90 unique values for Age, Capital-gain, and Capital-loss respectively. Moreover, only 65 values of Age, 33 values of Capital-gain, and 26 values of Capital-loss for occur in records with target class "$> 50K$", which means most values of Capital-gain and Capital-loss will be replaced by 0 conditional probability. In order to show the generated taxonomies more clearly, we just choose some typical values among those values that hold 0 conditional probability for both Capital-gain and Capital-loss, so there are only 55 and 72 values being plotted in figure 4.8 and 4.9.

Unfortunately, the taxonomies in figures 4.7, 4.8, and 4.9 just capture the natural structure of the *Adult* data. It is difficult to find some disjoint subintervals to present the nodes at higher levels of the taxonomies. For example, in figure 4.7, the ages between 17 to 23 are clustered with ages between 82 to 88, while age 90, 29/30, 75 are merged in another cluster. In figure 4.8, some pairs of nearby values are allocated to different clusters, e.g. 10566 and 10605, and 6497 and 6514. The same thing happens in figure 4.9, e.g. 2201 and 2205, and 1977 and 1974.

Ideally, we might hope numeric values in a certain interval would be clustered together. For the three numeric attributes of the *Adult* data, it seems more interested and reasonable to find the discretisation for the values, rather than the taxonomies. In the next section, partitional algorithms will be exploited for this purpose. We will only be interested in finding the optimal clusters for numeric attributes.

4.4 Exploiting Partitional Algorithms for Attribute-Value Taxonomy Construction and Discretisation

As described in section 4.2, the tree hierarchy of clusters can be built from bottom to top by applying the partitional clustering algorithm repeatedly to the resulting clusters.

Given a value set, $V = \{V_1, V_2, ..., V_l\}$, of an attribute, A, the procedure of partitional clustering based attribute-value taxonomy construction is described as below.

1. Let the number of clusters, k, equal the size of value set, V, then the leaves of the tree are $\{V_i\}$ for each value $V_i \in V$. Call this clustering, C.

2. Decrease k under certain criteria, and apply a partitional algorithm to C to find k clusters.

3. Replace each cluster with its centroid, and reset C to be the new k singleton clusters.

4. Go to step 2 until k reaches 2.

In this section, k-means and Fisher's algorithms are used to construct the attribute-value taxonomies for the five selected nominal attributes of the *Adult* data set. Both partitional algorithms will also be applied to the three numeric attributes, but only to find the optimal k clusters according to the average silhouette width. The resulting clusters can be then used for numeric value discretisation.

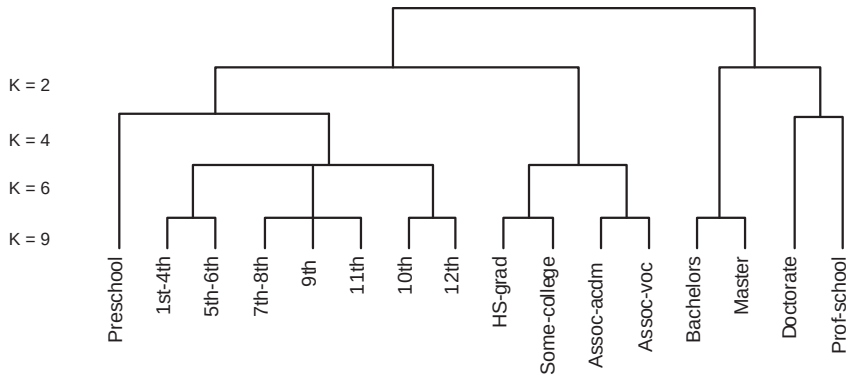
4.4.1 Nominal Attribute-Values Taxonomies Construction

According to above attribute-value taxonomy constructing procedure, the number of clusters, k needs to be preset before running a partitional algorithm at each iteration. In order to compare with the taxonomies generated by hierarchical algorithms, we choose the same or nearest k at each hierarchy level for each selected attribute. Figure 4.10 to figure 4.14 show the pair of nominal attribute-value taxonomies built by iteratively exploiting k-means and Fisher's algorithm.

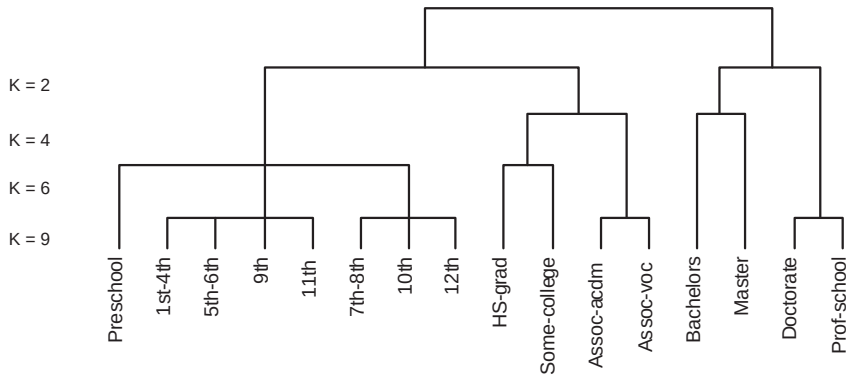
Generally, the same clusters can be found from the top levels of the taxonomies produced by k-means and Fisher's algorithm. In figure 4.10, the only difference between the two Education taxonomies is that the values in the subset {*1st-4th, 5th-6th, 7th-8th, 9th, 10th, 11th, 12th*} are clustered in different ways at the bottom level. The same situation can be found in figure 4.12, two subsets of Occupation {*Tech-support, Sales, Protective-serv*}, and {*Adm-clerical, Armed-Forces, Machine-op-inspct, Farming-fishing*} are clustered differently.

For the two attributes with the smallest domains, i.e. Marital-status and Workclass, the generated taxonomies are either totally different or identical, see figure 4.11 and figure 4.13. However, comparing the two taxonomies of Marital-status, we find the one generated by Fisher's algorithm is more close to the semantic taxonomy we normally use.

In the two Native-country taxonomies, most small set of clusters are identical at the second bottom-level when $K = 15$, e.g. {*Canada, Germany, Italy, Eng-*

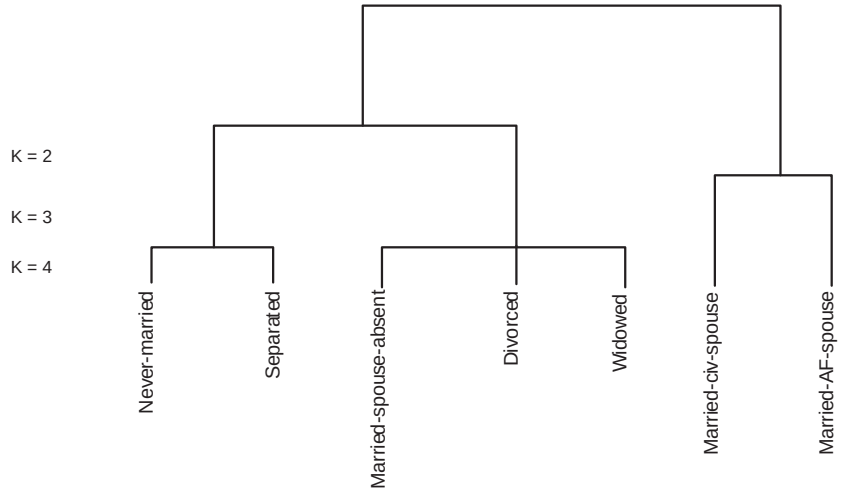


(a) Applying k-means Algorithm

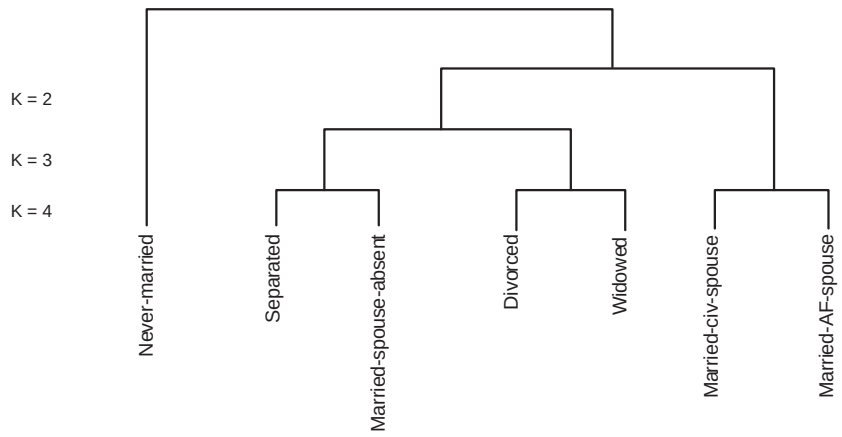


(b) Applying Fisher's Algorithm

Figure 4.10: Taxonomies of Education by using Partitional Algorithms

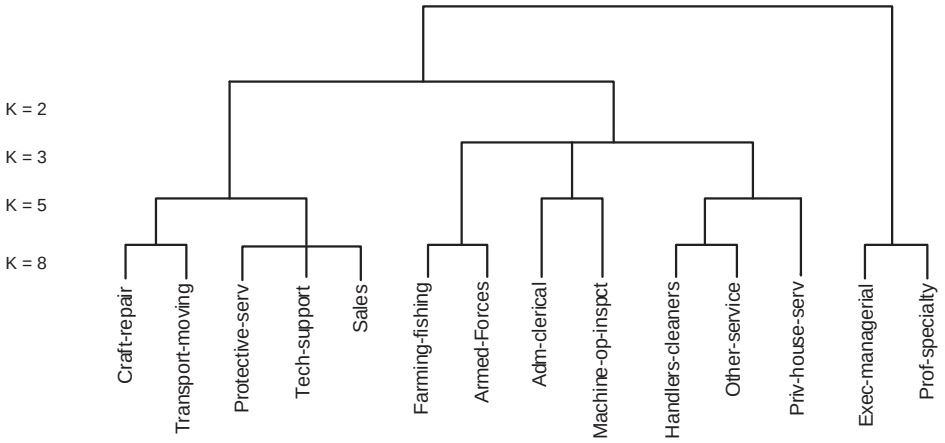


(a) Applying k-means Algorithm

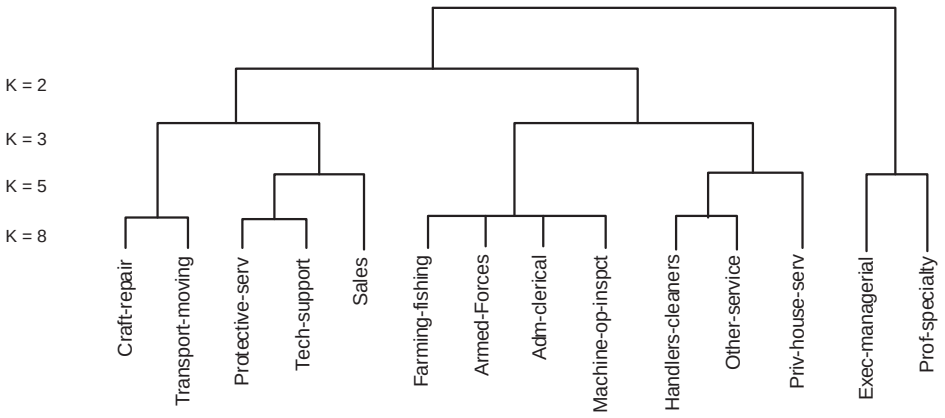


(b) Applying Fisher's Algorithm

Figure 4.11: Taxonomies of Marital-status by using Partitional Algorithms

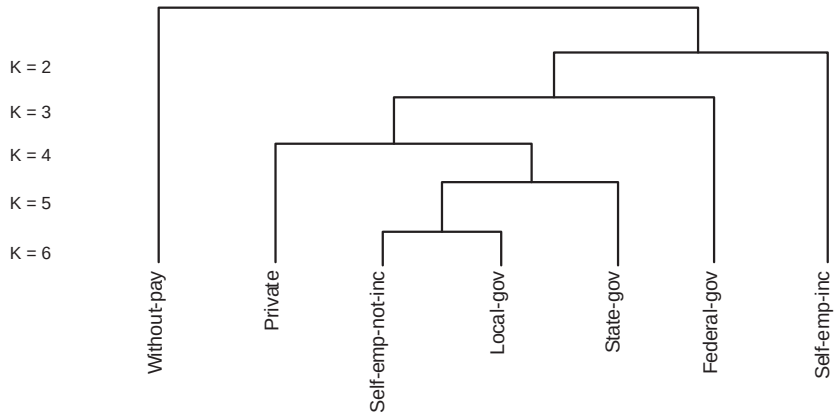


(a) Applying k-means Algorithm

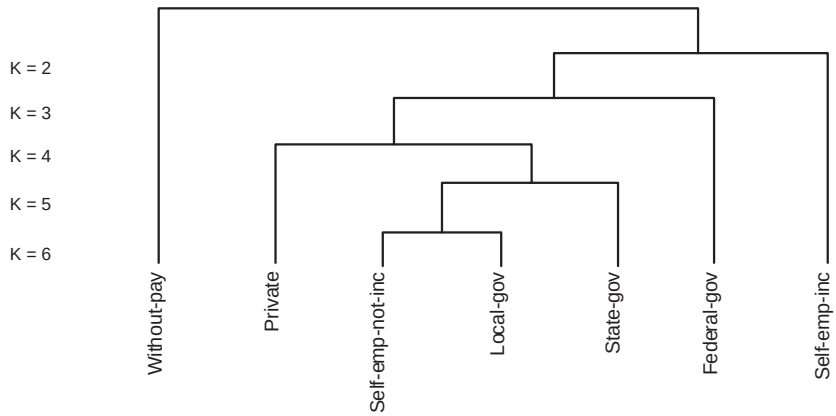


(b) Applying Fisher's Algorithm

Figure 4.12: Taxonomies of Occupation by using Partitional Algorithms

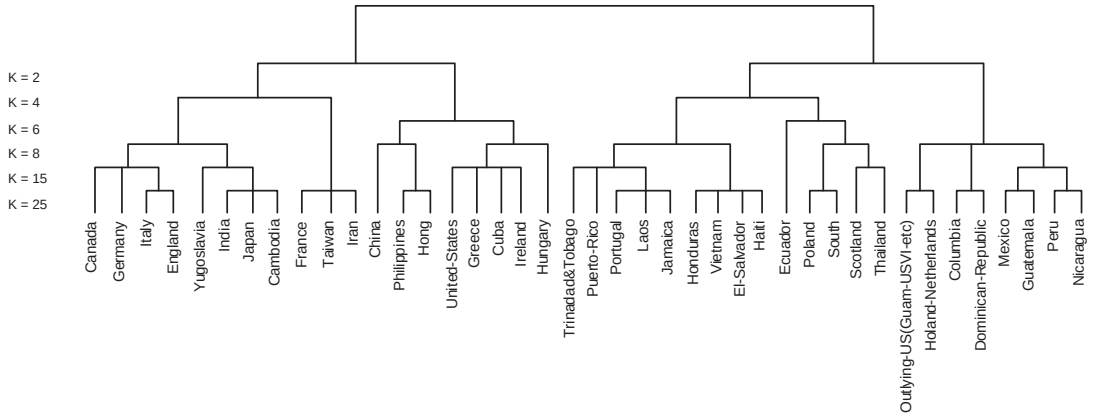


(a) Applying k-means Algorithm

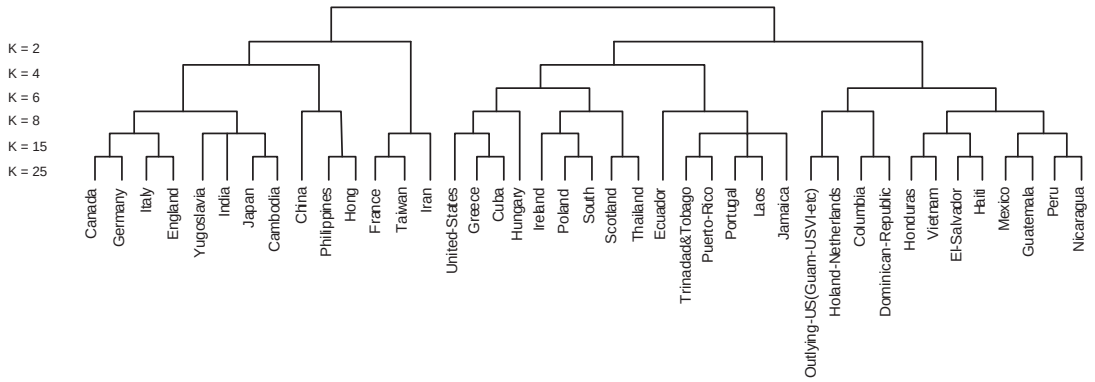


(b) Applying Fisher's Algorithm

Figure 4.13: Taxonomies of Workclass by using Partitional Algorithms



(a) Applying k-means Algorithm



(b) Applying Fisher's Algorithm

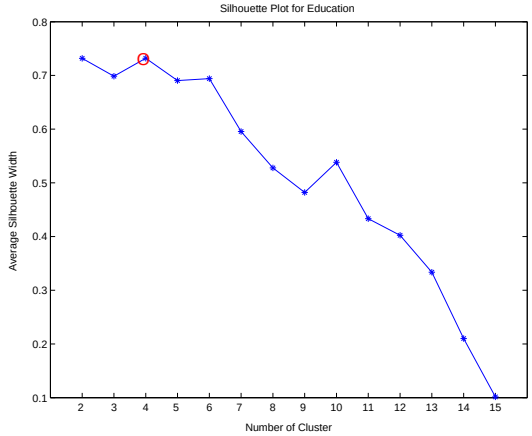
Figure 4.14: Taxonomies of Native-country by using Partitional Algorithms

land} and {*Yugoslavia, India, Japan, Cambodia*}, except {*Ireland*} is clustered with {*United-States, Cuba, Greece*} by k-means, while it is merged with cluster {*Poland, South*} by Fisher's algorithm.

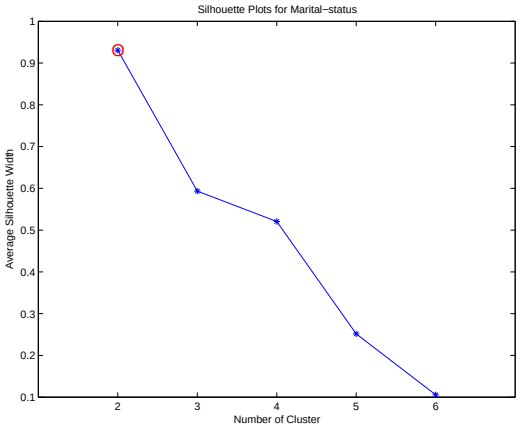
For each nominal attribute, the average silhouette widths are also calculated for k up to the number of attribute values, see figure 4.15. The optimal number of clusters are marked by a red circle in each plot. Interestingly, given the optimal number k, both k-means and Fisher's algorithm generate the same optimal clusters, see table 4.1 for details. These optimal number of clusters are also used when constructing the AVTs by partitional algorithms, therefore the optimal clustering will be guaranteed to appear in the taxonomies when k is set to be the optimal number.

Table 4.1: The optimal clusters for the nominal attribute values of the *Adult* data set

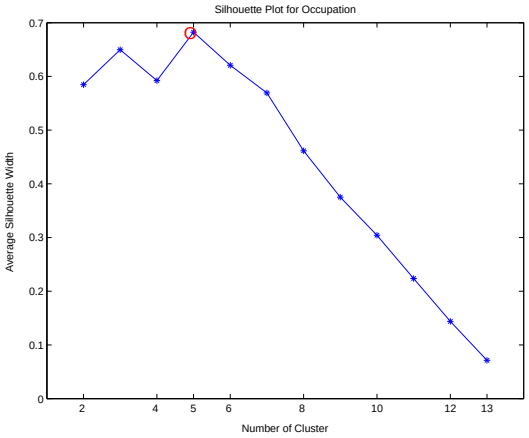
Attribute & Cluster No.	**Optimal Clusters**
Education (k = 4)	{Preschool, 1st-4th, 5th-6th, 7th-8th, 9th, 10th, 11th, 12th}, {HS-grad, Assoc-acdm, Assoc-voc, Some-college}, {Bachelors, Masters}, {Prof-school, Doctorate}
Marital-status (k = 2)	{Married-civ-spouse, Married-AF-spouse}, {Never-married, Widowed, Separated, Married-spouse-absent, Divorced}
Occupation (k = 5)	{Adm-clerical, Armed-Forces, Machine-op-inspct, Farming-fishing}, {Craft-repair, Transport-moving}, {Exec-managerial, Prof-specialty} {Handlers-cleaners, Priv-house-serv, Other-service}, {Sales, Tech-support, Protective-serv}
Workclass (k = 2)	{Without-pay}, {Self-emp-inc, Self-emp-not-inc, Local-gov, State-gov, Federal-gov, Private}
Native-country (k = 2)	{Canada, Italy, England, Germany, France, Hungary, Yugoslavia, Greece, United-States, Cuba, China, HongKong, Taiwan, India, Japan, Iran, Philippines, Cambodia}, {Outlying-US(Guam-USVI-etc), Holand-Netherlands, Guatemala, Mexico, Nicaragua, Peru, Columbia, Dominican-Republic, Haiti, Honduras, Vietnam, El-Salvador, Portugal, Laos, Jamaica, Puerto-Rico, Trinadad&Tobago, Ecuador, Ireland, Thailand, Scotland, SouthKorea, Poland}



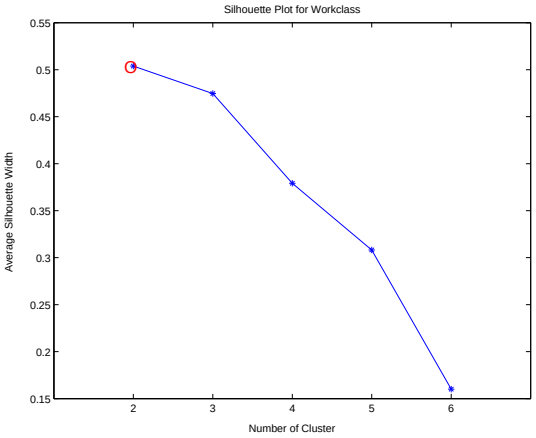
(a) Education



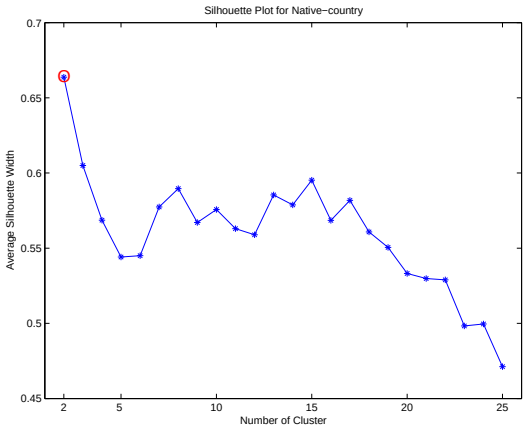
(b) Marital-status



(c) Occupation



(d) Workclass



(e) Native-country

Figure 4.15: Silhouette plot for Nominal Attributes

4.4.2 Numeric Attribute-values Discretisation

There are some approaches successfully dealing with numeric value discretisation. For instance, equal-width-interval method is one of the traditional methods, which divides all the attribute values into equal width bins/subintervals. But the problem is that it does not consider the natural structure of the specific data and the domain background knowledge.

In this section, we aim to find optimal partitions for the numeric values of three selected attributes by exploiting both k-means and Fisher's algorithm on the conditional probability based value set. These partitions can be directly used or provide a guide for numeric value discretisation. The average silhouette width will still play a key role in choosing the optimal number of clusters.

As mentioned in section 4.3.3, less than one third of values in the value set of both Capital-gain and Capital-loss correspond to the class "$> 50K$". Moreover, we notice that most of these "relevant" values only occur in the data whose target class is "$> 50K$", which means they will be replaced by 1 according to equation 4.11. This will greatly shrink the space of the value set to be clustered. Table 4.2 shows the number of unique values, or the size of the value set for the three numeric attributes under different conditions. CP presents the converted value set by using equation 4.11.

Table 4.2: The number of unique values for the numeric attributes of the *Adult* data set

Attribute	Mean	**Number of Unique Values** Whole Data	Classified as ">50K"	CP
Age	37.74	72	65	66
Capital-gain	1219.90	118	33	5
Capital-loss	99.49	90	26	9

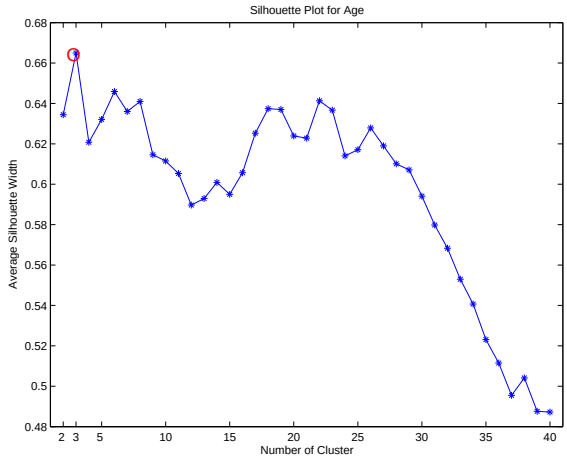
After the conversion, the size of value set for Capital-gain and Capital-loss is dramatically reduced to 5 and 9 respectively.

Figure 4.16 shows the silhouette plot for all the three numeric attributes over all the possible k. Similarly, by setting k to be the optimal number of cluster beforehand, both k-means and Fisher's algorithm produce identical optimal clusters for each attribute, see table 4.3 for detail.

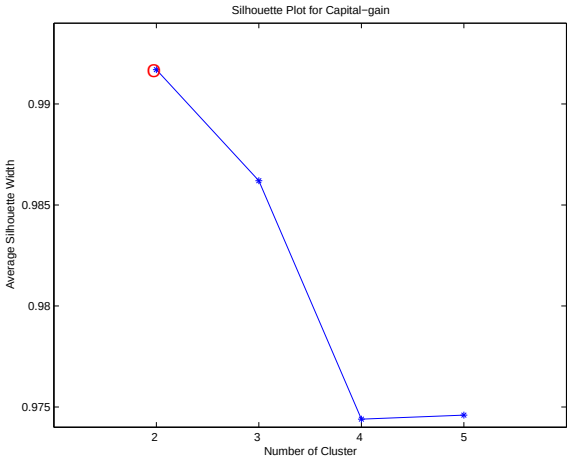
Table 4.3: The optimal clusters for the numeric attribute values of the *Adult* data set

Attribute & Cluster No.	**Optimal Clusters**
Age (k = 3)	{36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 74, 78, 79, 83}, {28, 29, 30, 31, 32, 33, 34, 35, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 75, 77, 81, 84, 90}, {17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 76, 80, 82, 85, 86, 88}
Capital-gain (k = 2)	{3103, 4386, 4687, 4787, 4934, 5178, 5556, 6097, 6418, 6514, 7298, 7430, 7688, 7896, 8614, 9386, 9562,10520, 10605, 11678, 13550, 14084, 14344, 15020, 15024, 15831, 18481 20051, 25124, 25236, 27828, 99999} {0, 114, 401, 594, 914, 991, 1055 1086, 1151, 1173, 1409, 1424, 1455, 1471, 1506, 1639, 1797, 1831, 1848, 2009, 2036, 2050, 2062, 2105, 2174, 2176, 2202, 2228, 2290, 2329, 2346, 2354, 2387, 2407, 2414, 2463, 2538, 2580, 2597, 2635, 2653, 2829, 2885, 2907, 2936, 2961, 2964, 2977, 2993, 3137, 3273, 3325, 3411, 3418, 3432, 3456, 3464, 3471, 3674, 3781, 3818, 3887, 3908, 3942, 4064, 4101, 4416, 4508, 4650, 4865, 4931, 5013, 5060, 5455, 5721, 6360, 6497, 6723, 6767, 6849, 7443, 7978, 10566, 22040, 34095, 41310},
Capital-loss (k = 5)	{0}, {653, 1485}, {2258, 2377, 3683}, {1564, 1755, 1825, 1848, 1887, 1902, 1977, 2174, 2201, 2231, 2246, 2282, 2392, 2415, 2444, 2472, 2547, 2559, 2824, 3004}, {155, 213, 323, 419, 625, 810, 880, 974, 1092, 1138, 1258, 1340, 1380, 1408, 1411, 1504, 1539, 1573, 1579, 1590, 1594, 1602, 1617, 1628, 1648, 1651, 1668, 1669, 1672, 1719, 1721, 1726, 1735, 1740, 1741, 1762, 1816, 1844, 1876, 1944, 1974, 1980, 2001, 2002, 2042, 2051, 2057, 2080, 2129, 2149, 2179, 2205, 2206, 2238, 2267, 2339, 2352, 2457, 2467, 2603, 2754, 3770, 3900, 4356}

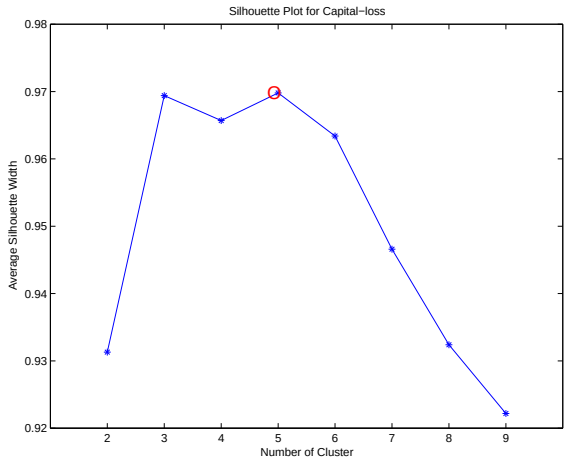
If we inspect the optimal clusters of Age, we observe that there are some subintervals that can be extracted from the clusters. For example, the subintervals, such as [36, 61], [28, 35], [62, 73], and [17, 27], only occur in one specific cluster, while most values within the range [80, 90] will be clustered with the subinterval [17, 27]. This is useful information which indicates a rule that if the person's age is below 27 or over 80, he will not earn more than $50K per year. In order to reflect



(a) Age



(b) Capital-gain



(c) Capital-loss

Figure 4.16: Silhouette plot for Numeric Attributes

the clustering results, the values of attribute Age can be discretised to following six boundary bins: [17, 27], [28, 35], [36, 61], [62, 73], [73, 79], and [80, 90].

For Capital-gain and Capital-loss, the situation is more complex. It is not easy to find some subintervals to represent each cluster without intersection, since for most values in cluster 1, their nearby values also occur in cluster 2.

Searching the optimal clusters of Capital-gain, we only find the values within the two subintervals, [8614, 10520] and [25124, 27828], can 100% guarantee the data hold those values will be classified to "$> 50K$" class. Besides, the values within [0, 3103) and (3103, 4101], can indicate the data belongs to class "$\leq 50K$". So the partition of values of Capital-gain is: [0, 4101], [4102, 8613], [8614, 10520], [10521, 25123], [25124, 27828], and [27829, 99999].

Similarly, the following rules can be found from the clusters of Capital-loss. If a person's capital-loss is 0, he/she can earn more than \$50K per year; if the capital-loss is within the range [1980, 2238] and (3004, 4356], he/she will earn less than \$50K per year. Then the vales of Capital-loss can be discretised as [0, 1), [1, 1979], [1980, 2238], [2239, 3004], and [3005, 4356].

4.4.3 Comparison and Discussion

The above introduction presents our work on constructing AVT using clustering algorithms on the *Adult* data set. Compared to the traditional hierarchical clustering algorithms, which prefers to generate a binary-tree structure, the use of partitional algorithms seems to provide a better solution. This is because, in our case study, the generated nominal taxonomies are more acceptable from the view of domain experts. However, it still has some drawbacks.

Firstly, nominal values are clustered based on conditional probability, which means the taxonomies reflect the statistical features of the data. To complete the taxonomy, we need to use some concepts to represent the internal nodes of

the taxonomies, but this is rather difficult in our case. For example, in Native-country taxonomy, see figure 4.14, it is a challenge to find some suitable terms to represent the clusters at the middle levels, unless just simply concatenate the terms of children nodes, e.g. {*United-States, Greece, Cuba*} {*Canada, Germany*} and {*Italy, England*}. In the numerical case, it becomes even more troublesome, since there always are some intersections between clusters if we want to use subintervals to cover all the values within the cluster.

Secondly, these conditional probability based taxonomies are built through training data. If the test set has a different distribution, this will cause an over-fitting or under-fitting problem. This occurs frequently in numeric attributes. For instance, the values of fnlngt in the training data range from 13769 to 1484705, while test data have a wide range, from 13492 to 1490400. The opposite example comes from attribute Capital-loss. In training data, its values range from 0 to 4356, but the range changed from 0 to 3770 in test data.

Actually, the automatic generation of concept taxonomies is itself a knowledge discovery process. Although the taxonomies generated using clustering algorithms are not perfect, they let us know the relationships among attribute values and the dependencies between the conceptual nodes at different levels in the tree like structure from the statistical point of view. This can be a good guide for us to adapt, transform, and even re-build the concept taxonomies for different tasks.

We take the Native-country taxonomy as an example. Intuitively, countries are often divided according to their geographical location, e.g. *America*, *Asia*, and *Europe*. However, considering their economic situation, they can also be divided into *Developed* and *Developing* country. For the *Adult* data, the attribute Native-country is the census of the native country of people who live in US. Since about 90% of the people are US citizen, not all the existing ontologies or taxonomies are applicable for this case. Furthermore, only 25% US people earn more than $50K per

year, so the clustering taxonomies reflect this characteristic. Given these various clustering taxonomies, we are encouraged to inspect the data more carefully, which may help us to reveal the above facts. In this case, the division of *America* and *Non-America* or simply *US* and *Non-US* for Native-country attribute seems more reasonable.

Finally, to discretise the three numeric attributes of the *Adult* data, we use some categories to represent the value intervals, see table 4.4 and table 4.5 for details. After analysing all the generated taxonomies in this chapter and referring to the taxonomies proposed in [42], five self-refined taxonomies of *Adult* nominal attributes are shown in figure 4.17 to figure 4.21.

Table 4.4: Discretisation for Age

Categorical Value	**Subintervals**
Young	[17, 27]
Middle	(27, 61]
Old	(61, 79]
Very Old	(79, 90]

Table 4.5: Discretisation for Capital-gain & Capital-loss

	Subintervals	
Categorical Value	*Capital-gain*	*Capital-loss*
Low	[0, 5000]	[0, 1980)
Medium	(5000, 11000]	[1980, 3100)
High	(11000, 25500]	[3100, 4356]
Very High	(25500, 99999]	

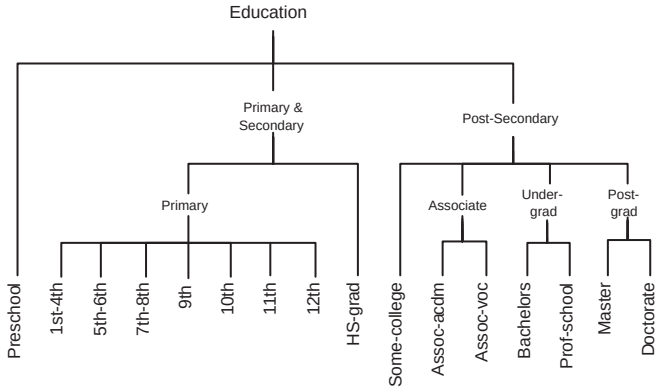


Figure 4.17: Taxonomy of Education

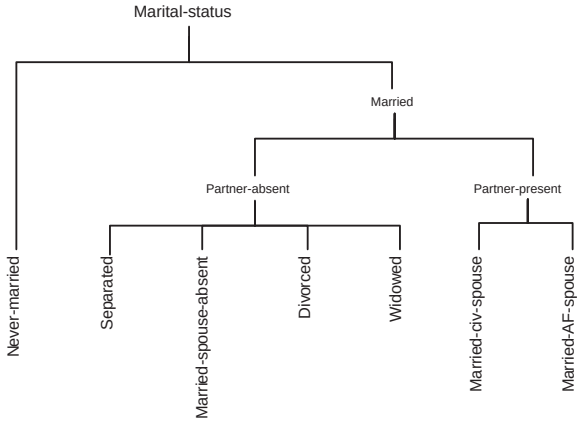


Figure 4.18: Taxonomy of Marital-status

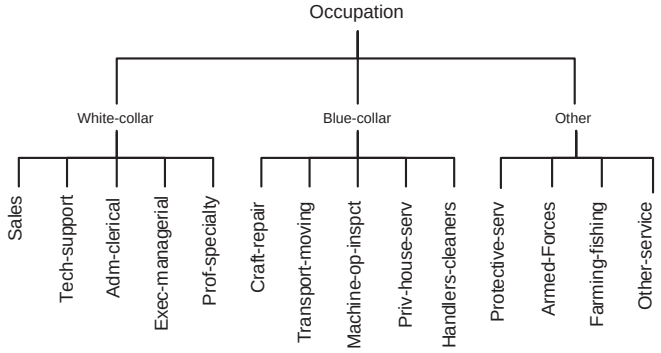


Figure 4.19: Taxonomy of Occupation

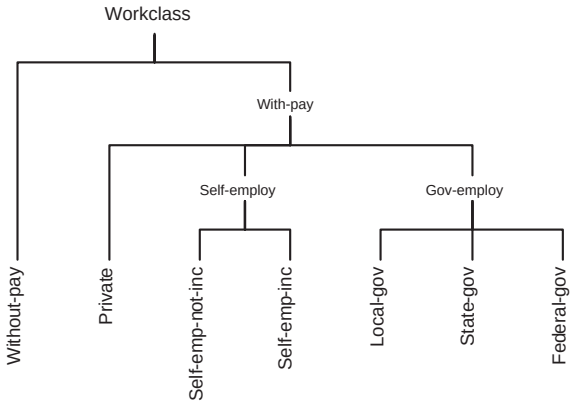


Figure 4.20: Taxonomy of Workclass

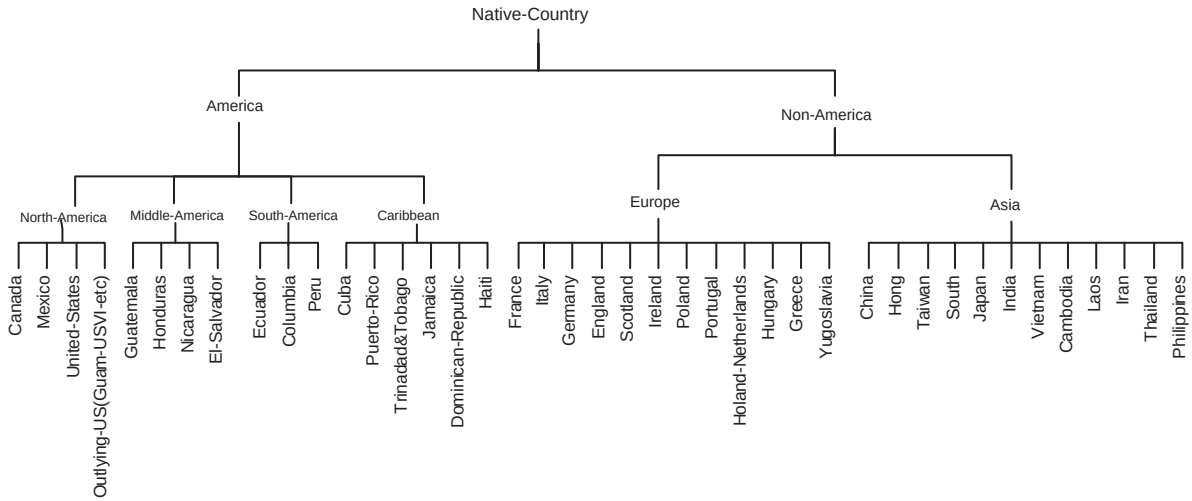


Figure 4.21: Taxonomy of Native-country

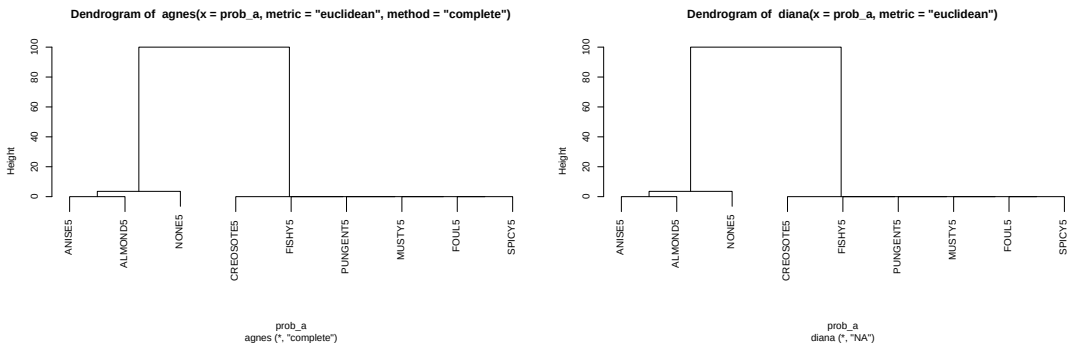
4.5 Construction of AVT on the *Mushroom* Data Set

In this section we will show the generation of attribute-value taxonomies on the *Mushroom* data set by using clustering algorithms. Seven nominal attributes of the *Mushroom* data set, Odor, Spore-print-color, Stalk-color-above-ring, Gill-color, Habitat, Stalk-color-below-ring, and Cap-color, are chosen for taxonomy construction.

Following the same steps on the *Adult* data set described in section 4.3 and 4.4, the same four clustering algorithms are applied to the *Mushroom* data set, respectively.

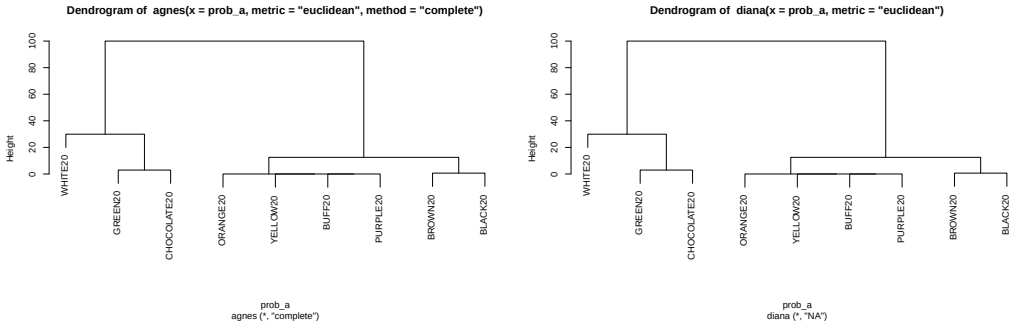
4.5.1 Construction of AVT Using Hierarchical Algorithm

We apply both the AGNES and DIANA algorithms to the matrix of each selected attribute, and the two generated taxonomies will be shown in the same figure, see figure 4.22(a) to figure 4.22(b). In these figures, the "Height" in AGNES presents the distance between merging clusters at the successive stages, whilst it is the diameter of the cluster prior to splitting in DIANA.



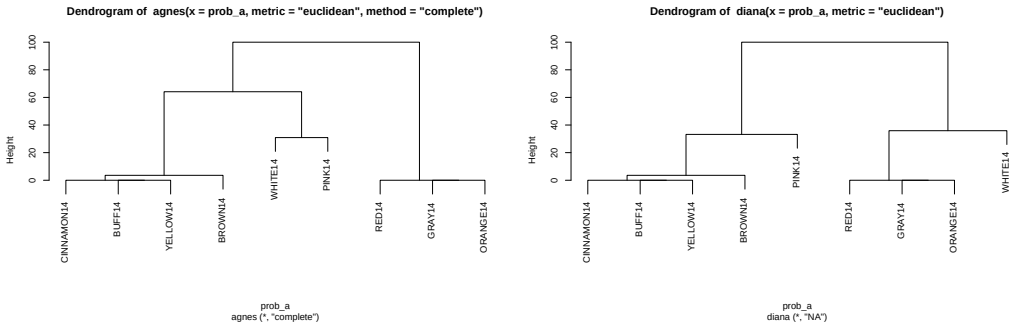
(a) Applying AGNES Algorithm (b) Applying DIANA Algorithm

Figure 4.22: Taxonomies of attribute *Odor* by using hierarchical algorithms



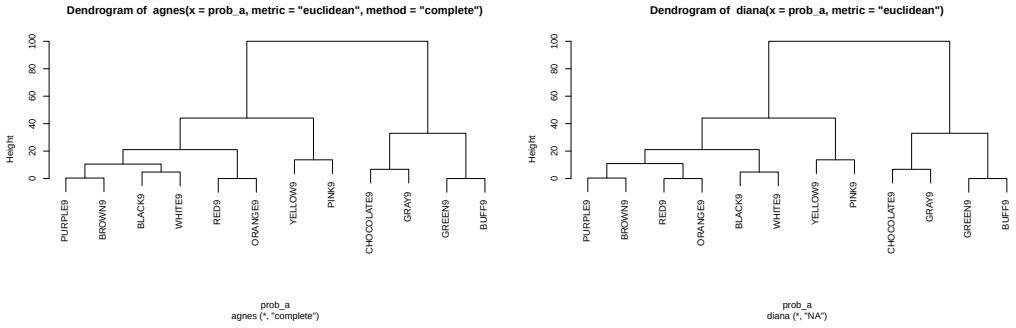
(a) Applying AGNES Algorithm (b) Applying DIANA Algorithm

Figure 4.23: Taxonomies of attribute *Spore_print_color* by using hierarchical algorithms



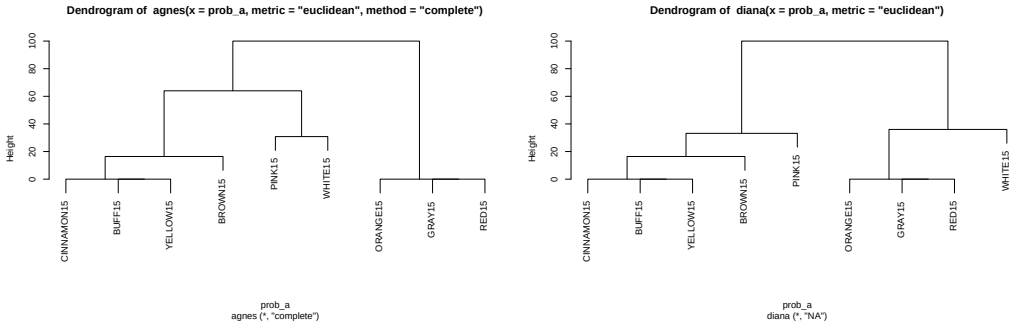
(a) Applying AGNES Algorithm (b) Applying DIANA Algorithm

Figure 4.24: Taxonomies of attribute *Stalk_color_above_ring* by using hierarchical algorithms



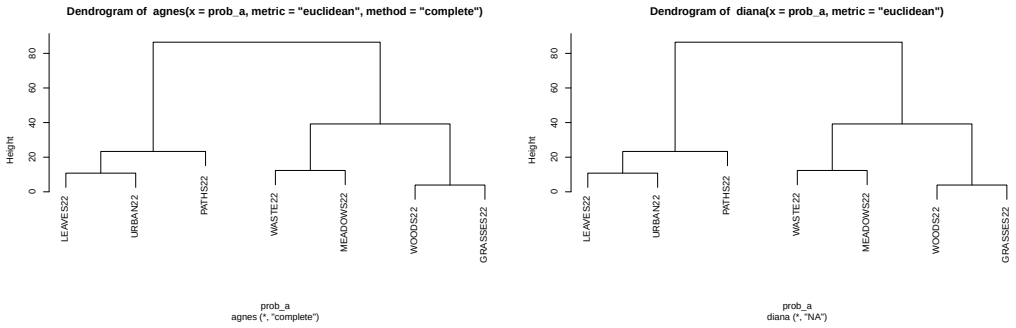
(a) Applying AGNES Algorithm (b) Applying DIANA Algorithm

Figure 4.25: Taxonomies of attribute *Gill_color* by using hierarchical algorithms



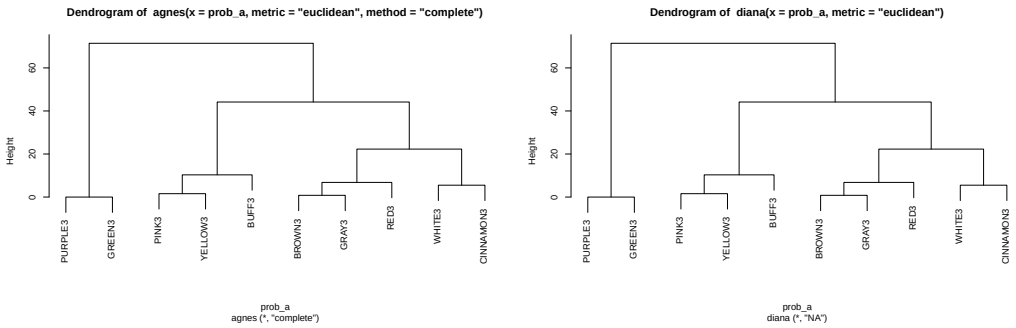
(a) Applying AGNES Algorithm (b) Applying DIANA Algorithm

Figure 4.26: Taxonomies of attribute *Stalk_color_below_ring* by using hierarchical algorithms



(a) Applying AGNES Algorithm (b) Applying DIANA Algorithm

Figure 4.27: Taxonomies of attribute *Habitat* by using hierarchical algorithms

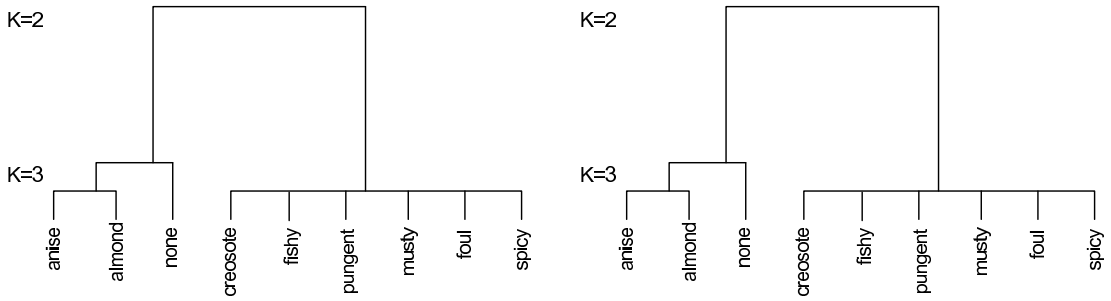


(a) Applying AGNES Algorithm (b) Applying DIANA Algorithm

Figure 4.28: Taxonomies of attribute *Cap_color* by using hierarchical algorithms

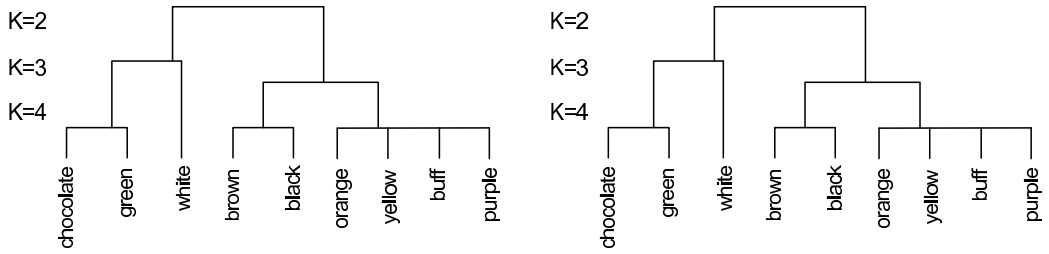
4.5.2 Construction of AVT Using Partitional Algorithms

In this section, as a comparison with the use of hierarchical algorithms, we will use k-means and Fisher's algorithms to construct the attribute-value taxonomies. We choose the same or nearest k at each hierarchy level for each selected attribute. Figure 4.29 $\sim$ 4.35 show the pair of nominal attribute-value taxonomies built by iteratively exploiting k-means and Fisher's algorithm.



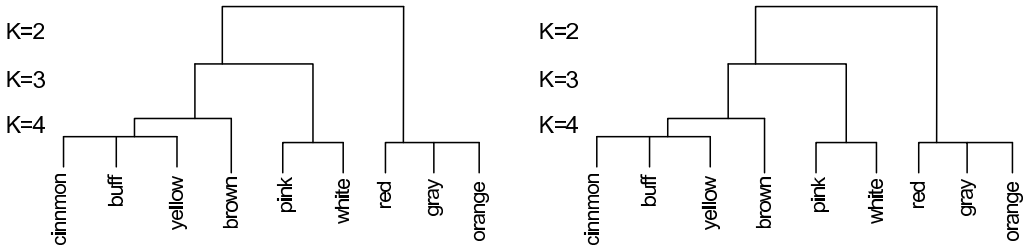
(a) Applying Fisher's Algorithm (b) Applying k-means Algorithm

Figure 4.29: Taxonomies of attribute *Odor* by using partitional algorithms



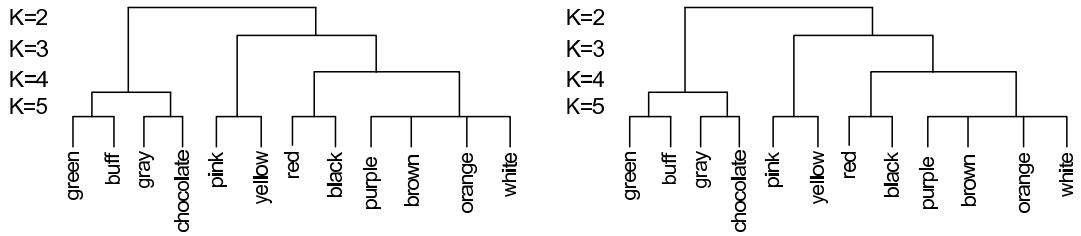
(a) Applying Fisher's Algorithm (b) Applying k-means Algorithm

Figure 4.30: Taxonomies of attribute *Spore_print_color* by using partitional algorithms



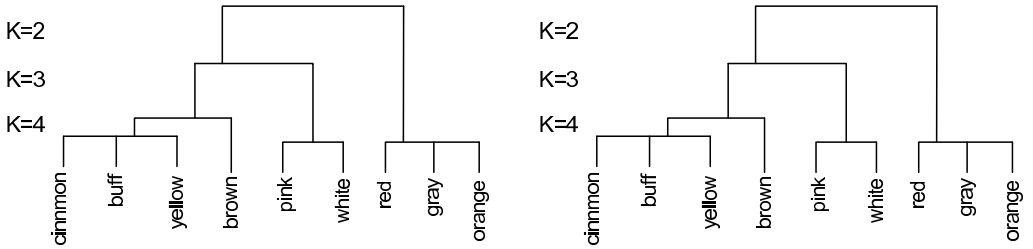
(a) Applying Fisher's Algorithm (b) Applying k-means Algorithm

Figure 4.31: Taxonomies of attribute *Stalk_color_above_ring* by using partitional algorithms



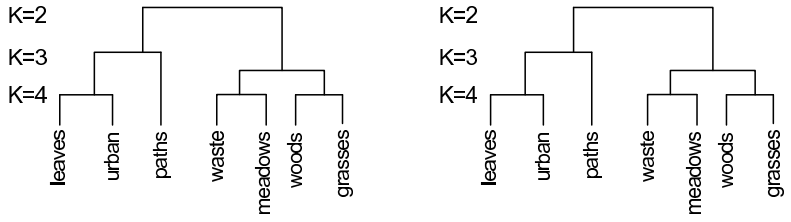
(a) Applying Fisher's Algorithm (b) Applying k-means Algorithm

Figure 4.32: Taxonomies of attribute *Gill_color* by using partitional algorithms



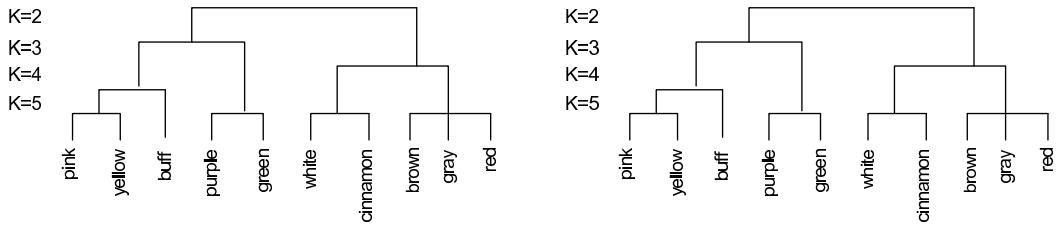
(a) Applying Fisher's Algorithm (b) Applying k-means Algorithm

Figure 4.33: Taxonomies of attribute *Stalk_color_below_ring* by using partitional algorithms



(a) Applying Fisher's Algorithm (b) Applying k-means Algorithm

Figure 4.34: Taxonomies of attribute *Habitat* by using partitional algorithms



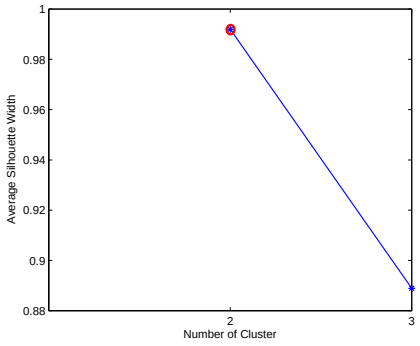
(a) Applying Fisher's Algorithm (b) Applying k-means Algorithm

Figure 4.35: Taxonomies of attribute *Cap_color* by using partitional algorithms

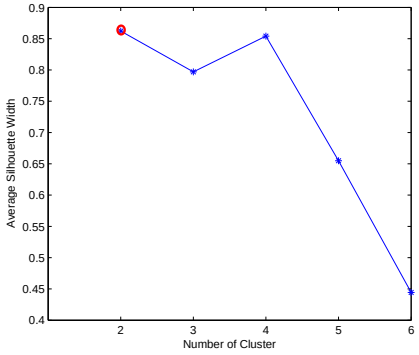
For each nominal attribute, the average silhouette widths are also calculated for k up to the number of attribute values, see figure 4.36. The optimal number of clusters are marked by a red circle in each plot. In this case, given the optimal number k, both k-means and Fisher's algorithm generate the same optimal clusters, see table 4.6 for details. These optimal number of clusters are also assigned to k appearing at the right level in the taxonomies.

Table 4.6: The optimal clusters for the nominal attribute values of the *Mushroom* data set

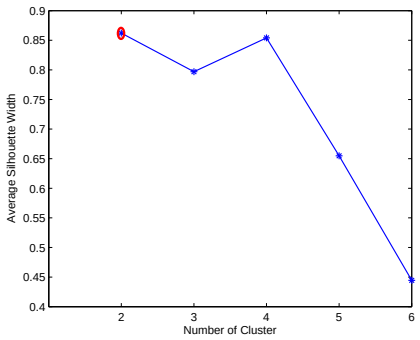
Attribute & Cluster No.	**Optimal Clusters**
Odor (k = 2)	{anise, almond, none}, {creosote, fishy, pungent, musty, foul, spicy}
Spore-print-color (k = 2)	{chocolate, green, white}, {brown, black, orange, yellow, buff, purple}
Stalk-color-above-ring (k = 2)	{red, orange, gray}, {cinnamon, buff, yellow, brown, pink, white}
Gill-color (k = 2)	{chocolate, green, buff, gray}, {red, pink, purple, orange, brown, yellow, white}
Stalk-color-below-ring (k = 2)	{cinnamon, buff, yellow, brown} {red, orange, pink, white, gray}
Habitat (k = 3)	{leaves, urban}, {paths}, {waste, meadows, woods, grasses}
Cap-color (k = 4)	{pink, yellow, buff}, {purple, green}, {cinnamon, white}, {red, brown, gray}



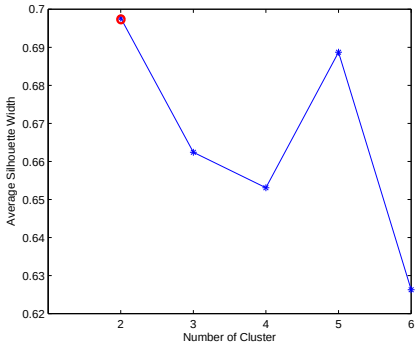
(a) Odor



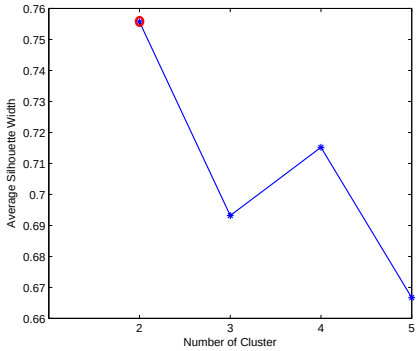
(b) Spore-print-color



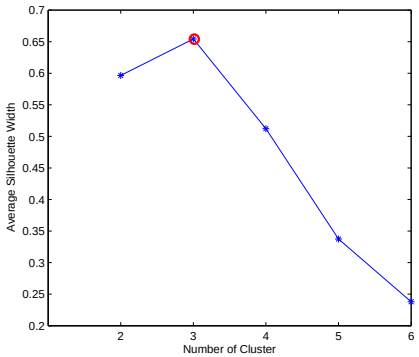
(c) Stalk-color-above-ring



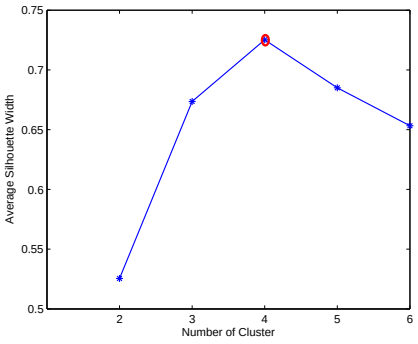
(d) Gill-color



(e) Stalk-color-below-ring



(f) Habitat



(g) Cap-color

Figure 4.36: Silhouette plot for the seven nominal attributes of the *Mushroom* data set

4.5.3 Discussion

In comparison with the *Adult* data set, all of the attributes of the *Mushroom* data set are natural attributes. The attribute Odor is related to the chemical material in mushroom. In general, most poisonous chemical materials emit an abnormal smell compared with non-poisonous materials, thus the taxonomy of Odor we obtained in this section is reasonable.

Other attributes, particularly color attributes are not reliable identifiers for poisonous mushrooms, therefore the generated taxonomies do not show such clear semantic relationships.

In addition, comparing with the taxonomies generated using different methods, we find that there are few differences between them. For the *Mushroom* data set, we will directly apply the taxonomies generated using the Fisher's algorithm to the work in the later chapters.

4.6 Summary

In this chapter, we firstly utilise two typical hierarchical clustering algorithms, AGNES and DIANA, to perform the automatic generation of attribute value taxonomies for some selected attributes of the *Adult* and the *Mushroom* data set.

Secondly, as the hierarchical algorithms prefer to generating a binary trees, we explore to use of partitional algorithm for AVT construction. An iterative approach is proposed to gradually construct the AVT under some supervision. Here supervision means setting the number of clusters before clustering. Two partitional algorithms, k-means and Fisher's, are selected to evaluate the performance. We also present a theorem about optimal clustering and provide the proof for it.

A related issue, the selection of the optimal number of clusters by using silhouette width for partitional algorithms, is also introduced and implemented for both

nominal and numeric attribute values. The obtained optimal numbers are then assigned to k appearing at the right level in the taxonomies. The two partitional algorithms have the same performance for a given optimal number of clusters.

All the generated taxonomies are compared, from which we revealed some statistical characteristic of the data. Although these taxonomies are not completely compatible with the semantic ones, they provide a good guide on constructing appropriate concept taxonomies for this particular data or for making adjustments to taxonomies extracted from existing ontologies. Such modification is likely to be required if a general taxonomy is to be used for a specific database.

Finally, we propose a concept taxonomy for each selected attribute of the *Adult* data set after adjusting and adding the interpretation for the internal nodes with semantic meaning. Whilst, for the *Mushroom* data set, we directly adopt the generated taxonomies using the Fisher's algorithm without a further adaptation.

Although the AVTs generated using clustering algorithms are not perfect in comparison with completely manual annotations, as a reference, they can be used for assisting users, analysts or specialists to figure out a better organization so as to avoid a user's subjectivity. The taxonomies generated in this chapter will be used either directly or indirectly (need some refinement) in the later chapters for performance evaluation by some data mining techniques.

Chapter 5

Exploiting AVTs in Tree Induction

Decision tree induction is a simple yet successful and widely used technique for classification and prediction in data mining. In this chapter, attribute-value taxonomies are exploited to generate improved decision trees in terms of less number of tree nodes. Prior to running a tree induction algorithm, the database is enhanced by the inclusion of new data constructed from the taxonomies. A procedure for constructing these data is described and methods for limiting potentially exponential explosion in the data are discussed. Experiments are carried out using the *Adult* and the *Mushroom* data set and some tree induction algorithms.

5.1 Tree Induction Algorithms

Tree induction algorithms mainly differ in the choice of different splitting measures to find the "best" test on input attributes and different pruning methods to overcome the overfitting. There are mainly three families of tree growing algorithms:

- The **CLS** (Concept Learning System) family: ID3 [118], C4.5 [119], and its more recent successor C5.0, etc.
- The **CART** (Classification and Regression Trees) family: CART [23], and its other versions in statistical packages, e.g. IndCART and S-Plus CART, etc.
- The **AID** (Automatic Interaction Detection) family: THAID [102], CHAID [78, 79], XAID [65], etc.

The algorithms in the CLS family were developed by the machine learning community, using information theory. The CART family is more oriented to statistics using the concept of impurity (e.g. GINI index), while the AID family grew out of the social sciences and uses statistical significance tests, e.g. χ^2 test, to find the best split.

In this chapter, the algorithms from the CLS family will be chosen to build decision tree for the prediction task. More details about CLS are present as follows.

C4.5 and C5.0

When building a decision tree, both C4.5 and C5.0 use the same splitting measure as ID3, ***information gain***, to seek a locally best test on an attribute. However, information gain tends to favour the attribute with the large number of unique values. We may observe that $Gain_{Info}$ defined in equation 2.1 is maximised when each subset partitioned by a test contains a single instance, but a test that results in a large number of outcomes may not be desirable since the small number of records associated with each partition is unlikely to lead to reliable predictions. So another measure, called ***gain ratio***, is exploited by C4.5 to overcome this problem. **Gain ratio** modifies the splitting criterion by taking into account the number of

outcomes produced by the test, and is defined by

$$Gain_{Ratio}(S,T) = \frac{Gain_{Info}(S,T)}{-\sum_{j=1}^{k} \frac{|S'_j|}{|S|} \log_2 \frac{|S'_j|}{|S|}} \ , \tag{5.1}$$

where, the denominator part represents the split information, which increases with the number of splits, which in turn reduces the gain ratio.

Furthermore, the ***windowing*** technique is still available as an option in C4.5 and C5.0 when dealing with very large data sets. With windowing, a provisional decision tree is first built from a randomly selected subset of the training set (called a *window*). The tree is then used to classify the remaining data outside of the window, usually with the result that some data are misclassified. These misclassified cases are then added to the window, and the tree construction process is repeated, until the updated tree classifies all the training data outside the window correctly.

C4.5 and C5.0 also introduce some other enhancements to ID3, see below for details.

- Handling training data with missing or unknown attribute values –

  (1) When doing the test evaluation, C4.5 modifies the splitting criteria by taking account the unknown rates of a given attribute, so equations 2.1 and 5.1 are modified to

  $$Gain_{Info}(S,T) = \frac{|S - S_0|}{|S|} \ Gain_{Info}(S - S_0, T), \tag{5.2}$$

  and

  $$Gain_{Ratio}(S,T) = \frac{Gain_{Info}(S,T)}{-\frac{|S_0|}{|S|} \log_2 \frac{|S_0|}{|S|} - \sum_{j=1}^{k} \frac{|S'_j|}{|S|} \log_2 \frac{|S'_j|}{|S|}} \ , \tag{5.3}$$

  where S_0 denotes the subset of cases in S whose values of the given attribute are unknown, and S'_j denote those subsets of S partitioned by test T on the

attribute with known outcomes;

(2) when partitioning the training set by a chosen test on an attribute, each case with missing value is divided into fragments and added to each subset corresponding to known test outcomes, with a probability proportional to the number of cases in the relative subset, i.e. with weight $|S'_j|/|S - S_0|$;

(3) when classifying a case with unknown value of a tested attribute, C4.5 provides a class probability distribution instead of a single class, determined by exploring all the outcomes and combining them with a weighted sum of the probabilities traversing down the tree model for each possible classification result, then the class with the highest probability is chosen as the predicted class for the test case.

- Handling continuous attributes – ID3 is restricted to deal with attributes with discrete or symbolic values, while C4.5 creates a threshold for each continuous attribute, and then splits the training set into those whose attribute value is above the threshold and those that are less than or equal to it [121].

- Pruning trees after creation – C4.5 uses the error-based pruning (EBP) technique [60] to prune the tree in a *bottom-up* fashion. Tree pruning is done by replacing a subtree either with one of its leaf nodes labelled with the majority class or by one of its subtrees with the most frequent outcome of the test (not necessarily the one with the most nodes). This replacement is performed when the root of the subtree has higher estimated error rate than its leaf nodes or its largest branch. The pruning terminates when no further improvement is observed.

- C5.0 supports boosting [120] – To improve the predictive accuracy of the decision tree, a sequence of tree classifiers are built and combined by using a boosting technique, so that more instances can be correctly classified.

Boosting maintains a weight for each instance in the training set (identical weights are assigned initially), updates them at the end of each round, and so leads to different classifiers. Finally, the multiple classifiers are combined by a weighted voting procedure to give improved prediction.

5.2 Applying AVT to Decision Tree Classifier Learning

5.2.1 Problem Specification

An important goal of machine learning is to discover comprehensible, yet accurate and robust classifiers [109]. The induction of the decision tree is a supervised classification process in which prior knowledge regarding classes in the database is used to guide the discovery. To generate a simple yet useful and easily interpretive tree is always challenging for data miners, data analyst, and data scientists. Helping to overcome this problem is the main objective of this chapter.

There is a considerable amount of research on simplicity in decision tree learning. Simply, the *tree size* is the measure most relevant to the simplicity of the tree. Inducing small decision trees is known to be NP-hard [69] and most learning algorithms perform heuristic pruning techniques to obtain small trees. Many DM software packages, e.g., C5.0, IBM Intelligent Miner, provide facilities that make the generation of decision trees a relatively easy task. The data mining analyst can determine how accurate or simple the generated decision tree is, by using various parameters, e.g. *Minimum Number of Instances per Leaf*, *Pruning Severity*, *Maximum Number of Branches from a Node*, *Maximum Depth of Tree*, etc.

Apart from using the pruning techniques, we also seek other ways for tree simplification. Motivated by the successful applications of mining multiple level

association rules by using a concept hierarchy, we aim to explore a methodology of utilising the attribute-value taxonomies for decision tree building. Since the AVTs provide an opportunity for attribute abstraction, we can expect that a compact and easily understanding decision tree will be finally generated after integrating the AVTs with the decision tree learning process.

Definition 5.1 (CUT) A *cut*, C, for a tree structured taxonomy, T, with a set of nodes, N, is a nonsingleton set of nodes, $C \subset N$, s.t.

1. every leaf in the tree is either in C or has an ancestor in C.
2. if $n \in C$, then either n is a leaf or no node in the subtree rooted at n is in C.

Thus every path, comprising a set of ordered nodes from the root to a leaf, can only contain a single node in C.

Let $C(n)$ be the number of cuts for the tree rooted at n ($n \in N$), then

$$C(n) = \begin{cases} C(n_1) \times C(n_2) \ldots \times C(n_k) & \text{if } n \text{ is the root and has children } n_1, n_2, \ldots, n_k, \\ C(n_1) \times C(n_2) \ldots \times C(n_k) + 1 & \text{if } n \text{ is not the root and has children } n_1, \ldots, n_k, \\ 1 & \text{if } n \text{ is a leaf.} \end{cases}$$

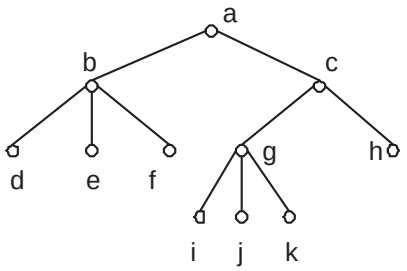


Figure 5.1: An Example of Taxonomy

For the tree in figure 5.1, we have $C(b) = 2,\ C(g) = 2,\ C(c) = 3$, and $C(a) = 6$. So the total number of cuts for the tree, rooted at a, is 6. All the six valid cuts

Table 5.1: Valid Cuts for Figure 5.1 Taxonomy

No	Cuts
1	$\{b, c\}$
2	$\{d, e, f, c\}$
3	$\{b, g, h\}$
4	$\{d, e, f, g, h\}$
5	$\{b, i, j, k, h\}$
6	{d, e, f, i, j, k, h}

are listed in table 5.1.

The *cut* comprising all the leaf nodes of a taxonomy corresponds to a column of original data in the database. We call this column of data, the *primitive* column for that attribute. In this case, the sixth cut is the primitive.

5.2.2 Methodology Description

In this section, we present the methodology for exploiting attribute-value taxonomies to generate simpler decision tree. The methodology contains four steps described as follows.

1. Given an AVT, all the possible attribute-value tuples, named as *cuts*, as defined in definition 5.1, can be searched and extracted, which comprise disjoint nodes at various levels of the taxonomy.

2. For each valid *cut*, its *Gain Ratio* (GR) value is calculated, and then all the *cuts* are ranked according to their GR. Each valid *cut* will be treated as a new variant of the original attribute. In this way, we create a multiple level attribute-value representation space.

3. Select a number of top ranked *cuts* for each nominal attribute with AVT to form the corresponding new expansions. New columns of data will be derived according to the following rule:

Given any non-primitive cut, C, of a taxonomy, for each node, $n \in C$, all its subtree nodes rooted at n are replaced by n.

To avoid the potential exponential computation in the tree induction, only the top few (say 5 for example) *cuts* will be chosen as new variants of the attribute instead of all the valid *cuts*. The attribute associated with the new column is the same as that associated with the primitive column, albeit with a suitable index. Though the use of these indices is necessary for use within the tree induction algorithm, they can be ignored when the tree is finally delivered.

4. Apply the decision tree algorithm to the updated data to generate a relatively simple tree. It is also desirable that this tree can still maintain reasonable accuracy.

The use of *Gain Ratio* to rank and select the *cuts* of the AVT, as described in step 2 and 3, will be tested in the following two case studies. This is because for a data set with a large attribute-value space and a large size, this kind of selection can save the memory occupation required by the algorithm, but also greatly reduce the size of the data file expanded with the new columns which is used by the tree induction algorithm.

5.3 Performance Evaluation

We conducted two case studies to demonstrate the impact on the generated decision tree by using attribute-value taxonomies for nominal attributes. Both of these two case studies will follow the steps described in the previous section.

5.3.1 Case Study 1 – the *Adult* Data Set

In this case study, we will use the four constructed attribute-value taxonomies of the *Adult* data set, as shown in figures 4.17 to 4.21.

In table 5.2, we ranked the four attributes according to their computed *information gain* (IG) values. A high IG value may indicate the priority of being selected as a test when building the decision tree.

Table 5.2: *Information Gain* of the four Nominal Attributes of the *Adult* Data Set

#Rank	Name	Information Gain
1	Marital-status	0.15747082
2	Education	0.09339399
3	Occupation	0.09319446
4	Workclass	0.01710448

Table 5.3 lists all the nodes' name of the four selected AVTs along with a code assigned for each node. Ideally, the name of the internal nodes of the AVT can hold some semantic meanings, which will make the generated decision tree moreeasy to understand.

Then all the possible cuts over the taxonomies of each attribute will be extracted. According to the formula of calculating the total number of *cuts* as defined in Definition 5.1, there are totally 5, 27, 8, and 5 cuts that can be extracted from the Marital-status, Education, Occupation and Workclass taxonomy, respectively.

In order to reduce unnecessary time-consuming computation, we will only select top five ranked cuts by computing the gain ratio (GR) value of each cut over the training data. Table 5.4 lists all top five ranked cuts of the four attributes along with their GR values. In fact, the top five ranked cuts generally contain almost all important conceptual nodes of the taxonomies, and these listed conceptual nodes will be treated as new attribute values.

Table 5.3: Coded Nodes of Four Taxonomies of the *Adult* Data Set

Attribute	Code of Node	Node of the Taxonomy
Marital-status	m1	Never−married
	m2	Separated
	m3	Married−spouse−absent
	m4	Divorced
	m5	Widowed
	m6	Married−civ−spouse
	m7	Married−AF−spouse
	m2−m3−m4−m5	Partner−absent
	m6−m7	Partner−present
	m2−m3−m4−m5−m6−m7	Married
Education	e1	Preschool
	e2	1st−4th
	e3	5th−6th
	e4	7th−8th
	e5	9th
	e6	11th
	e7	10th
	e8	12th
	e9	HS−grad
	e10	Some−college
	e11	Assoc−acdm
	e12	Assoc−voc
	e13	Bachelors
	e14	Prof−school
	e15	Masters
	e16	Doctorate
	e2−e3−e4−e5−e6−e7−e8	Primary
	e11−e12	Associate
	e13−e14	Under−grad
	e15−e16	Post−grad
	e2−e3−e4−e5−e6−e7−e8−e9	Primary&Secondary
	e10−e11−e12−e13−e14−e15−e16	Post−Secondary
Occupation	o1	Craft−repair
	o2	Transport−moving
	o3	Machine−op−inspct
	o4	Handlers−cleaners
	o5	Priv−house−serv
	o6	Tech−support
	o7	Sales
	o8	Adm−clerical
	o9	Exec−managerial
	o10	Prof−specialty
	o11	Protective−serv
	o12	Farming−fishing
	o13	Armed−Forces
	o14	Other−service
	o1−o2−o3−o4−o5	Blue−collar
	o6−o7−o8−o9−o10	White−collar
	o11−o12−o13−o14	Other
Workclass	w1	Without−pay
	w2	Private
	w3	Self−emp−not−inc
	w4	Self−emp−inc
	w5	Local−gov
	w6	State−gov
	w7	Federal−gov
	w3−w4	Self−Employ
	w5−w6−w7	Gov−Employ
	w2−w3−w4−w5−w6−w7	With−pay

Table 5.4: Ranked Top 5 cuts of Four Taxonomies (*Adult*)

Attribute	**Ranked cuts**		**GainRatio**
Marital−status	M1	m1 m2−m3−m4−m5 m6−m7	0.103877
	M2	m1 m2−m3−m4−m5 m6 m7	0.103362
	M3	m1 m2−m3−m4−m5−m6−m7	0.099352
	M4	m1 m2 m3 m4 m5 m6−m7	0.086894
	M5	m1 m2 m3 m4 m5 m6 m7	0.086535
Education	E1	e1 e2−e3−e4−e5−e6−e7−e8−e9 e10−e11−e12−e13−e14−e15−e16	0.042411
	E2	e1 e2−e3−e4−e5−e6−e7−e8−e9 e10 e11−e12 e13 e14 e15−e16	0.040801
	E3	e1 e2−e3−e4−e5−e6−e7−e8−e9 e10 e11−e12 e13 e14 e15 e16	0.040415
	E4	e1 e2−e3−e4−e5−e6−e7−e8−e9 e10 e11−e12 e13−e14 e15−e16	0.039950
	E5	e1 e2−e3−e4−e5−e6−e7−e8−e9 e10 e11−e12 e13−e14 e15 e16	0.039567
Occupation	O1	o1−o2−o3−o4−o5 o6−o7−o8−o9−o10 o11 o12 o13 o14	0.031086
	O2	o1−o2−o3−o4−o5 o6 o7 o8 o9 o10 o11 o12 o13 o14	0.030483
	O3	o1−o2−o3−o4−o5 o6 o7 o8 o9 o10 o11−o12−o13−o14	0.029177
	O4	o1−o2−o3−o4−o5 o6−o7−o8−o9−o10 o11−o12−o13−o14	0.028781
	O5	o1 o2 o3 o4 o5 o6 o7 o8 o9 o10 o11 o12 o13 o14	0.027438
Workclass	W1	w1 w2−w3−w4−w5−w6−w7	0.033005
	W2	w1 w2 w3 w4 w5−w6−w7	0.013538
	W3	w1 w2 w3 w4 w5 w6 w7	0.012118
	W4	w1 w2 w3−w4 w5−w6−w7	0.009664
	W5	w1 w2 w3−w4 w5 w6 w7	0.008767

Data Preprocessing:

- The *Adult* data set contains two types of attribute, nominal and numeric. For some numeric attributes, such as Capital-gain and Capital-loss, they can be quantised using a simple discretisation with values "*Low*", "*Medium*", "*High*" and "*Very High*" as described in chapter 4.

- According to the information gain value, the attribute, Native-country, is least important for decision tree construction. Moreover, 91.2% of people are Americans, thus it is quantised as "*United-States*" and "*non-US*".

- All twenty cuts listed in table 5.4 will be used as the expansions of the four original nominal attributes. This is also a way of assigning weight to some internal conceptual nodes of the taxonomy, since some nodes occur more than once in the five *cuts* of the same attribute. For example, the node "e2−e3−e4−e5−e6−e7−e8−e9" or value "*Primary & Secondary*" occurs in all five cuts.

Experiments:

We apply both C4.5 and C5 to the training data with different preprocessing, and conduct a series of experiments to compare performances using and without using AVTs. Details are given below.

1. Filtering out the Capital-gain and Capital-loss attributes, and marking the data as "Original – C";

2. Expanding the data with all the possible cuts and marking the data as "Original + All";

3. Expanding the data with the top five ranked cuts and marking the data as "Original + Top 5";

4. Discretising the three attributes, Capital-gain, Capital-loss and Native-country, with the values mentioned in 4.4.3 and 5.3.1 and marking the data as "Discretised".

All the experimental results on the *Adult* data set are shown in tables 5.5 to 5.8.

Table 5.5: Comparison of classification accuracy and depth & size of decision tree generated by C4.5

Test Data	Accuracy	Tree Depth	# Tree Nodes
Original Data	81.04%	4	21
Original + All/Top 5	81.04%	3	9
Original – C	81.82%	3	27
Original – C + All/Top 5	81.02%	3	8
Discretised Data	80.73%	4	24
Discretised + All/Top 5	79.38%	3	13

Table 5.6: Comparison of classification accuracy and depth & size of a simple decision tree generated by C5.0

Test Data	Accuracy	Tree Depth	# Tree Nodes
Original Data	85.55%	11	90
Original + All/Top 5	85.23%	9	70
Original – C	81.51%	2	29
Original – C + All/Top 5	82.22%	3	9
Discretised Data	83.92%	5	61
Discretised + All	84.03%	6	46
Discretised + Top5	84.04%	5	34

Discussion:

Our first set of experiments, shown in table 5.5 and 5.6, start with inspecting the impact of the two attributes, Capital-gain and Capital-loss, on the effectiveness of classification. After applying the C4.5 and C5.0 algorithms to build a simple

decision tree with the original data, we notice that these two attributes play an important role in the tree growth phase. However, as a chosen *decision node*, the tests on these two continuous attributes provide a large range, e.g. "*Capital-gain* $>$ 4865", for the further splitting; it is not very helpful to the end users if this kind of test appears too frequently in the generated tree. In this case, we consider either filtering out these two attributes or quantising them. The second set of experiments, shown in the middle of tables 5.5 and 5.6, presents the different performance of classification. It seems more reasonable to keep and discretise these two attributes. The corresponding experimental results are listed in both tables.

Some other experiments are conducted on the data with the AVT-based new expansions. Not only the top five ranked cuts, but also all the valid cuts extracted from each selected AVT, are used for the attribute expansion. The experimental results in both table 5.5 and 5.6 show that both cases generate similar performance. This is also proved that the decision of choosing top five ranked cuts is reasonable and effective.

Table 5.7: Comparison of classification accuracy and depth & size of an expert decision tree generated by C5.0 without Boosting but with 75% Pruning Severity and 20 Minimum Records/child branch

Data Set	**Accuracy(%)** Train / Test	**Tree Depth**	**# Tree Nodes**
Original Data	86.65 / 86.13	16	162
Original + All	86.87 / 86.14	17	160
Original + Top 5	**86.79 / 86.18**	**16**	**144**
Discretised Data	84.64 / 84.48	6	116
Discretised + All	84.94 / 84.64	10	100
Discretised + Top 5	**84.78 / 84.44**	**9**	**78**

Table 5.8: Comparison of classification accuracy and depth & size of an expert decision tree generated by C5.0 with Boosting and 75% Pruning Severity and 20 Minimum Records/child branch

Data Set	**Accuracy(%)** Train / Test	**Tree Depth**	**# Tree Nodes**
Original Data	86.70 / 86.18	25	162
Original + All	87.19 / 86.57	30	160
Original + Top 5	**87.32 / 86.29**	**26**	**144**
Discretised Data	84.01 / 84.10	7	116
Discretised + All	85.13 / 84.82	11	100
Discretised + Top 5	**84.90 / 84.58**	**10**	**78**

Tables 5.7 and 5.8 show the experimental results of the expert tree construction using C5.0. There are some parameters available in C5.0 package to allow us to produce various trees with different complexity.

From these two tables, we find the use of the top five ranked cuts obtained from the AVTs can generate a simpler decision tree by reducing the number of tree nodes by more than 10%. Simultaneously, the classification accuracy is kept stable. This is because the top five ranked cuts can be expected to contain more reasonable combinations of nodes when compared to those found in other cuts of the attribute-value taxonomies.

Conclusion:

1. Data preprocessing is an important way to reduce some possible redundant information by removing some attributes or doing the discretisation on numeric attributes. In addition, the selection of the top five ranked cuts from the AVTs by using *Gain Ratio* value also appears to be efficient and effective. From our experimental results, these data processing steps can effectively simplify the decision tree by reducing the number of tree nodes by more than 10% without undue sacrifice of classification accuracy.

2. Even if sometimes we might have to sacrifice the accuracy slightly, it is still

arguably worthwhile since a much simpler tree and more easily interpreted tree can be generated.

5.3.2 Case Study 2 – the *Mushroom* Data Set

In this section, we will conduct the same experiments on the *Mushroom* data set as we did on the *Adult* data set.

As shown in figures 4.29(a) to 4.35(a), the AVTs of the seven nominal attributes, Odor, Spore-print-color, Stalk-color-above-ring, Gill-color, Habitat, Stalk-color-below-ring, and Cap-color, will be used in our experiments.

- Table 5.9 lists the ranked attributes along with their *information gain* values.
- Table 5.12 lists all of the seven selected attributes together with their possible values and variants, as well as assigned code.
- All the possible *cuts* of the seven attributes are extracted, based on the attribute value taxonomies generated in chapter 4. Table 5.13 lists the top five ranked cuts of each taxonomy along with their *gain ratio* values.

Table 5.9: *Information Gain* of the seven Nominal Attributes of the *Mushroom* data set

#Rank	Name	Information Gain
1	Odor	0.89728426
2	Spore-print-color	0.46200583
3	Stalk-color-above-ring	0.37871288
4	Gill-color	0.26576852
5	Stalk-color-below-ring	0.24580019
6	Habitat	0.13583585
7	Cap-color	0.02980838

Then we use the algorithm C5.0 to generate decision trees on both original data and the new updated data after replacing the seven selected nominal attributes

Table 5.10: Comparison of classification accuracy and depth & size of an expert decision tree generated by C5.0 without Boosting but with 85% Pruning Severity and 5 Minimum Records/child branch

Data Set	**Accuracy** Train / Test	**Tree Depth**	**# Tree Nodes**
Original Data	98.73% / 100%	4	11
Original + Top 5	**98.56% / 100%**	**1**	**2**

Table 5.11: Comparison of classification accuracy and depth & size of an expert decision tree generated by C5.0 with Boosting and 75% Pruning Severity and 10 Minimum Records/child branch

Data Set	**Accuracy** Train / Test	**Tree Depth**	**# Tree Nodes**
Original Data	98.89% / 100%	4	9
Original + Top 5	**98.42% / 100%**	**1**	**2**

with the 35 new columns derived from the top five ranked cuts of the AVTs listed in table 5.13. The experimental results are shown in table 5.10 and 5.11.

It is clear that the number of the tree nodes are greatly decreased by more than 70% after applying AVTs on the data with only a slight sacrifice of the classification accuracy.

Table 5.12: Coded Nodes of Seven Taxonomies of the *Mushroom* data set

Attribute	Code of Node	Node of the Taxonomy
Odor (A5)	A51	anise
	A52	creosote
	A53	fishy
	A54	almond
	A55	pungent
	A56	musty
	A57	foul
	A58	spicy
	A59	none
	A51-A54	good smell
	A51-A54-A59	normal smell
	A52-A53-A55-A56-A57-A58	abnormal smell
Spore-print -color (A20)	A201	white
	A202	green
	A203	orange
	A204	yellow
	A205	buff
	A206	brown
	A207	black
	A208	purple
	A209	chocolate
	A202-A209	dark green
	A206-A207	dark brown
	A203-A204-A205-A208	bright colour
Stalk-color -above-ring (A14)	A141	cinnamon
	A142	red
	A143	buff
	A144	gray
	A145	white
	A146	orange
	A147	brown
	A148	pink
	A149	yellow
	A141-A143-A149	bright yellow
	A141-A143-A147-A149	bright+dark yellow
	A142-A144-A146	red+gray
Gill-color (A9)	A91	purple
	A92	brown
	A93	chocolate
	A94	green
	A95	red
	A96	yellow
	A97	pink
	A98	black
	A99	buff
	A910	gray
	A911	orange

Continued on next page

Table 5.12 – *Continued from previous page*

Attribute	Code of Node	Node of the Taxonomy
	A912	white
	A91-A92	purple+brown
	A94-A99	green+buff
	A95-A911	red+orange
	A98-A912	black+white
Stalk-color -below-ring (A15)	A151	cinnation
	A152	buff
	A153	pink
	A154	yellow
	A155	orange
	A156	brown
	A157	white
	A158	gray
	A159	red
	A151-A152-A154	bright yellow
	A151-A152-A154-A156	bright+dark yellow
	A155-A158-A159	red+gray
Habitat (A22)	A221	leaves
	A222	urban
	A223	waste
	A224	woods
	A225	grasses
	A226	meadows
	A227	paths
	A224-A225	woods+grasses
	A221-A222	leaves+urban
	A223-A226	waste+meadows
Cap-color (A3)	A31	purple
	A32	pink
	A33	yellow
	A34	brown
	A35	buff
	A36	white
	A37	cinnamon
	A38	red
	A39	green
	A310	gray
	A31-A39	purple+green
	A32-A33-A35	pink+yellow+buff
	A34-A38-A310	brown+red+gray
	A36-A37	white+cinnamon
	A34-A36-A37-A38-A310	white+cinnamo+brown+red+gray

Table 5.13: Ranked Top 5 Cuts of Seven Taxonomies (*Mushroom*)

Attribute		Ranked cuts	GainRatio
Odor	O1	A51-A54-A59 A52-A53-A55-A56-A57-A58	0.899592
	O2	A51-A54 A52-A53-A55-A56-A57-A58 A59	0.649771
	O3	A51 A52-A53-A55-A56-A57-A58 A54 A59	0.603608
	O4	A51-A54-A59 A52 A53 A55 A56 A57 A58	0.496059
	O5	A51-A54 A52 A53 A55 A56 A57 A58 A59	0.410120
Spore-print -color	P1	A201 A202-A209 A203-A204-A205-A208 A206-A207	0.283036
	P2	A201 A202-A209 A203 A204 A205 A206-A207 A208	0.274511
	P3	A201 A202 A203-A204-A205-A208 A206-A207 A209	0.273754
	P4	A201 A202 A203 A204 A205 A206-A207 A208 A209	0.265778
	P5	A201 A202-A209 A203-A204-A205-A208 A206 A207	0.218228
Stalk-color -above-ring	S1	A141-A143-A147-A149 A142-A144-A146 A145 A148	0.157661
	S2	A141-A143-A149 A142-A144-A146 A145 A147 A148	0.148087
	S3	A141-A143-A147-A149 A142 A144 A145 A146 A148	0.145732
	S4	A141 A142-A144-A146 A143 A145 A147 A148 A149	0.145057
	S5	A141-A143-A149 A142 A144 A145 A146 A147 A148	0.137598
Gill-color	G1	A91-A92 A93 A94-A99 A95-A911 A96 A97 A98-A912 A910	0.141402
	G2	A91-A92 A93 A94-A99 A95 A96 A97 A98-A912 A910 A911	0.140327
	G3	A91-A92 A93 A94 A95-A911 A96 A97 A98-A912 A99 A910	0.140203
	G4	A91-A92 A93 A94 A95 A96 A97 A98-A912 A99 A910 A911	0.139147
	G5	A91-A92 A93 A94-A99 A95-A911 A96 A97 A98 A910 A912	0.133152
Stalk-color -below-ring	B1	A151-A152-A154-A156 A153 A155-A158-A159 A157	0.137669
	B2	A151-A152-A154 A153 A155-A158-A159 A156 A157	0.133830
	B3	A151 A152 A153 A154 A155-A158-A159 A156 A157	0.130415
	B4	A151-A152-A154-A156 A153 A155 A157 A158 A159	0.127413
	B5	A151-A152-A154 A153 A155 A156 A157 A158 A159	0.124553
Habitat	H1	A221-A222 A223-A226 A224-A225 A227	0.093162
	H2	A221-A222 A223 A224-A225 A226 A227	0.091744
	H3	A221 A222 A223-A226 A224-A225 A227	0.086098
	H4	A221 A222 A223 A224-A225 A226 A227	0.085064
	H5	A221-A222 A223-A226 A224 A225 A227	0.064569
Cap-color	C1	A31-A39 A32-A33-A35 A34-A36-A37-A38-A310	0.031481
	C2	A31 A32-A33-A35 A34-A36-A37-A38-A310 A39	0.031298
	C3	A31-A39 A32 A33 A34-A36-A37-A38-A310 A35	0.025732
	C4	A31 A32 A33 A34-A36-A37-A38-A310 A35 A39	0.025612
	C5	A31-A39 A32-A33-A35 A34-A38-A310 A36-A37	0.022215

5.4 Summary

This chapter presents a methodology for exploiting attribute-value taxonomies to generate simple but easily understood decision tree with reasonable classification accuracy. Two real world data sets with a good number of nominal attributes are chosen to illustrate our methodology.

Once the AVT is constructed, no matter whether it is manually defined or automatically generated, the first thing is to search and extract all the valid cuts according to our definition.

We have observed that as AVTs increase in complexity, there is an exponential rise in the number of cuts and hence of potentially new columns of attributes. This growth will ultimately damage the performance of any tree induction algorithm. We have discussed how a simple measure such as *gain ratio* might be used to limit the number of new attributes created, but further research is required to determine the true effectiveness of this approach.

With the two case studies, we have shown how AVTs can be used to simplify decision trees quite dramatically without significant loss of accuracy.

Chapter 6

Exploiting AVTs in Rule Induction

In this chapter, we aim to explore how the use of attribute-value taxonomies (AVTs) can improve the classification performance when using rule induction. In our work, we first explore a rule based classifier using receiver operating characteristic (ROC) analysis; then we develop a new approach to integrate the search of the optimal cuts of the AVTs with rule learning into one framework. Our work is tested on the *Adult* and the *Mushroom* data sets.

6.1 Overview of the ROC Analysis for Rule Learning

The most popular strategy for learning classification rules is the covering or separate-and-conquer strategy. It has its roots in the early days of machine learning in the AQ family of rule learning algorithms [80]. It is popular in propositional rule

learning (CN2 [30, 31], Ripper [32], or CBA [90]).

For a two-class classifier, the examples in the training set are often labelled as positive and negative with respect to a specific requirement. Under this condition, learning a set of rules with the rule learning algorithm aims to cover the positive examples as much as possible whilst covering the negative examples as little as possible.

Table 6.1 shows a two-by-two confusion matrix, which can represent the dispositions of a set of instances given a classifier. In Table 6.1, we use capital letters to denote the total number of positive (P) and Negative (N) objects in the training set. p denotes the number of true positives and n the number of false positives covered by a rule. Here, the percentage p/P of correctly classified positive objects and the percentage n/N of incorrectly classified negative are respectively denoted by TPR and FPR.

Receiver Operating Characteristic (ROC) analysis has been introduced in machine learning as a powerful tool for evaluation of classifiers [116]. ROC analysis has several properties that could be explored in machine learning. For threshold-based classifiers, it is possible to spread the classification criterion over all possible trade-offs of hits (true positive) and error (false positive) rates. If we describe the ROC analysis in a two-dimensional space where the axes represent the false and true positive rates, the result is an ascending curve, rising from (0,0), where all the instances are classified as negative, to (1,1), where all the instances are classified as positive. The more sharply the curve rises, the better is the classifier. The area under the ROC curve (AUC) represents the probability that a randomly chosen positive example will be rated higher than a negative instance [115].

In this chapter, the ROC analysis provides a graphical plot of the fraction of true positives vs. the fraction of false positives for a binary classifier system in a two-dimensional plane, where the false positive rate (FPR) is on the x-axis against

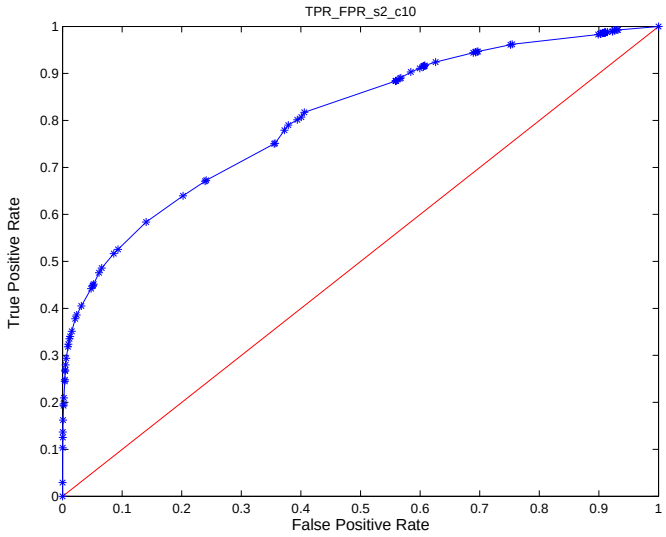


Figure 6.1: Illustration of the paths through coverage space

the true positive rate (TPR) on the y-axis [43].

Table 6.1: The confusion matrix

	True Target Class	
	Positive	Negative
Predict Positive	**T**rue **P**ositives **(p)**	**F**alse **P**ositives **(n)**
Predict Negative	**F**alse **N**egatives **(P-p)**	**T**rue **N**egatives **(N-n)**
Total	**P**	**N**

The rule learning is to add a rule to a rule set and increase the coverage of the rule set. This means that more examples are classified as positive. All positive examples that are uniquely covered by the newly assessed rule contribute to an increase of the true positive rate on the training data. Conversely, covering additional negative examples may be viewed as increasing the false positive rate on the training data. Therefore, adding rule R_{i+1} to rule set R_i effectively moves from a point $R_i = (n_i/N, p_i/P)$ to a new point $R_{i+1} = (n_{i+1}/N, p_{i+1}/P)$. Moreover, R_{i+1} will typically be closer to (N, P) and father away from (0,0) than R_i.

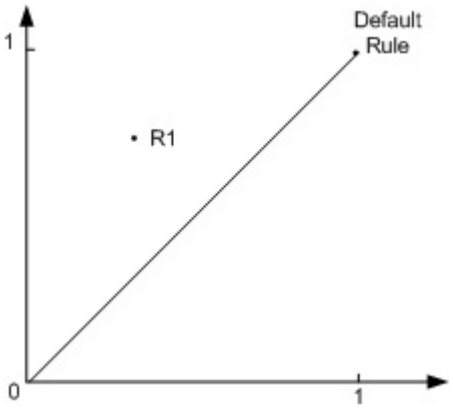
Figure 6.1 shows such a process of rule learning. From this figure, it can be seen that learning a rule set may be viewed as a path through coverage space, where each point on the path corresponds to the addition of a rule to the set. Such a coverage path starts at (0,0), which does not cover any examples. Adding a rule then moves to a new point in coverage space, which corresponds to a set consisting of all rules that have been learned so far. The final rule set learned by a classifier will typically correspond to the last point on the curve before it reaches $(n/N,p/P)$.

6.2 Rule Selection Using the ROC Analysis

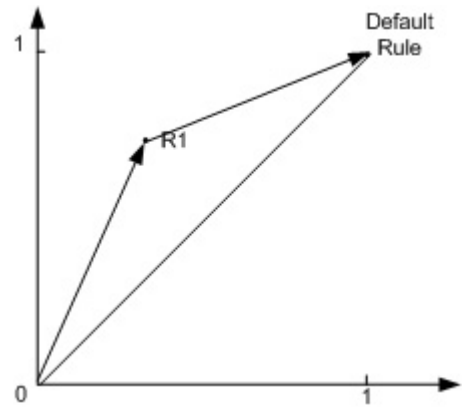
Our approach is to implement a selection step based on the ROC curve. The basic idea is to insert a rule in the rule list if the insertion leads to a point outside the current ROC convex hull, otherwise the rule is discarded . The selection of rule is based on the assumption that larger covering space can be obtained when selecting the rule locating outside the convex hull [114].

In the application of rule selection using the ROC analysis based method, rules are selected separately for each class, and are kept in an ordered rule list. For a two-class classifier, one class is labelled as positive and the other class is negative. The ROC analysis compares different classifiers according to their TPR and FPR, which also allows to plot classifier in a two dimensional space, one dimension for each of these two measurements.

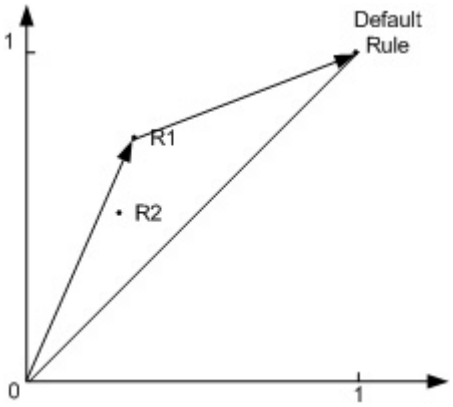
Figure 6.2 shows how a new rule is inserted in the convex hull. As we introduced in the last section, each rule can be described by two measures, fpr and tpr. So we can obtain pairs of points (fpr_i, tpr_i) according to the covered examples by various rules in a rule set. We try to insert a new rule R_1. As the actual convex hull is formed only by the line connecting (0,0) and (1,1), R_1 will be inserted if



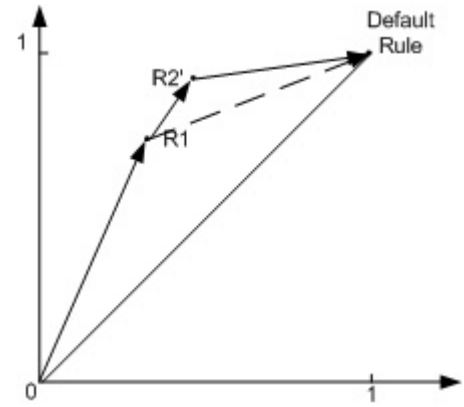
(a) The first rule learning



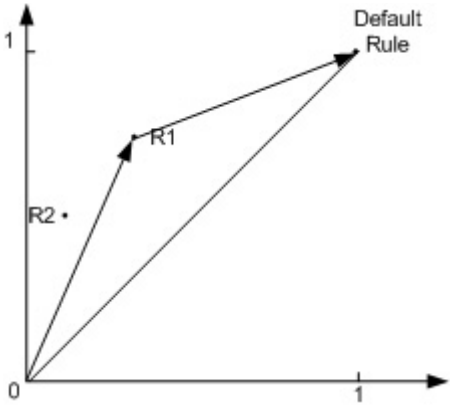
(b) The first rule is inserted in the rule set



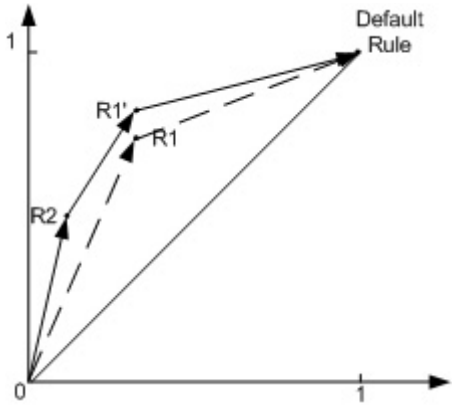
(c) The position of R2 is inside the convex hull, hence, R2 should not be inserted



(d) Update the position of R2



(e) The position of R2 is outside the convex hull



(f) Update the position of R1

Figure 6.2: Example of rule learning using ROC analysis

the slope, defined as tpr/fpr, of the new rule is larger than 1, which also means the position of the point is over the line. Then the convex hull is updated to be defined by the three points, (0,0), (fpr_1, tpr_1), and (1,1). This case can be found in figure 6.2(a) and 6.2(b). We now try to insert a second rule, R_2, after the first rule R_1 had been inserted. There are two cases that can occur. The first case is that the slope of R_2 is less than that of R_1 while the slope of R_2 is larger than R_1's slope for the second case. The two cases are shown in figure 6.2(c) and 6.2(e), respectively. As mentioned before, the new rule could share some examples with these learned rules (in this case R_1), so we will removing the shared examples and adapt the coordinates of the rules on the convex hull. For the first case shown in figure 6.2(c), the shared examples covered by R_2 will be removed, and the value of fpr_2 and tpr_2 will be updated to be fpr'_2 and tpr'_2. The coordinate of rule R_2, therefore, will be $(fpr_1 + fpr'_2, tpr_1 + tpr'_2)$, as shown in figure 6.2(d) . For the second case shown in figure 6.2(e), rule R_2 lies outside of the R_1. So we keep the value of fpr_2 and tpr_2, and only remove the shared examples covered by R_1. So the coordinate of rule R_1 will also be updated to be $(fpr_2 + fpr'_1, tpr_2 + tpr'_1)$ as shown in figure 6.2(f).

In our practical application of the ROC algorithm, we still meet some problems. The first is the occurrence of concavities in the convex hull, and the second is the order of rules presented to ROC analysis.

For the first problem, it is because some rules we are trying to insert do not keep the hull convex. Actions can be taken as described in [112] to detect the concavities and remove the rules that cause the concavities and try to reinsert the rule but the adaptation procedure is complex. Since we focus more on the application of taxonomy on the rule-based classification, the constructor of the rule-based classifier just provides us a platform. So, we utilise a simple method to try to remove the rules which may cause the concavities in the convex hull. The

Algorithm: An Algorithm for Rule Learning Using ROC Analysis

(1) **Input**: A rule set, $RS = \{R_1, R_2, ..., R_n\}$, for a given target class, which are generated by using Apriori association rule learning algorithm.

(2) **Pre-processing**:
I. Prune the rules according to the threshold of Support and Confidence of a rule.
II. Rank the order of the rules according to their gradient, where
$gradient_{R_i} = tpr_{R_i}/fpr_{R_i}$.
$RS' = \emptyset$
foreach $R_i \in RS$
| **if** ($(Support_{R_i} > Threshold_{supp})$ and $(Confidence_{R_i} > Threshold_{conf})$)
| **then**
| $RS' = RS' \cup \{R_i\}$.
| **endif**
endfor
Sort RS according to the gradient, then $RS = \{R_1, R_2, ..., R_n\}$

(3) **Rule Learning**:
$RS'_{refined} = \emptyset$
foreach $R_j \in RS$'
| **if** R_j belongs to convex hull of RS
| **then** $RS'_{refined} = RS'_{refined} \cup R_j$
| Update the values of tpr and fpr of rules in $RS - RS'_{refined}$by removing
| the examples covered by the new rule R_j;
| **endif**
| **foreach** $R_k \in RS'_{refined}$
| **if** $tpr_{R_k} < threshold_{tpr}$
| **then** remove R_k from $RS'_{refined}$
| **endif**
| **endfor**
endfor

(4) **Output**: The final rule set $RS'_{refined}$

Figure 6.3: Rule learning algorithm using ROC analysis

simple concavities' removal is dependent on the value of the slope of a rule to be inserted after the overlapped examples covered by those learned rules have been removed. If the latest slope of a rule to be inserted is less than a threshold, then the rule will be discarded.

For the second problem, [113] uses the Euclidean distance to the point (0,1) in the ROC space. Here, we adopt another way to make an initial order by only computing the initial gradient of a rule without any examples being removed. This ranking means the first processed rule has a lower gradient while the later ones have high gradient, which are often thought of as useful rules for the construction of a good classifier. The pseudo code of our modified rule learning algorithm is given in Figure 6.3.

By using the rule learning algorithm, we can build a classifier based on the refined rule set for each class. For classifying a new given instance, the algorithm evaluates the classification of the new instance in both rule sets for the positive and negative classes. Each rule set will return the values of tpr and fpr of each rule, which can cover this new instance. We therefore select a rule with respect to the value of gradient. The classification follows the rule with the largest value.

6.3 Applying the attribute-value taxonomies to the rule-based classifier

In this section, we propose exploiting AVTs within the rule based classifier. We aim to further reduce the number of rules whilst maintaining an appropriate classification performance.

We here still use the same attribute-value taxonomies, their related coded nodes and the five top-ranked cuts of each AVT as described in chapter 5. Although each attribute might only contain dozens of cuts over its AVT, the total number

of combinations of these cuts over the selected attributes increase exponentially. So a key problem is how to select the optimal cuts over the AVTs. We introduce two different approaches to handle this problem. The first approach (Top-1) is to update the original data set using the top-ranked cuts which are ranked by the computed *gain ratio* (GR) value. The second approach (Top-5) is to search the optimal path over the top five ranked cuts according to the number of generated rules.

6.3.1 Use of Top-ranked Cuts (Top-1)

We assume that the cuts with the largest *gain ratio* value will be the optimal path. After selecting the top-ranked cuts, we can hence update the original data using the attribute value nodes contained in the selected cuts, and then train a new rule based classifier using ROC analysis.

6.3.2 Use of Optimal Path over Top Five Ranked Cuts (Top-5)

In Top-1, we actually utilised two assumptions. The first assumption is that the attributes are independent to each other, and the second one is that the top-ranked cut is the best cut. However, the first assumption ignores the dependencies among these attributes, and the second assumption ignores a fact that the top one cut ranked by *gain ratio* value cannot be guaranteed to be in the optimal solution.

Because of these reasons, a divide-and-conquer strategy will be used and an illustration is shown in Figure 6.4. In this strategy, we will firstly rank the attributes according to their *information gain* (IG) values. The related IG values of the attributes have been listed in table 5.2 and 5.9 for the *Adult* and the *Mushroom* data set, respectively. We assume the attribute with the larger IG value should

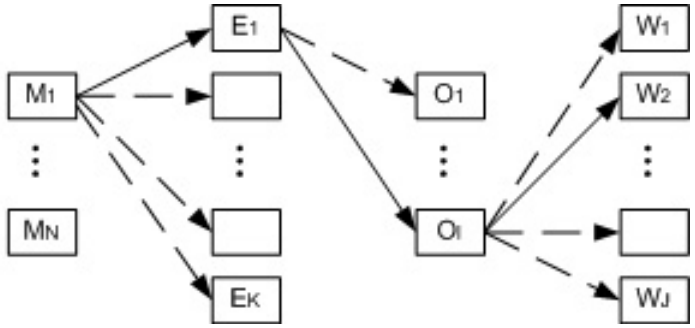


Figure 6.4: An illustration of searching the optimal paths over the top five ranked cuts of the attributes of *Adult* data set (M, E, O, W are the four nominal attributes)

be processed earlier because they are more important to classification than other attributes with smaller IG values.

In the following steps, we only test the attribute-value cuts of one attribute, whilst keeping other attributes unchanged. The cut of the current attribute generating the minimum number of learning rules will be selected, and then the cuts of the next attribute will be processed using the same steps. In Figure 6.4, each column represents all the possible cuts of an attribute-value taxonomy, the directed lines are the possible linkages between two different attributes. The dashed line denotes the sub-optimal choices when adding a cuts of a new attribute while the solid line denotes the optimal choice.

6.4 Performance Evaluation on the *Adult* Data Set

6.4.1 Performance Using the ROC Analysis

The *Adult* data set is selected to evaluate the performance using the ROC analysis. Table 6.2 shows the number of generated association rules using different numbers of antecedents. When the length of antecedent is 3 or 4, a large number of rules are

generated whilst a very small number of rules is obtained when only 1 antecedent is used. Here, it seems to be more appropriate to use 2 antecedents. In order to further reduce the time consumed in computation, we pre-prune the rules by setting the minimal threshold of *Support* and *Confidence*. The number of rules after being pruned based on various thresholds of *Support* and *Confidence* are listed in Table 6.3.

Table 6.2: The number of rules using different number of antecedents

Number of Antecedents	1	2	3	4
Number of Rules	148	4422	61118	432994

Table 6.3: The number of rules after being pruned by setting the minimal values of *Support* and *Confidence*

Confidence	**Support (%)**				
(%)	0.1	0.2	0.5	1	2
0	3068	2368	2048	1590	1140
10	2441	2119	1676	1320	973
20	2132	1849	1464	1164	863
30	1879	1625	1268	1008	739
40	1715	1486	1167	910	662
50	1534	1319	1024	795	570
60	1354	1153	881	680	478
70	1190	1013	762	528	401

After obtaining the pruned rules, we rank the presentation order of each rule according to their gradients. The earlier the rule is presented, the larger gradient value it has. Therefore, the position of each rule on the convex hull will be changed based on the adaptation as shown in figure 6.2(c) and 6.2(d). Considering the size of the *Adult* data set, which contains 30162 instances, we set the *Support* threshold as 0.1% ($\approx$ 30 instances). After setting the *Support* threshold, we test the rule based classifier based on the ROC analysis with different *Confidence* threshold varying from 40% to 70%. In addition, in our method, we set a TPR threshold. Each time when a new rule is inserted, the *tpr* and fpr of those learned rules will

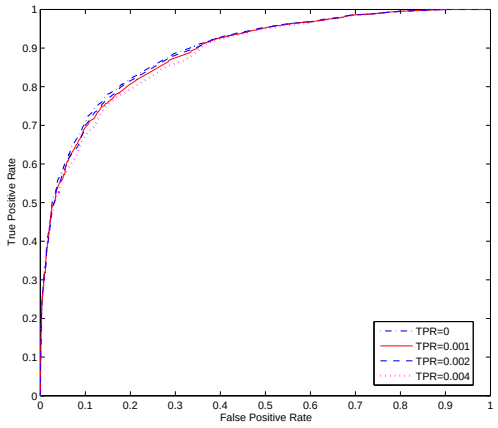
be updated. If the new *tpr* is less than the threshold, this rule will be removed from the convex hull.

Table 6.4 shows the performances using ROC analysis, where we list the number of rules in the final rule set, the accuracy rate on the training set and test set, and the covering rate of the rules of the two trained convex hulls for <=50K and >50K, respectively. Here, we can find that a reasonable performance can be obtained when setting the TPR and Confidence threshold as 0.4% and 60%, respectively.

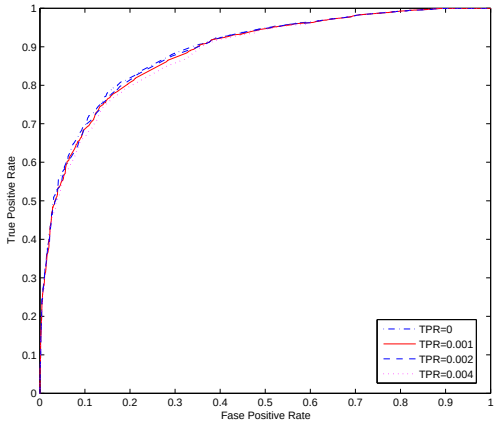
Table 6.4: Performances of rule based classifier using ROC analysis

Conf. (%)	**TPR** (%)	**#Rule** (> / <=)	**Covering Rate** (<=50K ,%) (FPR/TPR)	**Covering Rate** (>50K ,%) (FPR/TPR)	**Accuracy (%)** (train/test)
40	0	123/312	99.81/100	60.56/99.05	84.62/84.60
40	0.1	72/128	86.61/99.74	53.74/98.36	84.45/84.30
40	0.2	51/88	85.60/99.70	52.63/97.93	84.56/84.38
40	0.3	39/71	85.61/99.69	48.15/96.81	84.17/83.96
40	0.4	35/55	84.48/99.63	44.16/94.97	84.00/83.80
50	0	101/312	99.81/100	38.34/88.97	84.48/84.35
50	0.1	67/128	86.61/99.74	36.01/88.31	84.48/84.33
50	0.2	46/89	86.17/90.71	34.90/87.54	84.47/84.33
50	0.3	35/71	84.71/99.65	30.33/86.13	84.44/84.21
50	0.4	32/55	84.48/99.63	26.34/84.31	84.08/83.91
60	0	74/312	99.81/100	15.63/70.35	84.61/84.58
60	0.1	49/128	86.61/99.74	14.45/69.67	84.46/84.29
60	0.2	34/89	85.60/99.69	13.13/68.06	84.62/84.41
60	0.3	27/71	84.70/99.65	12.84/67.18	84.30/84.08
60	**0.4**	**25/55**	**84.48/99.63**	**12.82/67.08**	**84.20/84.02**
70	0	47/312	99.81/100	4.86/42.11	82.44/82.33
70	0.1	31/128	86.61/99.74	4.59/41.78	82.41/82.24
70	0.2	22/89	85.60/99.69	4.31/40.90	82.42/82.19
70	0.3	18/71	84.70/99.65	4.13/40.18	82.32/82.12
70	0.4	16/55	84.48/99.63	4.11/40.09	82.31/82.21

Figure 6.5 (a) and (b) further show the ROC curves of the results on the training and test data with setting different TPR thresholds when the confidence threshold is fixed to be 60%. In these two figures, TPR = 0 means there is no further rule



(a) training data



(b) test data

Figure 6.5: ROC curves example of the results on the *Adult* training and test data with different TPR thresholds when the confidence threshold is fixed (Conf.=60%)

prune caused by the TPR threshold, which seems to generate a slightly better classification performance than those with other thresholds. However, given the large number of pruned rules using TPR = 0, as shown in table 6.4, the loss of accuracy after setting the TPR threshold is trivial. This reflects the effectiveness of rule selection when using the ROC based approach.

6.4.2 Performance Evaluation Using AVT on Rule Learning

We still use the attribute-value taxonomies of the four nominal attributes of the *Adult* data set, *Education*, *Occupation*, *Martial-status*, and *Work-class*. Table 5.3 has listed all the coded nodes of the four taxonomies, whilst the top five ranked cuts, denoted by the coded nodes, of each attribute value taxonomy along with their *gain ratio* values were also given in table 5.4. We firstly utilise Top-1 by viewing the cut with highest *gain ratio* value as the optimal cut of the AVT, and apply it to the ROC based rule learning. Table 6.5 shows the performances using (*after*) and without using (*before*) attribute-value taxonomies(AVTs). It is easy to find that the number of the learned rules using AVTs is less than that without using AVTs when the TPR threshold varying in the range of 0.1% and 0.8%. Especially, when the TPR threshold is 0.8%, the number of rules can be reduced to 44, whilst the classification accuracy can also be improved by more than 3% in comparison with that of not using AVTs.

Table 6.5: Performances of a rule based classifier before and after using attribute-value taxonomies (Top-1)

TPR (%)	**# Rule**			**Accuracy (%) (train/test)**		
	before	*after*	Change(%)	*before*	*after*	Change(%)
0.1	177	94	-46.9	84.46/84.29	84.23/83.66	-0.27/-0.75
0.2	122	81	-33.6	84.62/84.41	84.21/83.66	-0.48/-0.89
0.3	98	67	-31.6	84.30/84.08	84.09/83.67	-0.25/-0.49
0.4	80	61	-23.8	84.20/84.02	84.19/83.77	-0.01/-0.29
0.6	57	50	-12.3	84.11/84.00	**84.17**/83.74	+0.07/-0.31
0.8	45	44	-2.2	81.13/81.06	**84.10/83.63**	+3.7/+3.2

Following the description of the Top-5 approach, we start from *Marital-status* as it is in the first place among the four attribute sets according to its *information gain* value. Table 6.6 shows the number of generated rules after searching through the cuts of the AVTs over the four attributes. Here, for each AVT, only the five top ranked cuts are used. From this table, we can find the top-ranked cut of the AVT of three attributes (*Marital-status, Education, and Workclass*) actually generate more rules than other cuts do. In addition, although the number of learned rules using cut *M4* and *M5* is the same, we select cut *M4* because it has larger gain ratio value. We then test all the possible cuts of the *Education* taxonomy after selecting *M4*. Following the same steps, we finally obtain the cut path over the four attribute-value taxonomies. When setting different TPR thresholds, e.g. 0.6% and 0.8%, the obtained cut path over all cuts of AVTs is *M4*, *E3*, *O1*, and *W4*, and the number of learned rules are 46 and 40, respectively.

As a comparison, the results using the original data, *Gain Ratio*, and the number of rules are listed in Table 6.7. We can hence find that the use of the second measurement can reduce the number of rules by more than 10%, while a slightly better classification accuracy (about 3%) is obtained on the test set when the TPR threshold is 0.8%.

We also run a 16-time 3-fold cross validation on the *Adult* data set. Figure 6.6 shows the classification accuracy when different methods are utilised. In this

Table 6.6: Selection of an optimal cuts over the four attribute-value taxonomies of the *Adult* data set

Step1		**Step2**		**Step3**		**Step4**	
Attr.	#R	Attr.	#R	Attr.	#R	Attr.	#R
Threshold = 0.6%; Supp. = 0.1%; Conf. = 60%							
M1	59	M4+E1	60	**M4+E3+O1**	**49**	M4+E3+O1+W1	48
M2	59	M4+E2	60	M4+E3+O2	54	M4+E3+O1+W2	48
M3	59	**M4+E3**	**56**	M4+E3+O3	51	M4+E3+O1+W3	49
M4	**57**	M4+E4	60	M4+E3+O4	56	**M4+E3+O1+W4**	**46**
M5	57	M4+E5	61	M4+E3+O5	52	M4+E3+O1+W5	47
Threshold = 0.8%; Supp. = 0.1%; Conf. = 60%							
M1	46	M4+E1	52	**M4+E3+O1**	**42**	M4+E3+O1+W1	42
M2	46	M4+E2	51	M4+E3+O2	47	M4+E3+O1+W2	42
M3	47	**M4+E3**	**48**	M4+E3+O3	43	M4+E3+O1+W3	43
M4	**45**	M4+E4	50	M4+E3+O4	48	**M4+E3+O1+W4**	**40**
M5	45	M4+E5	51	M4+E3+O5	47	M4+E3+O1+W5	42

figure, Top-1 indicates that we use the top-ranked cut to update the original date and Top-5 means the optimal searched path over the top five ranked cuts of the attributes. We can find the use of our approach can generate a slightly higher classification accuracy than that not using the attribute-value taxonomies. The number of rules is reduced in all cases.

6.5 Performance Evaluation on the *Mushroom* Data Set

In this section we present the performance evaluation on the *Mushroom* data set following the same evaluation procedures on the *Adult* data set as described in section 6.4.

6.5.1 Performance Evaluation Using the ROC Analysis

Table 6.8 shows the number of generated rules when using different numbers of antecedent. As with the *Adult* data set, the use of 2 antecendents is more feasible.

Table 6.7: Comparison of performances using different settings

Conf. (%)	**TPR** (%)	**#Rule**	**Covering Rate** (<=50K ,%) (FPR/TPR)	**Covering Rate** (>50K ,%) (FPR/TPR)	**Accuracy (%)** (train/test)
Original data without using Attribute-Value Taxonomies					
60	0.6	57	82.95/99.58	12.09/65.90	84.11/84.00
60	0.8	45	66.34/93.57	11.84/65.34	81.13/81.06
Using top-ranked cuts (Top-1)					
60	0.6	**50**	83.75/99.67	15.47/72.32	**84.17**/83.74
60	0.8	**44**	83.55/99.66	14.81/71.59	**84.10/83.63**
Using the optimal paths over the top-five cuts (Top-5)					
60	0.6	**46**	83.47/99.65	13.31/68.07	**84.05/84.19**
60	0.8	**40**	83.19/99.61	13.22/67.31	**83.82/83.94**

To reduce the time consumed in computation, we pre-prune the rules by setting the *Support* and *Confidence* threshold, and the number of pruned rules using different thresholds are listed in Table 6.9.

Table 6.8: The number of rules using different number of antecedents

Number of Antecedents	1	2	3	4
Number of Rules	119	14161	426703	6267767

For the *Mushroom* data set, we focus on the rules generated using larger *Support* and *Confidence* thresholds to reduce the interference caused by some redundant information. In our work, the *Support* threshold is set as 1%, while the *Confidence* threshold is set within the range from 80% to 98%. The TPR threshold ranges from 0.6% to 0.8%.

Table 6.9: The number of rules after being pruned by setting the minimal values of *Support* and *Confidence*

Confidence (%)	**Support** (%)			
	0.1	0.2	0.5	1
10	6845	6384	5800	4921
20	4486	4254	3939	3468
30	4111	3879	3571	3110
40	3826	3596	3290	2851
50	3564	3336	3044	2612
60	3327	3100	2811	2383
70	2829	2604	2334	1917
80	2548	2325	2057	1662
90	2127	1949	1688	1320
95	2127	1949	1688	1320
98	2005	1784	1525	1147

Table 6.10: Performances of rule based classifier using ROC analysis

Conf. (%)	**TPR** (%)	**#Rule** (Edi./Poi.)	**Covering Rate** (Poi. ,%) (FPR/TPR)	**Covering Rate** (Edi. ,%) (FPR/TPR)	**Accuracy (%)** (train/test)
80	0.6	13/11	0/98.92	0/99.16	99.07/99.28
80	0.7	11/11	0/98.92	0/97.82	98.36/98.57
80	0.8	11/11	0/98.92	0/97.82	98.36/98.57
90	0.6	13/11	0/98.92	0/99.16	99.07/99.28
90	0.7	11/11	0/98.92	0/97.82	98.36/98.57
90	0.8	11/11	0/98.92	0/97.82	98.36/98.57
95	0.6	13/11	0/98.92	0/99.16	99.07/99.28
95	0.7	11/11	0/98.92	0/97.82	98.36/98.57
95	0.8	11/11	0/98.92	0/97.82	98.36/98.57
98	0.6	13/11	0/98.92	0/99.16	99.07/99.28
98	0.7	11/11	0/98.92	0/97.82	98.36/98.57
98	0.8	11/11	0/98.92	0/97.82	98.36/98.57

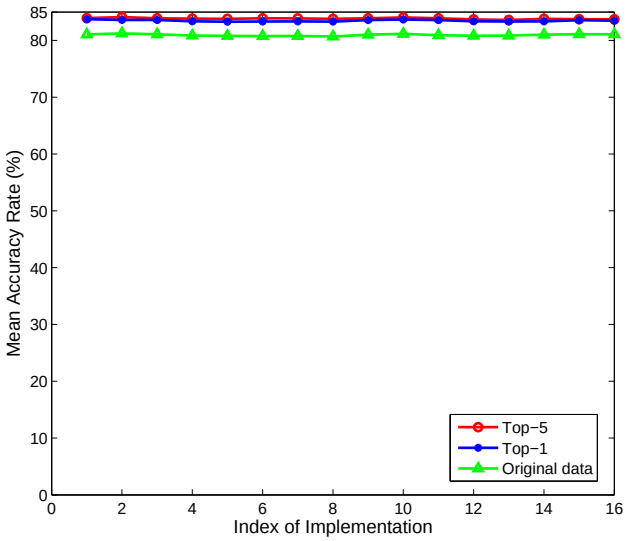
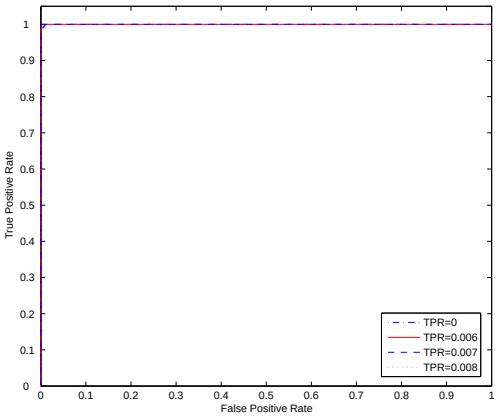


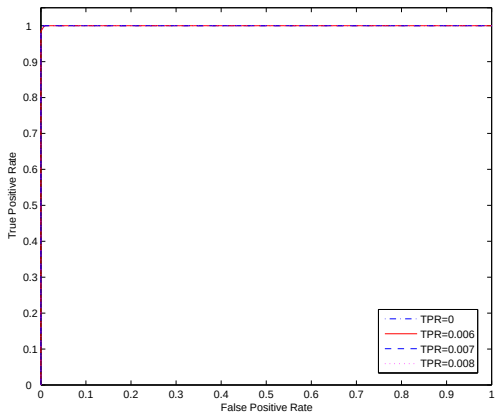
Figure 6.6: Classification accuracy on the *Adult* data set with 16-time 3-fold cross validations (Conf=60% and TPR=0.8%)

Table 6.10 shows the classification performances using the ROC analysis on the original data set when varying the *Confidence* and the TPR thresholds, respectively. We can find there are no variations when four different *Confidence* threshold values are set, respectively. This is because almost all the key rules generated to distinguish the EDIBLE mushroom (*Edi.*) from POISONOUS ones (*Poi.*) are kept even if the *Confidence* threshold value is set as large as 98%. Here, we choose three values (0.6, 0.7 and 0.8) as the TPR threshold, respectively. As an important constraint on the ROC analysis, the large TPR value (TPR=0.8%) can cause about 0.7% reduction in classification accuracy compared with a small TPR value (TPR=0.6%). This is because, when setting a larger TPR threshold, more rules are removed. Thus some instances are misclassified. However, the number of learned rules when setting the TPR threshold as 0.8% is smaller than when the TPR threshold is 0.6%.

Figure 6.7 (a) and (b), respectively, illustrate the ROC curves according to the classification results on the training and test data when the confidence threshold



(a) training data



(b) test data

Figure 6.7: ROC curves example of the results on the *Mushroom* training and test data with different TPR thresholds when the confidence threshold is fixed (Conf.=98%)

is fixed to be 98% and the TPR threshold changes from 0.001 to 0.004. In these two figures, we can find there are few differences among theses curves although four different TPR thresholds are set. This is because, after using the ROC based approach, we still obtain the most important rules, which can cover almost all instances in the training and test set.

6.5.2 Performance Evaluation Using AVT on Rule Learning

To apply the constructed attribute-value taxonomies on the *Mushroom* data set to the ROC analysis, similar to the work on the *Adult* data set, we also use the top-ranked cuts extracted from the AVTs of the *Mushroom* data set. The coded nodes of the attributes are listed in table 5.12 and the ranked cuts of each attribute are in table 5.13.

When using Top-1, the top-ranked cut with largest *gain ratio* value is selected to update the original data set, and then the new rules dependent on the ROC analysis is obtained. As a comparison, table 6.11 shows the results before and after updating the original data. It is clear that the use of an attribute value taxonomy can significantly reduce the number of rules by more than 50%, whilst there are also small improvements for the classification performances .

Table 6.11: Performances of a rule based classifier before and after using attribute-value taxonomies (Top-1) on the *Mushroom* data set

TPR	**# Rule**			**Accuracy (%) (train/test)**		
(%)	*before*	*after*	Change(%)	*before*	*after*	Change(%)
0.1	33	15	-54.5	99.73/99.76	99.82/99.64	+0.09/-0.12
0.2	27	14	-48.1	99.43/99.40	99.72/99.52	+0.29/+0.12
0.3	25	13	-48	99.22/99.40	99.61/99.52	+0.39/+0.12
0.4	24	13	-45.8	99.22/99.40	99.61/99.52	+0.39/+0.12
0.6	24	11	-54.1	99.22/99.40	99.61/99.52	+0.39/+0.21
0.8	23	10	-56.5	98.81/98.83	99.24/99.04	+0.43/+0.21

Following the same steps to find the optimal inter-AVT combinations for the *Adult* data set, we search through the top-five attribute value cuts of each attribute. The search is based on the measurement that the cuts generating the minimum number of rules will be selected. In each step, the original data set is updated using one of the cuts of the current attributes, and we then compare which cuts generates the minimum rules in the same condition. Table 6.12 shows the results of searching the optimal cuts through the AVTs of the seven attributes when the TPR threshold are set as 0.6% and 0.8%, respectively. From this table, we find that the number of learned rules is reduced more than 50% when the TPR threshold is 0.8%. In addition, the optimal path is the top-ranked cuts of each attribute (*O1, P1, S1, G1, B1, H1, C1*), which is different from the results we obtained on the *Adult* data set. This case is mainly caused by two factors:

1. The selected top ranked cuts are able to reasonably categorise different attribute values into their related semantic clusters.

2. The attribute *Odor* of the *Mushroom* data set plays a dominant role in searching the optimal cuts. This means there are few variations of the rule numbers after searching through the AVT of *Odor*.

In comparison with the *Adult* data set, the structure obtained on the attribute values of the *Mushroom* data set is simpler. In some complex cases, like the *Adult* data set, although the top-ranked cuts can generally generate a better performance than other cuts ranked behind it, it does not mean it will be an optimal choice as a whole. This also indicates the necessity of implementing our method to search for the optimal path.

Table 6.13 compares the recognition performances on the original *Mushroom* data set, the data set updated using Top-1 and the data set updated using Top-5, respectively. When the TPR threshold is set at 0.6%, better classification per-

Table 6.12: Selection of an optimal cuts over the four attribute-value taxonomies of the *Mushroom* data set

Step1		**Step2**		**Step3**		**Step4**		**Step5**		**Step6**		**Step7**	
Att.	#R	Att.	#R	Att.	#R	Att.	#R	Att.	#R	Att.	#R	Att.	#R
Threshold = 0.6%; Supp. = 1%; Conf. = 98%													
O1	13	O1+P1	12	O1+P1+S1	12	O1+P1+S1+G1	12	O1+P1+S1+G1+B1	12	O1+P1+S1+G1+B1+H1	11	O1+P1+S1+G1+B1+H1+C1	11
O2	16	O1+P2	12	O1+P1+S2	12	O1+P1+S2+G2	12	O1+P1+S1+G1+B2	12	O1+P1+S1+G1+B1+H2	11	O1+P1+S1+G1+B1+H1+C2	11
O3	16	O1+P3	12	O1+P1+S3	12	O1+P1+S3+G3	12	O1+P1+S1+G1+B3	12	O1+P1+S1+G1+B1+H3	11	O1+P1+S1+G1+B1+H1+C3	11
O4	21	O1+P4	12	O1+P1+S4	12	O1+P1+S4+G4	12	O1+P1+S1+G1+B4	12	O1+P1+S1+G1+B1+H4	11	O1+P1+S1+G1+B1+H1+C4	11
O5	24	O1+P5	13	O1+P1+S5	12	O1+P1+S5+G5	12	O1+P1+S1+G1+B5	12	O1+P1+S1+G1+B1+H5	11	O1+P1+S1+G1+B1+H1+C5	11
Threshold = 0.8%; Supp. = 1%; Conf. = 98%													
O1	12	O1+P1	10	O1+P1+S1	10	O1+P1+S1+G1	10	O1+P1+S1+G1+B1	10	O1+P1+S1+G1+B1+H1	10	O1+P1+S1+G1+B1+H1+C1	10
O2	14	O1+P2	10	O1+P1+S2	10	O1+P1+S2+G2	10	O1+P1+S1+G1+B2	10	O1+P1+S1+G1+B1+H2	10	O1+P1+S1+G1+B1+H1+C2	10
O3	14	O1+P3	10	O1+P1+S3	10	O1+P1+S3+G3	10	O1+P1+S1+G1+B3	10	O1+P1+S1+G1+B1+H3	10	O1+P1+S1+G1+B1+H1+C3	10
O4	20	O1+P4	10	O1+P1+S4	10	O1+P1+S4+G4	10	O1+P1+S1+G1+B4	10	O1+P1+S1+G1+B1+H4	10	O1+P1+S1+G1+B1+H1+C4	10
O5	23	O1+P5	11	O1+P1+S5	10	O1+P1+S5+G5	10	O1+P1+S1+G1+B5	10	O1+P1+S1+G1+B1+H5	10	O1+P1+S1+G1+B1+H1+C5	10

formances are obtained, 99.61%, on the training set and, 99.52%, on the test set, whilst a smaller number of rules (10 rules) are generated with the threshold set at 0.8%. When using AVTs, both Top-1 and Top-5 can significantly improve efficiency by reducing the number of learned rules in comparison with the original data set. To test the effectiveness, we also run a 10-fold cross validation.

Table 6.13: Comparison of performances using different settings

Conf. (%)	**TPR** (%)	**#Rule**	**Covering Rate** (Poi. ,%) (FPR/TPR)	**Covering Rate** (Edi. ,%) (FPR/TPR)	**Accuracy (%)** (train/test)
Original data without using Attribute-Value Taxonomies					
98	0.6	24	0/98.92	0/99.16	99.07/99.28
98	0.8	22	0/98.92	0/97.82	98.36/98.57
Using top-ranked cuts (Top-1)					
98	0.6	11	0/99.54	0/99.62	**99.61/99.52**
98	0.8	**10**	0/98.75	0/99.62	99.24/99.04
Using the optimal paths over the top-five cuts (Top-5)					
98	0.6	11	0/99.54	0/99.62	**99.61/99.52**
98	0.8	**10**	0/98.75	0/99.62	99.24/99.04

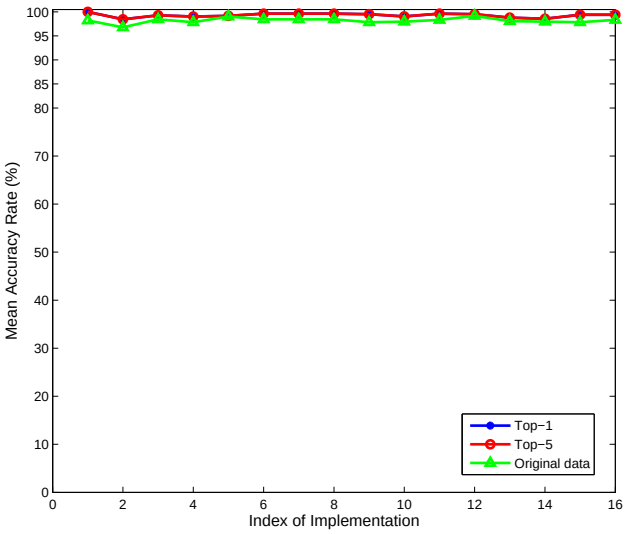


Figure 6.8: Classification accuracy on the *Mushroom* data set with 16-time 10-fold cross validations (Conf=98% and TPR=0.8%)

Figure 6.8 shows the classification accuracy by implementing cross validation on the *Mushroom* data set for 16 times when different methods are utilised. In

this figure, we find the use of Top-1 and Top-5 generates the same performance (the curves using Top-1 and Top-5 overlap with each other), but both methods have slightly better performances than obtained without using AVTs.

6.6 Summary

This chapter proposes a novel framework of integrating AVTs with ROC analysis for learning a better rule classifier. We used two approaches to implement the integration. The first approach (Top-1) is to update the original data set by using the top-ranked cuts. The second approach (Top-5) is to search for an optimal path over the top five ranked cuts of each selected attribute. The search procedure is based on the minimum number of learned rules over the attribute-value taxonomies. In our experiments, when comparing with the performances without using AVTs, we obtained better performances in both rule reduction and accuracy. The use of AVTs can reduce the number of learned rules by more than 10% on the *Adult* data set and by about 50% on the *Mushroom* data set. For the *Adult* data set, we improve the accuracy by 3% when the TPR threshold is 0.8%, while we also obtained a slightly better classification accuracy on the *Mushroom* data set.

The use of Top-5 outperforms Top-1 on the *Adult* data set after running cross validation for 16 times. This is because Top-1 assumes that the cut with largest *gain ratio* value is the optimal cut, and ignores the dependency between attributes. For the *Mushroom* data set, there is no difference in classification accuracy when using Top-1 and Top-5. This is mainly because that the "Odor" attribute plays a dominant role in performance evaluation. It is reasonable to expect Top-5 to be more suitable for processing more complex taxonomies compared with the first approach.

Chapter 7

Conclusions and Future Work

7.1 Overview

This chapter summarizes our research work as a whole, draws some conclusions according to the experimental performances described in previous chapters, discusses some phenomena in our work, and finally gives some suggestions for future work based on our discussion.

7.2 Thesis Summary

The overall aim of our research is to develop a methodology that exploits attribute-value taxonomies (AVTs) for compact and accurate classifier learning. In this thesis, we proposed the automatic construction of attribute-value taxonomies (AVTs) using ontology and clustering methods. We exploited the generated AVTs, with or without manual adjustment, for learning simpler decision trees and smaller sets of association rules without sacrificing too much accuracy. In our experiments, we

test our approaches on two relatively large data sets, the *Adult* and the *Mushroom* data set.

- In chapter 3, we gave a study of ontologies and explored the feasibility of exploiting some unique properties for taxonomy construction. Our work can be divided into two directions. The first direction is ontology building. we reviewed some important issues about ontologies, such as definition, types, representation languages, existing applications, etc. As a part of our early research, we created an illustrative *Student ontology*, using the DAML+OIL ontology language, and designed various examples for each property of DAML+OIL. This part of our work is shown in appendix A.

  The second direction is to extract taxonomies from predefined ontologies. After explaining the relationship between an ontology and a taxonomy, we proposed an algorithm for automatically extracting taxonomies from an ontology. In our work, we treat an ontology as a poly-hierarchy. Then taxonomy extraction transforms the ontology hierarchy into a directed acyclic taxonomy graph (DATG). In our DATG, the properties of ontology are used to guide the direction of the edges. Then our goal is to search and find all the possible subgraphs of the DATG that may become a taxonomy, named as Fully Leaved and Rooted Subgraphs (FLRS). We implemented the algorithm and illustrated this method with some case studies.

- In chapter 4, we utilised two hierarchical clustering algorithms (agglomerative and divisive clustering) and two partitional algorithms (k-means and Fisher's algorithm) to automatically construct concept based taxonomies. Such taxonomies were generated from the nominal and numeric values provided by the attributes of the *Adult* and the *Mushroom* data sets, respectively. Comparing with the hierarchical algorithms which prefer to generate a binary

tree, the use of the partitional algorithms is able to construct the non-binary taxonomy under some supervision, viz. setting the number of clusters before clustering. A related issue, the selection of the optimal number of clusters for partitional algorithms, is also involved and a technique is implemented for both nominal and numeric attribute values. Both partitional algorithms have the same performance for a given optimal number of clusters.

All the generated taxonomies are compared, which revealed some statistical characteristic of the data. Although these taxonomies are not completely compatible with the semantic ones, they provide a good guide on constructing appropriate concept taxonomies for this particular data or for making adjustments to taxonomies extracted from existing ontologies. Such modification is likely to be required if a general taxonomy is to be used for a specific database. We also proposed a concept taxonomy for each selected attribute after adjusting and naming the internal nodes of the taxonomy with suitable semantically intelligent terms.

The taxonomies constructed using the automatic methods help us to find the relationships and dependencies between attributes values. From a subjective point of view, we think the generated attribute-value taxonomies are useful to our work, and we used them in tree and rule induction in later chapters for objective performance evaluation.

- In chapter 5, we presented a methodology of exploiting attribute-value taxonomies (AVT) to generate a simple decision tree with reasonable classification accuracy. We trained a decision tree based classifier to test the effectiveness and efficiency of the classification performance on the *Adult* and the *Mushroom* data sets when using and not using the generated AVTs. In our experiments, we used *gain ratio* (GR) as a measure to select the top ranked

cuts through the attribute-value taxonomies (AVTs). This step can improve the efficiency when searching for the useful cuts in complex taxonomies. We found that there are salient reductions in the number of tree nodes, more than 10% on the *Adult* data set and more than 70% on the *Mushroom* data set, observed when using the taxonomies. At the same time, there is little loss of classification accuracy.

- In chapter 6, we developed a novel framework for integrating the attribute-value taxonomies (AVTs) with ROC analysis for learning a better rule classifier. In our work, we used two approaches to implement the integration. The first approach is to select a cut with the largest *gain ratio* value and update the original data set using the top-ranked cut. The second approach is to search for an optimal path over the top five ranked cuts of the selected attributes. The optimal cut path will be determined according to the number of learned rules. In comparison with the approach without using AVTs, our two approaches can reduce the number of rules on the *Adult* data set by about 10% and the number of rules on the *Mushroom* data set by about 50%. In addition, the classification accuracy using our approaches also consistently outperform the method without using AVTs on the *Adult* and the *Mushroom* data set after implementing cross validation for 16 times.

7.3 Discussion

Although the use of AVTs brings benefits, two points on the AVTs construction are still worth discussion. The first is the difference between the use of the ontology based method and that of the automatic methods using clustering algorithms. The other is about the impact of the data set on the construction of AVTs.

Construction of AVTs using ontology and clustering methods

In this thesis, we proposed two different approaches to build attribute-value taxonomies (AVTs). The first approach described in chapter 3 is to extract the taxonomies from the ontologies pre-defined by domain experts. The second one is dependent on the clustering methods for an automatic construction of AVT, which is presented in chapter 4.

Ontologies are broader in scope than AVTs. An ontology might encompass a number of taxonomies, with each taxonomy organizing a subject in a particular way. This is the motivation of our work described in chapter 3 exploring how to extract the related taxonomies from ontologies. Ontologies generally represent knowledge in a strict form so that computers can derive meaning by traversing the various relationships. This means complex and general ontologies require many subjective definitions and interpretations, provided by manual interference. For taxonomies, the linkages between parent and child branches were much simpler in nature compared to an ontology. We find that it is possible to build an AVT with an automatic method using the simple relationship, such as *is-a*. This is the motivation of our work in chapter 4.

Compared with the second approach, the first one has the advantage of being able to bring some precise and semantic relationships between the conceptual nodes. To generate taxonomies through the first approach, some possible false connections and dependencies can be avoided, which sometimes occur when using automatic methods. However, the second approach is more feasible and practical than the use of ontology, especially when building taxonomies without the domain specific ontologies pre-defined by experts in the field. This will arise in the vast majority of cases. Indeed, pre-defined ontologies on the *Adult* and the *Mushroom* data sets, used in this thesis, do not exist . Hence, we focus on the second approach in our work.

The impact of the characteristics of a data set on the construction of AVTs

When using an automatic method to build an AVT, the quality of the resultant AVT is dependent on two main factors. The first factor is the size of the data set, and the second one is the nature of the attributes and their values.

The automatic construction of the AVT finds correlations between attribute values. A larger data set can statistically describe the correlations more accurately, especially when processing a data set where many nominal attributes have a large number of values. This is one reason why, in our work, we selected the *Adult* and the *Mushroom* data set, which contains 48842 and 8124 instances, respectively.

Secondly, the construction of a meaningful AVT actually relies on the the characteristics of the attribute values. A meaningful AVT can generate a readable information structure, and let people easily understand what each node in the AVT represents. More importantly, a meaningful AVT can make it reusable between domains.

Although some data sets contain many attributes, their attribute values can only provide simple numeric information. This means that some possible latent semantic information that the attribute values possess might be ignored. However such information sometimes may be useful in AVT construction, e.g. age. This is another reason why we select the *Adult* data set in our work since the attributes and their values in this data contains rich semantic information. For example, the latent semantic information of the attribute value "PhD" is that people with a PhD degree are more likely to possess high skills and make greater contribution to society and thus find it easier to obtain a higher salary package.

It may be thought that attributes with few values will have relatively less information than those with more values. However, we notice that the attributes with more values can play a more important role in classification than others. For

example, the attribute "Odor" of the *Mushroom* data set only has nine values, which is less than some other attributes we selected, but this attribute and its AVT are dominant in the *Mushroom* data set. This also indicates the importance of our work in ranking the attributes along with their values according to their computed *gain ratio* values.

7.4 Conclusion

Our study demonstrates that the use of attribute value taxonomies (AVTs) is both practical and desirable when processing a data set with many attributes and a large number of attribute values.

- The construction of AVT using clustering methods provides an effective way to automatically analyse the correlations among attribute values and generate conceptual nodes in a hierarchical format. Moreover, it is also more flexible than the ontology based approach, whose application is limited to the pre-defined structure built using the domain specific expert knowledge.

- The use of gain ratio (GR) provides a reasonable measure to search the valuable cuts through the conceptual nodes of a complex AVT. When combining the extracted cuts with the decision tree, the size of the decision tree can be reduced without the loss of classification accuracy.

- The use of a ROC based approach is more efficient in rule learning when compared to conventional methods, such as the Apriori association rule learning algorithm. Moreover, the combination of the ROC based rule learning algorithm with the AVT can further reduce the number of learned rules, while the classification performance is still kept stable with few variations.

7.5 Future Work

Some promising directions for future work in using attribute-value taxonomies in classifier learning include:

- With increasing use of ontologies and the merging of data from various depositories, data mining is becoming an increasingly complex task. Since the availability of embedded taxonomies may be usefully exploited by researchers, the extraction of these taxonomies from ontologies needs to be automated. In chapter 3, we have shown how these can be defined using "Subclass-of" and "Disjoint-with". When databases are constructed from various sources, not all instances from different entries need necessarily be leaves of the taxonomy; the internal nodes (subclasses) can also appear. It is desirable to develop ways of handling such data in tree induction. In addition, to enhance the ability to construct a domain specific taxonomy using a general ontology, the use of some external semantic information resources, such as WordNet, to expand the space of conceptual relationships is also an interesting research topic.

- When constructing a taxonomy on a data set, especially on a data set with a large number of attributes and attribute values, there often exists the possibility that there will be several different hierarchy structures of the taxonomy that can be generated either through automatic methods or by domain taxonomists. This raises some problems, such as "which structure will be the most representative one?" or "how can we merge or integrate these different taxonomies into one taxonomy which is best suited to the domain application?" It seems worthwhile to develop some new measurements to select the optimal structure according to some possible specific requirements, context information, and computational efficiency. The methods used in

multi-model ensembles may be another good approach to do this work.

- From the taxonomists' point of view, a taxonomy is not a static and closed system. This is because new specimens will be discovered, existing classifications need to be revised, and new scientific knowledge may be gained through new methods. It is of interest to explore approaches for adjusting an existing taxonomy when adding new information to it. Some related techniques used in finite state automata (FSA) and graphical models may be used to handle this problem.

Bibliography

[1] Extensible Markup Language (XML) 1.0 (third edition). In T. Bray, J. Paoli, C. M. Sperberg-McQueen, and F. Yergeau, editors, *W3C Recommendation, http://www.w3.org/TR/REC-xml/*. Feb. 2004.

[2] OWL web ontology language guide. In M. K. Smith, C. Welty, and D. L. McGuinness, editors, *W3C Recommendation, http://www.w3.org/TR/owl-guide/*. Feb. 2004.

[3] RDF primer. In F. Manola and E. Miller, editors, *W3C Recommendation, http://www.w3.org/TR/rdf-primer/*. Feb. 2004.

[4] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proc. of the SIGMOD'93*, pages 207–216, Washington D.C., May 1993.

[5] R. Agrawal and R. Srikant. Fast algorithm for mining association rules. In *Proc. of the 20th International Conference on Very Large Databases (VLDB'94)*, pages 487–499, Santigo, Chile, June 1994.

[6] R. Agrawal and R. Srikant. Mining sequential patterns. In *Proc. of the 11th International Conference on Data Engineering (ICDE '95)*, pages 3–14, Taiwan, March 1995.

[7] S. Al-Harbi. *Clustering in Metric Spaces*. PhD thesis, School of Computing Sciences, University of East Anglia, Norwich, 2003.

[8] H. Alani, S. Kim, D. Millard, M. Weal, W. Hall, P. Lewis, and N. Shadbolt. Automatic ontology-based knowledge extraction from web documents. *IEEE Intelligent Systems*, 18(1):14–21, 2003.

[9] H. Almuallim, Y. Akiba, and S. Kaneda. On handing tree-structured attributes in decision tree learning. In *In Proc. ICML-95*, pages 12–20, 1995.

[10] H. Almuallim, Y. Akiba, and S. Kaneda. An efficient algorithm for finding optimal gain-ratio multiple-split tests on hierarchical attributes in decision tree learning. In *In Proc. AAAI-96*, pages 703–708, 96.

[11] A. Artale, E. Franconi, N. Guarino, and L Pazzi. Part-whole relations in object-centered systems: an overview. In *Data and Knowledge Engineering*, volume 20, pages 347–383. 1996.

[12] A. Asuncion and D. J. Newman. UCI machine learning repository. University of California, Irvine, School of Information and Computer Sciences, 2007.

[13] F. B. Backer and L. J. Hubert. A graph-theoretic approach to goodness-of-fit in complete-link hierarchical clustering. In *J. Am. Stat. Assoc.*, volume 71, pages 870–878. 1976.

[14] L. D. Baker and A. K. McCallum. Distributional clustering of words for text classification. In *Proc. of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, page 96103. ACM Press, 1998.

[15] J. A. Bateman. On the relationship between ontology construction and nutural language: A socio-semiotic view. *International Journal of Human-Computer Studies*, 43(2/3):929–944, 1995.

[16] S. Bechhofer, I. Horrocks, C. Goble, and R. Stevens. OilEd: a reasonable ontology editor for the semantic web. In *Proc. of KI2001, Joint German/Austrian conference on Artificial Intelligence*, volume 2174 of LNAI, pages 396–408, Vienna, Sep. 2001. Springer-Verlag.

[17] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American Magazine, http://www.scientificamerican.com/2001/0501issue /0501berners-lee.html*, 2001.

[18] F. Berzal, J. C. Cubero, N. Marin, and D. Sanchez. Building multi-way decision trees with numerical attributes. *Information Sciences: an International Journal archive*, 165(1-2):73–90, 2004.

[19] D. A. Binder. Bayesian cluster analysis. *Biometrika*, 1978.

[20] S. Bloehdorn and A. Hotho. Text classification by boosting weak learners based on terms and concepts. *Fourth IEEE International Conference on Data Mining (ICDM'04)*, pages 331–334, 2004.

[21] I. Borisova, V. Dyubanov, O. Kutnenko, and N. G. Zagoruiko. Use of the fris-function for taxonomy, attribute selection and decision rule construction. In *Proceedings of the First international conference on Knowledge processing and data analysis (KONT'07/KPP'07)*, volume 6581, pages 256–270, 2007.

[22] R. J. Brachman. What isa is and isn't: An analysis of taxonomic links in semantic networks. In *IEEE Computer*, volume 16, pages 30–36. 1983.

[23] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA., 1984.

[24] J. F. M. Burg. *Linguistic Instruments in Requirements Engineering*. IOS Press, 1997.

[25] Y. Cai, N. Cercone, and J. Han. *Attribute-oriented induction in relational databases*. In Knowledge Discovery in Databases. AAAI/MIT Press, 1991.

[26] M. Centelles. Taxonomies for categorisation and organisation in web sites. *Hypertext.net*, 2005.

[27] Y. Chen and T. C. Huang. A novel knowledge discovering model for mining fuzzy multi-level sequential patterns in sequence databases. In *IEEE Transactions on Knowledge and Data Engineering*, volume 66, pages 349–367, 2008.

[28] P. Cimiano, A. Hotho, and S. Staab. Comparing conceptual, divisive and agglomerative clustering for learning taxonomies from text. In *In Proceedings of the European Conference on Artificial Intelligence (ECAI)*, pages 435–439, 2004.

[29] P. Cimiano, S. Staab, and J. Tane. Automatic acquisition of taxonomies from text: Fca meets nlp. In *In Proceedings of the ECML/PKDD Workshop on Adaptive Text Extraction and Mining*, pages 10–17, Croatia, 2003.

[30] P. Clark and R. Boswell. Rule induction with CN2: Some recent improvements. In *Proc. of the 5th European Working Session on Learning*, pages 151–163, 1991.

[31] P. Clark and T. Niblett. The CN2 induction algorithm. *Machine Learning*, 3(4):216–283, 1989.

[32] W. Cohen. Fast effective rule induction. In *Proc. of the 12th International Conference on Machine Learning*, pages 141–168, 1995.

[33] D. Connolly and F. V. Harmelen. DAML+OIL reference description. *W3C Note, http://www.w3.org/TR/daml+oil-reference*, Dec. 2001.

[34] O. Corcho and A. Gómez-Pérez. Evaluating knowledge representation and reasoning capabilities of ontology specification languages. In *Proc. of the ECAI-00 Workshop on Applications of Ontologies and Problem Solving Methods*, Berlin, 2000.

[35] X. Dimitropoulos, D. Krioukov, B. Huffaker, K. Claffy, and G. Riley. Inferring as relationships: Dead end or lively beginnin. *WEA'2005 LNCS*, pages 113–125, 2005.

[36] A. Farquhar, R. Fikes, and J. Rice. The ontolingua server: a tool for collaborative ontology construction. In *Proc. of the 10th Knowledge Acquisition for KBS Workshop*, pages 44.1–44.19, Banff, Alberta, Canada, 1996.

[37] U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy. From data mining to knowledge discovery: An overview. In *Advances in Knowledge Discovery and Data Mining*, chapter 1. AAAI/MIT Press, Menlo Park, California, 1996.

[38] D. Fensel. Ontologies: Silver bullet for knowledge management and electronic commerce. Springer-Verlag, Berlin, 2000.

[39] D. Fensel, S. Decker, M. Erdmann, and R. Studer. Ontobroker: Or how to enable intelligent access to the www. In *Proc. of the 11th Banff Knowledge Acquisition for Knowledge-Based System Workshop (KAW98)*, 1998.

[40] W. D. Fisher. On grouping for maximum homogeneity. *Journal of the American Statistical Association*, 53:789–798, 1958.

[41] E. Forgy. Cluster analysis of multivariate data: Efficiency versus interpretability of classification. *Biometrics*, pages 768–780, 1965.

[42] B. C. M. Fung, K. Wang, and P. S. Yu. Top-down specialization for information and privacy preservation. In *Proc. of the 21st IEEE International Conference on Data Engineering (ICDE 2005)*, Tokyo, Japan, April 2005.

[43] J. Furnkranz and P. A. Flach. Roc 'n' rule learning – towards a better understanding of covering algorithms. *Machine Learning*, 58:39–77, 2005.

[44] A. Gangemi, R. Navigli, and P. Velardi. The OntoWordNet project: Extension and axiomatization of conceptual relations in wordnet. In *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE*, volume 2888 of LNAI, pages 820–838. Springer-Verlag, Berlin, 2003.

[45] A. Gangemi, D. Pisanelli, and G. Steve. An overview of the ONIONS project: Applying ontologies to the integration of medical terminologies. *Data Knowledge Engineering*, 31(2):183–220, 1999.

[46] V. Ganti, J. Gehrke, and R. Ramakrishnan. Cactus - clustering categorical data using summaries. In *fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, page 7383. ACM Press, 1999.

[47] M. Genesereth and R Fikes. Knowledge interchange format. Technical Report Logic-92-1, Computer Science Department. Stanford University, 1992.

[48] M. R. Genesereth and N. J. Nilsson. *Logical Foundation of Artificial Intelligence*. Kauffman, Los Altos, California, 1987.

[49] A. Gómez-Pérez and V. R. Benjamins. Overview of knowledge sharing and reuse components: Ontologies and problem-solving methods. In *Proc. of the IJCAI-99 workshop on Ontologies and Problem-Solving Methods (KRR5)*, Stockholm, Sweden, Aug. 1999.

[50] M. S. GOUIDE and A. Farhat. Mining multi-level frequent itemsets under constraints. In *International Journal of Database Theory and Application*, volume 3, Dec. 2010.

[51] J. C. Gower and G. J. S. Ross. Minimum spanning rees and single-linkage cluster analysis. In *Appl. Stat.*, volume 18, pages 54–64. 1969.

[52] W. E. Grosso, H. Eriksson, R. W. Fergerson, J. H. Gennari, S. W. Tu, and M. A. Musen. Knowledge modeling at the millennium – the design and evolution of protégé-2000. In *Proc. of the 12th Workshop on Knowledge Acquisition, Modeling and Management (KAW'99)*, Banff, Canada, Oct. 1999.

[53] T. R. Gruber. A translation approach to portable ontology specifications. In *Knowledge Acquisition*, volume 5, pages 199–220. 1993.

[54] N. Guarino. Formal ontology, conceptual analysis and knowledge representation. *International Journal of Human and Computer Studies*, 43(5/6):625–640, 1995.

[55] N. Guarino. Semantic matching: Formal ontological distinctions for information organization, extraction, and integration. In M. T. Pazienza, editor, *Information Extraction: A Multidisciplinary Approach to an Emerging Information Technology*, pages 139–170. Springer Verlag, 1997.

[56] N. Guarino. Formal ontology in information systems. In *Proc. of the 1st International Conference on Formal Ontology in Information Systems (FOIS98)*, pages 3–15, Trento, Italy, June 1998. IOS Press, Amsterdam.

[57] N. Guarino and C. Welty. Ontological analysis of taxonomic relationships. In *Proceedings of International Conference on Conceptual Modeling*, pages 210–224, 2000.

[58] S. Guha, R. Rastogi, and K. Shim. Rock: A robust clustering algorithm for categorical attributes. In *Proc. of IEEE International Conference on Data Engineering (ICDE'99)*, Sydney, Australia, Mar. 1999.

[59] M. Hahsler, C. Buchta, and K. Hornik. Selective association rule generation. *Computational Statistics*, 23(2):303–315, April 2008.

[60] L. O. Hall, R. Collins, K. W. Bowyer, and R. Banfield. Error-based pruning of decision trees grown on very large data sets can work. In *International Conference on Tools for Artificial Intelligence*, pages 233–238, 2002.

[61] J. Han and Y. Fu. Attribute-oriented induction in data mining. *Advances in Knowledge Discovery and Data Mining*, 4:399–421, 1996.

[62] J. Han and Y. Fu. Mining multiple-level association rules in large databases. *IEEE Transactions on Knowledge and Data Engineering*, 11(5):798–805, 1999.

[63] P. Hansen and B. Jaumard. Minimum sum of diameters clustering. *Journal. of Classification*, 4:215–226, 1987.

[64] J. A. Hartigan. *Clustering Algorithms*. New York: John Wiley & Sons, Inc., 1975. Pages 130-142.

[65] D. M. Hawkins and G. V. Kass. Automatic interaction detection. In D.M. Hawkins, editor, *Topics in Applied Multivariate Analysis*, pages 267–302. Cambridge Univ Press, Cambridge, 1982.

[66] W. L. Henschel, A. andWoon, T. Wachter, and S. Madnick. Comparison of generality based algorithm variants for automatic taxonomy generation. In *Proceedings of the 6th International Conference on Innovations in Information Technology*, pages 160–164, 2009.

[67] A. Hotho, S. Staab, and G. Stumme. Ontologies improve text document clustering. In *Proc. of the 3rd* IEEE *International Conference on Data Mining (ICDM03)*, pages 541–544, 2003.

[68] L. Hubert. Monotone invariant clustering procedure. *Psychometrika*, 38(1):47–62, 1973.

[69] L. Hyafil and R. L. Rivest. Constructing optimal binary decision trees is np-complete. *Information Processing Letters*, 5(1):15–17, 1976.

[70] A. K. Jain and R. C. Dubes. Algorithms for clustering data. In *Prentice-Hall advance reference series*. Prentice-Hall, Upper Saddle River, NJ, 1988.

[71] A. K Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Computing Surveys*, 31(3):264–323, 1999.

[72] R. A. Jarvis and E. A. Patrick. Clustering using a similarity method based on shared near neighbors. In *IEEE Trans. on Computers*, volume C, pages 1025–1034, Aug. 1973.

[73] J. Joo, J. Zhang, Yang. J., and V. Honavar. Generating avts using ga for learning decision tree classifiers with missing data. In *Discovery Science*, pages 347–354, 2004.

[74] D. Kang, A. J. Zhang, and V. Honavar. Generation of attribute value taxonomies from data for data-driven construction of accurate and compact

classifiers. In *Proc. of the Fourth IEEE International Conference on Data Mining(ICDM)*, pages 130–137. 2004.

[75] D. Kang, A. J. Zhang, and V. Honavar. Learning concise and accurate naïve bayes classifiers from attribute value taxonomies and data. In *Proc. of the Knowledge Information System*, pages 157–179. 2006.

[76] D. K. kang and K. Sohn. Learning decision trees with taxonomy of propositionalized attributes. *Journal of Pattern Recognition*, 42(1):84–92, 2009.

[77] G. Karypis, E.-H. Han, and V. Kumar. CHAMELEON: A hierarchical clustering using dynamic modeling. *COMPUTER*, 32(8):68–75, 1999.

[78] G. V. Kass. Significance testing in automatic interaction detection. *Applied Statistics*, 24(2):178–189, 1975.

[79] G. V. Kass. An exploratory technique for investigating large quantities of categorical data. *Applied Statistics*, 29(2):119–127, 1980.

[80] K. A. Kaufman and R. S. Michalski. An adjustable rule learner for pattern discovery using the aq methodology. *Journal of Intelligent Information System*, 14(2/3):199–216, 2000.

[81] L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. Wiley series in probability and mathematical statistics. John Wiley and Sons Inc., 1990.

[82] L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. New York: John Wiley & Sons, Inc., 1990.

[83] M. R. Keeler. *Nothing to Hide*. 2000.

[84] K. Knight and S. Luk. Building a large knowledge base for machine translation. In *Proc. of American Association of Artificial Intelligence Conference (AAAI-94)*, Seattle, WA, 1994.

[85] A. A. Knopf. The audubon society field guide to north american mushrooms. New York, 1981.

[86] F. Lehmann. Machine-negotiated, ontology-based edi (electronic data interchange). In *Proc. of CIKM-94 Workshop on Electronic Commerce*. Springer Verlag, 1995.

[87] D. B. Lenat and R. V. Guha. Building large knowledge-based systems. In *Representation and Interface in the Cyc project*. Addison-Wesley, Reading, MA, 1990.

[88] T. Li and S. Anand. Automated taxonomy generation for summarizing multi-type relational datasets. In *In Proceedings of The 2008 International Conference on Data Mining (DMIN)*, Las Vegas, 2008.

[89] W. Lian, D. W. Cheung, N. Mamoulis, and S. M. Yiu. An efficient and scalable algorithm for clustering xml documents by structure. In *IEEE Transactions on Knowledge and Data Engineering*, volume 16, pages 82–96, 2004.

[90] B. Liu, Y. Ma, and C. K. Wong. Integrating classification and association rule mining. In *Proc. of the 4th International Conference on Knowledge Discovery and Data Mining*, pages 80–86, 1998.

[91] H. Liu and H. Moroda. *Feature Extraction, Construction, and Selection: A Data Mining Perspective*. Kluwer Academic Publishers, Boston, 1998.

[92] H. Liu and H. Moroda. Feature selection for knowledge discovery and data ming. In *The Kluwer International Series in Engineering and Computer*

Science, volume 454, page 161168. Kluwer Academic Publishers, Boston, 1998.

[93] K. Mahesh. Ontology development for machine translation: Ideology and methodology. Technical Report MCCS-96-292, Mexico State University, Computing Research Laboratory, 1996.

[94] M. Makrehchi and M. S. Kamel. Automatic taxonomy extraction using google and term dependency. In *IEEE/WIC/ACM International Conference on Web Intelligence*, pages 321–325, 2007.

[95] C. J. Matheus and G. Piatesky-Shapiro. An application of kefir to the analysis of healthcare information. In *Proc. of the AAAI Workshop on Knowledge Discovery in Databases*, pages 441–452, Seattle, WA, 1994.

[96] A. McCallum, R. Rosenfeld, T. Mitchell, and A. Ng. Improving text classification by shrinkage in a hierarchy of classes. In *Proc. of the fifteenth international conference on machine learning*, pages 359–367. Morgan Kaufmann, 1998.

[97] D. McGuinness. Ontological issues for knowledge-enhanced search. In N. Guarino, editor, *Formal Ontology in Information Systems*. IOS Press, 1998.

[98] D. L. McGuinness. Ontologies come of age. In D. Fensel, J. Hendler, H. Lieberman, and W. Wahlster, editors, *Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential*. MIT Press, 2002.

[99] D. L. McGuinness, R. Fikes, J. Rice, and S. Wilder. An environment for merging and testing large ontolgoies. In A. G. Cohn, F. Giunchiglia, and B. Selman, editors, *Principles of Knowledge Representation and Reasoning:*

Proc. of the seventh International Conference(KR2000). Morgan Kaufmann, San Francisco, CA, 2000.

[100] J. B. McQueen. Some methods for classification and analysis of multivariate observations. In *Proc. of the 5th Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297, Berkeley, 1967.

[101] G. W. Milligan and M. C. Cooper. An examination of procedures for determining the number of clusters in a data set. *Psychometrika*, 1985.

[102] J. N. Morgan and R. C. Messenger. Thaid- a sequential analysis program for the analysis of nominal scale dependent variables. Technical report, Survey Research Center, Institute for Social Research, Univeristy of Michigan, 1973.

[103] K. Murthy, A. T. Faruquie, and L. V. Subramaniam. Automatically generating term-frequency-induced taxonomies. In *Proceedings of the ACL*, pages 126–131, 2010.

[104] S. K. Murthy and S. Salzberg. Decision tree induction: How effective is the greedy heuristic? In *Proc. of the 1st International Conference on Knowledge Discovery and Data Mining (KDD-95)*, pages 222–227, Montreal, Canada, 1995.

[105] M. Neshati and L. Hassanabadi. Taxonomy construction using compound similarity measure. In *Lecture Notes in Computer Science*, pages 915–932, 2007.

[106] N. F. Noy and D. L. McGuinness. Ontology development 101: A guide to creating your first ontology. Technical Report KSL-01-05 and SMI-2001-0880, Stanford Knowledge Systems Laboratory and Stanford Medical Informatics, Mar. 2001.

[107] Institute of Electrical and Electronics Engineers. *IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries.* New York, NY, 1990.

[108] J. S. Park, M. S. Chen, and P. S. Yu. An effective hash-based algorithm for mining association rules. In *Proc. of the 1995 ACM SIGMOD International Conference on Management of data*, pages 175–186, San Jose, CA, May 1995.

[109] M. Pazzani, S. Mani, and W. Shankle. Beyond concise and colorful: Learning intelligible rules. In *Proc. of the Third International Conference on Knowledge Discovery and Data Mining*, 1997.

[110] F. Pereira, N. Tishby, and L. Lee. Distributional clustering of english words. In *31st Annual Meeting of the ACL*, page 183190. 1993.

[111] R. Poli. Ontology and knowledge organization. In *Proc. of 4th Conference of the International Society of Knowledge Organization (ISKO 96)*, Washington, 1996.

[112] R. C. Prati and P. A. Flach. Roccer: a roc convex hull rule learning algorithm. In *Proc. of ECML/PKDD 2004 Workshop: Advances in Inductive Rule Learning*, 2004.

[113] R. C. Prati and P. A. Flach. Roccer: an algorithm for rule learning based on roc analysis. In *Proc. of the 19th International Joint Conference on Artificial Intelligence*, 2004.

[114] R. C. Prati and P. A. Flach. Roccer: An algorithm for rule learning based on roc analysis. In *Proc. of the 19th International Joint Conference on Artificial Intelligence(IJCAI05)*, pages 823–828, Edinburgh, 2005.

[115] F. Provost and T. Fawcett. Robust classification systems for imprecise environments. In *Proc. of the 15th International Conference on Artificial Intelligence*. AAAI Press, 1998.

[116] F. Provost, T. Fawcett, and R. Kohavi. The case against accuracy estimation for comparing induction algorithms. In *Proc. of the 15th International Conference on Machine Learning*, pages 445–453. Morgan Kaufmann, 1998.

[117] K. Punera, S. Rajan, and J. Ghosh. Automatic construction of n-ary tree based taxonomies. In *Proceedings of the 6th IEEE International Conference on Data Mining Workshop*, pages 75–79, 2006.

[118] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[119] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Francisco, CA, 1993.

[120] J. R. Quinlan. Bagging, boosting, and c4.5. In *Proc. of the 13th International Conference on Artificial Intelligence*, pages 725–730, Cambridge, MA, 1996. AAAI/MIT Press.

[121] J. R. Quinlan. Improved use of continuous attributes in C4.5. *Journal of Artificial Intelligence Research*, 4:77–90, 1996.

[122] M. R. Rao. Cluster analysis and mathematical programming. *Journal of The American Statistical Association*, 66:622–626, 1971.

[123] T. R. Rothenfluh, J. H. Gennari, H. Eriksson, A. R. Puerta, S. W. Tu, and M. A. Musen. Reusable ontologies, knowledge-acquisition tools, and performance systems: PROTÉGÉ-II solutions to sisyphus-2. *International Journal of Human-Computer Studies*, (44):303–332, 1996.

[124] S. Z. Selim and M. A. Ismail. K-means-type algorithms: a generalized convergence theorem and characterization of local optimality. In *IEEE Trans. on Pattern Analysis and Machine Intelligence*, volume 6, pages 81–86, 1984.

[125] L. Setia and H. Burkhardt. Learning taxonomies in large image databases. In *Proceedings of SIGIR Workshop on Multimedia Information Retrieval*, 2007.

[126] R. Shearer and I. Horrocks. Exploiting partial information in taxonomy construction. In *Proceedings of the 8th International Semantic Web Conference*, pages 569–584, 2009.

[127] W. Shen, K. Ong, B. Mitbander, and C. Zaniolo. Mataqueries for data mining. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 375–398. AAAI/MIT press, 1996.

[128] N. Slonim and N. Tishby. Document clustering using word clusters via the information bottleneck method. In *Proc. of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*, page 208215. ACM Press, 2000.

[129] J. F. Sowa. *Knowledge Representation: Logical, Philosophical, and Computational Foundations.* Brooks Cole Publishing Co., Pacific Grove, CA, 2000.

[130] M. Steinbach, G. Karypis, and V. Kumar. A comparison of document clustering techniques. In *Proc. of World Text Mining Conference,* KDD*2000*, Boston.

[131] R. Studer, R. Benjamins, and D. Fensel. Knowledge engineering: Principles and methods. *Data and Knowledge Engineering*, (25):161–197, 1998.

[132] P. N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*. Pearson Education, Boston, 2006.

[133] Merwyn G. Taylor, Kilian Stoffel, and James A. Hendler. Ontology-based induction of high level classification rules. In *In Proceddings of the SIGMOD Dataming and Knowledge Discovery Workshop*, 1997.

[134] E. Tsui, W. M. Wang, C. F. Cheung, and A. Lau. A conceptrelationship acquisition and inference approach for hierarchical taxonomy construction from tags. *Information Processing Management*, 46(1):44–57, 2010.

[135] R. Van de Riet, H. Burg, and F. Dehne. Linguistic issues in information systems design. In N. Guarino, editor, *Formal Ontology in Information Systems*. IOS Press, 1998.

[136] G. van Heijst, A. Th. Schreiber, and B. J. Wielinga. Using explicit ontologies in KBS development. *International Journal of Human-Computer Studies /Knowledge Acquisition*, (45):183–292, 1997.

[137] F. Vasile, A. Silvescu, D-K. Kang, and V. Honavar. Tripper: An attribute value taxonomy guided rule learner. In *Proceedings of the Tenth Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, pages 55–59, Berlin, 2006.

[138] Christopher Welty and Nicola Guarino. Supporting ontological analysis of taxonomic relationships. In *Proceedings of International Conference on Conceptual Modeling*, pages 51–74, 2001.

[139] M. A. Wong and T. Lane. A kth nearest neighbor clustering procedure. In *Journal of the Royal Statistical Society*, number 45 in Series B, pages 362–368. 1983.

[140] F. Wu, J. Zhang, and V. Honavar. Proceedings of the symposium on abstraction, reformulation, and approximation (sara 2005). In *Lecture Note of Springer-Verlag*, pages 313–320, 2005.

[141] H. Yi, B. Iglesia, and V. J. Rayward-Smith. Using concept taxonomies for effective tree induction. In *International Conference of Computational Intelligence and Security (CIS)*, volume 3802 of *Lecture Notes in Computer Science*, pages 1011–1016. Springer, 2005.

[142] H. Yi and V. J. Rayward-Smith. Using a clustering algorithm for domain related ontology construction. In *Proc. of the International Conference on Know ledge Engineering and Ontology Development(KEOD)*, pages 336–341, 2009.

[143] J. Zhang and V. Honavar. Learning decision tree classifiers from attribute value taxonomies and partially specified data. In *Proc. of the Twentieth International Conference on Machine Learning (ICML),*, volume 20(2), pages 880–887. 2003.

[144] J. Zhang and Honavar V. G. Silvescu, A. Ontology-driven induction of decision trees at multiple levels of abstraction. In *Proceedings of the 5th International Symposium on Abstraction, Reformulation and Approximation*, pages 316–323, 2002.

Appendix A

DAML+OIL Overview

DAML+OIL is a semantic markup language for Web resources. It builds on earlier W3C standards such as RDF [3] and RDF Schema, and extends these languages with richer modelling primitives. DAML+OIL provides modelling primitives commonly found in frame-based languages [33].

The following sections will focus on the introduction of DAML+OIL's syntax and usage through a self-defined *Student Ontology* in the style of the annotated DAML+OIL markup language. Figure A.1 shows a simple hierarchy of *Student Ontology*, where *Student*, *Faculty*, and *Accommodation* are three main classes, linked by some meaningful relationships described by the dashed lines.

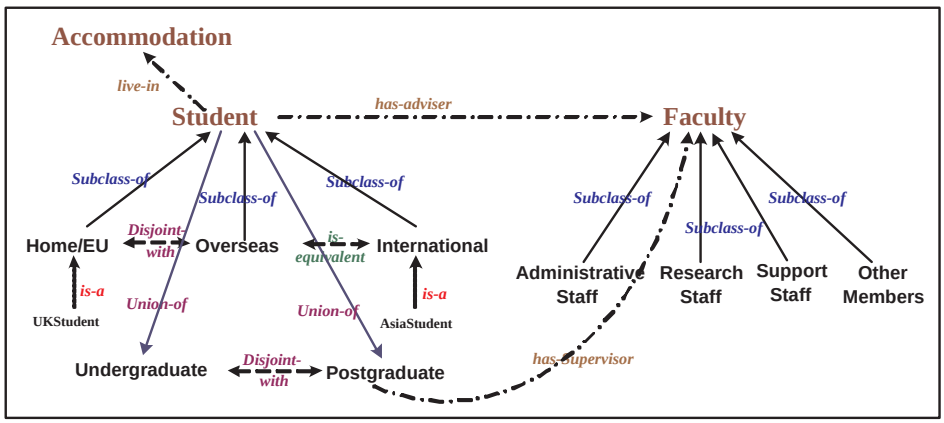


Figure A.1: Student Ontology

A.1 Namespace of DAML+OIL

A namespace in DAML+OIL refers to a schema or ontology, where terms associated with this namespace are defined. It can be represented by XML elements, since all the statements in DAML+OIL are RDF statements, which are written in XML using XML namespaces [XMLNS] and URIs(Uniform Resource Identifier). An example of the definition of a DAML+OIL namespace is shown below, starting with an XML document tags and the definitions of several entities.

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<!DOCTYPE rdf:RDF [
   <!ENTITY rdf  'http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
   <!ENTITY rdfs 'http://www.w3.org/2000/01/rdf-schema#'>
   <!ENTITY xsd  'http://www.w3.org/2000/10/XMLSchema#'>
   <!ENTITY daml 'http://www.daml.org/2001/03/daml+oil#'>
]>

<rdf:RDF
   xmlns:rdf  = "&rdf;"
   xmlns:rdfs = "&rdfs;"
   xmlns:xsd  = "&xsd;"
   xmlns:daml = "&daml;">
</rdf:RDF>
```

From the above definition, there are four different namespaces, `rdf`, `rdfs`, `xsd`, and `daml`, involved in the DAML+OIL ontology. Each of them is grouping identifiers for sets of vocabularies, shown as follows.

`rdf`: RDF specific vocabulary itself `rdfs`: RDF schema vocabulary

`xsd`: XML schema datatypes `daml`: DAML specification on the XML

All theses namespaces are defined within the RDF tag, where all the abbreviations following the ampersand(&) will be replaced with the full namespaces by the XML parser when parsing the RDF document.

A.2 Structure of DAML+OIL Ontology

A DAML+OIL ontology consists of zero or more **headers**, followed by zero or more **class** elements, **property** elements, and **instances**. We will give specific statement for each part with the example in the following section.

A.2.1 Header

There are several classes and properties defined in this header part, such as `title`, `date`, `creator`, `description`, etc. See the following definition for the detail.

```
<daml:Ontology rdf:about="">
   <dc:title>"An example of ontology"</dc:title>
   <dc:date>12 Jan. 2003</dc:date>
   <dc:creator>H Y YI</dc:creator>
   <dc:description>An case study on ontology and DAML+OIL language.
   </dc:description>
   <daml:versionInfo>V1.0<daml:versionInfo>
   <daml:imports rdf:resource="http://www.w3.org/2001/10/daml+oil"/>
</daml:Ontology>
```

In the header's definition, the `daml:Ontology` element asserts that this is an ontology; the `about` attribute will typically be empty, indicating that the subject of this assertion is this document; the `daml:versionInfo` element generally contains a string giving the information about this version, but it does not contribute to the logical meaning of the ontology.

Inside the Ontology element, using the `daml:imports` statement, we can list any other referenced DAML+OIL ontologies that apply to the current DAML+OIL resource.

A.2.2 Object and Datatype

DAML+OIL divides the world into two disjoint parts. One part, called the *Datatype* domain, consists of the values that belong to XML Schema datatypes.

Datatype values are used to help define classes, simply by including their URIs within a DAML+OIL ontology, but they are not DAML individual objects. For example, *"http://www.w3.org/2000/10/XMLSchema#decimal"* refers to the standard location for datatype decimal. The other part, called the *Object* domain, consists of objects that are considered to be members of classes described within DAML+OIL or RDF.

A.2.3 DAML+OIL Classes and Class Elements

DAML+OIL provides many classes to describe objects, for instance, **Class**, **Thing**, **Nothing**, **ObjectProperty**, etc. Here, we will introduce some important classes along with examples.

1. **Class**: In order to describe objects, it is useful to define some basic types. This is done by giving a name for a class, which is the subset of the universe which contains all the objects of that type, e.g.

```
<rdfs:Class rdf:ID="Class">
   <rdfs:label>Class</rdfs:label>
   <rdfs:comment>The class of all "object" classes.
   </rdfs:comment>
</rdfs:Class>
```

2. **Thing**: is the type of "Class", i.e., all the object X are of type "Thing". The properties, such as `unionOf` and `complementOf`, used in the following example will be introduced in the later section.

```
<Class rdf:ID="Thing">
   <rdfs:label>Thing</rdfs:label>
   <rdfs:comment>The most general (object) class in DAML. This is
                 equal to the union of any class and its complement.
   </rdfs:comment>
   <unionOf rdf:parseType="daml:collection">
      <rdfs:Class rdf:about="#Nothing"/>
      <rdfs:Class>
         <complementOf rdf:resource="#Nothing"/>
```

```
      </rdfs:Class>
   </unionOf>
</Class>
```

3. **Nothing**: is the value of "complementOf" applied to "Thing", i.e., "Nothing" is the complement of "Thing". The definition in DAML+OIL is:

```
<Class rdf:ID="Nothing">
   <rdfs:label>Nothing</rdfs:label>
   <rdfs:comment>The class with no things in it.
   </rdfs:comment>
   <complementOf rdf:resource="#Thing"/>
</Class>
```

4. **ObjectProperty**: properties that relate objects to other objects, e.g.

```
<daml:ObjectProperty rdf:ID="liveIn">
   <rdfs:domain rdf:resource="#Student"/>
   <rdfs:range rdf:resource="#Accommodation"/>
   <rdfs:comment> Students live in accommodation.
   </rdfs:comment>
</daml:ObjectProperty>

<daml:ObjectProperty rdf:ID="Upgraded">
   <rdfs:domain rdf:resource="#MPhilStudent"/>
   <rdfs:range rdf:resource="#PhDStudent"/>
   <rdfs:comment> MPhil students will become PhD students
                  after upgrading.
   </rdfs:comment>
</daml:ObjectProperty>
```

5. **DatatypeProperty**: properties that relate object properties to datatype values, e.g.

```
<daml:DatatypeProperty rdf:ID="Age">
   <rdfs:comment>Age is a DatatypeProperty whose range is xsd:decimal.
           Age is also a UniqueProperty.
   </rdfs:comment>
   <rdfs:range rdf:resource =
       "http://www.w3.org/2000/10/XMLSchema#nonNegativeInteger"/>
   <rdfs:type rdf:resource =
```

```
        "http://www.w3.org/2001/10/daml+oil#UniqueProperty"/>
  </daml:DatatypeProperty>
```

A class element, `daml:Class`, contains (part of) the definition of an object class. A class element refers to a class name and contains zero or more following elements.

1. **subClassOf**: asserts that the referred class C is a subclass of the class-expression mentioned in the property, e.g.

```
<daml:Class rdf:ID="Postgraduate">
   <rdfs:subClassOf rdf:resource="#Student"/>
   <rdfs:comment>It represents all the postgraduate students.
   </rdfs:comment>
</daml:Class>
```

2. **sameClassAs**: asserts that P is equivalent to the named class, e.g.

```
<daml:Class rdf:ID="Sophomore">
   <daml:sameClassAs rdf:resource="#2ndYearStudent"/>
</daml:Class>
```

3. **equivalentTo**: When applied to a property or a class, it has the same semantics as the **sameClassAs** element, e.g.

```
<daml:Class rdf:ID="Overseas">
   <daml:equivalentTo rdf:resource="#International"/>
</daml:Class>
```

4. **disjointWith**: asserts that referred class C is disjoint with the class expression in the property, e.g.

```
<daml:Class rdf:ID="Research">
   <rdfs:subClassOf rdf:resource="#Postgraduate"/>
   <rdfs:subClassOf rdf:resource="#Student"/>
   <daml:disjointWith rdf:resource="#Taught"/>
</daml:Class>
```

5.**oneOf**: contains a list of the objects that are its instances, e.g.

```
<daml:Class rdf:ID="Degree">
   <daml:OneOf rdf:parseType="daml:collection">
      <Degree rdf:ID="Bachelor"/>
      <Degree rdf:ID="GraduateDiploma"/>
      <Degree rdf:ID="Master"/>
      <Degree rdf:ID="MasterByResearch"/>
      <Degree rdf:ID="MPhil"/>
      <Degree rdf:ID="PhD"/>
   </daml:OneOf>
</daml:Class>
```

6. **unionOf**: defines the class that consists exactly of all the objects that belong to at least one of the class expressions from the list, e.g.

```
<daml:Class rdf:about="Student">
   <rdfs:comment> Students in a university are a union of Home/EU
                  and International students.
   </rdfs:comment>
   <daml:UnionOf rdf:parseType="daml:collection">
      <daml:Class rdf:about="#Home/EU"/>
      <daml:Class rdf:about="#Overseas"/>
   </daml:UnionOf>
</daml:Class>
```

7. **disjointUnionOf**: asserts that all of the classes defined by the class expressions of a disjointUnionOf element must be pairwise disjoint, and their union must be equal to the referred-to class, e.g.

```
<daml:Class rdf:about="Postgraduate">
   <rdfs:subClassOf rdf:resource="#Student"/>
   <rdfs:comment>Each postgraduate is either a taught or a research student.
   </rdfs:comment>
   <daml:disjointUnionOf rdf:parseType="daml:collection">
      <daml:Class rdf:about="#Research"/>
      <daml:Class rdf:about="#Taught"/>
   </daml:disjointUnionOf>
</daml:Class>
```

8. **intersectionOf**: defines the class that consists of exactly all the objects that are common to all class expressions from the list, e.g.

```
<daml:Class rdf:ID="PhD">
   <rdfs:subClassOf rdf:resource="#Postgraduate"/>
   <rdfs:subClassOf rdf:resource="#Student"/>
   <rdfs:comment> A PhD student must have been upgraded from MPhil.
   </rdfs:comment>
   <daml:intersectionOf rdf:parseType="daml:collection">
      <daml:Class rdf:about="#MPhil"/>
      <daml:Restriction>
         <daml:onProperty rdf:resource="#hasUpgraded"/>
      </daml:Restriction>
   </daml:intersectionOf>
</daml:Class>
```

9.**complementOf**: defines the class that consists of exactly all the objects that do not belong to the class expression, e.g.

```
<daml:Class rdf:ID="Undergraduates">
   <rdfs:subClassOf rdf:resource="#Student"/>
   <rdfs:comment> Undergraduate student must not be a postgraduate student.
   </rdfs:comment>
   <daml:complementOf rdf:resource="#Postgraduates"/>
</daml:Class>
```

A.2.4 DAML+OIL Property Restrictions

A property restriction is a special kind of class expression. It implicitly defines an anonymous class, namely the class of all objects that satisfy the restriction. So all the restrictions are enclosed in a special `subClassOf` element.

There are two kinds of restrictions: ObjectRestriction and DatatypeRestriction, working on object and datatype properties respectively.

A `daml:Restriction` element contains an `daml:onProperty` element, which refers to a property name, and one or more of the following elements.

1. **toClass**: defines the class of objects X for which it hold that if the pair (X, Y) is an instance of P, then Y is an instance of the class expression or datatype, e.g.

```
<daml:Class rdf:ID="Overseas">
   <rdfs:Comment>It presents the generic terms about overseas student.
   </rdfs:Comment>
   <rdfs:subClassOf rdf:resource="#Student"/>
   <rdfs:subClassOf>
      <daml:Restriction>
         <daml:onProperty rdf:resource="#hasNationality"/>
         <daml:toClass rdf:resource="#NonEurope"/>
      </daml:Restriction>
   </rdfs:subClassOf>
</daml:Class>
```

2. **hasValue**: contains an individual object or a datatype value. If we call the instance Y, then this property defines the class of object X for which (X,Y) is an instance of P, e.g.

```
<daml:Class rdf:ID="TaughtMasterStudent">
   <rdfs:Comment>It presents the generic terms about the taught
                 student who pursues a masters degree.
   </rdfs:Comment>
   <rdfs:subClassOf rdf:resource="#Postgraduate"/>
   <rdfs:subClassOf rdf:resource="#Student"/>
   <rdfs:subClassOf>
      <daml:Restriction>
         <daml:onProperty rdf:resource="#DegreeWillBeGained"/>
         <daml:hasValue rdf:resource="#Master"/>
      </daml:Restriction>
   </daml:subClassOf>
</daml:Class>
```

3. **hasClass**: defines the class of object X for which there is at least one instance Y of the class expression or datatype such that (X, Y) is an instance of P, e.g.

```
<daml:Class rdf:ID="PartTimeStudent">
   <rdfs:Comment>Part-time students may work part-time.
   </rdfs:Comment>
```

```
   <rdfs:subClassOf rdf:resource="#Student"/>
   <rdfs:subClassOf>
      <daml:Restriction>
         <daml:onProperty rdf:resource="#hasJob"/>
         <daml:hasClass rdf:resource="PartTimeJob"/>
      </daml:Restriction>
   </rdfs:subClassOf>
</daml:Class>
```

4. **cardinality**: defines the class of all objects that have exactly N distinct values for the property P, e.g.

```
<daml:Class rdf:ID="Research">
   <rdfs:subClassOf rdf:resource="#Postgraduate"/>
   <rdfs:subClassOf rdf:resource="#Student"/>
   <rdfs:subClassOf>
      <daml:Restriction daml:cardinality="2">
         <daml:onProperty rdf:resourcee="#hasSupervisor"/>
      </daml:Restriction>
   </rdfs:subClassOf>
</daml:Class>
```

5. **maxCardinality**: defines the class of all objects that have at most N distinct values for the property P, e.g.

```
<daml:Class rdf:ID="Student">
   <rdfs:subClassOf>
      <daml:Restriction daml:maxCardinality="1">
         <daml:onProperty rdf:resource="#hasStudentID"/>
      </daml:Restriction>
   </rdfs:subClassOf>
</daml:Class>
```

6. **minCardinality**: defines the class of all objects that have at least N distinct values for the property P, e.g.

```
<daml:Class rdf:ID="Student">
   <rdfs:Comment>Each student should have at least one Email address.
   </rdfs:Comment>
   <rdfs:subClassOf>
```

```
      <daml:Restriction daml:minCardinality="1">
         <daml:onProperty rdf:resource="#hasEmailAddress"/>
      </daml:Restriction>
   </rdfs:subClassOf>
</daml:Class>
```

7. **cardinalityQ**: defines the class of all objects that have exactly N distinct values for the property P that are instances of the class expression or datatype (and possibly other values not belonging to the class expression or datatype), e.g.

```
<daml:Class rdf:ID="ResearchPostgraduate">
   <rdfs:subClassOf rdf:resource="#Postgraduate"/>
   <rdfs:subClassOf rdf:resource="#Student"/>
   <rdfs:subClassOf>
      <daml:Restriction daml:cardinalityQ="2">
         <daml:onProperty rdf:resource="#hasSupervisor"/>
         <daml:hasClassQ rdf:resource="#Supervisor"/>
      </daml:Restriction>
   </rdfs:subClassOf>
</daml:Class>
```

8. **maxCardinalityQ** This property defines the class of all objects that have at most N distinct values for the property P that are instances of the class expression or datatype (and possibly other values not belonging to the class expression or datatype).

```
<daml:Class rdf:ID="StudentID">
   <rdfs:subClassOf>
      <daml:Restriction daml:maxCardinalityQ="1">
         <daml:onProperty rdf:resource="#hasIdentify"/>
         <daml:hasClassQ rdf:resource="Student"/>
      </daml:Restriction>
   </rdfs:subClassOf>
</daml:Class>
```

9. **minCardinalityQ**: defines the class of all objects that have at least N distinct values for the property P that are instances of the class expression or datatype (and possibly other values not belonging to the class expression or datatype).

```
<daml:Class rdf:ID="PartTimeStudent">
   <rdfs:subClassOf rdf:resource="#Student"/>
   <rdfs:subClassOf>
      <daml:Restriction daml:minCardinalityQ="1">
         <daml:onProperty rdf:resource="#hasJob"/>
         <daml:hasClassQ rdf:resource="PartTimeJob"/>
      </daml:Restriction>
   </rdfs:subClassOf>
```

A.2.5 DAML+OIL Property Elements

A DAML+OIL property element contains zero or more following properties.

1. **subPropertyOf**: states that every pair (X, Y) that is an instance of P is also an instance of the named property, e.g.

```
<daml:ObjectProperty rdf:ID="hasSon">
   <rdfs:subPropertyOf rdf:resource="#hasChild"/>
   <rdfs:domain rdf:resource="#Human"/>
   <rdfs:range rdf:resource="#Male"/>
</daml:ObjectProperty>
```

2. **domain**: asserts that the property P only applies to instance of the class expression of the property. It is already used in the above example.

3. **range**: asserts that the property P only assumes values that are instance of the class expression of the property. See the example of `subPropertyOf`.

4. **samePropertyAs**: asserts that P is equivalent to the named property, which allows us to establish synonymy, e.g.

```
<daml:ObjectProperty rdf:ID="hasSupervisor">
   <daml:samePropertyAs rdf:resource="#hasAdvisor"/>
   <rdfs:domain rdf:resource="#Student"/>
   <rdfs:range rdf:resource="#Supervisor"/>
</daml: ObjectProperty>
```

5. **inverseOf**: asserts that P is the inverse relation of the named property, then the pair (Y, X) is an instance of the named property.

```
<daml:ObjectProperty rdf:ID="hasChild">
   <daml:inverseOf rdf:resource="#hasParent"/>
   <rdfs:domain rdf:resource="#Human"/>
   <rdfs:range rdf:resource="#Human"/>
</daml: ObjectProperty>
```

6. **TransitiveProperty**: is the subclass of ObjectProperty. This asserts that P is transitive if the pair(X, Y) is an instance of P, and the pair(Y, Z) is an instance of P, then the pair(X, Z) is also an instance of P. e.g.

```
<daml:TransitiveProperty rdf:ID="hasAncestor">
   <rdfs:domain rdf:resource="#Human"/>
   <rdfs:range rdf:resource="#Human"/>
</daml:TransitiveProperty>
```

7. **UniqueProperty**: asserts that P can only have one (unique) value Y for each instance X, i.e., each subject uniquely identifies the object (value) of the property. e.g.

```
<daml:UniqueProperty rdf:ID="hasFather">
   <rdfs:subPropertyOf rdf:resource="#hasParent"/>
   <rdfs:domain rdf:resource="#Human"/>
   <rdfs:range rdf:resource="#Male"/>
</daml:UniqueProperty>
```

8. **UnambigousProperty**: is a subclass of ObjectProperty, which asserts that an instance Y can only be the value of P for a single instance X, that means a property whose object uniquely identifies its subject. So the inverse property of a UniqueProperty is always an UnambigousProperty and vice versa. e.g.

```
<daml:UnambigousProperty rdf:ID="isFather">
   <daml:inverseOf rdf:resource="#hasFather"/>
   <rdfs:domain rdf:resource="#Male"/>
   <rdfs:range rdf:resource="#Human"/>
</daml:UnambigousProperty>
```

A.2.6 Instances

Instances of both classes (i.e., objects) and of properties (i.e., pairs) are written in RDF and RDF Schema syntax, e.g.

```
<Home/EU rdf:ID="UKStudent">
   <rdfs:label>UKStudent</rdfs:label>
   <rdfs:comment>UKStudent is Home/EU student.</rdfs:comment>
</Home/EU>

<International rdf:ID="AsiaStudent">
   <rdfs:label>AsiaStudent</rdfs:label>
   <rdfs:comment>AsiaStudent is international student.</rdfs:comment>
</International>
```

The following two elements are used when stating two instances are same or distinct.

1. **sameIndividualAs**: is used to state that instance objects are the same. Note that equivalentTo can be also used here, but the former is preferred since it is exclusive for the identical instance object, e.g.

```
<Postgraduate rdf:ID="HY Yi">
   <rdfs:label>HY Yi</rdfs:label>
   <rdfs:comment>HY Yi is a postgraduate student.</rdfs:comment>
   <hasIDNumber><xsd:String rdf:value="a261008"/></hasIDNumber>
</Postgraduate>

<Postgraduate rdf:ID="Ellen Yi">
   <rdfs:label>Ellen</rdfs:label>
   <rdfs:comment>Ellen is HY Yi's English name.</rdfs:comment>
   <daml:sameIndividualAs rdf:resource="#HY Yi"/>
</Postgraduate>
```

2. **differentIndividualFrom**: is used to state that instance objects are distinct, e.g.

```
<Student rdf:ID="Adam">
   <rdfs:label>Adam</rdfs:label>
   <rdfs:comment>Adam is a student.</rdfs:comment>
   <hasIDNumber><xsd:String rdf:value="a123456"/></hasIDNumber>
   <daml:differentIndividualFrom rdf:resource="#Ellen"/>
</Student>
```

Appendix B

Data Description

B.1 Adult Data Set

The *Adult* data set in UCI data repository is extracted from the 1994 and 1995 current population surveys conducted by the U.S. Census Bureau. It contains information on various individuals, such as education, marital status, occupation, etc. Particularly, the income information is simply recorded as either more than \$50K or less than \$50K.

Given this information, some data mining tasks can be carried out, for example, clustering or classification. In this thesis, we are interested in finding the classifiers for prediction, which could identify whether a person's earning exceeds \$50K according to all the other information. This prediction is very practical for some government agencies, e.g. the taxation bureau to detect fraudulent tax refund claims.

B.1.1 Attributes

Table B.1 lists all the 14 attributes as well as their possible values. Attribute Fnlwgt[1] is a weight determined by considering people's demographic characteristics. *Class presents the person's income.

B.1.2 Distribution

The original *Adult* data is randomly split into train and test data, accounting for 2/3 and 1/3 respectively. The distribution of instances for the target class is shown in table B.2.

B.1.3 Missing and Unreliable Data

The attribute Workclass, Occupation, and Native-country contain missing data, marked by "?" in records. There are a total of 3620 records contain missing values.

Besides the missing values exist in data, following unusual cases also catch our attention.

1. If the value of attribute Relationship is either "*Husband*" or "*Wife*", there is a clear one-to-one correlation between these two values and the ones of attribute Sex. That is, "*Husband*" should correspond to "*Male*", and "*Wife*" corresponds to "*Female*". But there are two records have the opposite correspondence, which may be due to recording mistake.

2. There are 148 records, with 0.49% occurrences in the training data, have the maximum value of "99999" for attribute Capital-gain. If observing the distribution of the values of this attribute, we find there is a great gap between the second largest value, which is 41310, and the maximum value. So this is

[1]See *ftp://ftp.ics.uci.edu/pub/machine-learning-databases/adult/adult.names* for detail.

Table B.1: The Attributes of Adult Data Set

Attribute	**Values**	**No. Distinct Values**
Age	In the range [17,90]	72
Fnlwgt	In the range [13492,1490400]	20,263
Capital-gain	In the range [0,99999]	118
Capital-loss	In the range [0,4356]	90
Hours-per-week	In the range [1,99]	94
Workclass	Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Private, Without-pay.	7
Education	Preschool, 1st-4th, 5th-6th, 7th-8th, 9th, 10th, 11th, 12th, HS-grad, Assoc-acdm, Assoc-voc, Some-college, Prof-school, Bachelors, Masters, Doctorate.	16
Marital-status	Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse.	7
Occupation	Tech-support, Craft-repair, Handlers-cleaners, Exec-managerial, Prof-specialty, Other-service, Sales, Machine-op-inspct, Transport-moving, Armed-Forces, Admclerical, Priv-house-serv, Farming-fishing, Protective-serv.	14
Relationship	Wife, Own-child, Husband, Not-in-family, Unmarried, Other-relative.	6
Race	Asian-Pac-Islander, Amer-Indian-Eskimo, White, Black, Other.	5
Sex	Male, Female.	2
Native-country	United-States, Cambodia, England, Germany, Puerto-Rico, Outlying-US(Guam-USVI-etc), Greece, Canada, Japan, India, South, China, Cuba, Iran, Honduras, Poland, Jamaica, Peru, Portugal, Ireland, Italy, Dominican-Republic, Philippines, Mexico, Ecuador, Taiwan, Haiti, Guatemala, Nicaragua, Scotland, Yugoslavia, El-Salvador, France, Trinadad&Tobago, Laos, Columbia, Thailand, Vietnam, Hong, Hungary, Holand-Netherlands.	41
**Class* (Target Class)	"$<= 50K$", "$> 50K$".	2

Table B.2: Target Class Distribution

Data set	Target Class	%	Records
Train (with missing)	$\leq 50K$	75.91	24,720
Train (with missing)	$> 50K$	24.09	7,841
Test (with missing)	$\leq 50K$	76.38	12,435
Test (with missing)	$> 50K$	23.62	3,846

an unusual value, and it could be an assigned value for missing data, or the maximum possible value representing a real value which is equal or greater to it.

3. There are 68 records, with 0.23% occurrences in the training data, have the value less than 20 for attribute Hours-per-week and 0 for Capital-gain, but belong to class ">50K".

All these records could be considered as unreliable data or outliers, and need to be checked with the domain expert. However, for the experiments undertaken in this thesis, these records and attribute values will still be used, but the records with missing values will be removed.

B.2 Mushroom Data Set

The *Mushroom* data set was obtained by mushroom records drawn from the Audubon Society Field Guide to North American Mushrooms [85], and was donated to the UCI repository by Jeff Schlimmer.

This data set includes descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms in the Agaricus and Lepiota Family (pp. 500-525). Each species is identified as definitely edible, definitely poisonous, or of unknown edibility and not recommended. This latter class was combined with the poisonous

one.

B.2.1 Distribution

There are totally 8,124 instances in Mushroom data set, and table B.3 lists the instance distribution in two classes.

Table B.3: Target Class Distribution

Target Class	%	Records
edible	51.8	4208
poisonous	48.2	3916

B.2.2 Attributes

Table B.4 lists all the 22 attributes as well as their possible values. *Class presents whether it is edible or poisonous.

B.2.3 Missing Data

Only one of the attributes, *Stalk-root*, contains missing values. There are 2,480 missing values for this attribute, accounting for 30.5% of the total values.

Table B.4: The Attributes of Mushroom Data Set

Attribute	**Values**	**No. Distinct Values**
cap-shape	bell, conical, convex, flat, knobbed, sunken	6
cap-surface	fibrous, grooves, scaly, smooth	4
cap-color	brown, buff, cinnamon, gray, green, pink, purple red, white, yellow	10
bruises?	bruises, no	2
odour	almond, anise, creosote, fishy, foul, musty, none pungent, spicy	9
gill-attachment	attached, descending, free, notched	4
gill-spacing	close, crowded, distant	3
gill-size	broad, narrow	2
gill-color	black, brown, buff, chocolate, gray, green, orange pink, purple, red, white, yellow	12
stalk-shape	enlarging, tapering	2
stalk-root	bulbous, club, cup, equal, rhizomorphs rooted, missing=?	7
stalk-surface-above-ring	fibrous, scaly, silky, smooth	4
stalk-surface-below-ring	fibrous, scaly, silky, smooth	4
stalk-color-above-ring	brown, buff, cinnamon, gray, orange, pink red, white, yellow	9
stalk-color-below-ring	brown, buff, cinnamon, gray, orange, pink red, white, yellow	9
veil-type	partial, universal	2
veil-color	brown, orange, white, yellow	4
ring-number	none, one, two	3
ring-type	cobwebby, evanescent, flaring, large, none pendant, sheathing, zone	8
spore-print-color	black, brown, buff, chocolate, green, orange purple, white, yellow	9
population	abundant, clustered, numerous, scattered several, solitary	6
habitat	grasses, leaves, meadows, paths, urban waste, woods	7
**Class* (Target Class)	edible, poisonous.	2