

Dynamic Deduction of Induction Rules

by

Kalyani K. Manchi

**ARTHUR LAKES LIBRARY
COLORADO SCHOOL OF MINES
GOLDEN, CO 80401**

ProQuest Number: 10794505

All rights reserved

INFORMATION TO ALL USERS

The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if material had to be removed, a note will indicate the deletion.



ProQuest 10794505

Published by ProQuest LLC (2018). Copyright of the Dissertation is held by the Author.

All rights reserved.

This work is protected against unauthorized copying under Title 17, United States Code
Microform Edition © ProQuest LLC.

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 – 1346

A thesis submitted to the Faculty and the Board of Trustees of the Colorado School of Mines in partial fulfillment of the requirements for the degree of Master of Science (Mathematical and Computer Sciences).

Golden, Colorado

Date 05/08/2000

Signed: Kalyani K. Manchi
Kalyani K. Manchi

Approved: Xindong Wu
Dr. Xindong Wu
Associate Professor
Thesis Advisor

Golden, Colorado

Date 5/23/00

Graeme Fairweather
Dr. Graeme Fairweather
Professor and Head,
Department of Mathematical and
Computer Sciences

ABSTRACT

Knowledge acquisition from databases has seen sustained research for a number of years now, from both the machine-learning (ML) and the database technology fields. The ML perspective concentrates on designing agents that can learn from experience as represented by examples in databases. Inductive learning is a category of ML, where the attempt is to construct a description of a function from a set of input/output examples.

HCV is a heuristic induction algorithm, belonging to the extension-matrix based family of induction algorithms [23]. Given a set of training examples in the form of *(input, output)* pairs, it induces a set of rules that when applied to any input example, can come up with a target output or class for that example. At deduction time, these induced rules are applied to a pre-classified test set to evaluate their accuracy. Currently, there is no way to use these results to improve the rules' accuracy. Consequently, the induced rules are "frozen" on the training set, and they cannot adapt to a changing distribution of examples. In this thesis, we propose two approaches to dynamically refine the rules at deduction time, to overcome this limitation.

There are three possible match cases that have to be dealt with during deduction: *no match*, *single match* and *multiple match*. For each test example, we find the correct classification depending on the match case, using forced matching if necessary. Once

the correct class is thus found, we refine the associated rule in one of two ways: by increasing the coverages of all the conjunctions associated with the rule, or by increasing the coverage of the rule's most important conjunction only for the test example in question. These refined rules are then used for deducing the classifications for all remaining examples. Of the two deduction methods, the second method has been shown to significantly improve the accuracy of the rules when compared to the regular, non-dynamic deduction process.

TABLE OF CONTENTS

ABSTRACT	iii
LIST OF FIGURES	viii
LIST OF TABLES	ix
ACKNOWLEDGEMENTS	x
Chapter 1 INTRODUCTION	1
1.1 Problem Definition	1
1.2 Organization of Thesis	2
Chapter 2 REVIEW OF INDUCTIVE LEARNING	4
2.1 Introduction	4
2.2 AQ11	4
2.2.1 Background	5
2.2.2 Algorithm Description	7
2.2.3 Advantages and Disadvantages	8
2.3 ID3	9
2.3.1 Background	9

2.3.2	Algorithm Description	10
2.3.3	Advantages and Disadvantages	13
2.4	Extensions to AQ11 and ID3	14
2.5	The Extension Matrix Approach	16
2.5.1	Background	16
2.5.2	Terminology and Notation	17
2.5.3	Optimization Problems	19
2.5.4	Heuristic Strategies in AE1	21
2.5.5	Advantages and Disadvantages	22
Chapter 3	HCV AND DYNAMIC DEDUCTION	23
3.1	Introduction	23
3.2	The HFL algorithm	23
3.2.1	Four Strategies in HFL	23
3.2.2	Algorithm Description	28
3.3	The HCV algorithm	30
3.4	Deduction in HCV	31
3.4.1	Possible Match Cases at Deduction Time	31
3.4.2	Deduction in HCV	37
3.4.3	The Dynamic Deduction Approach	38
3.4.4	An Example Run of Dynamic Deduction	39

Chapter 4	EXPERIMENTS	45
4.1	Test Methodology	45
4.2	Results	46
4.3	A Comparison of Dynamic Deduction Methods	51
Chapter 5	SUMMARY AND FUTURE WORK	57
5.1	Summary	57
5.2	Future Work	57
REFERENCES		59

LIST OF FIGURES

4.1	Results of HCV using deduction method 2 on: (a) GLASS2	53
4.2	Results of HCV using deduction method 2 on: (b) IMP85	54
4.3	Results of HCV using deduction method 2 on: (c) SWISS	55
4.4	Results of HCV using deduction method 2 on: (d) WTP	56

LIST OF TABLES

3.1	Example Training Set for Dynamic Deduction	40
3.2	Example Test Set for Dynamic Deduction	42
4.1	Databases' Characteristics	47
4.2	Results on CLEVE database	49
4.3	Results on CLEVE2 database	49
4.4	Results on GLASS2 database	50
4.5	Results on HYPO database	50
4.6	Results on IMP85 database	50
4.7	Results on SOY database	50
4.8	Results on SWISS database	50
4.9	Results on VA database	51
4.10	Results on WINE database	51
4.11	Results on WTP database	51

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my thesis committee members Dr. Xindong Wu, Dr. Phillip Romig and Dr. Hugh King for their guidance towards my thesis. Specifically I am grateful to Dr. Wu, my thesis advisor, for his help, support and enlightening discussions. I am also grateful to Dr. King and Dr. Romig for their encouragement and support for me to reach my goal.

Chapter 1

INTRODUCTION

1.1 Problem Definition

Artificial intelligence (AI) is concerned with making machines perform “intelligent” tasks such as vision, planning and diagnosis. Machine learning (ML), as a sub-field of AI, concentrates on learning from experience. A system that can learn from the presentation of examples or from feedback provided by the environment or a friendly teacher can be employed in a number of ways. Classification and data-mining i.e., detection of new, useful and unsuspected relationships or patterns in data, are just two applications.

There are several types of machine learning [18] : Inductive learning [1], [16], [10] such as decision tree learning and version space learning, learning from network representations [2], [18], [6] as in neural networks and belief networks, and reinforcement learning [18] as in Q-learning and genetic algorithms. ML can also be categorized into supervised and unsupervised learning. Clustering techniques [26], [25], [11] fall under the latter category.

Many of the learning systems in commercial use today are based on inductive learning. In inductive learning, the two most widespread families of algorithms are

the ID3-like family and the AQ11-like family, both attribute-based. In contrast to the credit assignment and the generate-and-test processes in genetic algorithms, and numerical activity vectors in connectionist methods such as neural networks [18], attribute-based learning methods concentrate on symbolic and heuristic computations. These relate to models that operate at the level of symbols, and involve operations that manipulate symbolic expressions with an emphasis on heuristic rather than computationally explosive optimization strategies. No explicit credit assignment strategies are necessary in the attribute-based induction paradigm.

There are several limitations to the AQ11 and ID3 families of algorithms (as described in Chapter 2). Therefore, a competitive heuristic induction algorithm HCV [23] was developed based on the extension matrix approach. However, as with most other induction algorithms, once induction is done, the rules in HCV are “frozen” – there is no means to use the results of applying the induced rules to an unseen set of test examples to increase the rules’ accuracy, without re-inducing them from the unseen examples. The contribution of this thesis is to provide such a mechanism, whereby rule performance at deduction time can be used as feedback to modify rule weights, leading to increased prediction accuracy.

1.2 Organization of Thesis

This thesis is organized as follows:

Chapter 2 provides a review of inductive learning. It summarizes the three typical families of inductive algorithms, the AQ11-like, ID3-like and the extension matrix approach based algorithms, giving their main features, advantages and disadvantages.

A detailed description of HCV and the proposed approach to refine its rules dynamically are given in Chapter 3. Experiments and their results are given in Chapter 4. A comparison is made between the accuracy of classifications given by HCV and HCV with the proposed approach, dynamic deduction. Finally, Chapter 5 gives a brief summary of the thesis.

Chapter 2

REVIEW OF INDUCTIVE LEARNING

2.1 Introduction

This chapter first reviews the two most widespread induction algorithms in machine learning, AQ11 and ID3. It describes their approaches in brief and outlines their advantages and disadvantages. Then it mentions some of the enhancements made to them in recent years. Finally, it explains the relatively new extension matrix approach and again, analyzes it in terms of advantages and disadvantages.

2.2 AQ11

AQ11 belongs to the family of induction algorithms that are centered around the idea that learning can be viewed as a process of searching for a good hypothesis in a large *hypothesis space* defined by the representation language chosen for the task. We first look at the two main algorithms that were the basis for AQ11, then we examine AQ11 itself in some detail, and finally, we look at its advantages and disadvantages.

2.2.1 Background

AQ11, designed by Michalski *et al* [23], shares the basic generalisation-specialisation strategy to develop concept descriptions with Winston's ARCH program [21] and Mitchell's candidate elimination algorithm [14], [15].

Winston's ARCH program describes a mechanism that can learn concepts by looking for relationships between semantic network representations of block world configurations. There are two processes involved in this learning mechanism: finding and exploiting commonalities among structural descriptions for the same type of configuration, and finding significant differences between positive examples and negative examples. These two processes correspond to *generalisation* and *specialisation*. So the ARCH program generalises the concept representation so as to include all the positive examples, and specialises it to exclude all the negative ones. Each new example presented triggers an extension of the concept representation in one of two ways: generalization for a positive example, and specialization for a negative example.

Mitchell's *candidate elimination* algorithm [18], [14], [15] is also based on the basic generalization-specialization strategy, but it explores the solution (or hypothesis) space in a way different from Winston's ARCH program. In the ARCH program, hypotheses are generated and tested one by one, whereas in Mitchell's method a hypothesis space (viewed as a disjunctive sentence $H_1 \vee H_2 \vee \dots \vee H_n$) is maintained at all times. As examples are presented, those hypotheses found unsatisfactory are

deleted from the hypothesis space. This set of remaining hypotheses is called a *version space*.

Thus, Mitchell's method aims to ensure that at all times, the version space contains the complete set of hypotheses or concept representations consistent with all the examples seen so far. A simplified description of the candidate elimination algorithm is as follows:

The Candidate Elimination Algorithm

- 1 Initialize the version space to be the set of all hypotheses.
 - 2 Set G to be the set of most generous representations, initially "TRUE".
 - 3 Set S to be the set of most specific representations, initially "FALSE".
 - 4 For each new training example:
 - 5 If it is positive, update S so that it still contains the set of most specific, satisfactory representations.
 - 6 If it is negative, update G so that it still contains the set of most general, satisfactory representations.
 - 7 If $G=S$, exit.
-

AQ11 is a more sophisticated variant of the basic generalisation-specialisation method described above. It can be used to generate representations for a classification function, i.e., a target mapping with more than one output and more than two input groups.

2.2.2 Algorithm Description

The input to AQ11 is in the form of a set of (*attribute, value*) pairs. The target output is the classification for the input. The representations produced for the target mapping are rules written in *variable-valued logic*¹ [23], one rule for each distinct concept² or classification.

Suppose the training set is divided into n subsets, with each subset containing the inputs that should bring about a particular classification. AQ11 works as follows:

The AQ11 Algorithm

- 1 For each training subset do
 - 2 Convert all the examples in the subset into positive examples.
 - 3 Convert all the other examples (from other subsets) into negative examples. Use previously generated rules also as negative examples.
 - 4 Apply generalization-specialization and a set of heuristic strategies [9] to the training examples produced in Steps 2 and 3 above.
-

¹Variable-valued logic was a representation calculus developed by Michalski [12], for representing decision problems where decision variables can take on some range of values. Its principal entities are *selectors*, each having the general form

$$[X \# R] \tag{2.1}$$

where X is a variable or an attribute, “#” is a relational operator such as =, \neq , $<$, $>$, \leq , and \geq), and R (called a *reference*), is a list of one or more values that X can take. A well-formed rule in the logic is similar to a production rule, but it has selectors as the basic components of both its left and right sides.

²For example, in PLANT/ds [23] where AQ11 was first successfully applied, each of the fifteen soybean diseases is a concept or class.

2.2.3 Advantages and Disadvantages

One of the main advantages of AQ11 is that it will ensure that the most general representations do not cover any negative representations. That is, it will make sure that the current general boundary being constructed does not overlap any previously constructed general boundaries. However, there are several problems with AQ11.

First, as can be seen from Step 3 of the AQ11 algorithm, rules need to be in the same syntactic form as examples.

Second, the order of presentation of the training subsets influences the rules produced: The first rule induced will be more general than the ones induced later.

Third, the AQ11 algorithm is computationally more expensive than ID3. Whenever G or S is updated, we have to check to see whether all the examples that were previously satisfied are still satisfied, and no new unsatisfactory examples have been added. Also, the rules produced by AQ11 are more complex than those produced by ID3, although they may be more comprehensible.

Finally, since the AQ11 algorithm is bottom-up in nature it can be disrupted if the data is noisy, or if the domain contains insufficient attributes for exact classification [18].

2.3 ID3

ID3 is a decision-tree based algorithm. In general, a decision tree takes as input an object or situation described by a set of properties, and outputs a yes/no decision. Hence, decision trees best represent boolean functions, though they can be extended to represent functions with a larger range of outputs.

In this section again, we first describe the algorithm that led to the development of ID3. This is followed by an explanation of ID3, and its advantages and disadvantages.

2.3.1 Background

ID3 [16] is a “top-down” algorithm developed by Quinlan out of the Concept Learning System (CLS) by Hunt[10].

The CLS learning mechanism takes a set of training pairs and constructs a concept representation in the form of a decision tree in which each leaf node has a target output associated with it. This tree is then “applied” to any new input example by propagating the example down through the tree. The example ends up at a leaf node, and the target output associated with the leaf node is taken as the example’s classification.

The main steps in the CLS algorithm are:

The CLS Algorithm

- 1 $T \leftarrow$ the entire training set. Create a T node.
 - 2 If all examples in T are positive, create a “yes” node with parent T and stop.
 - 3 If all examples in T are negative, create a “no” node with parent T and stop.
 - 4 Select an attribute X with a value range of v_1, \dots, v_N and partition T into subsets (T_1, \dots, T_N) according to their values on X . Create N T_i nodes ($i = 1, \dots, N$) with parent T and $X = v_i$ as the label of the branch from T to T_i .
 - 5 For each T_i do: $T \leftarrow T_i$ and go to step 2.
-

2.3.2 Algorithm Description

Quinlan modified the CLS algorithm in two ways.

First, he added a process known as *windowing*, to enable the algorithm to cope with very large training sets.

For very large training sets, it may be more efficient to process a small sample first, rather than process the entire set as one. If this sample is representative of the complete set, the decision tree produced will be similar to the one that would have been obtained by processing the entire training set as one. The tree obtained from the sample set can now be refined by scanning the training set for examples that are not properly represented in the tree, and whenever such an example is found, modifying the tree appropriately. However, windowing does not feature very strongly in recent work, and there is evidence suggesting that it may provide very little benefit [23].

Second, and more importantly, Quinlan proposed a heuristic called the *entropy measure*, based on information theory. At each stage of the tree-growing process, this

measure helps decide the most “important” attribute on which to split the decision tree further. Thus, smaller and more efficient decision trees (i.e., decision trees with less depth) can be constructed.

ID3 works as follows:

Suppose the training set is represented by $T = PE \cup NE$ where PE is the set of positive examples and NE is the set of negative examples, $p = |PE|$ and $n = |NE|$. The probability of an example e belonging to PE is $p/(p+n)$ and that of its belonging to NE is $n/(p+n)$. A decision tree is considered as a source of the message “PE” or “NE”; using the information theoretic heuristic described above, the expected information needed to generate this message is given by

$$I(p, n) = \begin{cases} -\frac{p}{p+n} \log_2 \frac{p}{p+n} - \frac{n}{p+n} \log_2 \frac{n}{p+n} & \text{when } p \neq 0 \text{ \& } n \neq 0 \\ 0 & \text{otherwise.} \\ & \text{when } p \neq 0 \text{ \& } n \neq 0 \end{cases} \quad (2.2)$$

Suppose attribute X can take on values from the domain $\{v_1, \dots, v_N\}$. If it is used as the root of the decision tree, it will partition T into N subsets $\{T_1, \dots, T_N\}$ where T_i contains all the examples in T that take on the value v_i of X . Let T_i contain p_i examples of PE and n_i examples of NE. The expected information required for the subtree for T_i is $I(p_i, n_i)$. The expected information required for the tree with X as

root, $EI(X)$, is the weighted average

$$EI(X) = \sum_{i=1}^N \frac{p_i + n_i}{p + n} I(p_i, n_i) \quad (2.3)$$

where the weight for the i -th branch is the proportion of the examples in T that belong to T_i . Therefore, the information gained by branching on X , $G(X)$, is given by

$$G(X) = I(p, n) - EI(X). \quad (2.4)$$

ID3 examines all candidate attributes X_j , ($j = 1, 2, \dots, a$) where a is the total number of attributes, and chooses the X_j that maximizes $G(X)$ as the root. Then it uses the same process recursively to construct decision trees for subsets T_1, \dots, T_N , corresponding to the different values of attribute X_j . For each T_i ($i = 1, \dots, N$), if all the examples in T_i are positive, a 'yes' node is created; if all the examples in T_i are negative, a 'no' node is created. Further growth from these 'yes' and 'no' nodes is halted. Otherwise i.e., if the examples in T_i come from both PE and NE , another attribute is selected to grow the tree further using the information theoretic heuristic in the same manner as described above.

2.3.3 Advantages and Disadvantages

One of the advantages of the ID3 method is the fact that it does not require users to specify background knowledge. Version-space methods such as AQ11 may need the specification of generalization hierarchies to avoid problems with allowing unlimited disjunction in the hypothesis space [18]. This means that ID3 can be applied to any syntactically well-formed training set. In addition, it is efficient and fast. These factors have enabled ID3 to be used in a wide variety of commercial packages.

However, ID3 has some limitations:

First, in a domain where a single decision tree may not be sufficient to represent all the expertise in that domain, the decision trees cannot be applied directly as problem-solvers; they need to be decompiled into production rules first. Therefore, in these cases, decision trees are less convenient for manipulations than the variable-valued logic of AQ11. Further, production rules from decompiled decision trees are too simple to express things like memberships.

Second, in the tree growing phase, once an attribute is selected for splitting, all arcs labeled by values that attribute takes must be expanded. The resulting paths can be longer than those actually needed because, by the time specific concepts (i.e., leaf nodes on the decision tree) are developed, irrelevant variables may have been introduced.

Third, the number of branches (paths) might be very large since any arc out of

a non-leaf node can be labeled with only one value.

Finally, ID3 is heuristic in nature because the information gain heuristic (*entropy measure*) that it uses is not complete. This means that it is not guaranteed to find the simplest and optimal decision tree for the given training set.

2.4 Extensions to AQ11 and ID3

Several extensions have been made to the original AQ11 and ID3 algorithms in their successors such as AQ15, CN2, ID5R and C4, to improve their various capacities such as noise handling and incremental induction. These extensions are briefly described below:

- **Noise Handling:** Two approaches have been tried for AQ11 [23]: The first is to use a partial match procedure called TRUNC to execute rules in AQ15 [13]; the second, in CN2 [5], is to use the entropy measure from ID3 to produce rules in domains where problems of poor description language and/or noise may be present.

For ID3, tree growth can be halted in noisy environments when no more significant information gain can be found.

- **Incremental Learning:** Incremental learning means dividing a large example set into a number of subsets and considering one subset at a time. This method has been adopted in AQ15 [13], ID4 [19] and ID5R [20]. The window-

ing technique in ID3 can also be looked upon as a way to implement incremental learning.

- **Constructive Learning:** Explicit, built-in background knowledge is not needed in either AQ11-like or ID3-like algorithms. However, implicit background knowledge is always embedded in the formulation of solution spaces and in the representation of examples. If a solution space turns out to be inadequate, representation modification is needed. This modification process involves searching for useful new features (constructive induction) in terms of existing features or attributes. AQ17 [3] of the AQ11 family implements such a scheme.
- **Decision Tree Pruning**[18]: The ID3 algorithm tends to construct exact decision trees. This can be an issue in many real-world problems such as medical diagnosis and image recognition, since the actual classification cannot be exact due to the presence of noise and/or uncertainty in the data. As a result, a decision tree by ID3 may not be able to capture the relations in the data properly. Hence, mechanisms to prune decision trees have been devised and implemented in many systems such as ASSISTANT [4] and C4 [17]. Once a non-leaf subtree meets a specific criterion such as having an equal or smaller number of misclassifications after replacement, it is replaced by a leaf node.

In addition to the above, other techniques that have been tried include [23]: decompiling decision trees into production rules, binarization of decision trees, using

a new selection criterion (the *gain ratio* criterion) instead of the entropy measure, structured induction, and adding abilities to deal with real-valued attributes.

However, although each of the extensions mentioned above is useful in some specific cases, none of them has generally improved AQ11 or ID3 in noise-free environments. AQ11's time complexity and rule compactness remain the same. The core of the decision tree family of algorithms is still the entropy measure to select an attribute by examining all candidate attributes during the splitting of examples at non-leaf nodes.

2.5 The Extension Matrix Approach

This is a fairly recent approach to inductive learning. Since it forms the basis for the deduction methods presented in this thesis, we will look at it in more detail.

2.5.1 Background

The family of inductive algorithms based on the extension matrix approach was first developed at the University of Illinois by Hong *et al.* and then redesigned by Wu [23]. The algorithms of the extension matrix family take a kind of matrix called an *extension matrix* as their mathematical basis.

2.5.2 Terminology and Notation

Let a be the number of attributes (represented by $\{X_1, \dots, X_a\}$) in an example space, n be the set of negative examples $|NE| = |\{e_1^-, \dots, e_n^-\}|$ where e_i^- ($i = 1, \dots, n$) is the i -th negative example, and p be the set of positive examples $|PE| = |\{e_1^+, \dots, e_p^+\}|$ where e_i^+ ($i = 1, \dots, p$) is the i -th positive example. Let the negative example set NE be expressed by the matrix

$$NEM = \{e_1^-, \dots, e_n^-\}^T = (r_{ij})_{n \times a} \quad (2.5)$$

with the i -th row of matrix NEM corresponding to the i -th negative example e_i^- ($i = 1, \dots, n$). An element of matrix $NEM(i, j) = r_{ij}$, indicates that the value of e_i^- on attribute X_j is r_{ij} .

Now, if the k -th ($k = 1, \dots, p$) positive example is expressed as $e_k^+ = (v_{1k}^+, \dots, v_{ak}^+)$, its *extension matrix* against NE is given by

$$EM_k = (r_{ijk})_{n \times a} \quad (2.6)$$

where

$$r_{ijk} = \begin{cases} * & \text{when } v_{jk}^+ = NEM_{ij} \\ NEM_{ij} & \text{when } v_{jk}^+ \neq NEM_{ij} \end{cases}$$

Here, ‘*’ denotes a dead element that cannot be used to differentiate between the positive example e_k^+ and the negative example on row i of NEM .

For any EM_k , a set of n nondead elements r_{ij_i} ($i = 1, \dots, n, j_i \in \{1, \dots, a\}$) coming from the n different rows is called a *path* in the extension matrix.

In [9], it has been shown that a path $\{r_{1j_1}, \dots, r_{nj_n}\}$ in an EM_k corresponds to a conjunctive formula

$$L = \bigwedge_{i=1}^n [X_{j_i} \neq r_{ij_i}] \quad (2.7)$$

which covers the positive example e_k^+ against NE and *vice versa*.

Each $[X_{j_i} \neq r_{ij_i}]$ here corresponds to a *selector* in variable-valued logic. If r_{ij_i} appears on m ($m \in \{0, \dots, n\}$) rows in the same column j_i of an extension matrix EM_k , we say r_{ij_i} or $[X_{j_i} \neq r_{ij_i}]$ covers m rows of the EM_k .

The disjunction matrix of a set of positive examples $\{e_{i_1}^+, \dots, e_{i_k}^+\}$ against NE or the disjunction matrix of $EM_{i_1}, \dots, EM_{i_k}$, $EMD = (r_{ij})_{n \times a}$, can be defined as:

$$r_{ij} = \begin{cases} * & \text{when } \exists k_1 \in \{i_1, \dots, i_k\} : EM_{k_1}(i, j) = * \\ \bigvee_{k_2=1}^k EM_{i_{k_2}}(i, j) = NEM(i, j) & \text{otherwise} \end{cases} \quad (2.8)$$

In the EMD of a positive example set $\{e_{i_1}^+, \dots, e_{i_k}^+\}$ against NE , a set of n nondead elements r_{ij_i} ($i = 1, \dots, n, j_i \in \{1, \dots, a\}$) coming from the n different rows is also called

a *path*.

It has been proved [23] that a path $\{r_{1j_1}, \dots, r_{nj_n}\}$ in the *EMD* of a positive example set $\{e_{i_1}^+, \dots, e_{i_k}^+\}$ against the negative example set *NE* corresponds to a conjunctive *formula* or *cover*

$$L = \bigwedge_{i=1}^n [X_{j_i} \neq r_{ij_i}] \quad (2.9)$$

that covers all of $\{e_{i_1}^+, \dots, e_{i_k}^+\}$ against *NE* and *vice versa*.

If there exists at least one path in the disjunction matrix *EMD* of a positive example set $\{e_{i_1}^+, \dots, e_{i_k}^+\}$ against *NE*, all the positive examples in the set are said to “intersect” and the positive example set is called an *intersecting group*.

For a given set of examples, if *PE* and *NE* are persistent, i.e., they have no examples in common, there always exists at least one conjunctive formula covering any positive example e_k^+ from *PE* against *NE* [23].

Therefore, for a given set of examples, if *PE* and *NE* are persistent, at least one conjunctive formula cover always exists for each intersecting group of examples.

2.5.3 Optimization Problems

There are two optimization problems in the extension matrix approach [23]:

- The minimum formula (MFL) problem: Finding a conjunctive formula that covers a positive example or an intersecting group of positive examples against

the set of negative examples NE , and has the minimum number of different conjunctive selectors.

- The minimum cover (MCV) problem: Finding a cover that covers all the positive examples in PE (which can be divided into one or more intersecting groups) against NE and has the minimum number of conjunctive formulae with each conjunctive formula being as short as possible.

Since each of all the paths in the extension matrix EM_k of each positive example e_k^+ against NE corresponds to a conjunctive formula of e_k^+ against NE and since an optimal cover of PE against NE is a minimum set of formulae that is a logical combination of all the formulae from every EM_k ($k = 1, \dots, p$), both MFL and MCV problems are NP-hard [7].

HFL and HCV are two complete algorithms designed to solve the optimization problems MFL and MCV in [22], [24], with $O(na2^a)$ and $O(n2^a2^{2^a} + pa^22^{2^a})$ time complexity respectively, given that each attribute domain D_i ($i = 1, \dots, a$) satisfies $|D_i| = 2$. When there exists a domain D_j such that $|D_j| > 2$ or D_j is a real-valued interval ($j \in \{1, \dots, a\}$), a decomposition method that decomposes D_j into several sub-domains whose bases are each two is also designed there.

2.5.4 Heuristic Strategies in AE1

AE1 is a system that aims to solve the MFL and MCV problems described above by adopting some heuristic strategies. Thus, approximate rather than optimal solutions are found. These strategies are [7]:

- Starting solution search from the columns with the most nondead elements, and
- Simplifying selector or rule redundancy by deductive inference rules in mathematical logic.

There are two problems with the above strategies: First, the first strategy can lose optimal solutions in some cases. For example, in the extension matrix given below, the first heuristic strategy in AE1 cannot find the optimal formula ($[X_1 \neq 1] \wedge [X_3 \neq 1]$), since it will first choose the selector $[X_2 \neq 0]$. Second, simplifying redundancy for MFL and MCV problems is NP-hard [23].

$$\begin{pmatrix} 1 & * & * \\ * & 0 & 1 \\ 1 & 0 & * \\ * & 0 & 1 \\ 1 & 0 & * \\ * & * & 1 \end{pmatrix}$$

2.5.5 Advantages and Disadvantages

The major disadvantage of the extension matrix approach is its NP-hard nature described in section 2.5.3. AE1 does not remove this disadvantage, as shown in the previous section. An AE5 system [8] was developed based on AE1, but the basic algorithm remained the same. The only difference between AE5 and AE1 is that AE5 has some added facilities such as constructive and incremental induction. So, the two major problems of AE1 still apply to AE5. Chapter 3 talks about the HCV algorithm that removes them both.

Chapter 3

HCV AND DYNAMIC DEDUCTION

3.1 Introduction

This chapter starts with a description of the HFL and HCV algorithms. Then it narrows focus to the deduction process of HCV, and explains the approach followed by this thesis to improve the accuracy of the rules at deduction time.

3.2 The HFL algorithm

The HFL algorithm finds a heuristic conjunctive formula that corresponds to a path in an extension matrix (or a disjunction matrix) when there is at least one path in the extension (or disjunction) matrix. Disjunction matrices can be processed in the same way as extension matrices to find their conjunctive formulae; hence we will only refer to extension matrices below.

3.2.1 Four Strategies in HFL

The HFL algorithm adopts four strategies:

1. The *fast* strategy. In an extension matrix $EM_k = (r_{ij})_{n \times a}$, if there is no dead element in a column (say j), then $[X_j \neq r_j]$ where $r_j = \bigvee_{i=1}^n r_{ij}$ is chosen as the

one selector cover for the entire matrix EM_k .

For example, in the extension matrix given below, selector $[X_4 \neq \{normal, dry-peep\}]$ from the fourth column can cover all the five rows of the matrix:

$$\begin{pmatrix} absent & slight & strip & normal & * \\ * & * & hole & dry-peep & fast \\ low & slight & strip & normal & * \\ absent & slight & spot & dry-peep & fast \\ low & medium & * & normal & fast \end{pmatrix}$$

2. The *precedence* strategy. If there is a r_{ij} in column j that is the only nondead element of a row i in an extension matrix $EM_k = (r_{ij})_{n \times a}$, the selector $[X_j \neq r_j]$ where $r_j = \bigvee_{i=1}^n r_{ij}$ is called an *inevitable selector* and is chosen with top precedence.

For example, in the following matrix, $[X_1 \neq 1]$ and $[X_3 \neq 1]$ are two inevitable selectors:

$$\begin{pmatrix} 1 & * & * \\ * & 0 & 1 \\ 1 & 0 & * \\ * & 0 & 1 \\ 1 & 0 & * \\ * & * & 1 \end{pmatrix}$$

3. The *elimination* strategy. In an extension matrix EM_k , if for every appearance of some nondead element in the j_1 -th column of some row, there is always another nondead element in the j_2 -th column of the same row, $[X_{j_1} \neq r_{j_1}]$ where $r_{j_1} = \bigvee_{i=1}^n r_{ij_1}$ is called an eliminable selector and can be eliminated by selector $[X_{j_2} \neq r_{j_2}]$ where $r_{j_2} = \bigvee_{i=1}^n r_{ij_2}$.

For example, attribute X_2 can be eliminated by attribute X_3 in the extension matrix given below:

$$\begin{pmatrix} 1 & * & * \\ * & 0 & 1 \\ 1 & 0 & 1 \\ * & 0 & 1 \\ 1 & 0 & 1 \\ * & * & 1 \end{pmatrix}$$

4. The *least-frequency* strategy. In an extension matrix, when all inevitable selectors have been selected and all eliminable selectors have been excluded but all the selectors chosen so far have not yet covered all the rows, a selector having the least number of non-dead elements in its corresponding column in the extension matrix can be excluded. This selector is called a *least-frequency* selector.

For example, in the extension matrix given below, attribute X_1 can be eliminated and there still exists a path:

$$\begin{pmatrix} 1 & * & 1 \\ * & 0 & 1 \\ 1 & 0 & * \\ * & 0 & 1 \\ 1 & 0 & * \\ * & 0 & 1 \end{pmatrix}$$

It has been proved [23] that all of the *fast*, *precedence* and *elimination* strategies are complete. This means that if there exist one or more shortest conjunctive formulae in an extension matrix, the *fast*, *precedence* and *elimination* strategies will not lose it.

Also, in an extension matrix, if there exists at least one shorter path which has less than n different conjunctive selectors, the solution generated by the *least-frequency* strategy must be the shorter one.

Thus, even though the fourth strategy (the *least-frequency* strategy) is not necessarily complete i.e. it is a heuristic and thus could lose optimal paths at times, compared to the AE1 system, three complete strategies were adopted in HFL.

3.2.2 Algorithm Description

The HFL algorithm is given below:

The HFL Algorithm

- 0 Initialize variable Hfl to null.
 - 1 Try the fast strategy on all the uncovered rows.
If successful, add a corresponding selector to variable Hfl. Return Hfl.
 - 2 Apply the precedence strategy to the uncovered rows; If some inevitable selectors are found, add them to Hfl and go to Step 1.
 - 3 Apply the elimination strategy to all attributes that have neither been selected nor eliminated. If any eliminable selector is found, reset all the elements in the corresponding column with *. Go to Step 2.
 - 4 Apply the least frequency strategy to all attributes that have neither been selected nor eliminated. Find a least-frequency selector and reset all the elements in the corresponding column with *. Go to Step 2.
 - 5 Return Hfl
-

Steps 1, 2, 3 and 4 implement the *fast*, *precedence*, *elimination* and *least-frequency* strategies described in the previous section. When the *fast* strategy finds a column that has nondead elements on all the uncovered rows of the EM, the EM is fully covered by picking the selector corresponding to the column, and thus the *Hfl* variable can be returned. In Step 2, after one or more inevitable selectors have been chosen, more rows are covered. So HFL will come back to Step 1 to test the *fast* strategy on uncovered rows. Each time a selector has been chosen by either the *fast* strategy or the *precedence* strategy, there are two scenarios: all the selectors chosen till now have either covered or not yet covered all the rows of the extension matrix. After one

or more columns have been crossed out by the *elimination* strategy in Step 3, some rows may have only one non-dead element. Hence, the *precedence* strategy and the *fast* strategy are applied again, in that order. The least-frequency strategy is applied only in those cases when all inevitable selectors have been chosen and all eliminable selectors have been excluded but all the selectors chosen have not yet covered all the rows in the extension matrix. This strategy can cross out a column that has not been crossed out before. Therefore after executing it (Step 4), HFL comes back to test the *precedence* strategy. The *fast* strategy cannot be immediately applied after Step 3 or Step 4 since excluding a column by either the *elimination* strategy (Step 3) or the *least-frequency* strategy (Step 4) does not cover any uncovered rows. This is why the control in HFL comes back to Step 2 instead of Step 1 at the end of both Step 3 and Step 4. There is at least one column or selector that has been processed: it has been either chosen or crossed out, each time the control comes back to Step 2 or Step 1. Therefore, if there are a columns in an extension matrix, a loops in HFL are needed at the most.

The time complexity for Steps 1, 2, 3 and 4 is $O(na)$, $O(na)$, $O(na^2)$ and $O(na)$ respectively. The time complexity for the whole algorithm is thus [23]

$$O(n + 2a + 2 + a(na + na + na^2 + na)) \approx O(na^3).$$

3.3 The HCV algorithm

The HCV algorithm first partitions the positive examples (PE) of a specific class into p (p less than or equal to the total number of examples for that class) intersecting groups, then calls the heuristic algorithm HFL to find an *Hfl* (i.e. a conjunctive cover) for each intersecting group, and then obtains the covering formula for the entire class by logically ORing all the *Hfls* of the intersecting groups.

The HCV algorithm in brief is as follows:

The HCV algorithm

- 1 Partition PE into intersecting groups, initialize variable Hcv to null.
 - 2 For each intersecting group
 - 3 Initialize variable Hfl to null.
 - 4 Call HFL i.e., execute the fast, precedence, elimination and least-frequency strategies.
 - 5 Reassign Hcv: $Hcv = Hcv \vee Hfl$
 - 6 Return Hcv
-

The time complexity for HCV is $O(pna^3 + p^2na)$.

It has been proved [23] that the formula *Hcv* generated by Algorithm HCV covers all the positive examples against the negative examples in a given example set. Also, if there exists at least one conjunctive cover in a given training example set, HCV produces a conjunctive formula.

3.4 Deduction in HCV

Deduction is the time when the predictions made by the HCV algorithm in the form of induction rules are tested against pre-classified test examples to obtain a measure of the rules' accuracy. In general, the rule for any class is in the form of a disjunction ($conj_1 V conj_2 V \dots V conj_n$). If all the examples for the class fall into one intersecting group, this disjunction will have only one conjunction.

3.4.1 Possible Match Cases at Deduction Time

For a given test example, there are three situations that need to be dealt with:

- *No Match*: No rules match the example.
- *Single Match*: One or more rules match the example, and indicate the same class.
- *Multiple Match*: More than one rule matches the example, and these rules indicate different classes.

The third case does not apply to decision trees produced by ID3-like algorithms, but production rules derived from decompiling the decision the trees can face the same problem.

In the single match case, the test example's classification is naturally indicated by the rules. Deduction-time processing deals mainly with conflict-resolution in the

multiple match case, and probability estimation in the no match case, because the single match case is not a problem. Here, we describe the way HCV handles these cases:

No Match. In the case of no match, the training set is scanned to find a class that is “close” to the test example in question.

- **Largest Class.** This is a common method to deal with the no match case, where the example is assigned to the largest class, called the *default class*. The assumption behind this method is that the training set is representative of the example distribution in the domain under consideration. Therefore, the probability of a random example belonging to a large class is higher than that of it belonging to a small one.

The largest class method is good when the number of classes in the training set is small and one of the classes contains a predominant number of examples. However, as the number of classes grows and the examples are more evenly spread out over the classes, the results could deteriorate.

- **Measure of Fit.** Instead of relying solely on the probability of each class in the training example set, the *measure of fit (MF)* method [23] calculates the *MF* value of each class c_i for each no match test example e .

For a selector sel or X that can take on values $[V_1, \dots, V_n]$, its *MF* value for e is defined as

$$MF(sel, e) = \begin{cases} 1 & \text{if sel is satisfied by e} \\ \frac{n}{|X|} & \text{otherwise} \end{cases}$$

where n is the number of attribute values that the selector includes and $|X|$ is the number of values in the X domain.

If a conjunctive rule $conj$ has k selectors, its MF value for the test example is defined on the product of the MF values of its k selectors, adjusted by its own weight in the training set:

$$MF(conj, e) = \prod_k MF(sel_k, e) \times \frac{n(conj)}{N} \quad (3.1)$$

where $n(conj)$ is the number of examples in the training set that are covered by the $conj$ rule, and N is the total number of examples in the training set.

The MF of a class c_i for a test example e is the probabilistic sum of the MF values of all the conjunctive rules for the class. In the case of two rules, $conj_1$ and $conj_2$, it is given by the following formula:

$$MF(c_i, e) = MF(conj_1, e) + MF(conj_2, e) - MF(conj_1, e)MF(conj_2, e) \quad (3.2)$$

If there are more than two rules for the class, we use this formula recursively.

The measure of fit method interprets the MF value as the *closeness* of the test example to the class, and therefore chooses the class c_i that maximizes $MF(c_i, e)$ as e 's classification.

Multiple Match. Overgeneralization of the training examples at induction time causes multiple match to occur at deduction time. All rule induction algorithms explicitly or implicitly implement generalization and specialization. We try to generalize the positive examples as far as we can until they cover negative examples. Once negative examples are covered, we specialize the concept description to exclude them. However, generalization and specialization need to be carried out under the closed world assumption because the training set is generally incomplete. A training example (say $(X_1 = a, X_2 = 1)$) with a missing classification can be assumed to belong to a concept c when no negative examples of the concept taking on the value of a on X_1 can be found. It can also be assumed to be a negative example at the same induction process if we find that all the negative examples and no positive examples in the training set have the attribute value $X_2 = 1$. So the example in this case could well

be covered by descriptions of both concepts: “ c ” and “not c ”. Therefore a multiple match happens in the subsequent deduction process if this same example appears in the test set.

Multiple match resolution needs to provide some measurements or criteria to judge which class is closer or more reliable to the test example in question. The following are some common methods used in HCV for such conflict resolution:

- **First Hit.** The simplest way to solve the multiple match issue is to use the first rule that classifies the example to determine the example’s classification. This simple method can be expected to produce reasonable results if the rules from the induction process have been sorted and ordered according to their reliability (e.g., putting rules related to the largest class before others).

The advantage of this method is that it is straightforward and efficient in execution at deduction time. However, the price for this efficiency at deduction time is that the rules need to be sorted following induction.

- **Largest Rule.** Instead of using the coverage of the rules for a class, we can use the coverage of each conjunctive rule. Again, however, this method is good only when the number of rules is small and one of the rules covers a predominant number of examples.
- **Estimate of Probability.** The *estimate of probability* method [23] for handling the multiple match case assigns an *EP* value to every class by examining the

training set coverage of the satisfied conjunctive rules.

The EP value for a conjunctive rule $conj$ that is satisfied by a test example e is defined as

$$EP(conj, e) = \begin{cases} \frac{n(conj)}{N} & \text{if } conj \text{ is satisfied by } e \\ 0 & \text{otherwise.} \end{cases}$$

where $n(conj)$ is the weight of $conj$ in the training set and N is the number of examples in the training set.

The EP value of a class c_i is defined as the probabilistic sum of the EP values of all the conjunctive rules for the class. In the case of two rules, $conj_1$ and $conj_2$, it is given by the following formula

$$EP(c_i, e) = EP(conj_1, e) + EP(conj_2, e) - EP(conj_1, e)EP(conj_2, e) \quad (3.3)$$

If there are more than two rules for the class, we use this formula recursively.

Thus, for the multiple match test example, the *EP* method chooses the class with the highest *EP* value as its classification.

3.4.2 Deduction in HCV

The current implementation of HCV provides many methods for deduction. Out of these, the user can pick the one that may be best suited to the problem at hand. However, as we have seen in the previous section, all of these deduction methods are “frozen” on the training example set. For example, in the *epmf* method of deduction that combines both *EP* and *MF* (explained in the previous section), a major factor in determining the “closeness” of a test example to a particular class is the coverage of the training set by the rules for that particular class. That is, the closeness of any particular example to a class is influenced by the number of training examples that fell under that class at induction time.

This indicates that if the training set is not wholly representative of the actual distribution of examples in that particular domain, there is no way to refine the rules accordingly after induction. The only option is to re-induce another set of rules from a more representative training set.

This thesis explores whether the above limitation can be overcome by dynamically refining rule coverages at deduction time. A particular deduction method, *epmf*, has been chosen, and the results of applying the rules to the test examples at deduction time are used to modify rule coverages, to see whether the rules’ accuracy can

be improved.

3.4.3 The Dynamic Deduction Approach

The dynamic deduction approach is summarized below:

The Dynamic Deduction Algorithm

For each test example:

- 1 Get the initial classification.*
 - 2 If correct, increase the weight of the rule for that class (this applies to all three match cases: no match, single match and multiple match).*
 - 3 If incorrect,*
 - 3.1 No Match Case: Get the next best classification for the test example, until the correct class is found. Then, increase the weight of the rule that gave this classification.*
 - 3.2 Single Match Case: Force a no-match classification: Get the next best classification until the correct one is found. Then, increase the weight of the pertinent rule.*
 - 3.3 Multiple Match Case: First, try to get the next best classification from all the rules that match the given test example.*
 - 3.3.1 If the correct class is found, increase the weight of the associated rule.*
 - 3.3.2 Else, i.e. if none of the rules that match the test example give the correct classification, force a no-match classification. Get the next best classification until the correct one is found, then, increase the weight of the rule for this class.*
-

In the above steps, the following two methods were adopted to increase the weight of the rule giving the correct classification:

- Method 1: Get the list of all the conjunctions for the rule. Increase the weight of each of these conjunctions by 1. Also, increase the size of the training example set by 1.

- Method 2: Retrieve the conjunction from the rule's conjunction list that contributed the most towards the estimate of the rule/class. Increase its weight by 1, and the size of the training example set by 1.

If the number of test examples is t and the number of conjunctions is c , in the worst case, when *no match* is used for all test examples, the time taken by regular deduction would be $O(tc)$. The worst case for dynamic deduction will be when all the test examples are classified by multiple match, with the number of conjunctions matching each test example being close to the total number of conjunctions c , yet having to force a no-match selection for each test example. In this case, the time complexity would be $O(tc^2)$. But in practice, it has been found that the increase in time due to the use of dynamic deduction (as opposed to regular deduction) is of very small magnitude.

3.4.4 An Example Run of Dynamic Deduction

Suppose we have a training set as given in Table 3.1 (a “?” represents an unknown value):

Table 3.1. Example Training Set for Dynamic Deduction

Order	X1	X2	X3	X4	X5	CLASS
1	1.10	a	?	1	1.00	F
2	1.20	a	a	0	0.80	F
3	0.80	c	a	1	4.00	F
4	1.01	b	c	1	2.10	T
5	0.96	b	b	0	0.20	T
6	0.43	a	c	1	0.03	T
7	1.51	b	a	?	3.00	T
8	1.03	c	c	0	3.16	F
9	1.10	c	b	1	1.26	F
10	0.96	c	b	0	5.30	T
11	0.94	a	a	0	4.03	T

After running HCV on the above examples, we get the following rules:

The 1st conjunctive rule:

[$X2 \neq \{c\}$]

\rightarrow class T

covering 5 positive examples of the T class.

The 2nd conjunctive rule:

[$X3 = \{b\}$]

and

[$X_4 = \{ 0 \}$]

→ class T

covering 2 positive examples of the T class.

The 3rd conjunctive rule:

[$X_2 \neq \{ b \}$]

→ class F

covering 5 positive examples of the F class.

Suppose we have a test example set as given in Table 3.2.

The following are two instances of applying dynamic deduction (Method 2) on the above test set:

Table 3.2. Example Test Set for Dynamic Deduction

Order	X1	X2	X3	X4	X5	CLASS
1	1.10	a	a	0	1.00	T
2	1.10	a	a	1	1.00	T
3	1.10	b	b	1	1.00	F
4	1.10	a	c	0	1.00	F
5	1.10	b	b	1	1.00	F
6	1.10	a	c	1	1.00	F
7	1.10	b	a	0	1.00	T
8	1.10	c	a	1	1.00	F
9	1.10	c	c	1	1.00	T
10	1.10	a	b	1	1.00	F
11	1.10	a	b	0	1.00	T
12	1.10	b	c	0	1.00	F
13	1.10	b	b	1	1.00	F
14	1.10	b	b	1	1.00	F

- Example 1: Multiple match used. Based on the *EP* values given by the rules for the two classes, class *F* is picked as the classification for example 1 in Table 3.2. However, since this classification is wrong, the class with the next best *EP* value is obtained from the list of classes that match the test example, and this is class *T*. Now, for class *T*, there are two conjunctive rules. The first one i.e

$$X2 \neq \{c\}$$

→ class *T*

contributed more than the second one, towards the estimate of probability for class *T*. Hence its weight is increased from 5 to 6, and the size of the training

set is increased from 11 to 12.

- Example 3: Single match used. The conjunction that matches the given example is

$$X2 \neq \{c\}$$

→ class T .

But the actual class is F . Hence, a no match selection is forced, where the MF value (i.e. the closeness) of the test example for each class is computed in order to pick the one with the highest MF value.

But again, class T is picked since it has the highest MF value for the current test example. So the deduction algorithm tries to find the class with the next best MF value, and this is class F . Class F has only one conjunctive rule, hence its weight is increased from 5 to 6 and the training set size is also incremented by 1.

We proceed with all the examples in the test set and at the end of the deduction loop, we have the following values: the training set size is 25, and the weights of the three conjunctions induced from the training set are 10, 2 and 14, respectively.

So dynamic deduction as performed above adapts the rule weights to be more

representative of the test examples. This can be useful when a representative training set cannot be used for rule induction in a particular domain (in order to capture all possible cases of attribute values and their classifications), but the test examples *are* representative of the example distribution in that domain. At the end of the deduction process, the rule weights used in the *MF* and *EP* measures would be a more accurate reflection of actuality if dynamic deduction were used, whereas if regular deduction had been used, the conjunction weights would still be “frozen” to the inaccurate training set example distribution.

As demonstrated by the experimental results in the next chapter, using performance feedback from the classification of test examples in the manner described above does improve the prediction accuracy of the original rules. So, dynamic deduction provides an alternative where better results can be obtained simply by using the test performance to improve prediction accuracy without going through the process of induction again.

Chapter 4

EXPERIMENTS

This chapter provides a performance comparison between HCV with dynamic deduction and HCV.

4.1 Test Methodology

Ten databases were taken from the University of California at Irvine machine learning repository. These represent a good mix (See Table 4.1), with different proportions of nominal, continuous and unknown attributes and different numbers of classes (ranging from 3 to 19).

First, the HCV algorithm was run on each dataset, to generate rules for the different classes. The parameters used to run HCV were the default parameters, implying method *catlett* was used to discretize continuous attributes. Then, deduction was carried out using the two methods described in the previous chapter for increasing rule weights. The deduction experiments presented the same dataset numerous times in order to test whether the accuracy could be improved by repeated presentations of the same dataset. Thus, in Tables 4.2 to 4.11, “T2” means that the original test example set was presented twice to the deduction algorithms, “T3” means it was

presented three times, and so on. The results are compared to those of regular HCV (i.e. HCV without dynamic deduction). It must be noted that (regular) HCV has already been shown to be competitive with other induction algorithms such as those mentioned in Chapter 2 [24].

4.2 Results

Tables 4.2 to 4.11 and Figures 4.1 to 4.4 summarize the results. No significant increase in time was observed for HCV with dynamic deduction. Hence, time comparisons are not reported. In Figures 4.1 to 4.4, the number of test set presentations are shown on the X-axis, and the rules' accuracy, on the Y-axis. We can see from these tables and figures that:

- As expected, the performance of regular HCV does not change with the number of times the test set is presented.
- Method 1 of dynamic deduction is not very useful. In fact, except in three cases (“Cleve2”, “Swiss” and “Wtp”), its accuracy decreases (or does not change) with the number of times the test set is presented.
- Method 2 of dynamic deduction seems to be very useful. In all cases, without exception, it has increased the accuracy of the results with the number of examples presented. Even in cases where the initial results (given under column “T1”) are slightly worse than those of regular HCV, it has remarkably picked up

Table 4.1. Databases' Characteristics

Database	Instances	Attrrs.	Classes	Maj. Class(%)	Conts. (%)	Val./Attrr.	Unknown (%)
Cleve	303	13	5	54.0	38.0	3.00	2.00
Cleve2	303	13	5	54.0	38.0	3.00	2.00
Glass2	214	9	7	36.00	100.00	n/a	0.00
Hypo	3772	29	5	92.00	24.10	2.18	5.52
Imp85	205	25	7	32.70	60.00	6.00	1.15
Soy	683	35	19	13.50	0	2.80	12.00
Swiss	123	13	5	39	38	3	17
Va	200	13	5	28.00	38.00	3.00	26.80
Wine	178	13	3	40.00	100.00	n/a	0.00
Wtp	174	38	13	61	100.00	n/a	unknown

to exceed the accuracy of regular HCV. For all the cases, results only upto “T8” have been reported in the tables. From Tables 4.3 and 4.8, we see that even though the last reported accuracy (under “T8” column) does not exceed the initial result of regular HCV, there is clearly a trend of increasing accuracy, and this accuracy exceeded that of regular HCV after a few more presentations of the test set. These improved accuracies were observed for database “Cleve2” at T13 (Table 4.3), and for database “Swiss” at T9 (Table 4.8). They are reported under column “Best”.

- The greatest increases in accuracy seem to have occurred for databases “Glass2”, “Imp85”, “Swiss” and “Wtp”. From Table 4.1, we see that the attributes for these databases range from 9 to 38, and the number of classes, from 5 to 13. So Method 2 may not be dependent upon these two characteristics i.e. the number of attributes and the number of classes.
- For database “Hypo”, where 92% of the test examples belonged to just one class, the accuracy of regular deduction itself was very high (97.8%), hence Method 2 could increase the accuracy by only about 0.2% at the end of eight presentations of the test example set.
- Regular HCV itself did not perform well on databases “Swiss” and “Va”, but Method 2 was able to increase its own initial accuracies by 12.4% and 8.3%, respectively, at the end of eight presentations of the test example set. As can

be seen from Figure 4.3, Method 2 increased its own accuracy on “Swiss” by 20% to 30.0% after twenty presentations of the test example set, exceeding that of regular HCV (28.1%).

So, from the tables and figures, we clearly see that the performance of HCV with dynamic deduction (using Method 2) is better than that of regular HCV (i.e. HCV without dynamic deduction) — as the test examples are presented repeatedly, the performance of HCV with dynamic deduction improves, whereas that of regular HCV remains the same.

Table 4.2. Results on CLEVE database

CLEVE	T1	T2	T3	T4	T5	T6	T7	T8	Best
Regular HCV	56.0	56.0	56.0	56.0	56.0	56.0	56.0	56.0	56.0
HCV w/Method1	57.1	56.6	56.4	56.3	56.3	56.2	56.2	56.2	57.1
HCV w/Method2	56.0	56.0	56.4	56.6	56.7	56.8	56.8	56.9	56.9

Table 4.3. Results on CLEVE2 database

CLEVE2	T1	T2	T3	T4	T5	T6	T7	T8	Best
Regular HCV	78.0	78.0	78.0	78.0	78.0	78.0	78.0	78.0	78.0
HCV w/Method1	78.0	78.0	78.0	78.0	78.0	78.0	78.2	78.2	78.2
HCV w/Method2	76.9	76.9	76.9	76.9	76.9	76.9	77.2	77.5	78.1

Table 4.4. Results on GLASS2 database

GLASS2	T1	T2	T3	T4	T5	T6	T7	T8	Best
Regular HCV	72.3	72.3	72.3	72.3	72.3	72.3	72.3	72.3	72.3
HCV w/Method1	64.6	64.6	64.6	64.6	64.6	64.6	64.4	64.2	64.6
HCV w/Method2	72.3	75.4	76.4	76.9	77.2	77.4	77.6	77.7	78.2

Table 4.5. Results on HYPO database

HYPO	T1	T2	T3	T4	T5	T6	T7	T8	Best
Regular HCV	97.8	97.8	97.8	97.8	97.8	97.8	97.8	97.8	97.8
HCV w/Method1	97.7	97.6	97.6	97.6	97.6	97.6	97.6	97.6	97.7
HCV w/Method2	97.8	97.8	97.9	97.9	97.9	97.9	98.0	98.0	98.0

Table 4.6. Results on IMP85 database

IMP85	T1	T2	T3	T4	T5	T6	T7	T8	Best
Regular HCV	62.7	62.7	62.7	62.7	62.7	62.7	62.7	62.7	62.7
HCV w/Method1	62.7	61.9	61.6	61.4	61.4	61.2	61.3	61.2	62.7
HCV w/Method2	62.7	64.4	65.5	66.1	66.4	66.7	66.8	66.9	69.2

Table 4.7. Results on SOY database

SOY	T1	T2	T3	T4	T5	T6	T7	T8	Best
Regular HCV	66.9	66.9	66.9	66.9	66.9	66.9	66.9	66.9	66.9
HCV w/Method1	65.9	65.5	65.4	65.4	65.4	65.4	65.4	65.4	65.9
HCV w/Method2	66.5	67.7	68.3	68.6	68.9	69.1	69.2	69.3	69.3

Table 4.8. Results on SWISS database

SWISS	T1	T2	T3	T4	T5	T6	T7	T8	Best
Regular HCV	28.1	28.1	28.1	28.1	28.1	28.1	28.1	28.1	28.1
HCV w/Method1	25.0	28.1	29.2	28.9	28.8	28.6	28.6	28.9	28.9
HCV w/Method2	25.0	25.0	26.0	26.6	26.9	27.1	27.7	28.1	28.5

Table 4.9. Results on VA database

VA	T1	T2	T3	T4	T5	T6	T7	T8	Best
Regular HCV	26.8	26.8	26.8	26.8	26.8	26.8	26.8	26.8	26.8
HCV w/Method1	25.4	25.4	25.4	25.0	24.8	24.6	24.5	24.5	25.4
HCV w/Method2	25.4	26.1	26.3	26.8	27.0	27.2	27.4	27.5	27.5

Table 4.10. Results on WINE database

WINE	T1	T2	T3	T4	T5	T6	T7	T8	Best
Regular HCV	90.4	90.4	90.4	90.4	90.4	90.4	90.4	90.4	90.4
HCV w/Method1	90.4	90.4	90.4	90.4	90.4	90.4	90.4	90.4	90.4
HCV w/Method2	90.4	90.4	90.4	90.9	91.2	91.3	91.5	91.6	91.6

4.3 A Comparison of Dynamic Deduction Methods

As mentioned in the previous section, from Tables 4.2 through 4.11, we see that of the two dynamic deduction methods used, method 2 performed much better than method 1.

This is because of the way method 1 works: It gets the conjunction list of the rule giving the right classification, and increases the weights of all the conjunctions in this list by an equal amount. This implies that all these conjunctions match the test example equally well. We can see that such a conclusion need not be valid at all

Table 4.11. Results on WTP database

WTP	T1	T2	T3	T4	T5	T6	T7	T8	Best
Regular HCV	55.2	55.2	55.2	55.2	55.2	55.2	55.2	55.2	55.2
HCV w/Method1	50.0	50.6	51.1	51.4	51.6	51.8	52.0	52.2	52.2
HCV w/Method2	57.5	58.0	58.2	58.5	58.9	59.3	59.9	60.3	62.7

times. Also, the relative importance of the “right” conjunction is decreased because the weights of all the other conjunctions are also increased by an equal amount. Finally, since the weights of all the conjunctions are increased for the correct class, the *EP* and *MF* values for this class may turn out to be disproportionately high for other multiple match and no match test examples, leading to more misclassifications. So, as the number of times the test example set is presented increases, the results may deteriorate (as evidenced by the results in Tables 4.2 to 4.11).

On the other hand, in method 2, for each correct classification we increase the weight only of the most important conjunction (defined as the conjunction that contributed the most to the *EP* or *MF* value for the given test example). This refines the conjunction weights to be in line with the distribution of examples in the test set, in turn increasing the rules’ accuracy.

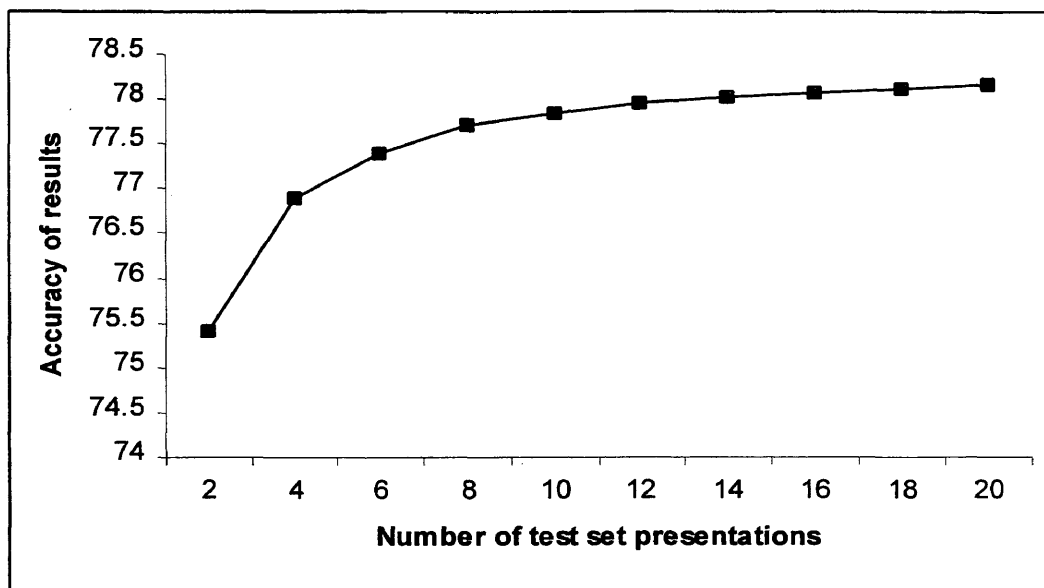


Figure 4.1. Results of HCV using deduction method 2 on: (a) GLASS2

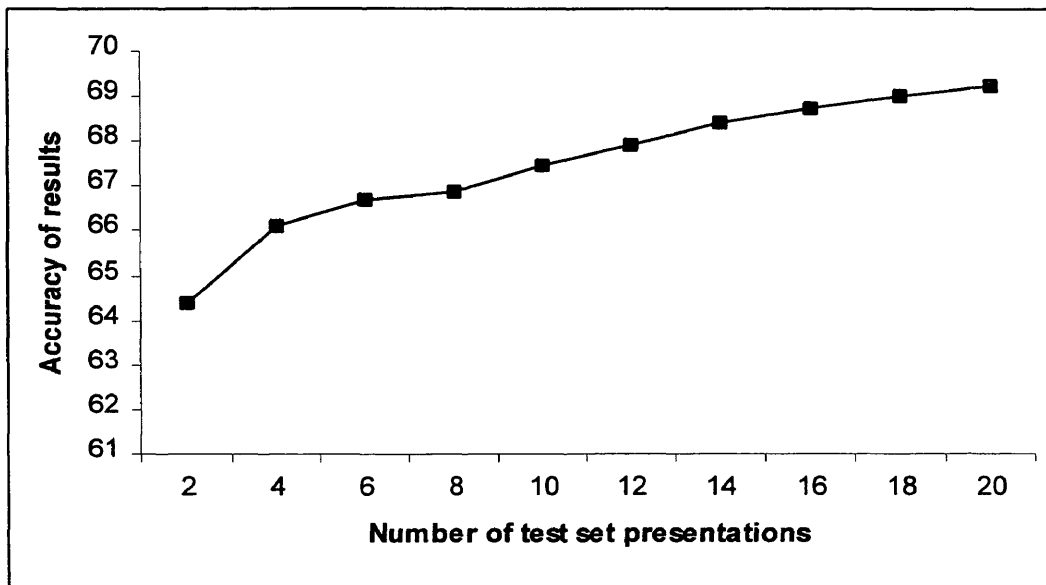


Figure 4.2. Results of HCV using deduction method 2 on: (b) IMP85

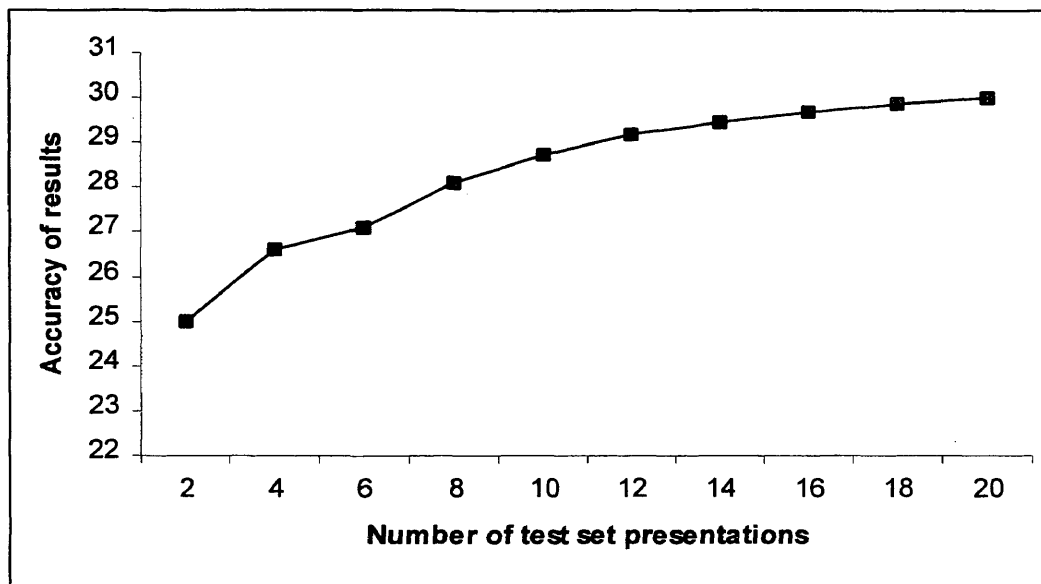


Figure 4.3. Results of HCV using deduction method 2 on: (c) SWISS

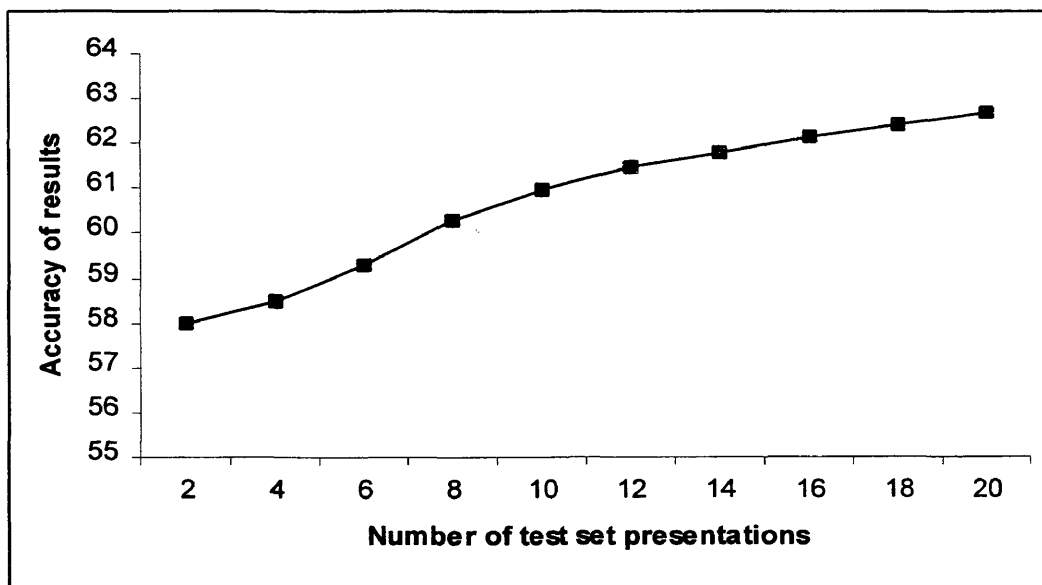


Figure 4.4. Results of HCV using deduction method 2 on: (d) WTP

Chapter 5

SUMMARY AND FUTURE WORK

5.1 Summary

In this thesis, we have taken a highly efficient and competitive induction algorithm, HCV, and explored whether its performance accuracy can be improved at deduction time by dynamically refining rule weights. The results in Chapter 4 prove that this can indeed be done, and thereby stand testimony to the significant advantage of dynamic deduction over regular or “static” deduction. Dynamic deduction can be especially advantageous when the training set used for induction is not wholly representative of the actual distribution of examples in the particular domain in question. At such times, dynamic deduction allows the rules to adapt themselves to the actual distribution, and thus increase their own classification accuracy.

5.2 Future Work

There are several other aspects of the above work that can be expanded upon:

- **Order of test examples.** The test examples are currently presented to the dynamic deduction algorithm in the same sequence at each presentation. This can be changed (for instance, the test set can be shuffled before each presentation)

to see whether the performance of dynamic deduction changes.

- **Rule weighting.** Other ways to increase the weight of the correct rule can also be explored. For example, if a rule has three conjunctions, and the contributions of these conjunctions towards the EP or MF value of a test example are 50%, 30% and 20%, respectively, we can possibly increase their weights by 5, 3, and 2 respectively, and increase the weight of the training set by ten.
- **New test sets.** Testing the accuracy of the dynamically refined rules on a new, unseen test set (i.e., different from the one used for dynamic deduction) is another aspect that might provide a further evaluation of the usefulness of dynamic deduction.
- **Mixing training and test sets.** Comparing rule accuracies in the following two cases may also provide additional insight into dynamic deduction: 1) Adding the test examples to the training set for regular induction and deduction processes, 2) Inducing rules from a training set and refining them using the test set, in the manner described in this thesis.
- **Cross Validation.** Six-fold cross-validation, i.e., dividing the entire collection of examples (including “training” and “test” sets) into six parts, and by turns, treating one of these six parts as a test set while using the other five parts for training, may also give further proof for dynamic deduction.

REFERENCES

- [1] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proceedings of the ACM SIGMOID International Conference on Management of Data*, 1993.
- [2] C.M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press Inc., New York, 1997.
- [3] E. Bloedorn, R.S. Michalski, and J. Wnek. Aq17 – a multistrategy constructive learning system. Technical report, George Mason University, 1992.
- [4] B. Cestnik, I. Kononenko, and I. Bratko. *Progress in Machine Learning*. Wilm-slow: Sigma Press, England, 1987.
- [5] P. Clark and T. Niblett. The cn2 induction algorithm. *Machine Learning*, 3, 1989.
- [6] David Heckerman. Bayesian networks for data mining. *Data Mining and Knowledge Discovery*, 1(1), 1997.
- [7] J. Hong. Ae1: An extension matrix approximate method for the general covering problem. *International Journal of Computer and Information Sciences*, 14(6), 1985.

- [8] J. Hong. Learning from examples and a multi-purpose learning. *Chinese Journal of Computers*, 12, 1989.
- [9] J. Hong. A new attribute-based learning algorithm and a comparison with existing algorithms. *Journal of Computer Science and Technology*, 4, 1989.
- [10] E.B. Hunt, J. Martin, and P.T. Stone. *Experiments in Induction*. Academic Press, New York, 1966.
- [11] G.J. Klir and B. Yuan. *Fuzzy Sets and Fuzzy Logic, Theory and Applications*. Prentice Hall Inc., NJ, 1995.
- [12] R. S. Michalski. Variable-valued logic and its applications to pattern recognition and machine learning. *Computer Science and Multiple-Valued Logic Theory and Applications*, 1975.
- [13] R. S. Michalski, J.R. Hong, and I. Mozetic. Aq15: Incremental learning of attribute-based descriptions from examples, the method and user's guide. Technical report, University of Illinois at Urbana-Champaign, 1986.
- [14] T. Mitchell. Version spaces: A candidate elimination approach to rule learning. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, 1977.
- [15] T. Mitchell. *Version Spaces: An Approach to Concept Learning*. PhD thesis, Stanford University, 1978.

- [16] J. R. Quinlan. Discovering rules by induction from large collections of examples. In *Introductory Readings in Expert Systems*. Gordon and Breach, London, 1979.
- [17] J.R. Quinlan, P.J. Compton, K.A. Horn, and L. Lazarus. *Applications of Expert Systems*. Addison-Wesley, 1987.
- [18] S. Russell and P. Norwig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, New Jersey, USA, 1995.
- [19] J.C. Schlimmer and D. Fisher. A case study of incremental concept induction. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, 1986.
- [20] P.E. Utgoff. Incremental induction of decision trees. *Machine Learning*, 4, 1989.
- [21] P. Winston. *Learning Structural Description from Examples*. PhD thesis, Massachusetts Institute of Technology, 1970.
- [22] X. Wu. Optimization problems in extension matrixes. *Science in China*, 35(2), 1992.
- [23] X. Wu. *Knowledge Acquisition from Databases*. Ablex Publishing Corp., USA, 1995.
- [24] X. Wu. Rule induction with extension matrices. *Journal of the American Society for Information Science*, 49(5), 1998.

- [25] L. Yi. Clustering by agglomeration of fuzzy object granules with application to web personalization. Master's thesis, Colorado School of Mines, Golden, CO, USA, 1999.
- [26] T. Zhang, R. Ramakrishnan, and M. Livny. Birch: A new data clustering algorithm and its applications. *Data Mining and Knowledge Discovery*, 1(2), 1997.