ABSTRACT

SACHDEV, MANISH P. On Learning of Ceteris Paribus Preference Theories. (Under the direction of Professor Jon Doyle).

The problem of preference elicitation has been of interest for a long time. While traditional methods of asking a set of relevant questions are still useful, the availability of user-preference data from the web has led to substantial attention to the notion of preference mining. In this thesis, we consider the problem of learning logical preference theories that express preference orderings over alternatives.

We present learning algorithms which accept as input a set of comparisons between pairs of complete descriptions of world states. Our first algorithm, that performs exact learning, accepts the complete set of preference orderings for a theory and generates a theory which provides the same ordering of states as the input. This process can require looking at an exponential number of data points. We then look at more realistic approximation algorithms and analyze the complexity of the learning problem under the framework of Probably Approximately Correct (PAC) learning. We then describe approximation algorithms for learning high-level summaries of the underlying theory.
On Learning of *Ceteris Paribus* Preference Theories

by

Manish Sachdev

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Computer Science

Raleigh, North Carolina

2007

Approved By:

______________________________  ________________________________
Dr. Dennis Bahler                Dr. Munindar P. Singh

______________________________
Dr. Jon Doyle
Chair of Advisory Committee
Dedication

To my parents...
Biography

Manish Sachdev was born on November 28, 1983 in Mumbai, India. He obtained his bachelor’s degree in Computer Engineering at Thadomal Shahani Engineering College, an affiliate of the University of Mumbai, in June, 2005. Since then, he has been a master’s student, majoring in Computer Science, at North Carolina State University, Raleigh, USA. After graduation, he plans to join Microsoft Corporation as a Software Design Engineer, at Redmond, Washington.
Acknowledgments

I would like to acknowledge the support and help of several people, some who have been instrumental in this research effort and others who have helped me get the most out of my stay at NCSU.

First and foremost, I would like to thank my family for all their support throughout my education, and specially through this thesis. Their faith in me is an invaluable source of motivation which has helped me achieve all that I have.

I would like to thank my adviser, Dr. Jon Doyle, for his guidance, encouragement and tremendous patience. His vision and approach are an inspiration I greatly value. It has been a great privilege to work under his supervision. I would like to thank Dr. Munindar P. Singh and Dr. Dennis Bahler for serving on my thesis committee. I would also like to thank Dr. Carla Savage for her help and guidance in this thesis.

Last, but certainly not the least, I wish to acknowledge my friends and colleagues here at NC State. I would like to thank Aditya Dalmia for all his help and for the endless enlightening discussions from which I have learned a lot. A thanks to my friends, Vinayak Devasthali, Siddharth Bhai, Jay Kothari and Ajit Gopalakrishnan who have been very supportive and have made my stay here most enjoyable. Among others, I would like to thank my colleague Satya Madhav Kompella for some fun times and great projects on which we partnered.
# Contents

List of Figures vii

1 Introduction 1

2 The Problem 3

3 *Ceteris Paribus* Preferences 7
  3.1 Features and Worlds ........................................... 7
  3.2 Preference Graph ............................................. 9
  3.3 Intermediate Representation ................................. 10

4 Learning Preference Theories 14
  4.1 Learning of Concepts .......................................... 14
  4.2 Learning Environment ......................................... 15

5 Exact Learning 18
  5.1 Notation ...................................................... 19
  5.2 From Preference Graph to Intermediate Representation ....... 22
    5.2.1 Complexity Analysis ..................................... 26
  5.3 Analysis: Learned Theory ..................................... 28
  5.4 Avoiding Redundant Theories ................................ 29
  5.5 From Intermediate Representation to Preference Logic ....... 32

6 Approximate Learning 34
  6.1 Preference Learning as a PAC problem ....................... 35
  6.2 Single Rule Theories .......................................... 37
    6.2.1 Algorithm ............................................... 38
  6.3 Multiple Rule Theories ...................................... 39
    6.3.1 Structural Assumptions ................................ 40
    6.3.2 Learning with Learner-Initiated Queries ............... 42
    6.3.3 Algorithm .............................................. 44
    6.3.4 Question Formulation .................................. 45
    6.3.5 Complexity Analysis ................................... 46
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.3.6</td>
<td>Error in Learning</td>
<td>47</td>
</tr>
<tr>
<td>6.3.7</td>
<td>An Alternate Approach</td>
<td>50</td>
</tr>
<tr>
<td>6.3.8</td>
<td>Going Beyond Pairwise Tradeoffs</td>
<td>51</td>
</tr>
<tr>
<td>6.4</td>
<td>Summary and Further Work</td>
<td>55</td>
</tr>
</tbody>
</table>

### 7 Conclusion

58

### Bibliography

59
List of Figures

2.1 Problem Space for Preference Learning ........................................... 4

3.1 Preference Graph for Lexicographical Ordering ................................. 10

5.1 $S$ Matrix - Avoiding Redundant Rules ........................................... 32
Chapter 1

Introduction

This research work is focused on the problem of preferences in decision making, specifically looking at the learning problem involved in this subject.

There has been a lot of research in decision theory, exploring means of expressing an agent's preferences and formalizing the process of making good decisions. The much-cited book by Keeney and Raiffa ([KR76]) talks about what is referred to as Multi Attribute Utility theory, in which we look at worlds described in terms of various attributes, each having their own marginal utility gains. It discusses means of factoring the agent’s preferences over these attributes, thereby choosing world states that make economic sense, under the given preferences.

The work of economists has been extended and modified in the domain of artificial intelligence by several researchers, along several different lines. Some of the early works involved formalizing these notions in a logical language, allowing us to talk in terms of preference orderings, rather than real-valued utility functions. Doyle and Wellman in [WD91] gave a means to bridge the semantics of goals popular from the classical planning systems of AI, to preference ordering over world states. Continuing on this work, the authors in [DSW91] formalized an all else being equal or ceteris paribus language of preferences. Other important works in this area included the work on conditional preference networks, or CP-Nets ([BBHP99]) which gives a means of representing and reasoning about an agent’s preferences.

While a substantial amount of research in the area talks about different represen-
ation and reasoning mechanisms, one problem that has been of interest since the early works of economic theories has been the elicitation of preferences from the users. The traditional technique, which is still relevant, has been to ask questions to the agent, in order to model the underlying preference structure. With the involvement of computers and indirect means of gathering preference information, such as buying patterns of users, the idea of preference mining has gained relevance. In this thesis we look at the general problem of learning a preference theory, by observing or asking the agent’s preference between pairs of world states.

The following is roughly the structure of the thesis: In the next chapter, we discuss the problem a little more in detail and outline our approach for the same. The theoretical discussion follows this, giving the background notation and definitions followed by two broad approaches towards a solution. We conclude with a discussion of related research problems and further work.
Chapter 2

The Problem

The core problem explored here is that of learning the preferences of an agent, by observing its actions or decisions, or by querying the agent. This chapter sketches the approach and identifies some of the interesting subproblems here.

We consider preference theories expressing ordinal utility functions over states of a world. The agent’s preferences are presented to the learner in the form of comparisons between different states of the world. Thus, each learning instance would be a statement expressing a preference of one specification of the world over another. We discuss the formal representation of world states in the next chapter.

In order to represent comparisons between world states, we employ a graphical notation patterned after the preference graph described in [McG02]. The graph consists of a vertex for each state of the world and a set of directed edges between vertices to depict the preference of one node to another.

We employ the propositional language described in [DSW91] for expressing preference theories. A preference theory is a set of preference statements describing a preference ordering over world states. The expected output of learning is a description of the agent’s preferences in this language.

The propositional language and the preference graph are discussed formally in the following chapters. In order to simplify our discussion in this chapter, we denote a preference theory as $PT$. We denote the preference graph corresponding to the agent’s preferences by $G$. In [McG02], the author describes an intermediate representation, denoted here as $IR$. 
which is a formal language used as an intermediate step to translate a preference theory to a preference graph. In this thesis, we use the intermediate representation for some of the learning algorithms. The following diagram shows the overall picture, depicting the position of the propositional language, intermediate representation and preference graph. It also shows some problems of research interest, some of which are discussed in this thesis.

![Diagram showing the problem space for preference learning](image)

Figure 2.1: Problem Space for Preference Learning

In the preceding diagram, $PT$ is the original underlying or target theory describing an agent’s preferences. The forward translation translates $PT$ to a theory in the intermediate representation $IR$ and then converts this into a preference graph $G$. This is described in [McG02]. We discuss this formally in the next chapter.

Several interesting questions arise going back from the preference graph to a theory
in the propositional language. We discuss some of these here:

1. **Exact Learning**: The first question of interest is whether we can model the agent’s preferences exactly. In our context this involves looking at some or all of the preference comparisons in the preference graph and generating a theory \( PT' \) which is *equivalent* to \( PT \), i.e., generates the same preference orderings as the theory \( PT \). The preceding diagram shows several paths for exact learning:

(a) *Exact IR with conversion*: Learn an exact theory in the intermediate representation and *convert* it to a theory in the propositional language. As is explained in the following chapters, in the general case, this approach would generate a theory having exponentially many more statements than the underlying theory, \( PT \).

(b) *Exact IR with reduction*: Learn an exact theory in the intermediate representation and *reduce* it to a theory in the propositional language. In this case, multiple statements in the intermediate representation would be reduced to a single statement in the propositional language. For simplicity and brevity, we do not considered this problem in detail in this thesis.

(c) *Exact PT*: Learn an exact theory directly in the propositional language. We do not consider this approach in this thesis. We note here that the use of the intermediate representation is not necessary. We use it for convenience and as an extension to the work done in [McG02]. It may be a better approach to skip the intermediate representation. We, however, do not analyze the same.

2. **Approximate Learning**: In most cases, a preference theory entails an exponential number of preference orderings, making the exact learning algorithm discussed in this thesis infeasible due to its high time complexity. This motivates the question of learning *approximations* of the original theory. The idea is to learn theories that have some bounded error. In this thesis, we consider the *Probably Approximately Correct* learning framework ([Va84]) wherein we try to learn, with high probability, a theory that has a bounded error. Once again, we can try to learn the theory using the intermediate representation or directly in the propositional language. In this thesis, we only consider learning the theory directly in terms of statements in the propositional language. As in the case of exact learning, we do not analyze which approach, if any,
Another related problem which is not depicted here is generating summaries of a theory. Given a theory $PT$ and its corresponding graph $G$, can we use the preference graph to learn simpler, exact or approximate, summaries of the theory? We discuss this problem as part of the approximate learning algorithms. We also briefly discuss the learning problem where the output is not in the form of statements in the propositional language. Specifically, we consider the graphical representation of conditional preference networks described in [BBHP99].

In the next chapter, we first formalize the notion of world states and preference theories. We discuss in detail the propositional language, the preference graph and forward translation using the intermediate representation. We then discuss an algorithm for exact learning of a preference theory given a preference graph.

The following chapter discusses approximation algorithms. We define the problem of learning preference theories as a probably approximately correct (PAC) learning problem and analyze the complexity of the problem and give some algorithms to approximate a theory. We look at means of learning a summary of the agent’s preference and analyze the same for certain different structures of the underlying theory.

We conclude by discussing some possible directions of extending this work.
Chapter 3

Ceteris Paribus Preferences

In this chapter, we formally discuss the notion and semantics of ceteris paribus preferences. The following sections discuss background concepts and notations which would be used throughout the discussion.

3.1 Features and Worlds

In order to express preferences over outcomes, we describe the outcomes or states of the world in terms of binary features. A complete description of a world state consists of an assignment of truth values to all the features in the feature set $F$. Although the cardinality of $F$ can theoretically be infinite, we make the simplifying assumption that $F$ is finite, consisting of only those features over which we wish to express preference information or are otherwise relevant in the relative desirability of world states.

We employ the language $\mathcal{L}$ described in [DSW91] with the restriction of allowing only the logical operators for negation $\neg$ and conjunction $\land$. The language $\mathcal{L}$ consists of propositional statements over the set of features $F$. The set of literals, denoted $\text{literals}(F)$, consists of the features and their negations. A model of the world is a complete and consistent set of literals, i.e., for each feature $f \in F$, the set contains either $f$ or $\neg f$. We denote $\mathcal{M}$ as the set of all models. For a given model, $f(m)$ denotes the truth value assigned to feature $f$ by model $m$. 
We say a model \( m \) satisfies a statement \( p \) of \( \mathcal{L} \), if the assignment in \( m \) makes the statement true. We denote this \( m \models p \). We denote \( \llbracket p \rrbracket \) to be the set of all models \( m \) such that \( m \models p \). Thus,

\[
\llbracket p \rrbracket = \{ m \in \mathcal{M} \mid m \models p \}
\tag{3.1}
\]

Preferences are expressed by a preorder (a reflexive and transitive relation) \( \succeq \) over the set of models \( \mathcal{M} \). We say \( m \succeq m' \) to mean outcome \( m \) is at least as preferred as \( m' \). We say here that \( m \) is weakly preferred to \( m' \). We write \( m \succ m' \) to express a strict preference of \( m \) over \( m' \). Formally, \( m \succ m' \), if \( m \succeq m' \) and \( m' \not\succeq m \).

In order to specify preference information in terms of statements of the language \( \mathcal{L} \), we use the concept of model modifications explained in the following.

We define the support of a statement \( p \) as the minimal set of features determining the truth of \( p \) and denote it as \( s(p) \). For example, the \( f_1 f_2 \), we have \( s(p) = \{ f_1, f_2 \} \).

We say model \( m' \) is equivalent modulo \( p \) to model \( m \), if they are the same outside the support of \( p \). Formally, \( m' \equiv m \mod p \), if \( m \setminus \text{literals}(s(p)) = m' \setminus \text{literals}(s(p)) \). Lastly, we write \( m[p] \) to denote the set of model modifications of model \( m \), making \( p \) true and define it as the set of all models \( m' \) having \( f(m') = f(m) \) for all features \( f \notin s(p) \). Thus,

\[
m[p] = \{ m' \in \mathcal{M} \mid m' \equiv m \mod p \text{ and } m' \models p \}
\tag{3.2}
\]

The preference relation is extended to statements \( p \) and \( q \) in \( \mathcal{L} \) in terms of the model modifications satisfying \( p \) and \( q \). We say, \( p \succ q \) just in case, for all \( m \in \mathcal{M} \), if \( m' \in m[p \land \neg q] \) and \( m'' \in m[\neg p \land q] \), then \( m' \succ m'' \) holds. For the case of strict preference, the definition in [DSW91] states that \( p \succ q \) holds only if, \( p \succeq q \) and if, for some \( m \in \mathcal{M} \), there exists \( m' \in m[p \land \neg q] \) and \( m'' \in m[\neg p \land q] \), such that \( m' \succ m'' \). In this thesis, we take a stronger definition of definition of strict preference, stating \( p \succ q \) if, for all \( m \in \mathcal{M} \), there exist \( m' \in m[p \land \neg q] \) and \( m'' \in m[\neg p \land q] \), such that \( m' \succ m'' \). We choose a stricter definition to avoid the complexity of indifference.

This defines formally the notion of ceteris paribus ("all else being equal") preferences. We refer to the statement \( p \succ q \) as a preference rule, the semantics being that models satisfying \( p \) are strictly preferred to models satisfying \( q \), ceteris paribus. For a preference rule \( r = p \succ q \) we refer to the more preferred side \( p \) as the greater side, denoted \( GS(r) \) and the less preferred side \( q \) as the lesser side, denoted \( LS(r) \).

One can express conditional preferences of the form \( t \rightarrow (p \succ q) \), in the complete
language by use of implications. The semantics of the above statement is that, whenever \( t \) holds, we strictly prefer the proposition \( p \) to \( q \). Statements of this form can be converted to simple rules with conjunctions of the form \( t \land p \succ t \land q \), provided the support of \( t \) is disjoint from that of \( p \) and \( q \). We assume this restriction on the preference rules.

A given preference rule, \( r \), describes an ordering between several pairs of models. We can look at the definition of a rule in the reverse direction to say, given a rule \( r = p \succ q \) in \( L \), we say \( r \) entails \( m' \succ m'' \) iff \( m' \in m[p \land \neg q] \) and \( m'' \in m[\neg p \land q] \) for some model \( m \) in \( M \).

If rule \( r \) entails \( m \succ m' \) for some \( m, m' \in M \), we write it as \( m \succ_r m' \). We use this notion to define the meaning of a rule. We define the meaning of a rule \( r \), denoted \([r]\), as the set of model pairs \((m, m')\), such that \( m \succ_r m' \). Formally,

\[
[r] = \{(m, m') | m \succ_r m'\} \tag{3.3}
\]

We note here that for each rule \( r = p \succ q \), \([p \land \neg q]\) is disjoint from \([\neg p \land q]\), and hence, the transitive closure \([r]^*\) of the above defined set is the set \([r]\) itself.

A preference theory \( T \) is a set of rules describing an agent’s preferences. The notion of meaning of a rule can be extended to that of a theory \( T \) as the combined meaning of all rules in the theory. We denote \([T]\) to be the simple union over the meanings of the rules \( r \in T \). Formally, \([T] = \bigcup_{r \in T} [r]\). In case of general theories, \( T \), the union of meanings of all rules does not give the entire set of preference orderings. In general, we have \([T]^* \neq [T]\). In cases of sets of rule, the transitive closure over the combined meaning of the constituent rules represents the complete set of pairwise comparisons of world states. Thus, the meaning of a theory is given by the transitive closure \([T]^*\). Although we would focus on the transitive closure \([T]^*\) for learning preference theories, the simple union set \([T]\) is also of interest for a different problem to which we will return in section 6.4.

Lastly, we say a preference theory \( T \) is consistent iff there exist no two models \( m \) and \( m' \) in \( M \) such that \((m, m') \in [T]^*\) and \((m', m) \in [T]^*\).

### 3.2 Preference Graph

We capture all the direct comparison of models using a preference graph similar to the one described in [McG02].
We denote the preference graph $G(V, E)$ of a preference theory $T$, where the set of vertices $V$ is the set of models $\mathcal{M}$. The edge set is constructed from the entire set of model comparisons given by the theory, i.e. $\llbracket T \rrbracket^*$, by having an edge directed from each $m$ to $m'$ such that $(m, m') \in \llbracket T \rrbracket^*$. Thus, $E = \llbracket T \rrbracket^*$.

Figure 3.1 shows a sample preference graph. The feature set in this example is $F = \{f_1, f_2\}$ and the preferences captured is a lexicographical ordering of models, having $\{f_1, f_2\}$ as the most preferred and $\{\neg f_1, \neg f_2\}$ as the least preferred model.

![Preference Graph for Lexicographical Ordering](image)

One way to construct the preference graph for a theory would be to enumerate all model comparisons that hold for each rule in the theory, and take a transitive closure of the orderings, giving us the edge set. A systematic approach to this is described in [McG02], by means of an intermediate representation. We explain the intermediate representation in the following section.

### 3.3 Intermediate Representation

We discuss here the forward translation of rules in the propositional language to the preference graph discussed in the preceding section. The intermediate language has also been employed in one of the learning algorithms discussed ahead.
The intermediate representation in [McG02] employs a language over what is referred to as a feature vector. A feature vector is an ordered list of features relevant in the domain, represented as \( V = \langle f_1, f_2, \ldots, f_N \rangle \), where \( f_i \in F \) and \( N = |F| \). We define a language over the domain of the feature vector, denoted \( \mathcal{L}(V) \). The alphabet for the language \( \mathcal{L}(V) \) is \( \Gamma = \{0, 1, \ast\} \). Each statement in \( \mathcal{L}(V) \) now consists of vectors of size \( N \) drawn from the alphabet, \( \Gamma \). Thus \( \langle 1, 1, \ast \rangle \) belongs to \( \mathcal{L}(\langle f_1, f_2, f_3 \rangle) \). In most of the discussion, we drop the vector notation and write statements in \( \mathcal{L}(V) \) as a string of characters. Thus, \( \langle 1, 1, \ast \rangle \) becomes \( 11\ast \). The value in \( \Gamma \) assigned to feature \( f \) by a statement \( p \) of \( \mathcal{L}(V) \) is denoted by \( f(p) \).

When a statement in \( \mathcal{L}(V) \) in expressed in the restricted alphabet \( \Gamma' = \{0, 1\} \), it denotes a model of \( \mathcal{L}(V) \). It is looked upon as a complete specification of the truth values of the features, such that 0 and 1 represent false and true respectively. We denote the set of all models of \( \mathcal{L}(V) \) by \( \mathcal{M}(V) \). If \( m \) is a model of \( \mathcal{L}(V) \), \( f(m) \) denotes the value in \( \Gamma' \) assigned to feature \( f \) by \( m \). We say model \( m \) of \( \mathcal{L}(V) \) satisfies a statement \( p \in \mathcal{L}(V) \), denoted \( m \models p \), if \( m \) assigns the same values as \( p \) to those features that do not have \( \ast \) letters in \( p \). Formly, \( m \models p \) only when \( f_i(m) = f_i(p) \), for all \( 1 \leq i \leq N \) such that \( f_i(p) \neq \ast \). For example, \( 110 \models 1\ast0 \) and \( 100 \models 1\ast0 \).

We define a language to specify preference rules in \( \mathcal{L}(V) \), denoted \( \mathcal{L}_r(V) \) to consist of pairs of statements of the form \( p \succ q \), where \( p, q \in \mathcal{L}(V) \) and \( p \) and \( q \) have matching \( \ast \) values. Formly, for each feature \( f \), \( f(p) = \ast \), if and only if \( f(q) = \ast \). Thus, \( 1\ast0 \succ 0\ast0 \in \mathcal{L}_r(V) \) and \( 10\ast \succ 0\ast0 \notin \mathcal{L}_r(V) \). In parallel to the rules in \( \mathcal{L} \), if \( r = p \succ q \), we refer to \( p \) and \( q \) as \( GS(r) \) and \( LS(r) \), respectively.

We say a pair of models \( (m, m') \) of \( \mathcal{L}(V) \) satisfies a rule \( r \) in \( \mathcal{L}_r(V) \), denoted \( (m, m') \models r \), whenever \( m \models GS(r) \) and \( m' \models LS(r) \) and \( m \) and \( m' \) assign the same values to those features which are assigned \( \ast \) letters by \( p \) and \( q \). Formly, \( (m, m') \models p \succ q \) only if, \( m \models p, m' \models q \) and \( f(m) = f(m') \) for all features \( f \) such that such that \( f(p) = f(q) = \ast \). The meaning \( [r] \) of a rule in \( \mathcal{L}_r(V) \) is the set of all model pairs satisfying the rule. As an example, consider the rule \( r = 10\ast\ast \succ 01\ast\ast \). We have, \( [r] = \{(1000, 0100), (1001, 0101), (1010, 0110), (1011, 0111)\} \). In case of a set of rules \( R \), the meaning of \( R \) is the transitive closure over the union of the meanings of all the rules in \( R \), i.e., \( (\bigcup_{r \in R} [r])^* \).

We now consider the translation of rules from the propositional language \( \mathcal{L} \) to \( \mathcal{L}_r(V) \). We first discuss the translation of models from \( \mathcal{M} \) to those of \( \mathcal{M}(V) \). The model
translation is achieved by the mapping $\alpha : M \to M(\mathcal{V})$ defined so that $f(\alpha(m)) = 1$ if $f \in m$ and $f(\alpha(m)) = 0$ if $\neg f \in m$, for all $f \in F$.

We next define model restriction. We define $M(S)$ to be the set of models restricted to a subset of features $S \subseteq F$. Consider a model $m$ of $M(S)$ and $S' \subseteq S$. We write $m \upharpoonright S'$ to denote the model $m'$ of $M(S')$ assigning the same values to the features in $S'$ as $m$. If we have $S' \subseteq S$, for some set of features $S$, we say model $m$ in $M(S)$ satisfies a model $m'$ in $M(S')$, written $m \models m'$, if it is the case that $m' = m \upharpoonright S'$.

The translation of a rule from $L$ to a set of rules in $L_r(\mathcal{V})$ requires the meaning of the rule to be retained. Thus, the model pairs generated by $r \in L$ should also be contained in the meaning of the translated rule set, i.e., $[R]^* = [\mu]$. We note here that if set $R$ is a translation of a single rule $r \in L$, $[R]^* = [R]$.

The translation involves the use of what is referred to as the characteristic model of statements in the intermediate language. We denote $\mu(p)$ as the characteristic model of $p$, where $p$ is a statement in $L(\mathcal{V})$ and define it as:

$$\mu(p) = \{ f \mid f(p) = 1 \} \cup \{ \neg f \mid f(p) = 0 \} \tag{3.4}$$

We note here that $\mu(p)$ is a model in $M(s(p))$, where $s(p)$ is the support of statement $p$.

Consider a rule $r = p \succ q$ in $L$. This rule specifies a preference of the models satisfying $p \land \neg q$ over those satisfying $\neg p \land q$, all else being equal. Let $s(r)$ denote the support for rule $r$, obtained as $s(r) = s(p \land \neg q) \cup s(\neg p \land q)$. Note here that these features would be the support features for all the rules obtained in the translated rule set $R$. Let $W_G(r)$ and $W_L(r)$ be the set of models in $M(s(r))$ satisfying $p \land \neg q$ and $\neg p \land q$, respectively. Formally,

$$W_G(r) = \{ w \in M(s(r)) \mid w \models p \land \neg q \}$$

Similarly,

$$W_L(r) = \{ w \in M(s(r)) \mid w \models \neg p \land q \}$$

We now define the statements in $L(\mathcal{V})$ for the greater and lesser sides as follows:

$$W_G'(r) = \{ w \in L(\mathcal{V}) \mid (\mu(w) \upharpoonright s(r)) \in W_G(r) \}$$

$$W_L'(r) = \{ w \in L(\mathcal{V}) \mid (\mu(w) \upharpoonright s(r)) \in W_L(r) \}$$
Lastly, we complete the translation by generating the rules in the intermediate representation as \( w'_G \succ w'_L \) for all \( w'_G \) in \( W'_G \) and \( w'_L \) in \( W'_L \). Formally,

\[
R(r) = \{ w'_G \succ w'_L \mid w'_G \in W'_G(r), \ w'_L \in W'_L(r) \} 
\]  

(3.5)

As an illustration, say \( F = \{ f_1, f_2, f_3, f_4 \} \). Consider the translation of rule \( r = f_1 \succ f_2 \land f_3 \). Thus, all models satisfying \( f_1 \land \neg(f_2 \land f_3) \) are preferred over those satisfying \( \neg f_1 \land f_2 \land f_3 \). We get, \( W_G(r) = \{100, 101, 110\} \) and \( W_L(r) = \{011\} \). This gives, \( W'_G(r) = \{100*, 101*, 110*\} \) and \( W'_L(r) = \{011*\} \). Finally, the translated set of rules is, \( R(r) = \{100* \succ 011*, 101* \succ 011*, 110* \succ 011*\} \).

The above explains generation of a set of rules in IR from a rule in \( L \). We require some additional steps to generate edges in the final graph. For each rule \( r \) in \( R \) above, we generate \( [r] \) by assigning all possible combinations of 0 and 1 to those features that have been assigned \( * \), as was illustrated with an example earlier. We can now generate the union over these to give \( [T] \). Finally, we get the set of edges as the transitive closure \( [T]^* \). We note here that in this case, for each \((m, m') \in [T]^* \) the models \( m, m' \) belong to \( \mathcal{M}(\mathcal{V}) \). Since our graph uses models from \( \mathcal{M} \) as vertices, we need to apply a reverse mapping of \( \alpha \) defined earlier. We denote this mapping \( \alpha' : \mathcal{M}(\mathcal{V}) \to \mathcal{M} \), defined so that \( f(\alpha'(m)) = true \) if \( f(m) = 1 \) and \( f(\alpha'(m)) = false \) if \( f(m) = 0 \), for all \( f \in F \).

This completes the forward translation of preference theories in \( L \) to the corresponding preference graphs.
Chapter 4

Learning Preference Theories

In this thesis, we analyze and give algorithms for the problem of learning a preference theory expressed in $\mathcal{L}$. This chapter formalizes this problem. In the following section we discuss some background concepts and notation from theory of machine learning. We then go ahead and discuss the learning environment for the problem under consideration.

4.1 Learning of Concepts

Concepts can be thought of as descriptions of classes of objects or events. We consider the case where the definition of a concept can be expressed as a conjunction of attribute values. Learning a concept entails learning such a definition of a concept. For example, one can consider the problem of learning a class of mammals given a set of examples consisting of animals described by a set of attributes and classification specifying whether the animal is a mammal. The set of examples is called the training set and it typically consists of positive and negative examples, where a positive example is one that belongs to the concept (in this example, mammals) and a negative example is one which does not.

The actual definition can be encoded in several ways, such as decision trees, weights of a neural network, or a set of classification rules. A generic way of looking at learning of a concept is to learn a function over the set of attributes which maps each instance of the concept to 1 and all other instances to 0.
Considering the target concept to be a function mapping each instance to \( \{0, 1\} \), we can look at the problem of learning as a search through the set of all possible functions for the one that fits the training data the best. In other words, the process of learning now involves searching for the function that best agrees with the training set. The entire set of functions is called the hypothesis space, denoted \( H \).

We now define some notions related to learning. We denote the set of all instances as \( X \). A learner, \( L \) is any agent that is given the training set and outputs a hypothesis. The accuracy or true error is defined in terms of the ratio of misclassified instances to the total set of instances. We also define training error as the fraction of training instances classified erroneously.

In the following section, we give the notation to be used in the discussion of preference learning and formalize the learning environment and desired properties of the output.

### 4.2 Learning Environment

In the present case of preference learning, the learner \( L \) takes input in the form of direct comparisons between models. The training instances are pairs of models of the world \((m, m')\) described over the set of features \( F \), such that \( m \succ m' \). As described earlier, a preference graph contains models \( m \) in \( \mathcal{M} \) as vertices and the graph edges represent preference comparisons. Thus, we use these edges as the training set for the learner. Formally \( X \subseteq E \), where \( E \) is the edge set of the preference graph. We note here that the mapping \( \alpha \) from \( \mathcal{M} \) to \( \mathcal{M}(\mathcal{V}) \) is a bijection, since each vector in \( \mathcal{V} \) has length \( N = |F| \). Keeping this in mind, we use either notation interchangeably, as convenient in the context of the discussion.

The target concept is a preference theory that would generate the same model comparisons. In the context of the preceding discussion, a preference theory \( T \) can be looked upon as describing a function mapping each pair of models \((m, m')\) to \( \{0, 1\} \), where 0 denotes the absence of the edge \((m, m')\) in the preference graph corresponding to \( T \), and 1 denotes its presence. The output of the learner is a set of rules or theory \( T' \) in \( \mathcal{L} \), which best describes the input training data. We note here that since the underlying preference theory is consistent, each training example \((m, m')\) consists of a positive example stating \( m \succ m' \) and a negative example stating \( m' \nprec m \).
We now look at some of desired properties of the output theory. The properties that one may want of any learner are speed and accuracy of learning. While in our case, the speed of learning is merely the asymptotic complexity of the algorithm, the notion of accuracy or more generally the quality of the output needs to be defined formally.

We restrict the learning algorithm to learn consistent theories as defined earlier. We now define the true error for the learned theory. The measure that we employ is the number of erroneous comparisons caused by the output theory. In other words, we test the relative desirability of each pair of models, \( m, m' \in \mathcal{M} \) as entailed by the output theory \( T' \) against that entailed by the original underlying theory \( T \). The error, as defined here, can be computed by reconstructing the preference graph from the learned theory \( T' \) and taking the symmetric difference of the edge set for \( T' \) and that of the graph for the original theory \( T \). Formally,

\[
\text{error}_{T'} = \frac{|[T']^* \Delta [T]^*|}{|[T]^*|} \tag{4.1}
\]

We now define two other desirable properties, namely, size of theory and redundancy of theory. We say a rule \( r \) is subsumed by rule \( r' \) if all the comparisons entailed by rule \( r \) are also entailed by rule \( r' \). Formally,

**Definition 4.1 (Subsumption)** A rule \( r \) is subsumed by rule \( r' \), denoted \( r' \supseteq r \), iff \([r'] \supseteq [r] \)

Note here, this definition is primarily defined for \( r \in \mathcal{L} \), but holds in the context of the language of the intermediate representation \( \mathcal{L},(V) \) as well.

Extending the notion of subsumption to sets of rules \( R, R' \), we say \( R' \supseteq R \), when \([R']^* \supseteq [R]^* \). We also define equivalence between to sets of rules in terms of the comparisons which are entailed by the rules in them. We say two rule sets \( R \) and \( R' \) are equivalent, denoted \( R \equiv R' \) iff \([R]^* = [R']^* \).

We say a theory \( T \) for a given graph is minimal, if every theory describing the same set of comparisons (i.e. edges) has at least as many rules as \( T \). Formally,

**Definition 4.2 (Minimal Theory)** A theory \( T \) has the minimum size for a given preference graph \( G(M,E) \) if,

1. \([T]^* = E \) and,
2. there does not exist a theory, $T'$ such that $T' \equiv T$ and $|T'| < |T|$

Finally, we define a redundant theory as follows:

**Definition 4.3 (Redundant Theory)** We say a theory $T$ is redundant iff there is a theory $T'$ such that $T' \subset T$ and $T' \equiv T$.

This can also be stated as $T' \subset T$ and $T' \supseteq T - T'$. It follows that a minimal theory can never be redundant.

The size of the theory and its redundancy form metrics for measuring the quality of the output. In general, we would prefer irredundant theories of small size.
Chapter 5

Exact Learning

In this chapter we focus on learning an exact theory. Specifically, we discuss an algorithm for learning a theory equivalent to the original underlying theory. Here, equivalent theories are as defined in the previous chapter, i.e., the two theories generate the same graph.

In this algorithm, we make use of the intermediate representation. We start with a preference graph and try to reduce it to a theory in the intermediate representation. This algorithm performs a complete conversion, which can involve looking at a number of edges exponential in the size of the feature set. We then discuss how the learned theory can be converted to a theory in the propositional language $\mathcal{L}$.

The algorithm presented here tries to derive a small size theory, generating a tight fit for the set of edges. The output of the algorithm is a set of rules in the intermediate representation. The idea is to generate a theory $T'$ such that there should not exist a rule $r$, which can replace two or more rules in $T'$, to give another theory $T''$ which is equivalent to $T'$.

A complete analysis of this algorithm including the run time complexity and an analysis of the output with respect to the original theory in intermediate representation, follows the discussion of the algorithm.
5.1 Notation

In the following, we denote an edge $e$ by the corresponding model pair $(m, m')$. Throughout the discussion of this algorithm, we assume the models (and the vertices) to be represented in the bit vector representation, i.e., $m, m' \in M(V)$. We write $l_m(f)$ to denote the literal corresponding to the assigned value of feature $f$ in model $m$. Thus, $l_m(f) = f$ if $f(m) = 1$ and $l_m(f) = \neg f$ if $f(m) = 0$.

Consider an edge of the graph, $(m, m')$. Let $R$ be a set of rules in the language $L_r(V)$, such that for each $r$ in $R$, $(m, m') \in [r]$. For example, $(1000, 0100)$ is entailed by the following rules:

1. $1000 \succ 0100$
2. $10*0 \succ 01*0$
3. $100* \succ 010*$
4. $10** \succ 01**$

The main difference between the above rules is the position of $*$ letters. We made the simplifying assumption that in case of rules in $L$ having conditionals (e.g. a rule of the form $t \rightarrow p \succ q$), the support of the conditional is disjoint from that of the left and right sides of rule and explained that this allows us to write such rules as simple conjunctions. On translating such a rule to its corresponding intermediate representation, we would obtain rules having the same value for the conditional on either side of the rule. For example, $p \land t \succ q \land t$, where $F = \{p, q, t\}$, translates to $R = \{101 \succ 011\}$. While translating in the reverse direction, we differentiate between the features that flip from the left side to the right and those which do not. The non-flipping features form a candidate list of conditionals. We call them candidates noting the fact that the underlying rule may contain all, some or none of these features as conditionals.

We now define these notions formally. We define the difference set for an edge $(m, m')$ as follows:

$$\text{diff}(m, m') = \{f \in F \mid f(m) \neq f(m')\}$$  

The semantics behind the difference set of the rule is an indication of the features over which the preference has been expressed in the particular rule. The difference set hides the candidate conditional features and extracts the essential features in the rule. We call
these features essential noting that any rules in $L_r(V)$ which entail this edge cannot assign ∗ values to these features.

On similar lines, the candidate set of conditionals, denoted $C(m, m')$, is defined as follows:

$$C(m, m') = \{ f \in F \mid f(m) = f(m') \}$$

We can compute the difference set for an edge by a simple XOR or addition operation of the left hand side and right hand side of the rule. Thus, if for two edges, $(m_1, m'_1)$ and $(m_2, m'_2)$, it is the case that $m_1 + m'_1 = m_2 + m'_2$, it implies the edges have the same difference set.

We define the unconditional rule or the Difference Rule, denote $DR(m, m')$ of an edge as follows:

**Definition 5.1 (Difference Rule)** Given an edge (or equivalently a model pair), $(m, m')$, the Difference Rule for the edge, $DR(m, m')$, is a rule $r$ in $L_r(V)$ such that,

1. $f(GS(r)) = f(m)$ for all $f \in \text{diff}(m, m')$ and
2. $f(GS(r)) = *$ for all $f \in C(m, m')$
3. $f(LS(r)) = f(m')$ for all $f \in \text{diff}(m, m')$ and
4. $f(LS(r)) = *$ for all $f \in C(m, m')$

The $DR$ for an edge is simply the rule generated by assuming none of the candidate features to be conditionals. For example, $DR(1000, 0100)$ is $10**, 01**$. Also, $DR(1010, 0110)$ is $10**, 01**$. We note here that different edges can have the same difference rule, as seen in the stated example.

We now define an equivalence relation on edges based on their difference rules. We say two edges $(m_1, m'_1)$ and $(m_2, m'_2)$, are $DR$-equivalent, denoted $(m_1, m'_1) \equiv_{DR} (m_2, m'_2)$, if the two edges have the same difference rule. Formally,

$$(m_1, m'_1) \equiv_{DR} (m_2, m'_2), \text{ iff } DR(m_1, m'_1) = DR(m_2, m'_2) \quad (5.2)$$

We note here that two edges are $DR$-equivalent if they have the same difference set and they additionally assign the same values on their corresponding sides to the features.
in the difference set. Formally, \((m_1, m'_1) \equiv_{DR} (m_2, m'_2)\) iff \(\text{diff}(m_1, m'_1) = \text{diff}(m_2, m'_2)\) and for each \(f\) in \(\text{diff}(m_1, m'_1)\), \(f(m_1) = f(m_2)\) and \(f(m'_1) = f(m'_2)\). In this sense, \(DR\text{-equivalence}\) extends the concept of a difference set by placing an additional restriction and grouping preference statements by their unconditional rules.

We denote the \(DR\text{-equivalence}\) class of edge \(e = (m, m')\) by \([e]\) and define it as follows

\[
[e] = \{e' \in E \mid e' \equiv_{DR} e\}.
\] (5.3)

\(DR\text{-equivalence}\), however, may also relate preference statements that have been derived from different rules in the underlying preference theory. This is due to the fact that it assumes no conditional preferences. Consider the following example:

**Example** Consider a preference graph with the edges \((001, 000), (111, 110)\). Here, \(DR(001, 000) = \text{**1} \succ \text{**0} = DR(111, 110)\). However, these edges could have been entailed by, among other possibilities, either one of the following rule sets:

1. \(\text{**1} \succ \text{**0}\)
2. \(001 \succ 000\) and \(111 \succ 110\).

The equivalence relation of edges defined here has an important property that follows from the translation of rules in the intermediate representation to edges in the graph. For each rule \(r\) in \(\mathcal{L}_r(V)\), any two model pairs \((m_1, m'_1), (m_2, m'_2)\) in \([r]\) are \(DR\text{-equivalent}\). We state and prove this formally in the following.

**Lemma 5.1.1.** All edges generated by the same rule in intermediate representation are equivalent under \(DR\). Formally, for each

\[e = (m, m') \in [r] \text{ we have } [r] \subseteq [e]\]

**Proof.** Let \(r\) be a rule in \(\mathcal{L}_r(V)\). We denote the model pairs (or equivalently edges) in the \([r]\) as \(e\). Let the statements \(w_G, w_L\) in \(\mathcal{L}(V)\) be \(GS(r)\) and \(LS(r)\), respectively.

For simplicity, we define \(S(w_G, w_L)\) to be the set of features which have been assigned the same value in \(\Gamma\) by \(w_G\) and \(w_L\). Formally,

\[S(w_G, w_L) = \{f \mid f(w_G) = f(w_L)\}\]
Now, for each feature $f$ in $S(w_G, w_L)$, we have either $f(w_G) \in \{0, 1\}$ or $f(w_G) = \ast$. In the first case, each of the model pairs in $[r]$ would assign the same value to $f$ as in $w_G$ (or, equivalently, $w_L$). Thus, in the difference rule $r_{DR}$, of these model pairs, we would have $f(GS(r_{DR})) = * = f(LS(r_{DR}))$. In the case that $f(w_G) = \ast$, the forward translation specifies that the models on either side in the generated edges should assign the same value to the features having $\ast$. Following the same argument, we get $f(GS(r_{DR})) = * = f(LS(r_{DR}))$. Thus, for all features having the same value in $w_G$ and $w_L$, the $DR$ for the edges assigns $\ast$.

This leaves us with the features having complementary values on either sides of $r$. In case of such a feature $f$, both the models in the model pairs assign the same values as the corresponding side in $r$. Formally, for each $f \in F$, such that $f(w_G) \neq f(w_L)$, for each pair $(m, m')$ in $[r]$, it would be the case that $f(m) = f(w_G)$ and $f(m') = f(w_L)$. Also, since $f(m) \neq f(m')$, $f(GS(r_{DR})) = f(m)$ and $f(LS(r_{DR})) = f(m')$. Since these value assignments are same for all models pairs generated by $r$, we get that the $DR$ for all the edges is the same.

The above is best illustrated by an example. Consider the rule $101\ast \succ 011\ast$. The generated edges would be: $[r] = \{(1010, 0110), (1011, 0111)\}$. Here, we have, $DR(1010, 0110) = 10** \succ 01** = DR(1011, 0111)$. We note here that, if a rule $r$ in intermediate representation contains no feature $f$ in $S(w_G, w_L)$ such that $f(w_G) \in \{0, 1\}$, the rule $r$ itself is the $DR$ of all the edges in $[r]$.

The algorithm described in the next section starts by partitioning the edge set into equivalence classes and further refines the rules for each of the edges.

### 5.2 From Preference Graph to Intermediate Representation

The equivalence class defined in equation (5.3) defines one set of edges that have the same set of changes. The complete graph can be partitioned by a set of such equivalence
classes. These can then be further refined by looking at the elements within each set.

The following algorithm looks at all the edges in a given preference graph and generates a theory in intermediate representation with rules of the form $p \succ q$. We discuss the basic idea in the algorithm here.

As mentioned, the first step is to divide the edges into a set of $DR$-equivalence classes. Given such an equivalence class, each edge in the class has as its underlying rule, either the unconditional difference rule itself, or a rule which has some conditionals along with the difference rule. The refinement step involves combining edges to form rules with * letters.

We first define edge matchings. We say edges $e_1 = (m_1, m'_1)$ and $e_2 = (m_2, m'_2)$ are matched under feature $f$ if,

1. $e_1 \equiv_{DR} e_2$,
2. $f(m_1) \neq f(m_2)$ and
3. $f'(m_1) = f'(m_2)$ for all features $f' \in F - \{f\}$

We write this as $e_1/f = e_2/f$. As an example for edge matching, let $e_1 = (1011, 0011)$ and $e_2 = (1010, 0010)$. Here, $DR(e_1) = 1 *** \succ 0 *** = DR(e_2)$ and $e_1/f_4 = e_2/f_4$. We note here, that if two edges match on feature $f$, $f$ cannot be in the difference set for either edge since by definition of $DR$, it follows that edges equivalent under $DR$ would have the same difference set and assign same values to the features in the different set.

The matching of edges as discussed here, helps us deduce possible * values in the underlying rule for the edges. Since for either value of the feature we get the same ordering of models with same values for the remaining features, it appears as a natural expansion of a * letter to obtain edges in the graph, as was explained in the forward translation. In the preceding example, we can deduce $101* \succ 001*$ to be a tight fit for the two edges. We can define here the specificity of a rule in $L_r(V)$ for a given edge in terms of the number of * letters in the rule. A rule having no * letters generates exactly one edge, and is hence the most specific rule for that edge. In this sense, the difference rule is the least specific rule, since it assigns a * value to all the features it can under the rules of the translation, namely all features not in the difference set. In the current example, the rule $101* \succ 001*$ is less specific than a rule given by either edge itself (i.e., $m \succ m'$), more specific than the difference rule and generates exactly the two given edges, making it a tight fit. The
difference rule, on the other hand, entails edges that may or may not exist in the graph. The basic idea, now, is to consider each edge \((m, m')\) in the equivalence class as a rule \(m \succ m'\) and generate increasingly general rules, which form a tight fit. We use the notion of edge matching discussed earlier to deduce \(*\) letters. If for some edge \(e\) all edges in \([DR(e)]\) exist in the graph, such pairwise combining would eventually give us the difference rule itself. This is the strategy followed in the algorithm.

The above example shows merging of edges to get a single rule. While this is straightforward for a single feature, for multiple features, we need more book keeping. Consider for example, the following three edges: \((100, 000)\), \((101, 001)\), \((110, 010)\). The first two edges can be combined to give \(10* \succ 00*\) and the first and third to give \(1*0 \succ 0*0\). We, however, cannot combine all three to get \(1** \succ 0**\) unless we also observe the edge \((111, 011)\). In order to track this, we use a separate data structure.

For book keeping purposes, we order the edges arbitrarily and give them an index number. We say \(*\) can be applied to edge \(e_i\) at feature \(f_j\) if the edge is matched to some other edge under \(f_j\), i.e., \(e_i/f_j = e_k/f_j\), for some edge \(e_k \in E\). We maintain the following data structure:

\[
S_{i, j} = \begin{cases} 
1 & \text{if we can apply } * \text{ to edge } e_i \text{ at } f_j \\
0 & \text{otherwise}
\end{cases}
\]

This data structure helps us track the matched edges. The next operation involves combining edges to obtain higher level rules. As was mentioned briefly earlier, we consider each edge \(e = (m, m')\) to be a rule in \(L_r(V)\) of the form \(m \succ m'\). We define the operation of applying \(*\) at feature \(f_i\) to a rule \(r\) in \(L_r(V)\) as replacing the value of feature \(f_i\) on either side by the letter \(*\). In the general case, this would create identical rules (one for each of the matched edges). We eliminate duplicates and label the new rule with the index numbers of the merged edges. We denote the set of indices for rule as \(I(r)\). Thus, in the beginning \(I(r)\) contains just the edge number. Combining of rule \(r\) with \(r'\) gives a single rule \(r''\) such that \(I(r'') = I(r) \cup I(r')\).

Consider an example. Let the following be the edge set \(E\):

\[
E = \{(100, 000), (101, 001), (110, 010), (111, 011)\}
\]
Thus, we get the following as the $S$ matrix:

$$
S = \begin{bmatrix}
0 & 1 & 1 \\
0 & 1 & 1 \\
0 & 1 & 1 \\
0 & 1 & 1
\end{bmatrix}
$$

Considering each edge as a rule, we can apply $\ast$ to all rules at feature $f_2$ giving:

1. $r_5 = 1 \ast 0 \succ 1 \ast 0$, $I(r_5) = \{1, 3\}$
2. $r_6 = 1 \ast 1 \succ 1 \ast 1$, $I(r_6) = \{2, 4\}$

Now, we have replaced edges 1 and 3 with $r_5$ and 2 and 4 with $r_6$. In order to apply $\ast$ at $f_3$ to rule $r_j$, we need to ensure that $S(i, 3) = 1$ for all indices $i$ in $I(r_j)$. Since this is the case for both $r_5$ and $r_6$, we can apply $\ast$ at $f_3$ for both to get the final merged rule: $r_7 = 1 \ast \succ 0 \ast$, with, $I(r_7) = \{1, 2, 3, 4\}$. Following is the complete algorithm:

1. Partition $E$ into equivalence classes $[e_1], \ldots, [e_k]$.
2. Initialize the marker array: for all $i, j$, Set $S_{i, j} = 0$.
3. Initialize the output theory $T' = \emptyset$.
4. for each $e = (m, m') \in E$, such that $[e]$ has not been visited
   (a) for each $f_i \notin \text{diff}(m, m')$
      i. Partition $[e]$ into $f_i^1$ and $f_i^0$ defined as:
         $f_i^1 = \{m \mid (m, m') \in [e] \text{ and } f_i(m) = 1\}$ and,
         $f_i^0 = \{m \mid (m, m') \in [e] \text{ and } f_i(m) = 0\}$
      ii. for each $m \in f_i^1$, if there exists $\hat{m} \in f_i^0$, such that
          for each $f_j$, $j \neq i$ $f_j(m) = f_j(\hat{m})$,
          Let $(m, m')$ and $(\hat{m}, \hat{m}')$ be the $r^{th}$ and $s^{th}$ edge respectively,
          Set $S_{r, i} = S_{s, i} = 1$
   (b) Set rule set $R = \{m \succ m' \mid (m, m') \in [e]\}$
   (c) for each rule $r \in R$, set $I(r)$ to be the singleton set containing only the edge number of $r$. 
(d) for each $f_i \notin \text{diff}(m, m')$

i. Set $M_i = \{ k \mid S_k, i = 1 \}$. (This denotes the set of edge numbers to which $\ast$ can be applied at feature $f_i$)

ii. Set $R' = \bigcup_{k \in M_i} \{ r \in R \mid k \in I(r) \}$. We note here, since $k$ is an edge number, for each $k$, there exists exactly one such rule. Update $R$ to be $R - R'$

iii. For each rule $r$ in $R'$, if $S_{k'}, i = 1$ for all $k' \in I(r)$, apply $\ast$ to $r$ at feature $f_i$.

iv. If the application of $\ast$ values generates set of identical rules $r_1, r_2, ..., r_n$, we merge them and replace them with single rule $r'$, such that $r' = r_i$ and $I(r') = \bigcup_{i=1}^{n} I(r_i)$

v. Set $R$ to $R \cup R'$

(e) Set $T'$ to be $T' \cup R$.

5. return $T'$.

5.2.1 Complexity Analysis

This section computes the worst-case runtime complexity for the preceding algorithm in terms of the number of edges and features.

The first step requires looking at all the edges of the preference graph, and placing it in the correct set. Using a trie-like structure for indexing the equivalence classes, placing one edge would take worst case $O(N)$ time, where $N = |F|$. If we denote the size of the graph (i.e., number of edges) by $|E|$, the total time for step (1) would be $O(N|E|)$.

The step (2) simply initializes the values of an $|E| \times N$ matrix and, hence, has complexity $O(N|E|)$. Since each edge appears in a single equivalence class, we need not reset this matrix for all iterations.

Consider the complexity for step (4.a). For each feature not in the difference set, we look at all the edges in the set once for partitioning (step 4.a.i). This would take time $O(N|E|)$ in the worst case. The next step would have a worst case when both sets have the same size (worst case $|E|/2$) for which the time taken would be $O(|E|^2)$. Over the entire set of features and one equivalence class, the time for step (4.a) would then be $O(N|E|^2)$.

Now considering all the equivalence classes, we can prove that the above case is indeed the worst case complexity of step (4.a). Consider the following two cases.
When the entire graph is partitioned into a single equivalence set, there would be one set with $|E|$ edges. This case is reflected in the preceding analysis and step (4.a) would be performed once, giving the overall complexity of $O(N|E|^2)$.

In case of multiple partitions, consider the case of equal division of edges across $k$ partitions. In this case, each pass in step (4.a) would have a set of $|E|/k$ edges to consider. The total time for step would then be:

$$\left(\frac{|E|}{k}\right)^2 N + \left(\frac{|E|}{k}\right)^2 N + \cdots k \text{ times}$$

$$= k \left(\frac{|E|}{k}\right)^2 N$$

$$= \frac{|E|^2}{k} N$$

$$= O(N|E|^2)$$

Thus, in either of these cases, the complexity of the step (4.a) is $O(N|E|^2)$. In the case of unequal partitions, the complexity would lie between the two extreme cases discussed, thus giving an average case complexity for this step to be $O(N|E|^2)$.

The steps (4.b) and (4.c) iterate once through all the edges in the class. Over the entire set of equivalence classes, the time required by these steps would be $O(|E|)$.

Consider step (4.d). The basic idea here is to look for pairwise combinations of rules in rule set $R$, based on the matrix $S$. Thus, for each feature $f$ not in the difference set, we look at all edges which are matched to some other edge under $f$ (step (4.d.i)). This is a look up of one column of the $S$ matrix. Thus, it takes time $O(N|E|)$ over the entire set of equivalence classes. Now, since the rules corresponding to the edge may have been combined with other rules due to application of * values, we need to find the rule $r$ in $R$, such that the index of the edge is present in $I(r)$ (step (4.d.ii)). Since the total number of rules across the entire graph is $O(|E|)$, this step would require time $O(|E|^2)$ in the worst case, for each feature $f$, giving a total running time of $O(N|E|^2)$. The actual application of * values to a rule $r$ (step (4.d.iii)) requires checking the $S$ matrix for each index in $I(r)$, which has a worst case complexity of $O(|E|)$ per feature. This gives a total complexity of this step to $O(N|E|)$. Similarly, the merging of identical rules (step (4.d.iv)) would require total worst case time $O(N|E|)$ since we may have to merge two rules $r_1$ and $r_2$ such that
\[ |I(r_1)| = |I(r_2)| = |E|/2. \] A similar analysis holds for step (4.d.v). Taking the worst case over the entire step, we get the worst complexity for step (4.d) to be \( O(N|E|^2) \).

The last step is merely a union of rule sets. In the worst case, there would be a single equivalence class and for each feature not in the difference set, \( |R'| \) would be half of \( |R| \). The union would take worst case time \( O(|E|^2) \) per feature, giving an overall worst case complexity of \( O(N|E|^2) \).

Thus, the overall complexity for step (4) is \( O(N|E|^2) \).

The total time would then be:

\[ O(N|E|) + O(N|E|) + O(N|E|^2) = O(N|E|^2) \]

### 5.3 Analysis: Learned Theory

The preceding algorithm makes an underlying assumption that for each edge \((m, m')\) in the graph, there exists a rule \( r \in L_r(V) \) in the underlying theory \( T \) such that \( m \succ_r m' \). We recall here that the input to the algorithm is the edge set \( E \) of the preference graph, which is the transitive closure \( [T]^\ast \). It may thus be the case that the edge \((m, m')\) exists due to transitivity, rather than being entailed by some rule in the underlying theory. A direct consequence of this is the generation of a redundant theory as defined in 4.3. Specifically, we would generate rules of the form \( p \succ p' \), \( p' \succ p'' \) and \( p \succ p'' \). One can try and eliminate these as a post-processing step to the algorithm, we, however, do not discuss a solution to avoid generating such theories in this thesis.

We discuss here one other situation under which a different kind of redundancy can occur in the output theory. One of the operations of the preceding algorithm which we had discussed earlier is that at each step we start with the most specific rules namely, \( m \succ m' \) where \((m, m') \in E\) and combine rules pairwise based on other edges. This ensures that the rule set generated is always accurate, i.e., would generate the same set of edges as the input graph. Also, since we explore all edges within the equivalence class, the property of the DR-equivalence class that no rule can generate edges that are not DR-equivalent (Lemma 5.1.1) ensures that our search is complete. Thus, the algorithm combines all rules that can be combined while being exact. However, the order in which \( \ast \) values are applied at different features, can lead to redundant theories. Consider the following example.
**Example** Consider the following set of edges, which are *DR-equivalent*:

\{(00111, 00011), (01111, 01011), (11111, 11011), (11101, 11001)\}. Here we would get the following $S$ matrix:

$$
S = \begin{bmatrix}
0 & 1 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 0
\end{bmatrix}
$$

In this case, we would first apply $*$ at $f_1$ to give $*1111 \succ *1011$ with labels \{2, 3\}. Performing this procedure at $f_2$ and $f_3$ would give the following set of rules:

1. $*1111 \succ *1011$
2. $0*111 \succ 0*011$
3. $111*1 \succ 110*1$

This theory is redundant, since the second and third rule generate the same set of edges as all the three together.

One can observe in this example that the edges matched under $f_1$ are also matched under two different features. Here if we apply the $*$ values in the reverse order, we would have ended up with only the second and third rule as desired. Thus, it indicates some sort of dependence on the order in which we visit the features. The following section discusses a property of the matching edges in redundant sets of rules and gives a means of selecting the order to avoid generating a redundant set of rules.

**5.4 Avoiding Redundant Theories**

The preceding section discussed two types of redundancy that arise in the output: one due to transitivity of rules, and the other due to a dependency on the order of feature combinations. In this section, we discuss and analyze the latter, and explain how can one select the order of combination to avoid this redundancy.

We take a look at some properties of redundant theories and how we can modify the preceding algorithm to avoid generating the same.

As defined in 4.3, for a redundant theory, $R$, there exists some $R' \subset R$, such that $R' \supseteq R - R'$.
We now formalize these two cases of redundancy in the output theory as follows:

**Definition 5.2 (Redundancy by Transitivity)** We say rule $r$ in $R$ is redundant due to transitivity, if there exist a set of rules $R' \subseteq R - \{r\}$ such that $[r] \not\subseteq [R']$ and $[r] \subseteq [R']^*$. We recall here that for a single rule $r$, $\llbracket r \rrbracket = \llbracket r \rrbracket^*$.

**Definition 5.3 (Redundancy by Overlap)** We say rule $r$ in $R$ is redundant due to overlap if there exists a set of rules $R' \subseteq R - \{r\}$, such that $\llbracket r \rrbracket \subseteq \llbracket R' \rrbracket$.

As explained in the preceding section, depending on the order in which the features are applied, we may generate rules which are redundant due to overlap. We analyze this case and explain how we can select the feature order to avoid such redundancy. The definition of redundancy by overlap given above can also be stated as: for each $e \in \llbracket r \rrbracket$, there exists $r' \in R'$, such that $e \in \llbracket r' \rrbracket$. This means that edge $e$ is matched to some other edge $e' \in E$ under all features $f$ which assume * in $r'$.

Now, it is not possible that both $r$ and $r'$ do not contain *, since in that case they would be the same rule. This is not possible since $R' \subseteq R - \{r\}$, giving $r \neq r'$.

Let $f^*_r = \{ f \mid f = * \text{ in } r \}$ and similarly define $f^*_{r'}$. Consider the following cases:

**Case 1** $f^*_r = f^*_{r'}$

Now, the remaining features have fixed values, 0 or 1. For both rules to generate the same edge, all such features should have the same value. But this implies the two rules are the same. Thus, this case is not possible.

**Case 2** $f^*_r \supset f^*_{r'}$ or $f^*_r \subset f^*_{r'}$

The application of * values in the algorithm precludes this case since when we apply * values to any feature, we combine the identical rules. Thus, during some point of application, the rule with fewer * values would be merged into the bigger one, when the remainder of the features are replaced by *. Note that this case also covers the case when either $f^*_r = \emptyset$ or $f^*_{r'} = \emptyset$.

**Case 3** $f^*_r - f^*_{r'} \neq \emptyset$ and $f^*_r - f^*_{r'} \neq \emptyset$

The second statement above states that some feature $f \notin f^*_r$ has the letter * in the
redundant rule $r$. Thus, each edge $e$ in the $[r]$ is matched under some feature $f \notin f^*_r$. Let $e_1$ and $e_2$ be edges in $[r]$ such that $e_1/f = e_2/f$. A rule in $\mathcal{L}_r(\mathcal{V})$ can generate both $e_1$ and $e_2$ only by assigning $f = *$ on either side. However, since $f \notin f^*_r$, we have either $e_1 \notin [r']$ or $e_2 \notin [r']$ or both. Thus, there exists at least one edge, $e' \in [r]$ such that $e' \notin [r']$. However, as stated earlier, since $r$ is redundant, for each edge in $[r]$ there exists some rule in $R'$ such that $e$ is entailed by that rule. Thus, there exists a third rule, $r''$ such that $e' \in [r''].$

We started our argument with an edge $e \in [r]$ such that $e \in [r']$ and deduced there exists an edge $e' \in [r]$ such that $e' \notin [r']$ and $e' \in [r'']$, where $r''$ is some rule in $R'$. Following a similar argument for $r''$ as we did for $r'$, we get the condition $f^*_{r''} - f^*_{r'} \neq \emptyset$, stated above. Thus, edge $e'$ is matched under some feature $f' \in f^*_{r''}$, such that $f' \notin f^*_{r'}$. Since both $r'$ and $r''$ share at least one edge each with $r$, the difference rule corresponding to all three rules is the same (Lemma 5.1.1). By an analogous argument as given in case 1 and 2, it cannot be the case that $f^*_{r''} \subseteq f^*_{r'}$ or $f^*_{r'} \subseteq f^*_{r''}$. Thus, for each edge $e \in [r]$, there exists edge $e' \in [r]$ such that $e'$ is not matched under some feature $f$ under which $e$ is matched and vice versa. Formally, for each edge $e_i \in [r]$, there exists $e_j \in [r]$, such that $S_{i,k} = 1$ and $S_{j,k} = 0$ for some feature $f_k$ and also $S_{j,m} = 1$ and $S_{i,m} = 0$, for some feature $f_m$. We note here that $f_k$, $f_m \notin f^*_r$, since $e$, $e' \in [r]$.

This condition can be observed in the $S$ matrix by noting that all edges in $[r]$ are matched under each feature in $f^*_r$. This allows us to deduce the set of features $f^*_r.$ If we do not apply * at any feature in $f^*_r$, we would avoid generating rule $r$. We formalize this as follows: Let $M(f_j)$ denote the set of edges which are matched under $f_j$. Formally, $M(f_j) = \{e_i \mid S_{i,j} = 1\}.$ A feature $f$ in $F$ should be suppressed during application of * values, if it is the case that for each edge $e_i \in M(f)$ there exists $e_j \in M(f)$, such that,

1. $S_{i,k} = 1$ and $S_{j,k} = 0$ for some feature $f_k$ and
2. $S_{i,m} = 0$ and $S_{j,m} = 1$ for some feature $f_m$.

We illustrate this using the $S$ matrix from our example in the previous section (figure 5.1).

Here, feature $f_1$ matches edges 2 and 3. We have
1. $S_{2,2} = 1$, $S_{3,2} = 0$ and
2. $S_{2,4} = 0$, $S_{3,4} = 1$.

Suppressing $f_1$ would avoid redundancy. One way of doing so would be to apply feature $*$ value at $f_1$ after applying for other features, as was explained earlier.

The algorithm discussed earlier needs to be modified in the part where we apply the $*$ values as per the $S$ matrix, such that features matching edges across different sets of features should be processed at the end. This would ensure generation of theories which are not redundant. Since this operation requires looking at the values in the $S$ matrix for all edges in the equivalence class, for each feature in the corresponding difference set, the time complexity would be no worse than $O(N|E|^2)$.

5.5 From Intermediate Representation to Preference Logic

We discussed the translation of a single rule $r$ in $\mathcal{L}$ to a set of rules $R$ in $\mathcal{L}(\mathcal{V})$. Here, we observe that a single rule can map to multiple rules in the intermediate representation. Ideally, we would want to reduce a set of rules obtained by the preceding learning algorithm to the smallest number of rules in the logical language. For example, let $F = \{f_1, f_2, f_3, f_4\}$ and the rule set output by the algorithm be:

- $101* \succ 011*$
- $110* \succ 011*$
In this case, the ideal output would be \( f_1 \succ f_2 \land f_3 \). This can be seen from the fact that translation of this rule gives the above rule set.

In order to achieve such reduction, we need to be able to identify which rules belong to the same rule set \( R \), such that there exists some rule \( r \) in \( \mathcal{L} \) which translates to \( R \) in intermediate representation. We avoid tackling the complexity of this problem in this thesis and employ the reverse translation described in [McG02].

We defined the characteristic model \( \mu(p) \) of statement \( p \) in \( \mathcal{L}(\mathcal{V}) \) to be a model in \( \mathcal{M}(s(p)) \) such that \( \alpha(f(\mu(p))) = f(p) \) for each feature \( f \in s(p) \) (equation (3.4)). Given a rule \( r = p \succ q \), where \( p \) and \( q \) are statements in \( \mathcal{L}(\mathcal{V}) \), we can translate it to a single rule \( a \succ b \) in \( \mathcal{L}(\mathcal{V}) \) by assigning \( a \) to be a conjunction over the literals in \( \mu(p) \) and, similarly \( b \) over those in \( \mu(q) \). Here, the features assigned \( * \) by \( p \) and \( q \) do not appear in either \( a \) or \( b \). The remaining features are assigned the same values as the corresponding side. This ensures the set of model pairs satisfying the two rules to be the same, thereby, retaining the meaning of the rule.

As an illustration, the rule \( 10* \succ 01* \) translates to \( f_1 \land \neg f_2 \succ \neg f_1 \land f_2 \).
Chapter 6

Approximate Learning

The algorithm discussed in the preceding chapter focussed on the problem of exact learning of preference theories. However, it has a time complexity of polynomial order in the size of the graph, i.e., the number of edges. In case of a preference graph, the number of vertices is exponential in the number of features \(2^N\) to be precise and the number edges for an average-sized theory can be expected to be much more than this. Although this technique can be useful for sparse graphs, where the number of edges is much smaller than the number of vertices, in general, we do not expect to encounter such graphs. In order to illustrate, consider a preference theory having a single rule of the form \(f \succ \neg f\) for some feature \(f\) in \(F\). The preference graph for this theory would have \(2^{N-1}\) edges, making the algorithm exponential in the number of features.

In the light of the preceding discussion, we look at approximation algorithms in this chapter. The idea here is to learn a theory with an acceptable error, but do so in time of the polynomial order in the number of features. In particular, we look at Probably Approximately Correct (PAC) learning techniques ([Va84]) and fit preference learning into this framework. In the following section we discuss the framework and analyze the learnability of preference theories.
6.1 Preference Learning as a PAC problem

Consider a learner $L$ trying to learn a target concept $c$ from a class of concepts $C$ using the hypothesis space $H$. We assume that the training set is drawn from the instance set $X$ according to some distribution $D$. The learner is expected to output a hypothesis $h$ in $H$, after having observed some number of examples. We write $c(x)$ and $h(x)$ to denote the classification assigned by the concept and hypothesis to an instance $x \in X$, respectively.

In this setting, the true error is defined in terms of the distribution, as the probability that a randomly drawn example would be misclassified by $h$. Formally ([Mit97]),

$$\text{error}_D(h) = Pr_{x \in D}[c(x) \neq h(x)].$$

Here, $Pr_{x \in D}$, denotes that $x$ is randomly drawn according to the distribution $D$ over the instance set $X$. Note here that if the distribution $D$ is uniform, the true error would be the ratio of misclassified examples to the size of the instance set, as was defined earlier.

We now define PAC-learnability as follows ([Mit97]):

**Definition 6.1 (PAC-Learnability)** Let $C$ be a concept class defined over a set $X$ of instances having instances of length $n$ and $L$ be a learner seeking to approximate a concept in $C$ using the hypothesis space $H$. $C$ is PAC-learnable by $L$ using $H$ if for all $c \in C$, probability distributions $D$ over $X$, $\epsilon$ such that $0 < \epsilon < 1/2$, and $\delta$ such that $0 < \delta < 1/2$, learner $L$ with probability at least $(1 - \delta)$ outputs a hypothesis $h \in H$ such that $\text{error}_D(h) \leq \epsilon$, in time that is polynomial in $1/\epsilon$, $1/\delta$, $n$ and $\text{size}(c)$.

In this definition, the length of an instance is defined for the domain under consideration. For example, in case of concepts defined as conjunctions over $n$ boolean variables, each instance would consist of a truth assignment to each attribute and, hence, have a length of $n$. The size of the concept, $\text{size}(c)$ is dependent on the representation. If the target concept is a conjunction of boolean variables, then $\text{size}(c)$ would be the number of literals in the target $c$.

As was discussed, we consider the problem of learning a preference theory expressed in the terms of logical statements over the feature set $F$. The learner $L$ in our setting is an algorithm, taking the training set as input. An instance in this case is an edge of the
preference graph, \((m, m')\), giving preference between the two models \(m, m'\) in \(\mathcal{M}\). Thus, each instance is of length \(n = 2 \times N\), where \(N = |F|\). Also, size of the concept, \(\text{size}(c)\) would be the summation of length of each statement. We define the length of a statement \(p \succ q\), where \(p\) and \(q\) are conjunctions, as the number features in the support of the statement, i.e., \(|s(p) \cup s(q)|\).

We analyze the time complexity of the learning algorithm by separating out the number of examples required to PAC learn the problem and the time spent on processing each of the examples. If both of these are polynomial, the overall learning time would be polynomial, as required. For most of the discussion, we analyze the number of examples to be seen. The second issue is discussed at the end each of the respective sections.

We denote \(O_T\) as the number of training instances required to be observed to approximate the concept. PAC learning requires the time taken by the learner to be polynomial in \(1/\epsilon\), \(1/\delta\), \(n\) and \(\text{size}(c)\). If we have an algorithm which processes each example in polynomial time, the number of examples should be bounded as:

\[
O_T = O(g(1/\epsilon, 1/\delta, n, \text{size}(c)))
\]

(6.1)

where \(g\) is polynomial function over its arguments.

We can now formally enumerate the conditions for PAC-learning as,

1. \(P(\text{error}_D \leq \epsilon) > 1 - \delta\),
2. having observed \(O_T = O(g(1/\epsilon, 1/\delta, n))\) examples,
3. processing each example in time polynomial in \(1/\epsilon\), \(1/\delta\) \(n\) and \(\text{size}(c)\).

Before going on to the algorithms, we discuss a result from PAC learning theory, discussing a general bound on the number of examples required to PAC-learn a concept. We use the following definitions and notations from the [Mit97]. A consistent learner is a learner that models the training data perfectly. In other words, a consistent learner is one which classifies all training instances correctly. Thus, a consistent learner has zero training error. We denote \(\text{VS}_{H,D}\) as the version space, which is defined as the subset of the hypothesis space \(H\) containing hypotheses which are consistent with the set of examples \(D\). Formally, denoting the underlying concept as \(c\),

\[
\text{VS}_{H,D} = \{h \in H \mid \text{for each } x \in D, c(x) = h(x)\}.
\]
Here, \( c(x) \) denotes the classification by the underlying concept and hence, the classification specified with the training instance.

Thus, a consistent learner always outputs a hypothesis from the version space. Note here that the true error for these hypotheses need not be zero.

Haussler, [Hau88], discusses how many examples are required to exhaust a version space of bad hypotheses, i.e., to eliminate hypotheses having true error more than \( \epsilon \). He shows that for a finite hypothesis space, \( H \), having seen a set \( D \) of examples, with \( |D| = m \), the probability that the version space \( VS_{H,D} \) contains a hypothesis with true error greater than \( \epsilon \), is less than or equal to \( |H| e^{-\epsilon m} \). This result is extended in [Mit97] to obtain a lower bound of \( \frac{1}{\epsilon} (\ln |H| + \ln(1/\delta)) \) on the number of examples sufficient for a consistent learner to learn a target concept with error less than \( \epsilon \) and probability at least \( 1 - \delta \). Thus,

\[
m \geq \frac{1}{\epsilon} (\ln |H| + \ln(1/\delta)) \quad (6.2)
\]

In the following sections, we use this result to analyze the learnability of a preference theory, having direct preference comparisons as the training instances. We first consider a special case in the following section, to reduce the complexity of the task at hand. The later sections analyze the generic case.

### 6.2 Single Rule Theories

We first consider a much simpler problem, in which the underlying preference theory consists of a single rule in the propositional language \( \mathcal{L} \). Formally,

\[
T = \{ r \}
\]

Here, \( r = p \succ q \), where \( p \) and \( q \) are statements in \( \mathcal{L} \). The target concept in this case would be a pair of statements \( p \) and \( q \) in \( \mathcal{L} \), such that \( p \succ q \). Here, \( p \) and \( q \) are conjunctions of length at most \( N = |F| \).

We show how such a theory can be learned using the same technique used to learn a concept defined by a conjunction of literals, which is known to be PAC-learnable.

The training instances given to the learner are model pairs \((m, m')\), such that the edge \((m, m')\) exists in \( E \). Since the underlying graph in the present case is generated from a single rule, \( r = p \succ q \), for each training instance \((m, m')\), we have \( m \succ_r m' \). Thus, for
each training example \((m, m')\),

\[ m \models p \land \neg q \text{ and } m' \models \neg p \land q \]

For any propositional statements \(r\) and \(r'\), if \(m \models r \land r'\) for some model \(m\), it holds that \(m \models r\) and \(m \models r'\). Thus, for each instance \((m, m')\), it is true that \(m \models p\) and \(m' \models q\).

In light of the preceding observation, we use the model pairs as instances to learn conjunctions \(p\) and \(q\) independently. The idea now is to learn two conjunctions from the same set of examples. Since we are not considering any dependencies between learning of the two conjunctions, our learning problem is reduced to learning two conjunctions in parallel, where each example has two parts, one for each conjunction. Now, given \(N\) features, number of possible conjunctions is \(3^N\). To see this, for each feature, \(f \in F\), either \(f\) or \(\neg f\) or neither is present. This gives \(3^N\) possibilities over the entire set of features. Thus, our hypothesis space would contain \(3^{3N} = 3^n\) (since \(n = 2N\)) hypotheses. Formally, \(|H| = 3^n\). Substituting this in equation 6.2, we get,

\[ m \geq \frac{1}{\epsilon}(n \ln 3 + \ln(1/\delta)) \quad (6.3) \]

which is polynomial in the size of the \(\epsilon, n\) and \(\delta\).

6.2.1 Algorithm

We now need an algorithm that takes polynomial time per example. The following algorithm is an adaptation of the FIND-S algorithm described in [Mit97]. We start with two sets \(c_{GS}\) and \(c_{LS}\) containing \(f\) and \(\neg f\) for each \(f \in F\). For each example \((m, m')\), we adjust \(GS\) and \(LS\) sets as follows: Let \(l(f)\) be a literal of feature \(f\), i.e. \(l(f) = f\) or \(l(f) = \neg f\). If it is the case that \(l(f) \in m\) and \(\neg l(f) \in c_{GS}\), for some feature \(f \in F\), we discard \(\neg l(f)\) \(c_{GS}\) and similarly check for all features in \(m'\) and \(c_{LS}\). This just requires a look up of two sets of size \(N\) each, which is linear in the size \(n\). Thus, the algorithm satisfies the complexity requirement. Also, with each example, we go from a more specific to a more general hypothesis. The adjustment allows to cover the new example. Thus, at any point, the hypothesis would correctly classify the entire training set. This meets the consistency requirement. The algorithm is given formally ahead:

1. Initialize each of \(c_{GS}\) and \(c_{LS}\) to the set \(\{f \mid f \in F\} \cup \{-f \mid f \in F\}\).
2. For each training instance, \((m, m')\)

(a) For each feature \(f\)

\[
\text{If } l(f) \in m \text{ and } \neg l(f) \in c_{GS}, \text{ drop } \neg l(f) \text{ from } c_{GS}.
\]

\[
\text{If } l(f) \in m' \text{ and } \neg l(f) \in c_{LS}, \text{ drop } \neg l(f) \text{ from } c_{LS}.
\]

3. Generate rule \(r = p \succ q\), such that \(p\) is a conjunction over all literals in \(c_{GS}\) and \(q\) over all literals in \(c_{LS}\).

The algorithm starts with \(c_{LHS} = c_{RHS} = \{f_1, \neg f_1, \ldots, f_n, \neg f_n\}\). On seeing the first example, half the literals from each set would get pruned and we would be left with the exact conjunctions that were presented (note here that the conjugations refer to what would be generated if the learning were to be terminated). This way the algorithm proceeds from most specific to general conjunctions. Since it requires examining two sets for each feature, it takes time \(O(n)\) per example. Thus, the preference theories containing a single rule are PAC learnable by the preceding learning algorithm.

### 6.3 Multiple Rule Theories

We now discuss the case of generic theories, having any combination of rules in \(\mathcal{L}\), as long as the theory is consistent. We defined a consistent theory as a set of rules \(T\), such that for no two models \(m, m'\) in \(\mathcal{M}\), it is the case that \((m, m') \in [T]^*\) and \((m', m) \in [T]^*\).

We first discuss the hypothesis space for this case. We consider all possible preference statements \(p \succ q\), where \(p\) and \(q\) are conjunctions in \(\mathcal{L}\). The number of possible conjunctions of \(N\) features is \(3^N\) as discussed in the previous section. Since we have a conjunction on each side, the total number of possible rules is \(3^N \times (3^N - 1)\), which we round up to \(9^N\) for simplicity of calculations. Note here that \(N = |F| = n/2\). Thus, the number of rules is \(3^n\). The number of possible subsets of these rules (which would be the candidate theories), is the cardinality of the power set. So the size of the hypothesis space is \(2^{3^n}\). One can note here that this hypothesis space also consists of inconsistent sets of rules, since we have both \(p \succ q\) and \(q \succ p\) for each pair of conjunctions \(p\) and \(q\). One simplification would be to eliminate one of the two opposing rules from the rule sets in the hypothesis space. This, however, reduces the space only linearly and our hypothesis space...
is still superexponential. Substituting $|H| = 2^{3^n}$ in equation 6.2, we get the following lower bound on the number of examples:

$$m \geq \frac{1}{\epsilon} (3^n \ln |2| + \ln(1/\delta))$$

(6.4)

which is exponential in $n$.

This relation exhibits the inherent complexity of the learning problem, posed by the size of the hypothesis space. Although, the above hypothesis space still contains inconsistent theories, it is in general difficult to construct the exact hypothesis space containing only consistent theories, making learning generic consistent theories a hard problem. In the following sections, we look at a subset of the problem set, making structural assumptions, or more generally, explore a smaller sized concept space.

We also look at a different type of learning approach, namely one where the learner is allowed to ask questions regarding the instance space. We analyze the problem using such a learner-initiated environment.

### 6.3.1 Structural Assumptions

The notion of preferences can be used to talk about goals and desires of an agent, where both of these talk about world states having higher utility for the agent. In the general case, we say proposition $p$ defined as conjunction in $L$ is a goal, if it is the case that $p \succ \neg p$. We briefly discuss the inherent difficulty in learning general goals in a later section. For our current problem, we focus on learning goal features, i.e., goals defined by single features. In this context, we say a feature $f$ is a goal, if its presence is preferred, other things being equal. This is simply saying, $f \succ \neg f$, other things being equal. More generally, one may say literal $l$ is a goal, where $l$ could be either $f$ or $\neg f$.

The statement $f \succ \neg f$ can be expressed as a rule in $L$, and one at a higher level, since it partitions the set of all vertices in the preference graph into two subsets, one containing $f$ and others $\neg f$ and gives a lower utility to the latter. We note here that such rules may have preconditions, specifying the situations in which a feature is desirable. Thus, in the unrestricted propositional language of [DSW91], one may write $f_2 \rightarrow f_1 \succ \neg f_1$. Under our restricted language, we can write this as $f_1 \land f_2 \succ \neg f_1 \land f_2$. We call such statements, expressing a preference of presence (or absence) of a feature over its absence (or presence), under (possibly) some pre-conditions, as statements of the desirability of the feature.
note here that learning statements of desirability of all the features in $F$ is similar to learning a CP-net ([BBHP99]). In this thesis, we do not explicitly learn a CP-net, but we briefly discuss how this can be done using our approach in a later section.

Once we have considered statements of desirability of features, we can look into trade-offs between goal features, as the next level of rules, expressing such preferences as $f_1 \succ f_2$, where both $f_1$ and $f_2$ are goals. This statement expresses a preference of models satisfying $f_1 \land \neg f_2$ over those satisfying $\neg f_1 \land f_2$, other things being equal. In other words, it attributes a higher desirability to $f_1$ than $f_2$. Once again, this may hold only under certain conditions laid by other features. We call such statements as tradeoff statements between goal features.

In general, these two types of statements can provide a fair summary of an agent’s preferences. In the following sections, we restrict our learning space to such rules. The idea behind this two-fold: one is to reduce the size of the hypothesis space and second is to try and learn a high level summary of the agent’s preference.

We now analyze the size of the hypothesis space for learning the two types of rules discussed. For a set of features $F$ such that $|F| = N$, there can be two rules per feature, expressing its desirability, assuming there are no preconditions for the rule. Thus, for each $f$ in $F$, we may have $f \succ \neg f$ or $\neg f \succ f$, giving a total of $2N$ rules. In case of tradeoff statements, we can express tradeoffs between any two pairs of features. We consider a general case, where either $f$ or $\neg f$ can be a goal, for any feature $f$. Thus, both possibilities for each feature can be compared with either possibility of any of the remaining $N - 1$ features, giving us $2N \times 2(N - 1)$ possibilities, again assuming there are no pre-conditions.

We can, however, note here that $\neg f_i \succ \neg f_j$ is merely the contrapositive of the statement $f_j \succ f_i$. This can be seen by looking at the semantics of the two rules: both specify a preference of models satisfying $\neg f_i \land f_j$ over those satisfying $f_i \land \neg f_j$, other things being equal. This reduces the number of rules by half, giving $2N(N - 1)$ rules, without any pre-conditions.

The preceding analysis considered only unconditional statements. We need to also consider rules of the form $p \rightarrow f_i \succ \neg f_i$ and $q \rightarrow f_i \succ f_j$, where $p$ and $q$ can be any conjunction over the remaining features. In the general case, the rule may hold under any set of preconditions. In other words, for each statement of the form $f \succ \neg f$, the theory may contain of set of rules of the form \( \{ p_1 \rightarrow f \succ \neg f, p_2 \rightarrow f \succ \neg f, \ldots, p_j \rightarrow f \succ \neg f, q_1 \rightarrow \neg f \succ f, q_2 \rightarrow \neg f \succ f, \ldots, q_k \rightarrow \neg f \succ f \} \), where $p_1, \ldots, p_j$ and $q_1, \ldots, q_k$ are conjunctions.
over $F - \{f\}$, as long as no model $m$ of $\mathcal{M}$ satisfies two conjuncts $p_l$ and $q_m$ for some $l$, $1 \leq l \leq j$ and some $m$, $1 \leq m \leq k$. Formally, $[p_1 \lor p_2 \lor \ldots \lor p_j] \cap [q_1 \lor q_2 \lor \ldots \lor q_k] = \emptyset$.

Note here that we chose $f \succ ¬f$ as an example; the same holds for tradeoff statements also.

Now, the number of possible pre-conditionals is simply the number of possible conjunctions over the remaining features, i.e., $3^{N-1}$ and $3^{N-2}$ respectively for the two types of statements under consideration. We only consider the features not already present in the statements to avoid statements of the form $f_i \rightarrow f_i \succ ¬f_i$. Each of the rules can have any subset of conjunctions, thereby giving a hypothesis space defined by the power set of these sets of conjunctions. Considering our previous calculations on the number of possible rules of each type, we get a total of $2N \times 2^{3^{N-1}}$ rules of the form $p \rightarrow l \succ ¬l$ and $2N(N-1) \times 2^{3^{N-2}}$ rules of the form $p \rightarrow (l_i \succ l_j)$, where $l_i$ is a literal over feature $f_i$. Once again, the size of our hypothesis space is superexponential (even considering $N = n/2$), posing a learning problem.

In order to overcome this, we pose a length restriction on the size of the antecedent and learn only those rules having at most $k$ literals in the antecedent. This structural restriction is analogous to that of learning $k$-CNF or $k$-DNF boolean statements to restrict the hypothesis space. The justification in doing this is simply that rules with a large number of literals in the antecedent would generate fewer number of edges in the graph. Thus, as the size of rules increase (size being defined in terms of the number of features in the support of the rule) the gain in accuracy (i.e., reduction in error) decreases monotonically. The formal analysis of the learning error and effects on algorithmic complexity are explained in further sections, after discussing the actual algorithm.

### 6.3.2 Learning with Learner-Initiated Queries

As mentioned earlier, we solve the preceding learning problem with a slightly different approach, namely one in which the learner can ask membership queries to a teacher. The teacher is merely an oracle having complete knowledge of the instance space and underlying concept. A membership query in context of preference theories is a question of the form “$m \succ m'$?” for two models $m$, $m'$ of $\mathcal{M}$. The reply can be yes, if there exists an edge from $m$ to $m'$ in the preference graph or no, if either they are incomparable or there is an edge from $m'$ to $m$. Since the latter ambiguity can be resolved by asking one more question (a linear increase), we assume for simplicity that the oracle replies as yes, no, or
incomparable as the case may be. Lastly, the efficiency of the learner is measured in terms of number of questions asked and time taken to process each reply. The idea is to still keep the total learning time to be polynomial in the length of each instance, i.e., $n = 2N$. Here, total learning time would be the time to process each reply times the number of questions.

We now describe the starting hypothesis that is used. We refer to rules without conditionals of the form $l \succ l'$, with $l$, $l'$ being literals over $F$, as simple statements. We denote the feature corresponding to literal $l$ as $f(l)$. We note here that in the statement $l \succ l'$, having $l \neq l'$ but $f(l) = f(l')$ denotes a statement of desirability (or undesirability if $l = \neg f(l)$) of feature $f(l)$. As was discussed earlier, for each such simple statement, there may exist a set of conjunctions $\{p_1, \ldots, p_m\}$ over features $F - \{f(l), f(l')\}$ of length at most $k$, as the antecedent for the statement. We denote true as a tautology, used to denote unconditional preferences. Thus, $f \succ f'$ is the same as true $\rightarrow f \succ f'$. We denote $\mathcal{C}(F)$ to be the set of all conjunctions over the feature set $F$. For any conjunct $p$ in $\mathcal{L}$, we write $size(p)$ to denote the length or the size of $p$ as the number of features in support of $p$. Formally, $size(p) = |s(p)|$. Finally, we denote $A_k(s)$ as the set of all possible antecedents of size at most $k$ for the simple statement $s = l \succ l'$. We can write this formally as follows:

$$A_k(l \succ l') = \{p \in \mathcal{C}(F - \{f(l), f(l')\}) \mid size(p) \leq k\}$$  \hspace{1cm} (6.5)

We note here that in this definition, we may have $l = \neg l'$, giving $f(l) = f(l')$.

The starting hypothesis is now the set of all rules of the form $p \land l \succ p \land l'$, for all $p \in A_k(l \succ l')$, where $l \neq l'$. In case that $p = \text{true}$, the rule is simply the underlying simple statement, $l \succ l'$. 
Example Starting hypothesis for $F = \{f_1, f_2\}$ with $k = 2$:

$$H = \begin{cases} 
  f_1 \succ -f_1 & -f_1 \succ f_1 \\
  f_2 \land f_1 \succ f_2 \land -f_1 & f_2 \land -f_1 \succ f_2 \land f_1 \\
  -f_2 \land f_1 \succ -f_2 \land -f_1 & -f_2 \land -f_1 \succ -f_2 \land f_1 \\
  f_2 \succ -f_2 & -f_2 \succ f_2 \\
  f_1 \land f_2 \succ f_1 \land -f_2 & f_1 \land -f_2 \succ f_1 \land f_2 \\
  -f_1 \land f_2 \succ -f_1 \land -f_2 & -f_1 \land -f_2 \succ -f_1 \land f_2 \\
  f_1 \succ f_2 & f_2 \succ f_1 \\
  f_1 \succ -f_2 & -f_2 \succ f_1 
\end{cases}$$

We note here that $-f_1 \succ f_2$ and $f_2 \succ -f_1$ are contrapositives of $-f_2 \succ f_1$ and $f_1 \succ -f_2$, respectively. Similarly, $-f_1 \succ -f_2$ and $-f_2 \succ -f_1$ are contrapositives of $f_2 \succ f_1$ and $f_1 \succ f_2$. Thus, this hypothesis space covers all possibilities. We discuss the details of the learning algorithm in the following section.

6.3.3 Algorithm

The idea here is to learn one simple statement $l \succ l'$ at a time, pruning the space of conditionals. Now, since the theory cannot contain both $p \land l \succ p \land l'$ and $p \land l' \succ p \land l$, we consider such opposing pair of statements together. Note here that $A_k(l \succ l') = A_k(l' \succ l)$ according to our definition in equation (6.5). Thus, considering one such pair at a time, we ask questions allowing us to remove those conjunctions from one or both sides which are inconsistent with the reply. The pruning is explained in the following.

The forward translation from the rules in $L$ to preference graph converts each rule $r$ in the theory $T$ to a set of rules $R$ in $L_r(V)$, each of which in turn generate edges in the graph. Thus, for each rule $r$ in the preference theory $T$,

$$[r] \subseteq E$$

where, $E$ is the edge set of the preference graph, and $[r]$ is the set of edges generated by rule $r$, as defined earlier. This can also be stated as follows: for each $e \in [r]$, we have $e \in E$.

The converse is, however, not true since several rules could generate the same edge. Also, since the edge set is the transitive closure $[T]^*$, some edges might be generated by
transitivity. We can, though, look at the contrapositive of the preceding statement in order to get some pruning: if there exists $e \in [r]$ such that $e \notin E$, then $r \notin T$.

Thus, in order to prune the hypothesis space, we ask questions regarding presence of certain edges. For each edge $(m, m')$ where $m, m'$ are models in $\mathcal{M}$, we can prune from the hypothesis space all rules $r'$ such that $(m', m) \in [r]$. Since, the learner would be asking questions and pruning depending on the reply, we need to formalize which rules can be pruned upon receiving each reply.

Let the pair of statements under consideration be $l \succ l'$ and $l' \succ l$. We discuss the question formulation in the next section, but we note here that the questions compare models $m_1$ and $m_2$ such that $m_1 \in m[l \land \neg l']$ and $m_2 \in m[l \land \neg l']$ for some model $m$. Say in this case the reply is $m_1 \succ m_2$. This would then allow us to prune all those rules of the form $p \land l' \succ p \land l$, where $m_1 \models p$. Note here that, since support of $p$ is disjoint from $\{f(l), f(l')\}$, it would be the case that $m_1 \models p \iff m_2 \models p$. We also note that if the reply is that the models are incomparable, we would prune rules for both the base statements being considered.

### 6.3.4 Question Formulation

The question formulation requires comparing for each pair of statements, $l \succ l'$ and $l' \succ l$, those models such that each possible conjunct in $A_k(l \succ l')$ can be pruned.

We start with the underlying statement and set all other features to false (i.e. $\neg f$). Thus, the models would be $m_1 = \{l, \neg l'\} \cup \{\neg f \mid f \in F - \{f(l), f(l')\}\}$ and $m_2 = \{\neg l, l'\} \cup \{\neg f \mid f \in F - \{f(l), f(l')\}\}$. We note here that it may be the case that $f(l) = f(l')$. We write $C(s)$ to denote the set of features which can appear in the antecedent for the simple statement $s = l \succ l'$, given by $F - \{f(l), f(l')\}$. We note here that $C(l \succ l') = C(l' \succ l)$.

For the next set of questions, we use positive literals for each feature in $C(s)$, one at a time, keeping the remaining features negated. After this, we use positive literals for every pair of features, then for every three features and so on up till $k - 1$ features, to cover all possible conjuncts having at least one negative literal. Finally, we ask one question having all literals to be positive, to cover all conjunctions having only positive literals.
6.3.5 Complexity Analysis

We now analyze the complexity of the preceding learning algorithm in terms of number of questions required to learn the theory and the time spent in processing each reply.

For each rule of the form \( l \succ l' \), we ask two questions, one with all negative literals and the last one having all positive literals. In between, we set each set of two, three and so on up to \( k - 1 \) to be positive, keeping the remaining negated. Thus, the total number of questions would be:

\[
2 + \sum_{j=2}^{k-1} \binom{N}{j}
\]

We note here that the set \( C(s) \) has fewer than \( N \) features, but this does not make any difference to the asymptotic complexity of the algorithm, and we choose the above for simplicity. For constant \( k \), this equation is bounded by \( N^k \), giving a polynomial number of questions per rule.

The preceding calculation looks at number of questions per simple statement. The total number of such simple statements, as calculated earlier, is \( 2N \) statements of the form \( l \succ \neg l \) and \( 2N(N-1) \) statements of the form \( l \succ l' \) where \( f(l) \neq f(l') \). Thus, the total number of questions across the entire hypothesis space is polynomial in \( N \). Since \( N = n/2 \), we get it is polynomial in \( n \).

Lastly, we look at the time required to process each answer. For each reply, we have to look at all possible conjunctions in \( A_k(l \succ l') \) to prune the ones which do not satisfy the rule. The size of the set \( A_k(l \succ l') \) can be bounded as follows. \( A_k(l \succ l') \) consists of conjunctions of literals of up to length \( k \) over features in \( F - \{f(l), f(l') \} \). In other words, it contains all possible conjunctions containing all features in subsets of \( F - \{f(l), f(l') \} \) of length one, two and so on, up till \( k \). The number of possible conjunctions containing all of \( j \) features is \( 2^j \). Thus,

\[
|A_k(l \succ l')| < \sum_{i=0}^{k} \binom{N}{i} \times 2^i
\]

This is an inequality because \( |F - \{f(l), f(l') \}| < N \). For constant \( k \), this value grows as \( O(N^k \times 2^k) \). Since \( k \) is a constant, this reduces to \( O(N^k) \), giving a polynomial bound on the time spent per question.
6.3.6 Error in Learning

We define the true error as the size of the symmetric difference of the original edge set and the edge set generated by the learned theory. This was defined mathematically in equation (4.1) in section 4.2.

We restricted the learning problem to small rules, in terms of the number features factored in each preference statement. This allowed us to learn rules having antecedents of up to a constant length \( k \) with a polynomial number of questions. If the underlying theory only contains rules having such a size bound, this algorithm would learn these two types of rules with complete accuracy.

We now look at other structures of the underlying theory and analyze the error in those cases. Consider rules having the underlying simple statement of the form \( l \succ l' \) for some literals \( l \) and \( l' \). We briefly mentioned that as the number of features in the antecedent increases, the corresponding rule would generate fewer edges in the preference graph. This can be illustrated by the following example: Let the set of feature set be \( F = \{ f_1, f_2, f_3, f_4 \} \). Consider the rule \( f_2 \rightarrow f_1 \succ \neg f_1 \). The corresponding rule in the intermediate representation is \( 11** \succ 01** \), which generates four edges in the preference graph. Now consider the rule \( f_2 \land f_3 \rightarrow f_1 \succ \neg f_1 \). The corresponding rule here is \( 111* \succ 011* \), which generates only half the number of edges as the previous one. In general, a rule \( r = p \rightarrow (l \succ l') \), where \( p \) is conjunction over \( F - \{ f(l), f(l') \} \), would generate \( 2^{N-\text{size}(p)+k} \) edges, where \( k = |\{f(l), f(l')\}| \). Thus, each rule in the hypothesis space generates either \( 2^{N-\text{size}(p)-1} \), in case \( f(l) = f(l') \), or \( 2^{N-\text{size}(p)-2} \) otherwise, where \( p \) is the antecedent. For simplicity of calculations, we consider the strict upper bound of \( 2^{N-\text{size}(p)} \), noting that this would not affect the asymptotic analysis. We can now generalize the problem as follows: We define a distribution over the number of rules of the form \( p \rightarrow l \succ l' \). Let the number of rules of the form \( p \rightarrow l \succ l' \), be given by \( g(\text{size}(p)) \). Thus, the following gives the total number of edges in the graph generated by rules having the form \( p \rightarrow l \succ l' \):

\[
|E| = \sum_{i=1}^{N} g(i) \times 2^{N-i} \quad (6.6)
\]

We note here that the hypothesis space consists of all possible rules and hence, would also learn rules which can be deduced by transitivity. The preceding calculation, thus, includes edges generated by transitivity.

Now, all rules having at most \( k \) literals in the antecedent would be learned with
complete accuracy and the remaining pruned. Thus, the error would now be given by the ratio of the edges generated by the pruned rules and the entire edge set given by equation (6.6). This gives us the following true error:

\[
\frac{\sum_{i = k+1}^{N} g(i) \times 2^{N-i}}{\sum_{i = 1}^{N} g(i) \times 2^{N-i}}
\]

We consider the following cases for \( g(i) \):

**Case 1** \( g(i) = c \): Here, \( c \) is a constant. This case considers a uniform distribution of rules of all lengths. This is equivalent to saying for each statement of the form \( l \succ l' \), the antecedent could contain with equal probability some, all or none of the remaining features. In practice, this may be assumed if little is known about the dependencies between features and the underlying structure. The error for this case is:

\[
\frac{\sum_{i = k+1}^{N} g(i) \times 2^{N-i}}{\sum_{i = 1}^{N} g(i) \times 2^{N-i}}
= \frac{\sum_{i = k+1}^{N} \left( \frac{1}{2} \right)^i}{\sum_{i = 1}^{N} \left( \frac{1}{2} \right)^i}
= \frac{1}{2^k}
\]

This indicates a constant error, dependent on the constant \( k \). Since \( k \) would be an input to the learning algorithm, it can be looked upon as a stopping criteria.

**Case 2** \( g(i) = i \): In the general case, one may expect to encounter theories consisting of a some higher level or small-sized rules, generating more number of edges and some number of exceptional cases, specified by rules of longer length. Consider for example the rules, \( f_2 \succ f_1 \) only when \( p = f_3 \land f_4 \land f_5 \land f_6 \) is satisfied and \( f_1 \succ f_2 \) all other
times. Formally, \( p \rightarrow f_2 > f_1 \) and \( \neg p \rightarrow f_1 > f_2 \). In such a case, if we limit \( k = 3 \), the rule \( p \rightarrow f_2 > f_1 \) would be pruned, since \( \text{size}(k) = 4 \) and the learned rule would be \( f_1 > f_2 \).

In order to consider theories having such structures, we consider two cases. The present case assumes a linear growth in the number of rules, in terms of the length of the rule. The error for this case is:

\[
\sum_{i = k+1}^{N} \frac{g(i) \times 2^{N-i}}{g(i) \times 2^{N-i}} = \frac{\sum_{i = k+1}^{N} i \times 2^{N-i}}{\sum_{i = 1}^{N} i \times 2^{N-i}} = \frac{2^{N-k}(k+2) - (N+2)}{2 \cdot 2^N - (N+2)}
\]

If we assume here that \( N+2 \ll 2^{N-k} \), this equation reduces to \( 1/2^k \). The assumption \( N+2 \ll 2^{N-k} \) restricts \( k \) to be small enough such that \( k \ll N - \log_2 N \).

**Case 3** \( g(i) = i^2 \): In this case, we assume a quadratic increase in the number of rules. This case considers theories having a significantly larger number of exception cases,
as compared to base statements. Following is the result:

\[
\sum_{i = 1}^{N} g(i) \times 2^{N-i} \sum_{i = 1}^{N} g(i) \times 2^{N-i}
\]

\[
= \sum_{i = 1}^{N} i^2 \times 2^{N-i}
\]

\[
= \frac{2^{N-k}\{(k + 2)^2 + 2\} - (N + 2)^2 - 2}{6 \times 2^N - (N + 2)^2 - 2}
\]

Once again, if we assume \((N + 2)^2 \ll 2^{N-k}\), we get an error of \(1/2^k\).

The preceding examples indicate that the error can be bounded to a small value choosing \(k\) accordingly. We note here that increasing \(k\) reduces the expected error, but increases the time required, \(O(N^k)\).

6.3.7 An Alternate Approach

The preceding learning problem can be handled by a slightly different technique, which we briefly outline in this section.

In the above problem, we learn rules of the form \(p \rightarrow l \succ l'\), for all those cases having \(l \neq l'\) and all conjunctions \(p\) over feature set \(F - \{f(l), f(l')\}\). Since there may be multiple conjunctions \(p_i\) such that \(p_i \rightarrow l \succ l'\), we can merge these as a single rule of the form \((p_1 \lor p_2 \lor \ldots \lor p_m) \rightarrow l \succ l'\). We can denote the antecedent as a disjunction of conjunctions \(P = p_1 \lor p_2 \lor \ldots \lor p_m\). Such statements are referred to be statements in Disjunctive Normal Form or DNF. Learning a general DNF is not known to be PAC-learnable, but the class of k-DNF problems, i.e., DNF statements in which each conjunction has at most \(k\) literals has been proven to be PAC-learnable [Va85].

We can now modify our learning technique as follows. The starting hypothesis would be rules of the form \(P \rightarrow (l \succ l')\), where \(P\) is the disjunction over \(A_k(l \succ l')\). We recall here that \(A_k(l \succ l')\) is the set of all conjunctions having at most \(k\) literals, defined
over features $F - \{f(l), f(l)\}$. We can adapt the algorithm defined in [Va85] to PAC-learn the DNF for each simple statement of the form $l \succ l'$. We note here that the algorithm described above for pruning the conjunctions is very similar to the algorithm described in [Va85], with the difference that we seek to eliminate each of the conjunctions from at least one of the two sides, namely $l \succ l'$ or $l' \succ l$. This is done to avoid a theory that generates both $(m, m')$ and $(m', m)$ for some models $m$ and $m'$. If we PAC-learn the antecedent for the rule, we would be asking fewer number of questions (or observing fewer examples), but we may need some kind of post-processing step, to prune some of the conjunctions from either side in order to have a consistent theory. We note here that in this technique we repeat the PAC learning algorithm of Valiant a polynomial (in $N$) number of times since there is a polynomial number of simple statements $l \succ l'$. This leads to compounding of the error, $\epsilon$, occurring while learning each of the rules. Thus, the number of questions asked per rule would have to be modified, so as to have the overall error to be less than $\epsilon$ with probability of at least $1 - \delta$. We do not analyze this formally in this thesis.

6.3.8 Going Beyond Pairwise Tradeoffs

The approximation algorithms discussed in the earlier section looked at rules of the form $p \rightarrow (l \succ l')$, having $\text{size}(p)$ at most some constant $k$. The idea behind this approach was to learn a summary of the preferences and do so in time polynomial in $N = |F|$. We can extend this idea to learn generic rules of up to a fixed length and still bound the error as we had done in the preceding discussion. In this section we formalize this idea and explain how one can extend the preceding algorithm for the same. We also present an analysis of the complexity and error during learning.

We denote $\text{size}(p \succ q)$ as the size or the length of the rule $p \succ q$, which is defined as the number of features present in the rule. Formally, $\text{size}(p \succ q) = |s(p) \cup s(q)|$, where $s(p)$ is the support of $p$. In the following, we look at a learning algorithm for learning all rules over $F$ having size at most some constant $k$. We note that this is very similar to the earlier analysis, with the difference that the constant $k$ describes the size of the entire rule, rather than just the antecedent. Thus, for all rules of the form $f \succ -f$, the hypothesis set would consist of all rules $p \rightarrow f \succ -f$, having $\text{size}(p)$ at most $k - 1$.

This appears to be a simple extension of the earlier algorithm, with the difference that depending on the number of features in the underlying simple statement, we would
learn antecedents having monotonically decreasing sizes. While this is the basic theme, there are some other issues that are required to be discussed before porting the algorithm to this idea. Consider a statement of the form \( f_1 \succ f_2 \land f_3 \). In order to learn this rule, the queries generated by the algorithm would need to cover all of following rules:

1. \( f_1 \land \neg f_2 \land \neg f_3 \succ \neg f_1 \land f_2 \land f_3 \)
2. \( f_1 \land \neg f_2 \land f_3 \succ \neg f_1 \land f_2 \land f_3 \)
3. \( f_1 \land f_2 \land \neg f_3 \succ \neg f_1 \land f_2 \land f_3 \)

The reason behind this is simply the semantics of the rule: all models satisfying \( f_1 \land \neg (f_2 \land f_3) \) are preferred to ones satisfying \( \neg f_1 \land f_2 \land f_3 \). The first conjunction gives rise to a disjunction, complicating the problem slightly. The algorithm discussed previously constructs queries to deduce which conjunctions should be eliminated from the antecedent, keeping the features in the underlying base statement the same. In order to simplify the learning procedure, we take the redundant approach here of learning, in the preceding case, the three rules enumerated rather than the rule \( f_1 \succ f_2 \land f_3 \). This can be generalized by considering only those rules for which the conjunctions on either side have the same features. Thus, the hypothesis set would consist of the rules \( p \succ q \), such that \( s(p) = s(q) \). We note here that pairwise tradeoff rules of the form \( f_1 \succ f_2 \) would be treated as \( f_1 \land \neg f_2 \succ \neg f_1 \land f_2 \), which have the same semantics.

We now define the hypothesis space formally. Continuing from the earlier discussion, we denote \( C(F) \) to be the set of all conjunctions over feature set \( F \). The hypothesis space now would be all rules of the form \( p \succ q \), such that,

1. \( p, q \in C(F) \)
2. \( p \neq q \)
3. \( s(p) = s(q) \). Note here that this also implies \( \text{size}(p) = \text{size}(q) \)
4. \( \text{size}(p) \leq k \), for some constant \( k \).

We denote the hypothesis space by \( H \).

For some rules \( p \succ q \), it may be the case for some feature \( f \) in \( s(p) \), the same literal \( f \) or \( \neg f \) is present in both \( p \) and \( q \). Formally, there may exist \( f \) in \( s(p) \), such that \( f(p) = f(q) \). Such a literal can be considered to be a conditional, i.e., a part of antecedent
for the rule. The underlying simple statement for a rule \( p \succ q \) in our current hypothesis space, would be the statement \( p' \succ q' \) obtained by eliminating from either side of \( p \succ q \), each feature \( f \) such that, \( f(p) = f(q) \). For each rule \( r = p \succ q \) in the hypothesis space, let \( S_u(r) \) denote the underlying simple statement for \( r \). Formally, \( S_u(r) = p' \succ q' \) such that \( s(p') = s(q') = \{ f \in s(p) \mid f(p) \neq f(q) \} \) and for all \( f \) in \( s(p') \), \( f(p') = f(p) \) and \( f(q') = f(q) \). We note here that the concept of the underlying statement defined for rules in \( H \) is analogous to the concept of difference rule (definition 5.1) defined for rules in the intermediate language, in the chapter on exact learning. On similar lines, we say two rules \( r_1 = p_1 \succ q_1 \) and \( r_2 = p_2 \succ q_2 \) in \( H \) are difference equivalent, denoted as \( r_1 \equiv_D r_2 \), if they have the same underlying simple statement. Thus, \( r_1 \equiv_D r_2 \) iff \( S_u(r_1) = S_u(r_2) \). For any rule \( r \) in the \( H \), we write \([r]\) to denote the set of all rules \( r' \) in \( H \) such that \( r \equiv_D r' \). Thus, \([r]\) = \( \{ r' \in H \mid r \equiv_D r' \} \). As stated earlier, for rule \( p \succ q \), the features assigned same values by \( p \) and \( q \) form the antecedent for the rule. We write \( A(r) \) to denote the conjunction forming the antecedent for rule \( r \). Formally, for \( r = p \succ q \), \( A(r) \) is a conjunction over literals \( l \) such that \( l \) is present in both conjunctions \( p \) and \( q \). Given the set \([r]\) for any rule \( r \) in \( H \), the union of \( A(r') \) for all rules \( r' \) in \( [r] \) gives us the set of all possible antecedents for the underlying statement \( S_u(r) \) having length at most \( N - \text{size}(S_u(r)) \). Note here that these conjunctions are defined over features \( F - s(p') \), where \( p' \) is \( GS(S_u(r)) \). In case of the earlier algorithm, we had defined the set of possible antecedents as \( A_k(s) \), where \( s \) is the underling simple statement. We redefine this slightly for this case: given a rule \( r = p \succ q \) in \( H \), \( A_k(r) = \bigcup_{r' \in [r]} A(r') \). Note here that we retain the value \( k \) to indicate the hypothesis contains rules of length at most \( k \).

The basic strategy of the algorithm is to consider each equivalence class, \([r]\) and ask questions comparing models \( m \) and \( m' \) such that, \( m \succ m' \) satisfies the statement \( S_u(r) \) and \( m, m' \) satisfy the conjunctions in \( A_k(r) \), one at a time. Once again, we can ask two questions: one assigning 0 and the other assigning 1 to all features not in \( s(S_u(r)) \). Employing a similar question formulation technique as earlier, for each \([r]\), we ask the following number of questions:

\[
2 + \sum_{i=1}^{k - \text{size}(S_u(r))} \binom{N}{i} \times 2^i
\]

We note here that this value is a strict upper bound since for rules \( r \), having \( S_u(r) = r \), we need only ask one question. This equation, for constant \( k \), is bounded by \( N^k \times 2^k \), which
is $O(N^k)$. Now, the next step is calculating the total number of classes $[r]$. This is given by the number of different underlying statements. In other words, this is the number of rules $r$, having size($r$) at most $k$ and $S_u(r) = r$. This can be calculated as the number of rules $p \succ q$ over $N$ features having $s(p) = s(q)$, size($p$) $\leq k$ and $f(p) \neq f(q)$ for all $f$ in $s(p)$. As an example, for size($p \succ q$) = 1, we get all rules of the form $f \succ \neg f$ and $\neg f \succ f$, which is $2N$. For size($p \succ q$) = 2, we get all possible rules defined over pairs of features, i.e., $\binom{N}{2} \times 2^2$ rules. Continuing so on, gives us the total as:

$$\sum_{i=1}^{k} \binom{N}{i} \times 2^i$$

Once again, this bounded as $O(N^k)$ for constant $k$. This gives use the total number of questions to $O(N^k) \times O(N^k) = O(N^{2k})$. We note that a tighter bound on the number of questions can be calculated by observing that for higher values of $i$, the number of classes $[r]$ is higher (given by $\binom{N}{i} \times 2^i$) but the number of questions asked is less (given by $(\binom{N}{k-i} \times 2^{k-i})$). However, the asymptotic bound of $O(N^k)$ is tight and is polynomial as required.

The analysis of the error in learning remains very much similar. If the underlying theory has a structure such that no rule has a length of more than $k$, the algorithm would learn the theory with complete accuracy. As earlier, we also consider theories with certain different structures. Particularly, we consider theories having some distribution over the size of the rules.

While defining the hypothesis space, we specified that all rules $p \succ q$ in the hypothesis have the form that $s(p) = s(q)$. The underlying theory, however, may contain any rule in $L$, as long as it is consistent. In order to define a distribution over the rules, we convert the underlying theory $T$ to an expanded form, so that each rule in the modified theory has the above restricted structure. Each rule in $L$ can be expanded to generate a set of rules having the structure of the rules in $H$ by converting it into the set of rules in the intermediate representation and converting each rule back by using the reverse translation described in [McG02], which was discussed briefly in section 5.5. The example discussed at the start of this section is an illustration of such a translation. Let $T'$ denote the theory obtained by converting all rules in the underlying theory $T$ in this manner. We define $g(i)$ as the function defining the number of rules $r$ in the theory $T'$ having size($r$) = $i$. Now, number of edges generated by a rule $r$ having the above structure would be $2^{N-size(r)}$. Thus,
the total number of edges would be:

$$|E| = \sum_{i=1}^{N} g(i) \times 2^{N-i}$$  \hspace{1cm} (6.7)

We note here that this is the same as the equation (6.6), obtained for the earlier algorithm. Also, the algorithm described in this section would learn all rules having size at most \(k\). Thus, the error would be given by,

$$\sum_{i=k+1}^{N} g(i) \times 2^{N-i}$$

$$\sum_{i=1}^{N} g(i) \times 2^{N-i}$$

This reduces the error analysis for this algorithm to the same as that of the earlier algorithm, retaining the results.

### 6.4 Summary and Further Work

As was mentioned earlier, the preference graph has the edge set, \(E = [T]^*\). The transitivity of the edges has certain implications. The justification behind considering the transitive closure rather than simply the union over rules is that if instead of a preference graph, the queries are posed directly to the agent, then a consistent set of answers would be indeed the transitive closure. This allows us to model the agent’s preferences directly, although with some redundancy in the output. The question now arises whether the output theories would generate the same preference ordering as the original graph. One can look at the utility functions defined over the preference graph in [McG02]. The ordinal utility functions in this work are defined over a graph containing only those edges \((m, m')\) such that \(m \succ_r m'\) for some rule \(r\) in the underlying theory. However, while learning a preference theory, we consider all orderings in the transitive closure and generate rules to cover all of these. One way of comparing equivalence between this output theory \(T'\) and the original theory would be construct utility functions as described in [McG02] and compare the utility of the models as given by the two theories \(T\) and \(T'\). We can observe here that translating the theory \(T'\) into a preference graph which does not contain edges due to transitivity, would, in the general case, generate a graph \(G'\) having more edges than the corresponding graph.
For the underlying theory $T$. We note here that although we have extra edges, all these are generated due to transitivity. Intuitively, this would not generate different orderings of the models. The individual utility values would certainly be different for some of the utility functions described in [McG02]. These are, however, ordinal utility functions and hence the values matter only as far as they give the same ordering. We do not give formal proofs for this, but the intuition is that since we would generate all the edges in $G$, and in addition, generate the edges due to transitivity, the utility of high valued models would increase a little further, without affecting the ordering. As an example, if $(m_1, m_2)$ and $(m_2, m_3)$ are edges in the original non-transitive graph, $G$, we would generate in addition, the edge $(m_1, m_3)$. This increases utility of model $m_1$, without affecting the ordering.

In the approximate case, the form of the input poses another interesting question. If we have an oracle that has the underlying theory and we now ask of it the edges in the non-transitive graph, can we effectively summarize the theories? It may be the case that we have a much bigger and redundant description of preference theory. Using analysis as the one discussed in this thesis, one could look at means of simplifying the underlying theory and learning higher level description of the agent’s preferences.

We discussed an algorithm learning higher level rules describing goal features and tradeoff information and mentioned briefly that we can use the same approach, with some modifications, to learn a conditional preference network or CP-net, [BBHP99]. A CP-net is a directed network having one node per feature, $f$ in $F$ and associated, with each node, a conditional preference table (CPT) storing preference comparisons between $f$ and $\neg f$, under different conditions. The conditions are nothing but the antecedent for the rule. We say $f_i$ is a parent node of $f_j$ is there is a directed edge from $f_i$ to $f_j$. The set of parents $P(f)$ of a feature $f$ indicates that the preference between $f$ and $\neg f$ is independent of all features in $F - P(f)$, given some assignment for all features in $P(f)$. Thus, the CPT for each node $f$ contains a set of preference statements between $f$ and $\neg f$, under different assignments of features in $P(f)$. We can now use the approximation algorithm discussed in section 6.3.3, to learn a CP-net over the features $F$. The idea would be to learn rules of the form $f \succ \neg f$ under all possible conditionals, as was done. An additional post-processing step would be required to prune the repeated conjunctions from the antecedent. In order to illustrate, if $f_i \land f_j$ is a part of the antecedent, we would also have $f_i \land f_j \land f_k$ and $f_i \land f_j \land \neg f_k$ in the learned DNF. These need to be pruned in order to be able to deduce exactly which features form the parent features of the current feature being learned. For each feature $f$, we can
now look at the DNF to deduce the features in $P(f)$, allowing us to construct the network. We note here that we had imposed a restriction on the length of the antecedent to be some constant $k$. The implication this has in terms of CP-nets would be that no feature should have more than $k$ parent nodes. Formally, for each feature, $f$ in $F$, we have $|P(f)| \leq k$. A more detailed analysis of this approach for learning CP-nets is one possible extension of this work.

We defined the notion of goals as propositions $p$ such that $p \succ ¬p$, where $p$ is a conjunction in $\mathcal{L}$. We restricted our learning problem to only goal features, instead of general goal statements. In case of general goal propositions, the difficulty arises noting that goal propositions of higher length generate more edges and require more questions to be asked. Consider for example a goal proposition $p = f_1 \land f_2$ for feature set $F = \{f_1, f_2, f_3, f_4\}$. The rule $p \succ ¬p$ expresses a preference of models satisfying $p$ over those satisfying $¬p$, other things being equal. Now, considering models in $\mathcal{M}(s(p))$, there is exactly one model satisfying $p$ and all the remaining $2^{\text{size}(p)} - 1$ satisfy $¬p$. Thus, in this case, the number of edges would be given by $(2^2 - 1) \times 2^2$. In general, for goal statements, $r = p \succ ¬p$, number of edges in $[r]$ would be $(2^{\text{size}(p)} - 1) \times 2^{N - \text{size}(p)} = 2^N - 2^{N - \text{size}(p)}$. Thus, the number of edges generated increases with $\text{size}(p)$. In case of goal features, we restricted our length to some constant $k$, noting that gain in accuracy, decreases for rules of higher sizes. This does not hold true for general goal propositions, due to which we cannot bound the error like we did in case of learning goal features. Thus, the algorithm discussed here has been restricted to the case of goal features.
Chapter 7

Conclusion

The work described in this thesis treated formally the notion of learning preference theories. We presented and analyzed an algorithm to generate equivalent theories. In general, this does not turn out to be feasible owing to the exponential number of preference orderings generated by moderate-sized theories. Considering approximate algorithms, we expressed preference learning under the Probably Approximately Correct learning framework. Although, theories with single rules are PAC-learnable, an arbitrary preference theory has a superexponential hypothesis space, making it a hard problem. We then considered approximating high-level summaries of the theories, and depicted certain practical assumptions under which we can learn a theory with a bounded error. Among other extensions to this work, we mentioned one technique to learn a CP-net given a set of preference comparisons.
Bibliography


