Inducing Default Knowledge Bases

Anna Hopfgarten
c98aho@cs.umu.se

March 11, 2003

Abstract

This thesis explores the notion of generating default rule-bases in place of decision trees. It will be shown that the resulting knowledge is more compact, perspicuous and may be constrained by prior knowledge. An algorithm will be presented that is based on recording examples within a lattice structure. A simple traversal algorithm is able to generate a set of default rules that generalize over the training examples. The approach is incremental and able to incorporate prior knowledge. The results of this algorithm will be compared and contrasted with established decision tree induction techniques.

Contents

1 Introduction **5**

1.1 Organization of this Thesis 6

2 Background **7**

2.1 Decision Trees 7

2.1.1 The Decision Tree Learning Algorithm 8

2.1.2 Performance 9

2.1.3 Choosing features to test - Information Theory 10

2.1.4 Decision Tree Example 11

2.1.5 See5 14

2.2 Default Reasoning 15

2.3 Lattices 17

3 A Lattice-based Approach **20**

3.1 The Positive Case 21

3.1.1 Example 22

3.2 The Full Case 24

3.2.1 Lattice data structure algorithm 24

3.2.2 Sort and Insert 24

3.2.3 Generalization Process 25

3.2.4 Cleansing Process 26

3.2.5 The Knowledge Base 26

3.2.6 Querying the Lattice 27

3.2.7 Example 28
3.3 System Properties 29
3.3.1 Incremental 29
3.3.2 Incorporating Prior Knowledge 30

4 Experiments **31**
4.1 Waiting for a table 31
4.2 Generalization and Specification 32
4.3 House Votes example 33

5 Discussion **34**
5.1 Extending the Implementation 34
5.1.1 Completing the Knowledge-base 34
5.1.2 Using Information Theory 35
5.2 Comparison to Previous Work 35
5.2.1 See5 35
5.2.2 Version space learning 36
5.2.3 Rough Sets 36
5.3 Mechanisms and policies 37
5.4 Future Work 37
5.5 Conclusions 38
5.6 Acknowledgments 39

6 Appendix **40**
6.1 Glossary 40
6.2 Lattice showing the pizza-example 44

Bibliography **45**

List of Figures

2.1 Decision Tree Data Structure 8
2.2 Decision Tree Example 14

3.1 Lattice Data Structure 21
3.2 Lattice Example 28

6.1 Pizza Example Lattice 44

List of Tables

2.1 Decision Tree Algorithm 9
2.2 Will Wait Training Set 11

3.1 Lattice Data Structure Algorithm 24

4.1 Example of General versus Specific Knowledge 32

Chapter 1

Introduction

To data-mine is to apply computationally intensive analysis to large stores of data, looking for useful patterns[EN99]. Though data mining spans many techniques from statistical analysis to association rule mining, a significant portion of these techniques generate knowledge that *classifies*[1] records (or tuples) as being, or not being members of a *concept*. More specifically consider that we have an *object* o with *features* $f_{i_1}, ..., f_{i_m}$ all drawn from a set of possible features F. These features are qualities that an object possesses. Given this we may speak of a *class membership function* that decides whether an object is or is not a member of a class based on the *presence* or *absence* of features. Data mining in this context is looking for such a classification function.

The classical approach to this problem is the induction of decision trees[Qui90]. Decision tree induction is a mature technique to generate knowledge, for the classification of objects, that has found its way into several commercial implementations[RR]. This said, decision trees have drawbacks. Of concern here we note: 1.) it is difficult to incorporate prior knowledge into decision tree induction; 2.) it is difficult to obtain human readable knowledge from the decision tree induction; 3.) the knowledge generated from decision tree induction is not minimal.

Default reasoning, with default rules, is a different approach to representing a class membership function, and also the approach used in this thesis. Default reasoning is

[1] See the glossary, in section 6.1, for definitions of the terms used in this thesis

similar to how humans reason with common sense. General beliefs about the world are kept that are true unless an exception is found to contradict that belief. Default rules, as opposed to decision trees, have the virtues of giving more perspicuous knowledge and being designed to handle uncertainty.

1.1 Organization of this Thesis

In chapter 2 some necessary background concepts are covered. This includes an in-depth description of decision tree techniques and some mathematical properties of lattices. Various default semantics are covered as well. In chapter 3 we cover our lattice based approach to default rule induction. The first section describes the basic techniques of this approach. The second section considers the restricted case where only the presence (not absence) of features may be mentioned over an object. The third section deals with the full case, which is a generalization that allows for both presence and absence of features to be noted for an object. In chapter 4, results from experiments are presented and contrasted with the performance of an established decision tree induction system. In chapter 5 problems with this approach and their remedies are discussed. Chapter 5 also includes comparisons to previous work in this area. Finally future work is discussed and conclusions presented.

Chapter 2

Background

2.1 Decision Trees

As stated before, decision trees is one branch in machine learning and data mining that has made its way into the heart of practical implementations. The idea is to build a tree, as in figure 2.1, for each concept to test an objects membership in. An *internal node* n_k in the tree tests the presence or absence of an object o's feature from the set of features F. Branching from a node is equivalent to establishing the absence or presence of a feature and when a *leaf node* is reached, the tree will state whether or not the object o was a member of the concept or not.

The decision tree induction technique can also be amended so as to handle non-boolean features, such as the feature 'color' - which is not merely present or absent - it can take on any of the values *red, green, blue, yellow*, etc. Continuous features are handled in the same way, by approximating the continuous feature values to a discrete number of value-sets or by using comparison operators $(>, \leq, etc.)$ over numeric or other total order domains.

The next section will give a deeper description of the decision tree induction algorithm.

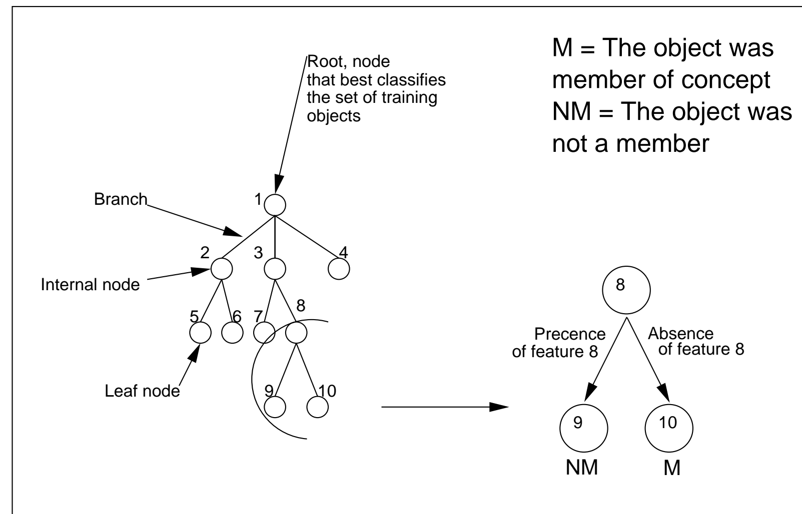

Figure 2.1: Decision tree data structure - testing concept membership for objects

2.1.1 The Decision Tree Learning Algorithm

The induction algorithm in table 2.1.1 from [RN95], builds the decision tree, one tree-node at a time, one feature at a time, from the root to the leaves, by choosing the feature that best classifies[1] the training examples that need to be considered at a given node. The procedure to choose the best feature is designed to minimize the depth of the final tree and when all examples are properly classified, a decision tree covering those examples is returned.

There are three states which makes the algorithm terminate. The first state, if the training set is empty and the second state, if all objects in the training set have the same concept membership, will make the algorithm return the given default membership of the concept. The third state handles the case when there are no more features to ground[2], but there still exist example objects to classify. In that case, the algorithm simply calculates the majority class among the examples and uses that as the classification for the leaf node where those examples fall under.

Otherwise, the decision tree induction algorithm determines the feature f_i with

[1] This corresponds to the function CHOOSE-FEATURE in the decision tree learning algorithm, which is described further in section 2.1.3

[2] To ground a feature simply means that the value of the feature is set to a specific value from the possible value set

```
function DECISION-TREE-LEARNING(examples, features, default) returns a decision
  tree
inputs:
    examples, set of example objects
    features, set of features
    default, default value for membership in concept
if examples is empty then return default
else if all examples have the same classification then return the classification
else if features is empty then return MAJORITY-VALUE(examples)
else
    best ← CHOOSE-FEATURE(features, examples)
    tree ← a new decision tree with root test best
    for each value v_i of best do
        examples_i ← {elements of examples with best = v_i}
        subtree ←
          DECISION-TREE-LEARNING(examples_i, features − best, MAJORITY-
          VALUE(examples))
        add a branch to tree with label v_i and subtree subtree
    end
    return tree
```

Table 2.1: Decision Tree Algorithm

the highest information gain on the training set, see section 2.1.3 below. Then it uses this feature f_i as the *root node* of the tree, creates a branch for each of the values f_i can have and, for each branch, it creates a subtree, by grounding one feature at a time, using the subset of the training set where the value of f_i corresponds to that of the branch. Each leaf created in this process will state any object's concept membership that have the same value of its features as the branch values from the leaf up to the root.

2.1.2 Performance

The performance of a decision tree is measured by dividing the given set of examples into two disjoint sets, a *training set* and a *test set.* The decision tree is built using the training set and then the accuracy of the generated tree is checked against the test set. If multiple tests like this are performed, using different disjoint training and test sets, both randomly selected, in the end the average prediction accuracy can be determined and used as a performance measurement.

2.1.3 Choosing features to test - Information Theory

Information theory is a mathematical model that is used to choose the best feature f_i from a set of features F, given the probability set for each feature in F. The probability set for a feature f_i is the probability for each value that f_i can take on. For example, if the possible values are either absent or present, think of a value set to be $\{absent = 0.5, present = 0.5\}$.

Definition 1 *If $v_{1..n}$ are values of an object o's feature and $P(v_{1..n})$ are the probabilities for those features, then the **Information content**, I, of that feature is* $I(P(v_i), ..., P(v_n)) = \sum_{i=1}^{n}(-P(v_i)log_2P(v_i))$

Information Theory measures information content (entropy) I of a feature in **bits**, where one bit is sufficient to represent a concept membership state that one has no information on the outcome of. The information content function uses the possible values of an objects feature to determine the information held in that feature. The information content for a feature, along with the distribution of the values for that feature, is used to determine the information gain of that feature.

Definition 2 *If f is a feature from a set of features F for an object o and v are the number of values that that feature can hold, then the **Remainder** of that feature, R(f), is the sum of all information contents where f is given one of its values at a time, multiplied with the proportion of objects that have that value for f.*
$R(f) = \sum_{i=1}^{v}(\frac{p_i+n_i}{p+n}I(\frac{p_i}{p_i+n_i}, \frac{n_i}{p_i+n_i}))$ [3]

Definition 3 *If p is the number of positive example objects and n is the number of negative example objects then the **Information Gain**, G(f), is the gain of information received when the feature f's value is given.*
$G(f) = I(\frac{p}{p+n}, \frac{n}{p+n}) - R(f)$

The information gain $G(f_i)$ of a feature f_i is measured by taking the original information content and subtracting the remainder for f_i. That is, subtracting the

[3] Here p_i and n_i are the number of positive and negative examples left in the example set that have the value v_i assigned to feature f

sum of the information content for each value of f_i times the distribution for that value among the examples given.

The feature that has the largest information gain is the feature that will be placed as the next node in the resulting decision tree.

2.1.4 Decision Tree Example

This following text walks through an example [RN95] showing how decision trees work. The concept over which the decision tree will test membership is if a person should wait for a table at a restaurant and there are ten features recorded for every example. From the table of training examples below a decision tree will be built, using the decision tree algorithm from section 2.1.1 and information theory explained in the previous section.

Ex	Features										Concept
	Alt	Bar	Fri/Sat	Hungry	Patrons	Price	Rain	Reserve	Type	Estimate	*Wait?*
X_1	Y	N	N	Y	Some	$$$	N	Y	French	0-10	**Yes**
X_2	Y	N	N	Y	Full	$	N	N	Thai	30-60	**No**
X_3	N	Y	N	N	Some	$	N	N	Burger	0-10	**Yes**
X_4	Y	N	Y	Y	Full	$	N	N	Thai	10-30	**Yes**
X_5	Y	N	Y	N	Full	$$$	N	Y	French	>60	**No**
X_6	N	Y	N	Y	Some	$$	Y	Y	Italian	0-10	**Yes**
X_7	N	Y	N	N	None	$	Y	N	Burger	0-10	**No**
X_8	N	N	N	Y	Some	$$	Y	Y	Thai	0-10	**Yes**
X_9	N	Y	Y	N	Full	$	Y	N	Burger	>60	**No**
X_{10}	Y	Y	Y	Y	Full	$$$	N	Y	Italian	10-30	**No**
X_{11}	N	N	N	N	None	$	N	N	Thai	0-10	**No**
X_{12}	Y	Y	Y	Y	Full	$	N	N	Burger	30-60	**Yes**

Table 2.2: The *Will wait?* training set

The algorithm will, given the examples $X_1, ..., X_{12}$, pick one feature at a time as the next node in the decision tree starting with the feature that best classifies these examples - the feature with largest gain. In this case the *Patrons* feature will be the one with largest gain looking at all twelve examples. By studying the gain calculations for all the features, this will be obvious[4].

[4] The full calculation of the gain for feature Patrons is given and will give a guide how to calculate the other features' gain

CHOOSE BEST FEATURE

$n = number\ of\ negative\ examples$

$p = number\ of\ positive\ examples$

$Information\ Content\ given\ no\ feature\ values:$

$I(\frac{p}{p+n}, \frac{n}{p+n}) = -\frac{6}{12}log_2\frac{6}{12} - \frac{6}{12}log_2\frac{6}{12} = 1$

$To\ choose\ the\ best\ feature,\ the\ Gain\ for\ all\ features\ is\ calculated:$

$Gain(Patrons) = I(\frac{p}{p+n}, \frac{n}{p+n}) - Remainder(Patrons) =$

$1 - \sum_{i=1}^{3}(\frac{p_i+n_i}{p+n}I(\frac{p_i}{p_i+n_i}, \frac{n_i}{p_i+n_i})) =$

$1 - (\frac{2}{12}I(0,1) + \frac{4}{12}I(1,0) + \frac{6}{12}I(\frac{2}{6}, \frac{4}{6})) =$

$1 - (0 + 0 + \frac{1}{2}I(\frac{1}{3}, \frac{2}{3})) =$

$1 - \frac{1}{2}(-\frac{1}{3}log_2\frac{1}{3} - \frac{2}{3}log_2\frac{2}{3})[log_2 x = \frac{log_{10}x}{log_{10}2}] \approx 0.54$

$---$

$Gain(Alternative) = 0$

The gain for the feature *Alternative* will be zero, since each possible value of *Alternative* gives a remaining set of examples with the same number of positive and negative classifications.

$Gain(Bar) = 0$

$Gain(Fri/Sat) \approx 0.021$

$Gain(Hungry) \approx 0.20$

$Gain(Price) \approx 0.20$

$Gain(Rain) = 0$

$Gain(Reservation) \approx 0.021$

$Gain(Type) = 0$

$Gain(Estimate) \approx 0.21$

This calculation determines that in fact *Patrons* is the best feature to pick as the

root node. The same calculation is made for the examples that remains to be properly classified, in this case all those examples that have the value `full` for the feature *Patrons*, X_2, X_4, X_5, X_9, X_10 *and* X_12. The other values for *Patrons* will result in leaf nodes, properly marked with the classification of the examples in the training set which had the corresponding value on the feature *Patrons*. The value `Some` will result in a membership while `None` will not.

When there are multiple features with equal gain, one policy for picking the best feature could be to pick the one with the smallest value domain and the first one calculated. In this example, the feature *Hungry* fits that policy[5] and is placed as the first internal node in the decision tree following the branch that corresponds to the value `full` for the *Patrons* feature.

Now, four examples still remain to be classified. Recalculating the gain for the remaining features results in the feature *Type* having the largest gain, but still not properly classifying two of the four remaining examples. In spite of that, *Type* is the best classifying feature, and therefore placed as an internal node after following the branch that corresponds to the value `No` for feature *Hungry*. The gain calculation must continue though, to cover those last two examples.

Fri/Sat is the last feature that needs to be placed as an internal node in the decision tree. It properly classifies the remaining two examples and creates the two last leaf nodes. The final decision tree can be viewed in figure 2.2.

A classical way of representing the decision tree in figure 2.2 is with eight rules, one for each path from the root to a leaf node. Four rules stating the positive outcomes and four rules stating the negative outcomes. A virtue of default reasoning is seen in this light, since the same information can be represented with a smaller set of rules. In fact only five rules are needed to represent the decision tree, independent of the choice for default class.

[5] *Price*, *Reservation*, *Type* and *Estimate* are features with equal gain as *Hungry*, but *Hungry* has the smallest value domain and was the first feature to be calculated

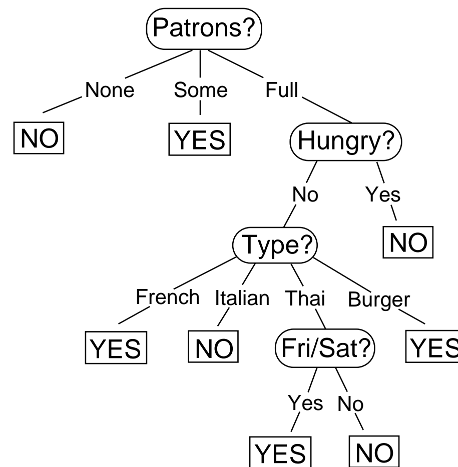

Figure 2.2: Decision tree generated with the DT-algorithm from the *Wait?* example

2.1.5 See5

See5[RR] is a complete system and a commercial product that uses decision trees, among other techniques, to produce different class membership functions - “to predict outcomes for future situations as an aid to decision-making”.

Since See5, in its current state, is a full commercial product, there are limited possibilities to gain insight into how the system actually works. One can merely observe the effects and results of using the product and, with the sparse information on their web site, conclude that there is no way to fully explain the parts that lies beyond that of decision trees.

In addition to decision trees, See5 offers the possibility to generate sets of if-then rules from the decision trees. The rule sets claim to give a noticeably higher predictive accuracy and are usually more compact then the information held in the corresponding decision tree. Rule sets as classifiers in See5 use some notion of default reasoning. A rule set contain a default class rule, which is used whenever an example that should be classified does not fall under the other rules in the rule set. The default class is determined by the majority class of the training examples, to cover as many examples as possible. However, the notion of default reasoning is not used in its full capabilities. See5 has a default class but could extend rule sets to hold specificity override, which would lead to fewer rules.

The policy for choosing between two rules with conflicting predicted classes is based on voting and rules promoting different amounts of influence on the concept membership. The amount of influence a rule has is based on the predictive accuracy every rule has associated with it, the greater predictive accuracy, the greater influence it has on the resulting classification.

See5 uses a technique called *adaptive boosting*. The technique is used to minimize errors and by that increase the predictive accuracy of the class membership function. Adaptive boosting is based on the work by Freund and Schapire[FS96] and combines generated classifiers to minimize the influence of the mistakes that the induction process is likely to make. Though working beyond the basic decision tree technique, this restricted way of trying to incorporate prior knowledge[6] into the induction process can be seen as clever. It attempts to correct previously erroneous predictions, that is, predictions from previously induced decision tree classifiers.

2.2 Default Reasoning

Imagine a world where drawn conclusions would never be retracted, where a person never would change his or her opinion, where a decision could never be withdrawn. A monotonic world. This is a very hard world to imagine. But if an agent were forced to live by such rules, the agent would rarely, if ever, jump to conclusions. The agent would most likely be paralyzed and not make any conclusions at all.

The reasoning we use, however, is non-monotonic. That is, if and when new facts are found about the world then the previously derived assumptions are retracted or revised in order to fully comprise the new facts.

Take this example for instance, a person, let us call him Ted, finds himself enjoying a ham and tomato sandwich on his way to work. This makes him come to the conclusion that he likes ham and tomato sandwiches and that he will eat a ham and tomato sandwich on his way to work from now on. But one day when he had his ham and tomato sandwich on his way to work he slips and drops the sandwich on the sidewalk. Mr. Ted is also known to be very picky about what he eats so he had to

[6]Prior knowledge since the features for the classifier has been previously learned

revise his decision to eat a ham and tomato sandwich every time he was on his way to work, and add this new information into the decision process. He would eat a ham and tomato sandwich every time he was on his way to work - unless the sandwich fell on to the ground and got dirty. This example can be extended further in saying, for example, that his roommate had teased him about being so picky with his food and had also been eating up all their food, so he was especially hungry this day, then he might in fact have continued with eating that sandwich, even if it fell on to the ground.

Default reasoning captures that nuance of the world as it derive default assumptions from typicality statements and use these assumptions to reason over the world, minimizing the information needed - filling in gaps of knowledge, similarly to how humans reason with common sense. In the example with Mr. Ted above a default reasoning system might induce the following set of rules for eating a sandwich:

holding a sandwich $\wedge$ *walking to work* $\rightarrow$ *eat the sandwich*
dropped the sandwich on the ground $\wedge$ *now holding a sandwich* $\wedge$ *walking to work* $\rightarrow$ *do not eat the sandwich*

The existing default semantics are 1.) P-entailment by Adams followed by the equivalent Epsilon-semantics by Pearl [Pea88]; 2.) Circumscription by McCarthy [McC86]; 3.) Reiter's Default Logic [Pea88]; 4.) Maximum entropy by Pearl, Goldszmidt and Morris [PGM93] [Bou99]; 5.) System-Z by Pearl and Goldszmidt [GP95]; and 6.) Lexicographic entailment by Lehmann [Leh95].

This work has a simple semantic of specificity override. This simplicity can be seen as both an advantage and a disadvantage. In some cases one might desire to extend the approach taken in this thesis towards handling reasoning with multiple concepts, combining the knowledge bases from different concepts. In that case, the semantics of this work has to be extended.

All the semantics mentioned above will behave the same way under the limited assumptions of my semantic. Introducing any of them in an extended version of this work will be possible.

2.3 Lattices

In chapter 3 we shall see that lattices play a critical role in the approach presented in this thesis. In this section the basic definitions and properties of lattices will be presented[Gri99].

The lattice-based approach uses the notion of a lattice as a skeleton for how training examples should be placed and sorted. The lattice helps the objects to order themselves to simplify the process of generating conditional default rules.

Definition 4 *If A, B are sets, a* ***relation*** *from A to B is any subset of $A \times B$. Subsets of $A \times A$ are called relations on A.*

A relation from A to B is any subset of $A \times B$, for example a relation on the set of all integers $\mathbf{Z}$ can be the $\leq$ relation. In the approach presented in this thesis, the notion of relations refer to *subnodes* and *supernodes*[7] used to order nodes in the lattice.

Definition 5 *A relation $\mathcal{R}$ on a set A is called* ***reflexive*** *if for all $x \in A$, $(x, x) \in \mathcal{R}$.*

A reflexive relation on A states that A is related to itself.

Definition 6 *A relation $\mathcal{R}$ on set A is called* ***symmetric*** *if $(x, y) \in \mathcal{R} \Rightarrow (y, x) \in \mathcal{R}$, for all $x, y \in A$.*

A symmetric relation says that if a is related to b, then b is also related to a. In the lattice data structure this property does not hold by simply looking at one relation.

Definition 7 *For a set A, a relation $\mathcal{R}$ on A is called* ***transitive*** *if for all $x, y, z \in A$, $(x, y), (y, z) \in \mathcal{R} \Rightarrow (x, z) \in \mathcal{R}$. (So if x "is related to" y, and y "is related to" z, we want x "related to" z, with y playing the role of "intermediary").*

[7]The definition of subnodes and supernodes can be found in the glossary or in chapter 3

Definition 8 *Given a relation $\mathcal{R}$ on a set A, $\mathcal{R}$ is called* ***antisymmetric*** *if for all $a, b \in A$, $(a\mathcal{R}b$ and $b\mathcal{R}a) \Rightarrow a = b$. (Here the only way we can both have a "related to" b and b "related to" a is if a and b are one and the same attribute from A.)*

The fact that both subnodes and supernodes exist in the lattice, the lattice could be seen as not having the antisymmetric property. This is an incorrect observation though, since subnodes and supernodes are essentially different relations.

Definition 9 *A relation $\mathcal{R}$ on a set A is called a* ***partial order****, or a* ***partial ordering relation****, if $\mathcal{R}$ is reflexive, antisymmetric and transitive.*

Definition 10 *If $(A, \mathcal{R})$ is a poset*[8]*, then an element $x \in A$ is called a* ***maximal element*** *of A if for all $a \in A, a \neq x \Rightarrow x \neg\mathcal{R}a$. An element $y \in A$ is called a* ***minimal element*** *of A if whenever $b \in A$ and $b \neq y$, then $b \neg\mathcal{R}y$.*

Theorem 1 *If $(A, \mathcal{R})$ is a poset and A is finite, then A has both a maximal and a minimal element.*
Proof: Let $a_1 \in A$. If there is no element $a \in A$ where $a \neq a_1$ and $a_1\mathcal{R}a$, then a_1 is maximal. Otherwise there is no element $a_2 \in A$ with $a_1 \neq a_2$ and $a_1\mathcal{R}a_2$. If no element $a \in A$, $a \neq a_2$, satisfies $a_2\mathcal{R}a$, then a_2 is maximal. Otherwise we can find $a_3 \in A$ so that $a_3 \neq a_2, a_3 \neq a_1$, while $a_1\mathcal{R}a_2$ and $a_2\mathcal{R}a_3$. Continuing in this manner, since A is finite, we get to go to an element $a_n \in A$ with $a_n \neg\mathcal{R}a$ for all $a \in A$ where $a \neq a_n$, so a_n is maximal. The proof for a minimal element follows in a similar way.

Definition 11 *If $(A, \mathcal{R})$ is a poset, then an element $x \in A$ is called a* ***least element*** *if $x\mathcal{R}a$ for all $a \in A$. Element $y \in A$ is called a* ***greatest element*** *if $a\mathcal{R}y$ for all $a \in A$.*

Theorem 2 *If the poset $(A, \mathcal{R})$ has a greatest (least) element, then that element is unique.*
Proof: Suppose that $x, y \in A$ and that both are greatest elements. Since x is a greatest element, $y\mathcal{R}x$. Likewise, $x\mathcal{R}y$ because y is a greatest element. As $\mathcal{R}$ is antisymmetric, it follows that $x = y$. The proof for the least element is similar.

[8] A poset is a set of partial order relations

Definition 12 *Let $(A, \mathcal{R})$ be a poset with $B \subseteq A$. An element $x \in A$ is called a* ***lower bound*** *of B if $x\mathcal{R}b$ for all $b \in B$. Likewise, an element $y \in A$ is called an* ***upper bound*** *of B if $b\mathcal{R}y$ for all $b \in B$.*
An element $x' \in A$ is called a ***greatest lower bound*** *(glb) of B if it is a lower bound of B and if for all other lower bounds x'' of B we have $x''\mathcal{R}x'$. Similarly $y' \in A$ is a* ***least upper bound*** *(lub) of B if it is an upper bound of B and if $y'\mathcal{R}y''$ for all other upper bounds y'' of B.*

The lattice data structure maintains a greatest lower bound as the empty set ϕ, since the structure needs to maintain the property of having a default concept membership state when no information is given.

Theorem 3 *If $(A, \mathcal{R})$ is a poset and $B \subseteq A$, then B has at most one lub (glb).*
Proof: Suppose that $x', y' \in A$ and that both are least upper bounds. Since x' is a least upper bound, and y' is an upper bound, $x'\mathcal{R}y'$. Likewise, $y'\mathcal{R}x'$ because y' is a least upper bound and x' is an upper bound. Since $\mathcal{R}$ is antisymmetric, it follows that $x = y$. The proof for the greatest lower bound is similar.

Definition 13 *The poset $(A, \mathcal{R})$ is called a* ***lattice*** *if for all $x, y \in A$ the elements lub$\{x, y\}$ and glb$\{x, y\}$ both exist in A.*

Chapter 3

A Lattice-based Approach

The approach to default rule induction taken in this thesis, is to record classification examples in a lattice data structure, see fig 3.1. From this lattice data structure a set of rules will be generated that fully explain the training set and generalize to unseen cases.

This lattice structure incorporates all features in the feature set F. Certainly we do not need to maintain structure of $2^{|F|}$ nodes to represent this lattice, rather, a node represents an example *object* from a training set. The *content* of a node is the set of observed features of the corresponding object. The *sign*(+ or -) of the node states whether the object is a member of the classification concept or not. Finally we keep on each node an indication of whether a rule is needed to *cover* that node or not.

Definition 14 *If two nodes n_i and n_j exists in the lattice and a partial ordering relation $n_i \mathcal{R} n_j$ holds between them, then n_j is said to be a **supernode** to n_i and n_i is said to be a **subnode** to n_j.*

An example, if $n_i = \{a, b\}$ and $n_j = \{a, b, c\}$, then n_i is a subnode to n_j and n_j is a supernode to n_i.

Definition 15 *A node n in the lattice is **covered** if all subnodes S_n agree on sign (all subnodes have the same classification as n) or if the node is rule-generating.*

When the first example is given to the lattice the *minimal element* and the *maximal element* will be generated and as more example objects are entered into the system the maximal element will grow with the increase of the possible features.

An insert will place a node n in the lattice so that it is connected to all its subnodes and supernodes. When the system has properly placed the node n into the lattice it will continue to propagate its influence on its supernodes and make sure that those nodes are properly covered.

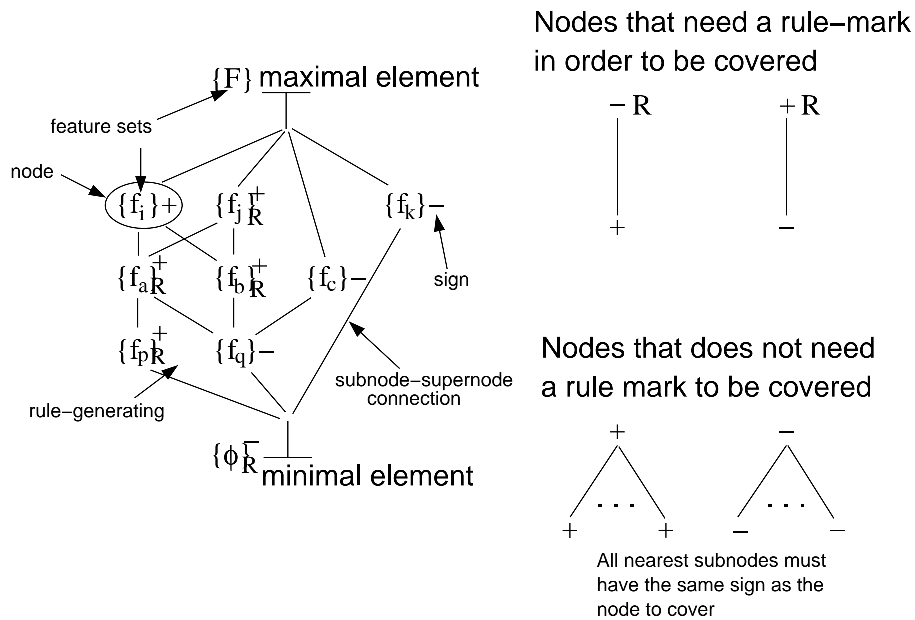

Figure 3.1: The lattice data structure and the notion of coverage

Before the full algorithm is presented, a special case will be considered that does not allow for the representation of the absence of features.

3.1 The Positive Case

As said, the positive case is not able to represent the absence of features, it can only state the presence of features for objects in the lattice. This case can be seen as a subset of the full case described later in this chapter. It can be used as a separate system, and is, without exceptions, faster than the full case. Thus, if the properties

of the domain allow the use of the positive case, it is preferred to do so.

3.1.1 Example

There may be a natural set of application domains that are restricted enough to be represented by only presence of features. Some ideas are:

- Compatible medicines, which combinations are safe and which are a potential hazard?
- A personalized coupon, checking buying habits.
- Combining friends at a social event.

Let us consider the rather silly example of what combinations of ingredients make a pizza. Consider the feature set F to be {*cheese, tomato-sauce, ham, pineapple, shrimp, mushroom, anchovy, blueberry*}. The target is to determine if any subset of F is a pizza or not.

A pizza with no ingredients does not count as a pizza, nor does one with only tomato-sauce on it, any combination of ingredients with blueberry would not make a pizza and only some combinations with anchovy makes a pizza.

The original knowledge base KB consists of one rule, $\{\} \rightarrow false$, which claims that a pizza with nothing on it is not a pizza. Here, the system is fed with the following training set[1]:

[1] Parts of this training set is actually a subset of the pizzas available at the Taormina restaurant in Umeå, Sweden

tomato-sauce is not a pizza,
cheese, tomato-sauce is a pizza,
cheese, tomato-sauce, ham is a pizza, *
cheese, tomato-sauce, ham, pineapple is a pizza,
cheese, tomato-sauce, ham, mushroom is a pizza,
cheese, tomato-sauce, ham, shrimp is a pizza,
cheese, tomato-sauce, shrimp is a pizza,
cheese, tomato-sauce, pineapple is a pizza,
cheese, tomato-sauce, ham, pineapple, mushroom is a pizza,
cheese, tomato-sauce, ham, shrimp, mushroom is a pizza,
cheese, tomato-sauce, ham, shrimp, pineapple is a pizza,
cheese, tomato-sauce, blueberry is not a pizza,
cheese, tomato-sauce, anchovy is not a pizza, *
cheese, tomato-sauce, anchovy, ham is not a pizza, *
cheese, tomato-sauce, anchovy, shrimp is a pizza

This training set generates the following rules for testing membership in the concept pizza:

1. {*cheese, tomato-sauce, blueberry*} $\rightarrow false$
2. {*cheese, tomato-sauce, anchovy, ham*} $\rightarrow false$
3. {*cheese, tomato-sauce, anchovy, shrimp*} $\rightarrow true$
4. {*cheese, tomato-sauce, anchovy*} $\rightarrow false$
5. {*cheese, tomato-sauce*} $\rightarrow true$
6. {} $\rightarrow false$

The knowledge base consisting of these six rules can now properly classify all input-examples and will try to answer any other example object given to it. Rule number 2 in the knowledge base might seem unnecessary to some. But if we look at the second rule again - in the context of the * marked examples in the training set, the rule is necessary to properly classify the *cheese, tomato-sauce, anchovy, ham* pizza example. The full lattice with all the pizza's added into it can be viewed in figure 6.1 in the appendix.

3.2 The Full Case

The full case can handle both the presence and absence of features. That is, you can state whether an object o has, or does not have, a feature f_i.

3.2.1 Lattice data structure algorithm

function CONSTRUCT-LATTICE(*examples, default*) **returns** a conditional knowledge base
inputs:
    examples, set of example object nodes with corresponding features
    default, default value for membership in concept
if *examples* is empty **then return** *default*
else
    for each $example_i$ of *examples*
        $node_i$ ← CREATE-NODE($example_i$)
        location ← SORT-AND-INSERT-INTO-LATTICE($node_i$)
        PROPAGATE-INFLUENCE(*location*, $node_i$)
        if $node_i$ is not *covered*, *Cover* it.
        discovered-nodes ← DISCOVER($node_i$)
        if *discovered-nodes*
          CONSTRUCT-LATTICE(*discovered-nodes*, *default*)
          REMOVE all *discovered nodes* that are not rule-generating.
    When no more input, GENERATE-THE-KB.

Table 3.1: Lattice Data Structure Algorithm

3.2.2 Sort and Insert

In the algorithm from section 3.2.1, a *sort* of a node n_i with the feature set f_i, into the lattice will determine the *nearest subnodes* and *nearest supernodes* of n_i. A supernode to n_i is a node that have a feature set which is a proper superset of f_i and a subnode to n_i is a node that have a feature set which is a proper subset of f_i. A *nearest supernode* to n_i has the additional condition of not having a subnode n_j with a feature set f_j that is a proper superset of f_i. The same condition applies for *nearest subnodes*, with the only difference that no supernode of a subnode of n_i can have a feature set that is a proper subset of f_i. When all subnodes and supernodes to n_i have been found, an insert of n_i into the lattice is executed, that is, n_i is connected

with all its supernodes and subnodes.

SORT-AND-INSERT(*node*)
supernodes ← *All nearest supernodes to node*
subnodes ← *All nearest subnodes to node*
for each $subnode_i$ *in subnodes*
  connect $subnode_i$ *as a subnode to node*
  connect node as a supernode to $subnode_i$
for each $supernode_i$ *in supernodes*
  connect $supernode_i$ *as a supernode to node*
  connect node as a subnode to $supernode_i$

After the node n_i is inserted n_i is likely to have affected the cover-status of its supernodes, in fact, n_i's supernodes must each be be asserted to be properly covered, which is done in the PROPAGATE-INFLUENCE step.

PROPAGATE-INFLUENCE(*node, supernodes*)
for each $supernode_i$ *in supernodes*
  if $supernode_i$ *is not covered, cover it*

3.2.3 Generalization Process

Definition 16 *If a node n in the lattice has a set of features F_n (present and absent), it has a **partial complement** in node m iff $F_n = F_m$ and $n \neq m$.*

Definition 17 *A node n is said to be **discovered** if it holds the resulting feature set from a meet operation between two partially complementing nodes that agree on sign.*

A generalization process will now be initiated[2]. The inserted node will be compared with all its *partially complementing nodes* in the lattice. If a pair of partially complementing nodes are found, a meet-operation[3] between these two nodes will be

[2]Note that in the positive case no generalization is performed
[3]The *meet* between two nodes n and m is defined as an intersection of the features sets for those nodes ($f_n \cap f_m$), and can be seen as finding the greatest lower bound for f_n and f_m

performed. As a result a *discovered node* will be inserted into the lattice. As the discovery process continues, generalization of the input knowledge is performed. The newly discovered nodes will cover at least the nodes it was generated from - cutting down representation in half.

3.2.4 Cleansing Process

When all discovered nodes have been inserted into the lattice, all discovered nodes that are not rule-generating are removed from the lattice. This process can be seen as a cleansing process from unnecessary information in the lattice. The original knowledge in the lattice will not be affected since the nodes that are removed are only discovered nodes.

When all inserts and discoveries have been properly taken care of and the example set is empty, a final traversal through the lattice is made to generate and return the knowledge base.

3.2.5 The Knowledge Base

The knowledge base is the set of rules that are returned after all the examples have been inserted and after all discovered nodes have been properly generated. This resulting set of rules can be used within any logical reasoning system using default reasoning if one wishes to go beyond the single concept classification approach taken in this thesis.

In some problems, or rather in some specially configured training sets, the regularly generated knowledge base was proven by experiments to not hold the desired property of being minimal. The approach taken in this thesis that focuses on removing irrelevant features by their occurrence in both present and absent forms, has the desired, though insufficient, property of being justified with common sense. However, not surprisingly, the generalization process needs additional policies in order to be satisfactory.

One such policy has been to minimize the generated knowledge base itself, when properly returned. The idea is to keep those rules that, in a greedy search approach,

cover all the training examples and seem to result in the least number of rules. More specific, the rules that cover the greatest number of examples with the same classification are desired over others. Rules that cover examples with different sign than the rule itself are considered low priority rules, and are only kept if they cover some example that has not been covered earlier. In most cases this collapses the knowledge base into something that resembles a minimal knowledge base, given the chosen policies.

How well the knowledge base manage to generalize is, without exceptions, in the end always dependent on the quality of the observations fed to the system. The system can perform well if consistent observations are given to it, but needs additional policies to perform outstanding results. This will be discussed further below section 5.1.

3.2.6 Querying the Lattice

When querying the lattice there are two states in which it can position itself. Either sleep mode or full mode. The full mode is when all examples, that were used when building the lattice, still remain as nodes in the lattice. In this mode the examples are used as knowledge in themselves. In sleep mode all nodes that are not rule generating are removed from the lattice. This is equivalent to querying the knowledge base that is returned after building the lattice.

The lattice gives answers to concept membership in three categories. If the example was a member of the concept, if the example was not a member of the concept or if it is uncertain if the example is a member or not.

Depending on the mode the lattice is in, confirmation on whether or not the answer comes from an actual example or from the generalized knowledge is given. The answer to the query will be *Certainly True* or *Certainly False* if it is confirmed that the decision is based on a previously entered example. If the decision is based on generalized knowledge the answer will be *Likely True*, *Likely False*, *Possibly True* or *Possibly False* based on which rule(s) that cover the query example. If multiple rules with different outcomes cover the query example then the system will state that

the answer is *Undecided*.

3.2.7 Example

This example will make a thorough walk-through of all the steps in the process of building the lattice data structure.

The chosen example tests the membership in the concept of *Will I go out tonight?*. The lattice data structure is fed with a training set of three examples that matches simplified explanations for why a person might go out. For the sake of simplicity we will get reacquainted with Mr. Ted from section 2.2, who, by the way, does not have perfect vision. All the steps in the example can be viewed in figure 3.2.

Training set:

Once Mr. Ted went out when he had time, money and friends who joined him but he had no contacts lenses.

Once Mr. Ted went out when he had time, money, friends who joined him and contacts lenses.

Once Mr. Ted went out when he had time and money but no friends who joined him and no contacts lenses.

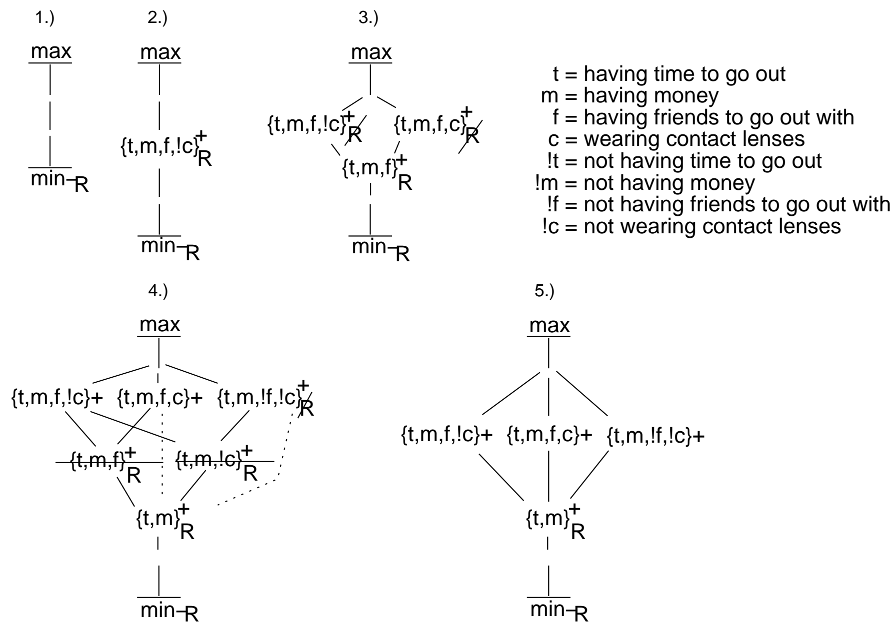

Figure 3.2: Lattice showing the *Will I go out tonight?* concept

Taking one example at a time and inserting it into the lattice, initially empty as in step 1 in figure 3.2, will result in the lattice below step 5. The first insert is shown in step 2, the inserted node became a rule generating node, since its nearest subnodes (the glb) did not agree on sign. After inserting the second example from the training set, shown in step 3, the lattice can start to generalize. The inserted examples are partially complementing and they agree on sign, so therefore a discovered node gets inserted into the lattice. The discovered node does not have the *contact lenses* feature and since it is a subset of both examples, it will cover both those - hence the removal of the rule generating mark on both example nodes.

If the training set only held these examples, the lattice would now generate a knowledge base that says that Mr. Ted would go out if we knew that he had money, time or friends. And, if we did not know if he had money, time or friends he would stay home.

The insertion of the third example in the lattice generates two new discovered nodes, shown in step 4, since the third example is partially complementing to both previously entered examples. The first and the second discovered node will be removed, those are the ones with a line crossed over them in step 4. The reason is the third discovered node, which is a subnode to both of the others. Since that node covers both, the rule markings will be removed and those nodes are now seen as unnecessary.

In step 5 the lattice shows its final state and from this two rules are generated. The first rule says that if no information is known about the properties for a night out then Mr. Ted will stay home. The second rule says that if Mr. Ted has time and money he will go out.

3.3 System Properties

3.3.1 Incremental

The lattice data structure holds an incremental property. That is, the addition of information can be done incrementally. A large set of training data can always be

followed by another large set of data, without having to regenerate any knowledge from the previous training set. This results in a positive effect on the performance of the lattice data structure when new information is known and must be added into the knowledge base.

3.3.2 Incorporating Prior Knowledge

The system that has been presented in this thesis can incorporate prior knowledge, it has been designed to leverage on other systems. For instance, the system can take rule-sets generated from See5[RR] and incorporate them into the lattice before adding more examples or before querying it.

There is, however, no restriction on the form of the prior knowledge.

Chapter 4

Experiments

4.1 Waiting for a table

The example from [RN95], previously shown in section 2.1.4, is a simple example of 12 objects where it is given if the object is a member of the concept in question or not. The concept is if one should wait for a table in a restaurant. If an object is a member of that concept it simply means that the outcome of the question "Will I wait for a table?" is *Yes*. If the object is not a member of the concept to wait for a table at a restaurant, then the answer to the question is *No*.

This example generated the decision tree shown in figure 2.2 when using the decision tree algorithm presented earlier in section 2.1.1. See5 however, does not generate the same tree. It picks the *Hungry* feature as the root and then it stops there, generating an 25% error in its predictive accuracy - in the training data alone. The same is true for See5's rule sets that generate three rules, including the default class.

The lattice data structure generates 4 rules, but is able to cover all given examples in the training data. And when run with the prior knowledge from See5 the system generates 5 rules, in order to cover those examples that See5 fails to classify.

4.2 Generalization and Specification

In the table below the 16 examples that make up the training set are displayed. The training set is run on both the See5 system and the lattice.

Features				Concept
¬a	¬b	¬c	¬d	**Yes**
¬a	¬b	¬c	d	**Yes**
¬a	¬b	c	¬d	**Yes**
¬a	¬b	c	d	**Yes**
¬a	b	¬c	¬d	**Yes**
¬a	b	¬c	d	**Yes**
¬a	b	c	¬d	**Yes**
¬a	b	c	d	**Yes**
a	¬b	¬c	¬d	**Yes**
a	¬b	¬c	d	**Yes**
a	¬b	c	¬d	**Yes**
a	¬b	c	d	**Yes**
a	b	¬c	¬d	**No**
a	b	¬c	d	**No**
a	b	c	¬d	**No**
a	b	c	d	**Yes**

Table 4.1: Example of General versus Specific Knowledge

See5 produces, with rule sets, for this distribution of examples, four rules including the default class:

1. Default class - Yes
2. If b = no → Yes
3. If a = no → Yes
4. If a = yes and b = yes → No

This proves that See5 cannot handle these types of problems and it is easy to see that any extension of this problem would brake See5's predictive capability.

The lattice generates three rules. Two exception rules, $\{a, b\} \rightarrow false$ and $\{a, b, c, d\} \rightarrow true$, and then the default membership rule, $\{\} \rightarrow true$. This representation of knowledge for these types of problems certainly seems to be the best one.

With the rules generated from See5 on this training set, the lattice based system still produces the same set of rules.

4.3 House Votes example

This example is taken from a machine learning test bench database [Rec84]. It is a database with 1984 United Stated Congressional Voting Records. The classifications are Republican or Democrat and all features are boolean. There are 400 training examples where 150 examples are used for the training set and the rest of the sample is used for the test set.

See5 generates on average three rules plus the default class. It fails to properly classify 2 examples from the training set and 16 from the test set, a total of 18 erroneously classified examples.

The lattice produces 12 rules after minimization, which properly classifies all examples from the training set, and it fails to classify 17 examples from the test set.

Chapter 5

Discussion

5.1 Extending the Implementation

5.1.1 Completing the Knowledge-base

A desirable property for the system would be if it was able to fully classify all possible examples under given set of features. A claim could then be made towards having a complete knowledge base, a knowledge base able to answer any query posted to it.

A question is, for this approach, if it is really possible to have a complete system without needing to generate a full lattice, covering all the possible feature combinations for an object, and then minimizing it with the use of default reasoning. The system can however, be complete without giving the correct answers to any given query. It seems likely that if one always requires the correct answer, then the full evidence base would be needed.

The system presented in this work uses default reasoning, which is designed to handle uncertainty. Default reasoning should be used when incomplete knowledge must be a basis for decision making. It is not clear if the system is in fact complete, but there seems to be an indication towards that. While running tests and experiments the answer "undecided" rarely, in the end never, came up. This would be a favorable result to have and must be explored further.

5.1.2 Using Information Theory

Information theory in combination with decision trees has proved to be a successful approach when a class membership function for single concepts is needed. That is however not a guarantee that it would be an appropriate policy in the lattice based approach. But, intuition says that, as an additional policy within the lattice, it could be used to guide generalization. This policy incorporated in the lattice based approach would give the system the benefits decision trees have, but the policy cannot be justified in the same way as for the policy of removing irrelevant attributes based on their occurrence in both present and absent form.

5.2 Comparison to Previous Work

5.2.1 See5

Though it is difficult to obtain human readable knowledge from the DT induction, See5, with rule sets, actually generate human readable knowledge. Or at least knowledge more perspicuous than decision trees. See5 as a system does perform well in the real world, where this lattice based approach at this point can make no such claim. However, there is one case, one category of problems where my system always does better than See5. That is, in the case of general patterns with a set of exceptions, which was presented in section 4.2.

Ultimately, these kinds of problems will brake See5's predictive accuracy since it chooses to ignore the exceptions and treats them as just exceptions - where See5 is allowed to err. The aim of See5 is to generalize, so one might say that excluding exceptions is the intended approach, and therefore not a drawback. However, depending on the goal or the target group for the system, high predictive accuracy may be required and in those target areas the lattice based system can claim to perform better than See5.

5.2.2 Version space learning

A way to learn more general logical representations is using version space learning [Mit78]. Version space learning is based on having a hypothesis set where inconsistent hypothesis are removed from the version space - reducing the number of available hypothesis that could be correct. The version space can be defined since a generalization/specification ordering exists between the hypothesis in the hypothesis set. From this ordering two boundary sets can be defined, a most general boundary, the G-set, and a most specific boundary, the S-set. Every hypothesis between these two sets is guaranteed to be consistent with the examples.

The concept of version spaces have two major deficiencies 1.) there exist no way of handling noise and insufficient attributes in the input 2.) if unlimited disjunction is allowed the S-set will always contain the disjunction of the positive examples and the G-set will always contain the negation of the disjunction of the negative examples.

5.2.3 Rough Sets

The rough set theory was designed as a tool to deal with uncertain and vague knowledge in AI applications, and it is all about ways to classify objects in order to reason about the world.

The central theory in rough sets is that of defining a relation between a subset of all attributes for a set of objects that are equal. Or more precise, given only that subset of attributes, one cannot determine the concept in which an object belongs.

In [MS96] rough set theory is used to test objects for membership in a set of classes by inducing default rules. They use the rough set theory to determine which set of attributes that are irrelevant to determine the concept membership, and generate default rules from the other attributes.

They have one advantage over the system presented in this thesis, they can decide membership within a set of concepts whereas the approach presented in this thesis only states if the object to classify is a member of the concept or not. Of course, one can always use the lattice based approach multiple times on one object in order to decide membership within a set of concepts, but the rough set approach seem to

simplify this procedure.

However, the approach they are using requires the same set of features for every object to classify, or generate rules from. That is, they need all example objects to have a predefined set of features (though this is true one might say that they can try to detach the rough set approach from that restriction since they allow for uncertain and vague information in the input objects). In defining what set of allowed features an input object can hold, the lattice based approach has the advantage, since it does not impose any restriction on the feature set to its input objects.

5.3 Mechanisms and policies

In this work the mechanism consists of three major parts, 1.) The lattice consisting of a *least upper bound* and a *greatest lower bound* for the entire example set 2.) the *insert* operation which inserts an object into the lattice 3.) the *delete* operation which removes nodes from the lattice.

The approach taken in this thesis has also been chosen as to give a clear distinction between mechanism and policy. The mechanisms are the lattice itself, the insert operation and the delete operation. The policies of the system lies within the discovery process and the rule generating process, where the generalization of knowledge is made - where the system is learning.

If the machinery is separated from the policy one can later on easily improve or alter the system.

The See5 system is, as stated earlier, a complete system and a commercial product. As such, I presume, it is difficult to picture the distinction between policy and mechanism - a clear disadvantage. It was impossible, with its current release, to strip off all chosen policies and test on the decision tree technique alone.

5.4 Future Work

Since the intention of this implementation was foremost to confirm the ideas presented, it lacks some of the features one might require. First of all the system is

written in Lisp[ALU], a relatively slow programming language performance wise, used because of its advantages when testing ideas.

Though one can blame Lisp for some portion of the deficient performance, not all can be said to come from the use of Lisp. Reconstructions to some portions of the algorithm presented in section 3.2.1 might prove to be efficient and work done in [AKBLN88] can be explored further.

Future work might also include finding an additional policy to generalize knowledge in the lattice, maybe among those presented in this thesis.

Moreover, there are one additional issue that need to be addressed before this work can be considered done. The system in its present state is very sensitive to noise in the training data. This is not a desirable property to have, therefore some way to handle noise must be implemented before the system can be used in the big leagues.

5.5 Conclusions

Comparing default rules and decision trees gives favor towards default rules. Default rules are 1) more compact than decision trees; 2) more human readable than decision trees; and 3) more sophisticated than See5 rule sets[1].

The lattice based approach taken in this thesis generates default rules from data and have the desired properties of being able to update the knowledge base incrementally and being able to incorporate prior knowledge.

Decision trees have been around for quite a while now and a lot of research effort have been put into the techniques surrounding it. Not surprisingly, this has led to results. In spite of all that, results from this work indicate that there exist a way to improve the performance with the use of default reasoning. In the combination of lattice data structures and default reasoning paths still remains unexplored. The initial results are promising, even though they are not fully satisfactory, and certainly gives you a taste for more.

[1] If See5 were to encompass default reasoning into their system, the representational cost for their rules could be greatly decreased

5.6 Acknowledgments

This paper would not have made it to its current form without the help from my supervisor, Michael Minock, the support from my family - my mother, my father, Barbro, Oskar, Jessica, Andrea, Anja, Helena, Christer, Emilia, Catarina, Erik, Robin, Evelina, Cecilia and Gustav, my aunt Berith, my friends Maria Hansson, Elin Kröger Nygren, Marcus Björklund and Kane Neman. A thought also goes to all of you that gave me carrots to run for and I would like to thank all those working at the department of computing science that have helped me finish.

Chapter 6

Appendix

6.1 Glossary

- **Classify** - Determine whether or not an object is a member of a given concept.

- **Class membership function** - A class membership function is a function that decides whether an object is or is not a member of a class based on the features the object possess.

- **Concept** - The class to test for membership in. The concept of waiting for a table in a restaurant is one example, another concept is a pizza, a third and final example is the concept of going out to a bar.

- **Object** - In this context, an object is an example that has a set of features connected to it. An object can be a pizza with a set of ingredients as features. Or, for instance, an object can be a state that determines if a person is going out to a bar or not, where the features are the set of facts that describe the state.

- **Node** - An object inserted into the lattice.

- **Lattice** - The lattice structure used in this thesis to hold example objects for determining concept membership. Consists of a set of nodes, possibly empty, and a minimal element and a maximal element - between which all nodes can be found.

- **Feature set** - A set of features that is connected to each object, a feature in the feature set however, can be either absent or present itself. The feature set only holds the features, not the state of the feature. the feature set for an object is always contained within the lub, and always greater than the glb.

- **Feature** - A property of an object.

- **Supernode** - A supernode n_j to a node n_i is a node that has a feature set that is a proper superset to n_i's feature set.

- **Nearest supernode** - A nearest supernode n_j to a node n_i is a node that has a feature set that is a proper superset to n_i's feature set, but there must not exist a third node n_k with a feature set that is a proper superset of n_i and a proper subset of n_j. (No node n_k with feature set f_k can exist as an intermediate node where $f_i \subset f_k \subset f_j$).

- **Subnode** - A subnode n_i to a node n_j is a node that has a feature set that is a proper subset to n_j's feature set.

- **Nearest subnode** - A nearest subnode n_i to a node n_j is a node that has a feature set that is a proper subset to n_j's feature set, but there must not exist

a third node n_k with a feature set that is a proper subset of n_j and a proper superset of n_i. (No node n_k with feature set f_k can exist as an intermediate node where $f_i \subset f_k \subset f_j$).

- **Covered node** - A node in the lattice is covered if all its nearest subnodes agree on sign.

- **Sign of a node** - The sign of a node in a lattice is simply its membership status in the concept that the lattice captures.

- **Partial complement** - A node n_i has a partial complement in a node n_j if they share the same feature set, $f_i = f_j$ but the nodes themselves are not equal, $n_i \neq n_j$.

- **Discovered node** - A node in the lattice is called a discovered node if it is generated from the meet between two partially complementing nodes that agree on sign.

- **Decision tree** - A tree that captures the determination of objects membership in concepts, based on the absence or presence of features.

- **Root node** - The upper-most node in a decision tree, or in other words the node that corresponds to the first feature to test the value of.

- **Branch** - Corresponds to one state of a feature in a decision tree.

- **Internal node** - A node in the decision tree that tests the absence or presence of a feature.

- **Leaf node** - A termination node in the decision tree, if a leaf node is reached the objects membership in the concept can be stated.

- **Training set** - A set of examples that a class membership function uses to build its knowledge from.

- **Test set** - A set of examples that are used to test the accuracy of a class membership function.

6.2 Lattice showing the pizza-example

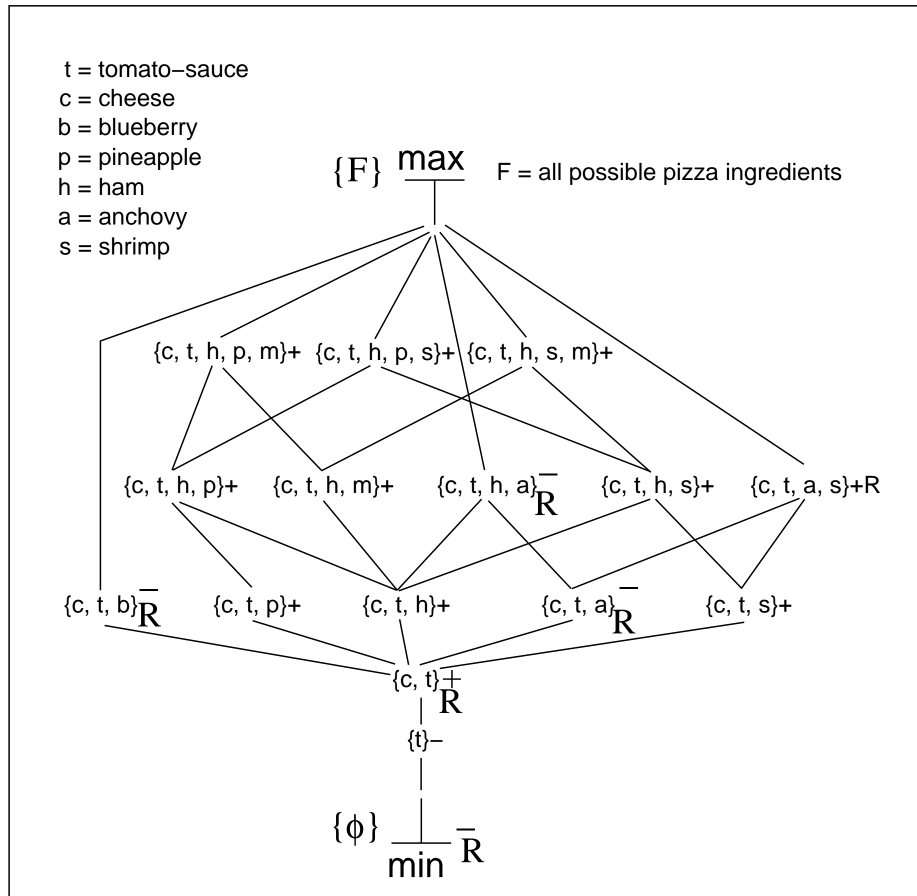

Figure 6.1: The Lattice after inserting all pizza example

Bibliography

[AKBLN88] H. At-Kaci, R. Boyer, P. Lincoln, and R. Nasr. *Efficient Implementations of Lattice Operations*, 1988.

[ALU] ALU. *The Association of Lisp Users*, `http://www.lisp.org/`.

[Bou99] R. A. Bourne. *Default Reasoning using Maximum Entropy and Variable Strength Defaults*, 1999. Dissertation submitted in partial fulfillment for Ph.D. at University of London.

[EN99] Elmasri and Navathe. *Fundamentals of Database Systems.* Morgan Kaufmann, San Mateo, CA, fourth edition, 1999.

[FS96] Y. Freund and R. E. Schapire. *Experiments with a New Boosting Algorithm*, 1996.

[GP95] M. Goldszmidt and J. Pearl. *Qualitative Probabilities for Default Reasoning, Belief Revision, and Causal Modeling*, 1995.

[Gri99] R. P. Grimaldi. *Discrete and Combinaltiorial Mathematics.* Addison-Wesley, Reading, Massachusetts; Menlo Park, California; New York; Harlow, England; Don Mills, Ontario; Sydney; Mexico City; Madrid; Amsterdam, fourth edition, 1999.

[Leh95] D. Lehmann. *Another Perspective on Default Reasoning*, 1995.

[McC86] J. McCarthy. *Circumscription - A Form of Nonmonotonic Reasoning*, 1986.

[Mit78] T. Mitchell. *Version Spaces: An Approach to Concept Learning*, 1978. Ph.D. Thesis, Electrical Engineering Department, Stanford Uneversity, CA.

[MS96] T. Mollestad and A. Skowron. *A Rough Set Framework for Data Mining of Propositional Default Rules*, 1996.

[Pea88] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann, San Mateo, CA, first edition, 1988.

[PGM93] J. Pearl, M. Goldszmidt, and Morris. *A Maximum Entropy Approach to Nonmonotonic Reasoning*, 1993.

[Qui90] J. R. Quinlan. *Induction of Decision Trees*, 1990. Originally published in *Machine Learning* 1:81–106, 1986.

[Rec84] United States Congressional Voting Records. *Congressional Voting Records Database*, 1984, `http://www.ics.uci.edu/ mlearn/MLSummary.html`. data can be found at: ftp://ftp.ics.uci.edu/pub/machine-learning-databases/voting-records/.

[RN95] S. J. Russel and P. Norvig. *Artificial Intelligence A modern Approach.* Prentice Hall, Inc., Upper Saddle River, NJ, first edition, 1995.

[RR] R. Quinlan RuleQuest Research. *Rulequest research data mining tools*, `http://www.rulequest.com/`.