DEFT Guessing:
Using Inductive Transfer to Improve Rule Evaluation from Limited Data

By
Mark Darren Reid

A thesis submitted for the degree of
Doctor of Philosophy

THE UNIVERSITY OF
NEW SOUTH WALES

SYDNEY · AUSTRALIA

School of Computer Science and Engineering,
The University of New South Wales.

May 2007

Originality Statement

'I hereby declare that this submission is my own work and to the best of my knowledge it contains no materials previously published or written by another person, or substantial proportions of material which have been accepted for the award of any other degree or diploma at UNSW or any other educational institution, except where due acknowledgement is made in the thesis. Any contribution made to the research by others, with whom I have worked at UNSW or elsewhere, is explicitly acknowledged in the thesis. I also declare that the intellectual content of this thesis is the product of my own work, except to the extent that assistance from others in the project's design and conception or in style, presentation and linguistic expression is acknowledged.'

Signed:_____

Mark Darren Reid

Abstract

Algorithms that learn sets of rules describing a concept from its examples have been widely studied in machine learning and have been applied to problems in medicine, molecular biology, planning and linguistics. Many of these algorithms used a separate-and-conquer strategy, repeatedly searching for rules that explain different parts of the example set. When examples are scarce, however, it is difficult for these algorithms to evaluate the relative quality of two or more rules which fit the examples equally well.

This dissertation proposes, implements and examines a general technique for modifying rule evaluation in order to improve learning performance in these situations. This approach, called Description-based Evaluation Function Transfer (DEFT), adjusts the way rules are evaluated on a target concept by taking into account the performance of similar rules on a related support task that is supplied by a domain expert. Central to this approach is a novel theory of task similarity that is defined in terms of syntactic properties of rules, called descriptions, which define what it means for rules to be similar. Each description is associated with a prior distribution over classification probabilities derived from the support examples and a rule's evaluation on a target task is combined with the relevant prior using Bayes' rule. Given some natural conditions regarding the similarity of the target and support task, it is shown that modifying rule evaluation in this way is guaranteed to improve estimates of the true classification probabilities.

Algorithms to efficiently implement DEFT are described, analysed and used to measure the effect these improvements have on the quality of induced theories. Empirical studies of this implementation were carried out on two artificial and two real-world domains. The results show that the inductive transfer of evaluation bias based on rule similarity is an effective and practical way to improve learning when training examples are limited.

Acknowledgements

The School of Computer Science and Engineering has been a thoroughly stimulating place to research. My sincere thanks go to my supervisors, Professors Claude Sammut and Arun Sharma. Their encouragement and insight has been invaluable in seeing this thesis through to completion. Other academics within the School, such as Norman Foo, Mike Bain and Eric Martin, were instrumental in clarifying some of my early research ideas, as were talks with visiting researchers such as Peter Flach. My fellow post-grads, Eric McCreath, Malcolm Ryan, Waleed Kadous, and James Westendorp (amongst others) also helped make my time at CSE very enjoyable, with many and varied discussions on, around and instead of our research.

Malcolm Ryan is thanked a second time for his effort proof-reading this thesis, as is Julieanne Lamond. My anonymous reviewers also provided some useful suggestions and corrections. Of course, any remaining erors are mine alone.

Implementing the DEFT system would not have been possible were it not for the ever-helpful members of the YAP Prolog and ALEPH mailing lists. Vitor Santos Costa and Ashwin Srinivasan were especially gracious in responding to my questions and resolving my issues with their respective systems.

I was fortunate enough to work in a number of stimulating research roles during my candidature. For this reason, I'd like to thank Dan Oblinger, my mentor during my internship at IBM's T. J. Watson Research Center, as well as my managers and team-mates at CISRA: Ian Gibson, Joe Thurbon, Steve Hardy and Barry Drake. Thanks also to Bernd Hammann, Sherali Karimov and Etienne Deleflie at Proxima Technology.

I cannot overstate how much I appreciate the love and support my friends and family provided me during the ups and downs of my Ph.D. For that, many thanks go to Bron & Anthony, Kat & Stu, Brom & Em, Bill, Nick, Jack & Anja, Ben & Shel, Peter & Ange, all the folk from the dog-park, and many others including my wonderful in-laws Mark, Marg, Alison, Alexander and the two Daves. Extra heapings of gratitude must go to my own Mum and Dad, my brothers, Paul and Darren, and my grandmother Doreen - to whom this work is dedicated - for their unflagging love and understanding.

Finally, for being my companion through the darkest parts of the research tunnel and making the light at its end so enticing, my deepest love and gratitude go to my beautiful wife, Julieanne.

For Nana

Contents

At each increase of knowledge, as well as on the contrivance of every new tool, human labour becomes abridged.

- Charles Babbage [1832]

CHAPTER 1

Introduction

Humans have a remarkable ability to infer general concepts from specific examples. Learning a new language, recognising faces, choosing food from a menu, running a pharmaceutical trial, tuning an engine or designing a house all require us to recognise patterns amongst the particular and extract from them the characteristic. This skill, called induction, enables us to summarise our knowledge of the world and use it to predict, decide and act in novel situations. Induction's role in our everyday lives as well as our scientific, industrial, and artistic endeavours has driven attempts to create tools that can mimic our ability to generalise, extending our inductive reach to problems which are too complicated or too time-consuming to solve without them. The theoretical and empirical study of algorithms that form the basis of these tools is a fundamental area of research within machine learning, found at the intersection of probability theory, statistics and the cognitive and computer sciences.

This dissertation proposes an inductive tool to extend our inductive ability for concept learning problems defined by the following three characteristics. Firstly, the induced concepts are to be expressed using sets of *rules*. These are statements of the form "if an example has particular properties then it is an instance of the concept". Secondly, the examples available to learn from are *limited*. That is, the number of examples is significantly fewer than required to learn a good generalisation. The third characteristic is that examples of *related* concepts are also available when learning. The proposed approach to this type of problem is called *Description-based Evaluation Function Transfer* (DEFT). This DEFT approach to concept learning improves the generalisations made from limited examples by exploiting extra information available in the examples for the related concepts. The remainder of this dissertation presents the theory behind this approach, including what is meant by "related concepts" in this context, its implementation and application.

1.1. Tools for Induction

Thanks to continuing advances in computing and the rise of the internet, we presently have the ability to cheaply and easily store, retrieve and transmit

huge quantities of information. This ever increasing amount of information that pervades our lives has driven the need for better tools for its management. Especially important are inductive tools that allow us to intelligently summarise this information in order to derive meaning. Without them we are left asking, "Where is the knowledge we have lost in information?" [Eliot, 1961]

Today, data-mining algorithms are commonly used on large commercial databases to uncover sales patterns, possible fraud, and network outages [Fayyad et al., 1996]. The proliferation of data generated in the biological sciences has led to the creation of systems such as the "Robot Scientist" which are able to repeatedly propose, run and analyse experiments that investigate gene functionality [King et al., 2004, Muggleton, 2006]. Also, at the personal level, managing information has been aided by inductive tools for document filing [Bao et al., 2006], email organisation [Ho et al., 2003, Crawford et al., 2002] and spam detection [Graham, 2002].

The learning systems used in all of the above cases are capable of learning using a small amount of expert guidance to draw generalisations from data that would otherwise be impossible for humans. In his *Introduction to Cybernetics*, Ashby [1956] calls this sort of cognitive enhancement "intelligence amplification", likening the "intellectual power" or "power of appropriate selection" that humans display when solving problems to other forms of power that can be amplified by machine such as sound and electricity. The cost to the expert when supplying input to these systems can be broken down into a variety of categories. These include the cost of collecting and classifying training examples as well as other "Human-Computer Interaction" (HCI) costs such as "finding the right features for describing the cases, finding the right parameters for optimizing the performance of the learning algorithm, converting the data to the format required by the learning algorithm, analyzing the output of the learning algorithm, and incorporating domain knowledge into the learning algorithm or the learned model" [Turney, 2000].

The relative burden of these costs will depend on the problem and the learning system. For some problems, the cost of collecting and classifying examples is much greater than any HCI costs. In these cases, the inductive transfer system described in this work can be seen as a method for improving the amplificatory power or efficiency of rule learning systems. When used successfully, the relatively small cost of providing a related task can result in good generalisation from significantly fewer collected and classified examples than would otherwise be required, thus achieving the same output from the system for a lower cost of input.

1.2. A Motivating Example

In his discussion of costs for inductive learning, Turney [2000] notes that "every human is a potential case for medical diagnosis, but we require a physician to determine the correct diagnosis for each person". This example of a problem with high classification costs is expanded upon below to motivate the approach described in this work.

Consider a physician who, after performing several expensive but conclusive tests has determined that five patients have a new type of disease while another seven do not. She would like some method of predicting whether or not other patients have the same disease based on their attributes, the symptoms they present or the results of some cheaper and less time-consuming tests. In addition it is important that the predictive model be communicated to the physician's colleagues to allow them to identify new cases in their hospitals. Ideally, some simple set of rules would be used to make new predictions such as "if the patient has high-blood pressure, is over 50, chest pain and a fever then the disease is likely to be present".

The hospital the physician works for has a large database of all the case histories and other medically relevant information for each of the patients in question. Sifting through all of this information looking for meaningful patterns amongst the 12 patients would be extremely time-consuming. Ideally, some learning tool would be connected to this database which can help her find models that discriminate between those patients with and without the new disease. Naïvely applying such a tool would fail since due to the extremely large number of chance correlations between the presentation of the disease and collections of patient features such as eye colour, presence of a birthmark, or height. Without some kind of guidance, the learning tool will suggest models that are virtually meaningless to the physician even though they fit the available data perfectly.

In order for the learning tool to be useful it must take into account not only the fit of the models to the examples when assessing their quality but also how well they fit with the physician's expectations. For example, suppose two models for the disease, one that tests whether the patient has chest pain and the other that tests the patient eye colour, both fit the data equally well. For most diseases, the physician would quickly disregard the latter model. In making this decision she brings much of her training, general knowledge and experience with other diseases to bear.

Translating the inductive preferences of an expert into what is called an *inductive bias* for a learning algorithm can be difficult. Not only does an expert

need to know about the learning task but also how the learning algorithm's settings affect the resulting theory. The primary research objective of this work is to allow domain experts to communicate a bias for a task by saying they expect the good theories on this task to be similar to the good theories on that task. The thesis is that when training data is limited this approach to defining bias can improve the performance of induced theories.

1.3. Inductive Transfer of Evaluation Bias

The aim of an inductive transfer method is to allow a domain expert to express their expectations to a learning system by saying, "I believe the theories that perform well on task A will be similar to the theories that perform well on task B". Here, task A is the learning problem to be solved and is called the *target* task and task B is called the *support*. It is the job of an inductive transfer system to use the information available in the support task to turn this statement into a bias that can be used to aid learning on the target task. A bias in this context is any decision a learning system might make to choose between two generalisations which is not strictly based on the task's examples [Mitchell, 1980].

When experts use a rule learning system to solve a learning task they bring domain knowledge to bear in the choice of representation, algorithm, background information, constraints and more. These methods for communicating bias to a learning algorithm can be extremely useful when the expert can articulate her knowledge and is skilled with those methods. Systems that perform inductive transfer can complement these methods in situations in which domain knowledge is more difficult to articulate or the expert has less experience with these traditional ways of expressing biases. This is especially true when there are a limited number of examples available for the concept to be learnt. Expressing a strong and correct bias for a learning task such as this requires that the expert already have a very clear idea of what the unknown concept might be.

The biases used by most rule learning algorithms can be categorised into three main types: the *language bias* which determines which rules will constructed and tested by the learner, the *search bias* which determines the order in which the rules will be tested and the *evaluation bias* which determines how the quality of rules will be assessed[1]. Existing inductive transfer systems for rule learning have focused on modifying a learner's language and search biases.

[1]The relationship of these three categories to existing characterisations of bias are discussed in Section 2.3.4.

The approach to inductive transfer put forward here is novel in its modification of evaluation bias.

The method DEFT uses for modifying evaluation bias can be understood with reference to the opening example. When the epidemiologist is presented with two rules to assess, one testing chest pain and the other eye colour, her preference for the former over the latter could be explained as follows. Although both rules fit the training examples equally well the researcher knows that rules which test for eye colour have had no predictive value for past diseases that are, in her mind, similar to the present one. That is, they are equally likely to misclassify an example as correctly classify it. On the other hand, rules that test for chest pain have, on similar diseases in the past, tended to classify more examples correctly than not. An evaluation that takes these extra-evidential assessments into account will result in a preference for the chest pain rule over the eye colour rule even though they have the same assessment on the training examples.

It is important to note that inductive transfer is only another way to express bias. If the epidemiologist were to express a wrong bias for the task by erroneously believing it similar to another task then rules learnt by the learning system would be poor. As Englebart [1962] puts it,

> [J]ust as the mechanic must know what his tools can do and how to use them, so the intellectual worker must know the capabilities of his tools and have good methods, strategies, and rules of thumb for making use of them.

1.4. Research Outline and Contributions

Several concepts and mechanisms need to be clarified to implement an inductive transfer system that functions in the manner described above. Firstly, some way of expressing and incorporating extra-evidential beliefs about the classification errors made by rules must be expressible to a rule learning algorithm. Secondly, some way of using support tasks to derive an evaluation bias that can be used in this way is required. Thirdly, to be useful, these mechanisms should be efficient and not place too large an overhead on the expert who must use them. Finally, some theory of task similarity is required in order to understand when the use of these mechanisms will result in improved generalisation performance.

Bayesian methods of probability estimation [Good, 1965] are used to address the first of these requirements as it is a theoretically well-founded way of combining prior beliefs with observations. Many evaluation functions rely

on estimates of classification probabilities which are poorly estimated when training samples are limited. These estimates can be improved through the use of priors that reflect extra beliefs about the true classification probabilities of a rule. The precise prior to be used to evaluate a given rule depends on its syntactic features, such as the number and type of conditions it tests. To address the second requirement, these priors can be derived from support tasks by evaluating a random sample of rules and determining how frequently each syntactic feature of a rule coincides with a correct or incorrect classification. By assuming the independence of the rule features, the collection and use of these derived frequencies as priors can be efficiently implemented. Finally, the focus on similarity between rules and the use of priors gives rise to a natural definition of similarity between tasks.

The main contribution of this work is a novel technique for modifying rule evaluation in order to improve learning performance from limited data. Flowing from this centrepiece is a theory of task similarity. To the author's knowledge this is the first theory of inductive transfer that is directly applicable to rule learning systems. The implementation of this approach is the first such system to modify evaluation heuristics used during rule search. A comprehensive literature survey of all the existing systems that implement transfer for rule learning is an additional contribution to scholarship in this area.

Some of the ideas explored here have been the basis of several publications during the author's candidature. An early attempt at handling limited data in inductive logic programming is described in [McCreath and Reid, 1999]. Work on concept reuse in reinforcement learning [Ryan and Reid, 2000] and the application of rule learning to symbolic planning for reinforcement learning [Reid and Ryan, 2000] also helped to shape some of the ideas explored here. The use of support tasks for bias learning were explored experimentally within a text-mining domain in [Oblinger et al., 2002]. Finally, the core of the research presented here was first described in [Reid, 2004].

1.5. Overview

The remainder of this dissertation is organised as follows. Chapter 2 examines the problem of rule learning when training data is limited. A brief survey is made of literature addressing the problem in general before going into detail about approaches to inductive transfer for learning rules expressed in first-order logic. Focus is given to a large and important class of rule learning algorithms, called "covering" or "separate and conquer" algorithms which include many well known learning systems such as CN2 [Clark and Niblett, 1989], Foil [Quinlan, 1990] and Progol [Muggleton, 1995]. With all of these

algorithms, the assessment of the quality of individual rules is a crucial task. When data is limited this rule assessment becomes unreliable and is identified as a major cause of poor generalisation.

Chapter 3 presents the theory behind description-based inductive transfer. The theory defines successful transfer in terms of estimated *classification probability error*, a measure of how well evaluation over training examples estimates the true misclassification rates. Sufficient conditions are given for when this error can be reduced through similarity-based transfer. These conditions use novel but natural definitions of the *similarity* between the support and target task and the *regularity* of the target task. An implementation of description-based transfer for the inductive logic programming system ALEPH [Srinivasan, 2001] is described in Chapter 4. This implementation splits the calculation of priors into two phases: a consolidation phase, which builds a statistical summary of a support task (called a Description Frequency Table) and a transfer phase, which uses the summary to compute classification priors. Efficient algorithms for both phases are presented and analysed.

To test the theory and implementation, DEFT was applied to four environments. The first of these is an artificial set of learning tasks in which the concepts are the reading preferences of five different people. The second environment involves concepts describing the movement of chess pieces. This domain has been used by other researchers to test representational approaches to inductive transfer [Khan et al., 1998] and makes it ideal for comparing them to DEFT's functional approach. The third environment involves predicting heart disease in patients from three different hospitals, tasks that have also been used previously in inductive transfer studies [Silver, 2000]. The fourth domain consists of the well-known tasks from the inductive logic programming literature: predicting mutagenesis and carcinogenesis [Srinivasan et al., 1994, 1997]. The results of these experiments are presented in Chapter 5. Finally, in Chapter 6 the main discoveries are reviewed in light of the objectives presented here and concludes with some new questions and suggestions for future research.

You're telling me the French word for "croissant" is "croissant". So French is just the same as English... Well, "see you later" or as the French would say, "see you later".

- Nick, from *Family Ties*

CHAPTER 2

Rule Learning from Limited Data

This chapter examines predictive concept learning of sets of rules and the difficulties that arise when training examples are not available in sufficient quantity. Section 2.1 provides an overview of terminology and definitions commonly found in the concept learning literature. As evaluation is a central theme of this dissertation most of this section is dedicated to measurements of the error and misclassification rates of hypotheses. Section 2.2 then focuses attention on relational rule learning: concept learning using theories which are represented by sets of rules which use predicate logic to express relational conditions. To further limit the scope, Section 2.3 discusses systems that use the popular "covering" strategy for constructing sets of rules by repeatedly finding single rules that account for unexplained examples. Rule induction by these systems is an optimisation problem and, when training data is limited, the accurate evaluation of rule quality is identified as a key difficulty.

Some approaches to learning from small sets of training examples are reviewed in Section 2.4. Of particular interest is the growing number of techniques aimed at reducing the dependence on expertly provided bias. In these approaches learning tasks are considered to be part of an environment of tasks that share some common bias requirements. Under this assumption it is possible for machine learning algorithms to transfer inductive experience from one learning task to another. While much of this research has been explored using artificial neural networks there is a small body of work on inductive transfer for relational rule learning systems which is reviewed in Section 2.5. Finally, the summary in Section 2.6 argues that there is no existing approach to inductive transfer for relational learning that modifies the procedures used to evaluate rules. As evaluation bias is considered an important bias for learning and the one most affected when training data is limited it is a natural candidate for inductive transfer. This observation opens the way to the main contribution of this thesis: the DEFT algorithm for learning and transferring evaluation bias described in the remainder of this thesis.

2.1. Concept Learning: Terminology and Definitions

The aim of this section is to introduce some of the basic terminology and definitions that will be used throughout the rest of this thesis. In very broad terms, the problem investigated here is one of solving concept learning tasks. Such a task involves developing a hypothesis that will correctly classify future instances given specific examples of when an object belongs to a particular category or not. This section provides an overview of problems of this type along with the methods used to evaluate the quality of the learnt hypotheses. Evaluation is one of the key themes of the work presented here and so some space is dedicated to discussing *contingency tables* and *classification probability matrices* or CPMs. These are summaries of the number and type of examples a hypothesis correctly or incorrectly classifies. When training examples are limited these summaries are estimated poorly and any evaluation scheme that relies on their values will be affected. These ideas play an important part in the discussion of rule evaluation in Section 2.3 and in Chapters 3 and 4 where the theory and implementation of DEFT is introduced.

2.1.1. Concept Learning. Following Mitchell [1997, §2.1], a *concept* can be thought of as a boolean function which classifies instances (objects, situations, *etc.*) into two sets: those said to be in the concept and those which are not. This abstraction captures much of the everyday use of the word "concept". When the instances are furniture they can be split into the concepts "chairs" and "not chairs", instances of mushrooms into "toxic" or "non-toxic" or each book in a library into "books I enjoy reading" and "those I do not". Formally, a concept will be defined as a boolean function $f : X \to \{+, -\}$ that maps *instances* $x \in X$ to *class labels* $y \in \{+, -\}$.

Given a concept, the class labels for specific instances can be deduced from the general case. The aim of *concept learning* is to do the inverse, that is, induce the general case from specific examples of it. Put formally again, an *example* (x, y) is an instance x with an associated label y. Whenever $y = f(x)$ the pair (x, y) is said to be an example of the concept f. An example is called *positive* or *negative* depending of whether its label is $+$ or $-$ respectively. A *concept learning task* consists of a set of examples E of some unknown *target* concept t. To solve a task a learner must provide a boolean function $h : X \to \{+, -\}$, called a *hypothesis*, that agrees with the target concept on

most or all of the instances.[1]

2.1.2. Evaluating Hypotheses. When a hypothesis is used to classify examples its performance on those examples can be summarised by counting the types of mistakes a classifier makes. These summaries, called *contingency tables*, can be used to assign a number of different types of measures of quality to a hypothesis depending on the relative importance of each kind of mistake. For example, a hypothesis to predict a customer's reading preferences might suggest a book the customer does not like. In many respects this type of error is worse than not suggesting a book the customer may have liked. Similarly, it is more important for a hypothesis about heart disease to correctly predict when someone has heart disease than it is to correctly predict when someone does not.

The weighing of the importance of different classification mistakes when assessing the final quality of a hypothesis will vary from learning task to learning task depending on the intended application of induced theories. Regardless of the specific function used to convert error counts into an assessment, the quality of any evaluation function is only as good as the values it depends on. For this reason, the theory developed in this dissertation will focus on the values in contingency tables for hypotheses rather than specific functions of them. In Chapter 3, a method is proposed for improving these estimates by making use of extra information provided by an expert in the form of tasks related to the one being solved. By examining these improvements independently of any specific evaluation function, this new method is applicable to a wider variety of problems. Definitions of contingency tables and their related terms are given below.

When a classifier h is tested against an example (x, y) its *predicted label* for x is given by $h(x)$ while the *actual label* for x is y. The result of the test can be summarised as the *classification pair* $[h(x), y]$. As both $h(x)$ and y can take on the values $\{+, -\}$ there are four possible classification pairs. The classification pair is called *positive* or *negative* whenever the predicted label is $+$ or $-$. The modifiers *true* or *false* are added as a prefix to these names to denote whether the actual label in the pair agrees or disagrees with the predicted one. Each of the four classification pairs are shown in Table 2.1 along with their names and abbreviations.

[1]Concept learning tasks can be framed more generally by making the weaker assumption that the target concept is described by a distribution over $X \times \{0, 1\}$ rather than a function. A stronger, non-*agnostic* [Kearns et al., 1994] assumption is used here to simplify the exposition.

TABLE 2.1. Classification pairs with their names and abbreviations.

$[i, j]$	**Predicted**	**Actual**	**Name**	**Abbreviation**
$[+, +]$	$+$	$+$	True Positive	TP
$[+, -]$	$+$	$-$	False Positive	FP
$[-, +]$	$-$	$+$	False Negative	FN
$[-, -]$	$-$	$-$	True Negative	TN

When a set of examples is tested against a classifier a classification pair is created for each example in the set. A summary of these tests can be obtained by counting how many of each type of classification occurs over the examples. These counts are defined as a function of the classifier h and example set E as follows

$$n_{ij}(h, E) = |\{(x, j) \in E : h(x) = i\}| .$$

The arguments h and E will usually be dropped when there is no confusion as to the classifier or examples under consideration. A *contingency table* is a convenient way of storing these four counts for any particular classifier h and example set E. It is defined as

$$\mathbf{n}_E(h) = \begin{bmatrix} n_{++} & n_{+-} \\ n_{-+} & n_{--} \end{bmatrix} .$$

The total number of examples with actual label $+$ can be obtained by summing the values in the first column while the total number of examples predicted to be $+$ by the classifier can be read off as the sum of the values in the first row. Similarly, these values for the examples or predictions with negative labels can be determined from the value in the second column or row, respectively.

A matrix containing the relative frequencies of each type of classification pair can be derived from a contingency table. This *classification probability matrix* (CPM) is defined to be

$$\mathbf{p}_E(h) = \frac{1}{N}\mathbf{n}_E(h)$$

where $N = \sum_{ij} n_{ij}$ is the total number of examples in E. Assuming the examples are all of some concept t and their instances have all all been drawn independently according to some distribution P over X, the values in a CPM can be viewed as frequentist estimates of classification pair probabilities. That is, if a new instance $x \in X$ is drawn according to P, the probability the classification pair for x will be $[i, j]$ is

$$\Pr_{x \in X}(h(x) = i, t(x) = j) \approx p_{ij}(h, E) = \frac{1}{N} n_{ij}(h, E).$$

2.1.3. Functions of Classification Probabilities. There are several well-known functions of classification probabilities that will be referred to in this thesis, especially in Chapters 4 and 5 where the DEFT system is implemented and experimentally tested. Two of these, generalisation accuracy and the area under a Receiver-Operator Characteristic (ROC) curve, are used when empirically assessing the predictive performance of a hypothesis. The rest, discussed in Section 2.3.5 below, are used by rule learning algorithms to assess the quality of individual rules.

The most commonly used measure of predictive performance is the generalisation accuracy of a hypothesis. That is, the probability that the hypothesis will correctly predict the class label of a new example drawn from the same distribution used when inducing that hypothesis. The true generalisation accuracy of a hypothesis h is given by

$$\mathrm{acc}(h) = \Pr_{x \in X}(h(x) = t(x))$$

where t is the target concept and X is a complete set of examples for that concept. Given an incomplete set of examples E, the true positive and true negative rates in the classification probability matrix $\mathbf{p}_E(h)$ for a hypothesis can be used to estimate the true generalisation accuracy as follows

$$\mathrm{acc}_E(h) = p_{++} + p_{--}.$$

Generalisation accuracy places equal importance on misclassifying a positive example as it does a negative example. For some learning tasks this can mean high accuracies for a classifier can be reported even though it may be overgeneral and incorrectly classifying every (rarely occurring) negative example. This can misleadingly suggest that an induced classifier is very similar to the target concept when in fact it is not. This problem can be avoided in two ways. One is to ensure that the example set has an equal number of positive and negative examples even when the true distribution of class labels may not be equal. Alternatively, the area under a receiver-operator curve (ROC) can be used as a measure that is independent of class distribution.

A ROC curve provides an informative, visual summary of the generalisation performance of a learning algorithm by plotting points defined by the true positive and false positive rates of a classifier [Fawcett, 2004]. These rates are the values that appear in a CPM divided by their respective column totals. The true positive rate tpr and the false positive rate fpr for a hypothesis with

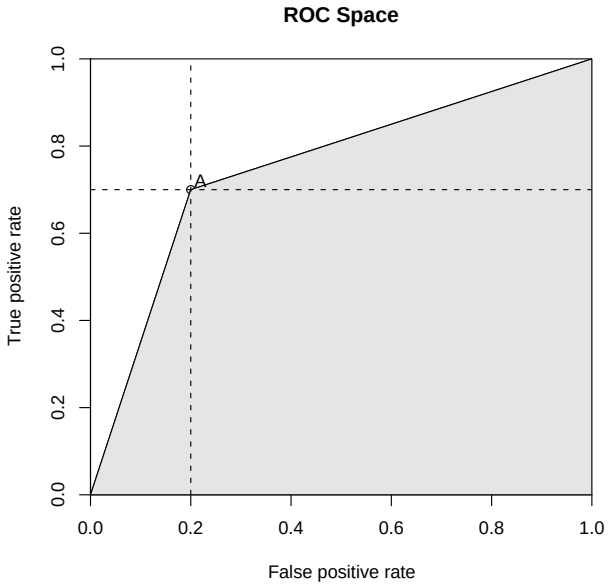


FIGURE 2.1. An example ROC plot. The point A represents a classifier with a true positive rate of 0.7 and a false positive rate of 0.2. The shaded area under the curve (AUC) is a measure of classifier A's performance.

CPM $\mathbf{p}(h)$ are given by

$$\begin{aligned} tpr &= \frac{p_{++}}{p_{++} + p_{-+}} \\ fpr &= \frac{p_{-+}}{p_{-+} + p_{--}}. \end{aligned}$$

The false negative rate fnr and true negative rate tnr are defined similarly. These rates are independent of class distribution as the denominator in the true positive (respectively, false negative) rate is the proportion of positive (respectively, negative) examples.

ROC curves are most useful for learning algorithms that produce classifiers which are controlled by a parameter, such as the cutoff probability for naïve Bayes classifiers or number of nearest neighbours for clustering algorithms. By varying the parameter, the resulting set of (tpr, fpr) points form a curve which shows learning performance under a range of conditions. When only one point (tpr, fpr) is available (as in the case with most rule learning algorithms) a curve can still be formed by connecting the point with (0,0) (the "always false" classifier) and (1,1) (the "always true" classifier). An example of such a curve for a classifier with a true positive rate of 0.7 and a false positive rate of 0.2 is shown in Figure 2.1.

As the true positive and false positive rates are independent of the class

distribution of the examples, so too is the *Area Under a ROC Curve* (AUC). The one-point ROC curve for a hypothesis h, as described above, has an area equal to the average of the true positive rate tpr and the true negative rate tnr, that is

$$\text{auc}(h) = \frac{tpr + tnr}{2}.$$

Statistically, the AUC is equivalent to the Wilcoxon test of ranks [Hanley and McNeil, 1982]. This measurement of hypothesis quality will be the preferred one in the experimental evaluation of DEFT in Chapter 5.

The belief that an estimate of the generalisation accuracy, AUC or any other measure of performance based on a training set of examples is a good approximation of the true values of those functions is known as the *inductive learning hypothesis* [Mitchell, 1997, §2.2.2]:

> Any hypothesis found to approximate the target function well over a sufficiently large set of training examples will also approximate the target function over the other unobserved examples.

This assumption forms the cornerstone of inductive learning and hinges on the condition that the training set be "sufficiently large". Section 2.4 below considers the problem of learning when this condition does not hold. This is done in the context of rule learning, where the hypotheses are expressed as sets of rules as introduced in the next section.

2.2. Relational Rule Learning

A well-studied class of concept learning algorithms represent theories as collections of statements of the form "*if* an instance satisfies these conditions *then* classify it like so"[2]. One advantage of this representation is that the induced theories are relatively easy for humans to comprehend. Rule learning algorithms have been successfully applied to problems in chemistry, medicine, linguistics, and engineering [Langley and Simon, 1995].[3] Rules can be particularly expressive when their conditions can include *relations* between objects that constitute instances. Molecules [Srinivasan et al., 1994], web pages [Cohen, 1998], sentences [Cussens, 1998], citation and movie databases [Jensen and Neville, 2002] and genomic pathways [King et al., 2004] are just a few examples of domains where examples and concepts are most naturally expressed using relations.

[2]One might say that these statements satisfy the "rule" rule: if *if* then *then*.

[3]Langley and Simon use the term "rule learning" to refer to systems which induce both sets of rules or decision trees. The examples listed here are for the applications they survey which used induced sets of rules.

2.2.1. Definitions. In general, a *rule* is an expression of the form "*if* condition A is true for an instance *then* condition B is also true". In the context of classification, they can be used quite naturally for prediction: "if a mushroom has spots then it is toxic", "if a book's genre is science-fiction then I will enjoy reading it". As this thesis is primarily concerned with binary classification tasks like these the focus of this section will be on rules that predict the membership of an instance in a concept. A rule r like this can be written as

$$c \rightarrow +$$

which is understood to mean "if condition c is true for an instance then that instance is labelled +".

As all the rules considered here have the same consequent, "assign the label +", the important part of a rule is its antecedent c. If c is true for an instance x we will write $x \in c$ and say that c *matches* (or *covers*) x. Exactly how this matching relationship is defined depends on the representation used for both instances and the rule conditions.

In one common representation instances have a number of primitive *attributes* that can be inspected, such as a book's genre or date of publication. Rule conditions then perform one or more tests comparing these attributes with a known *value* ("the book's genre is drama", "the book was published before 1984"). In these representations, a rule condition will match an instance if all the condition's tests are satisfied. The learning of rules using this representation is known as *attribute-value rule learning*. Systems such as CN2 [Clark and Niblett, 1989], RIPPER [Cohen, 1995a] and their derivatives fall into this category of learners.

Given a rule r and a matching relationship $\in$ for its condition c, a classifier h_r can be defined like so

$$h_r(x) = \begin{cases} + & \text{if } x \in c \\ - & \text{otherwise.} \end{cases}$$

Assigning a negative label to non-matching instances by default is known as *negation as failure*: any instance that fails to match a rule is assumed not to be part of the concept. Because of the close relationship between a rule $r = c \rightarrow +$, its condition c and its interpretation as a classifier h_r these terms will often be deliberately conflated and throughout the discussion there will be talk of instances being matched ($x \in r$) or classified ($r(x) = +$) by rules.

Classifiers can also be built for *sets of rules* by assigning an instance a positive label if any rule in the set matches it and assigning it a negative label

otherwise. If $R = \{r_1, \ldots, r_n\}$ is a set of rules then it can be interpreted[4] as a classifier h_R by defining

$$h_R(x) = \begin{cases} + & \text{if } x \in r_i \text{ for some } r_i \in R \\ - & \text{otherwise} \end{cases}.$$

The matching relationship between an instance and a rule is central to the definition of rule-based classifiers. As mentioned earlier, this relationship is determined by the representation of both the rules and instances. The majority of this dissertation is concerned with a particular type of representation and matching for rules that is based on first-order logic.

2.2.2. Relational Rules. The attributes in attribute-value learning can be thought of as functions from instances to values. That is, only a single value can be associated with each attribute for a given instance. A book can only have one genre and one date of publication, for example. A limitation of an attribute-value representation becomes apparent when trying to express a condition like "the book has an author who was born in Spain". The problem here is that a book may have more than one author and so the relationship between a book and its author cannot be expressed as a function.

One approach to this problem, known as "propositionalisation" [Krogel et al., 2003], is to create new attributes such as "nationality of the first author", "nationality of the second author", and so on. Another approach is to let the value of an attribute be a set ("nationalities of all authors of this book") and add set membership queries to the representation language [Cohen, 1996]. An earlier, more general approach, and the one considered here, is to do away with the requirement that attributes be functions and allow arbitrary relations, such as "author-of", between objects in rule conditions. *Relational rule learning* is the name given to the induction of these types of rules.

2.2.3. Syntax. Relations and relational rules are often expressed using first-order logic[5] or a subset or first-order logic that can be implemented in the programming language Prolog [Bratko, 1990, is a standard reference]. In Prolog, specific things such as a particular book, its author or the year it was written are identified using *constants* and written in lower-case (*e.g.*

[4]This is common way of interpreting a set of rules as a classifier but it is not the only way. Fürnkranz [1999, §2.3] surveys other methods for both binary and multi-class classification tasks.

[5]First-order logic is a large and technical field and it is beyond the scope of this thesis to discuss it in a detailed and formal manner. The summary here is based on the one found in Table 10.3 of [Mitchell, 1997]. For an in-depth discussion the reader is pointed to [Nienhuys-Cheng and de Wolf, 1997].

`don_quixote`, `cervantes`, `1650`). *Variables* are used to stand for non-specific things such as "some book" or "some author" and customarily denoted with an upper-case first letter (*e.g.* `Book`, `Author`). Collectively, constants and variables are known as *terms*. *Functions* of terms, such as the successor of a number (*e.g.* `succ(1650)`) are also terms. Any term can be used to replace a variable in a logical expression. `Book` might be replaced by `don_quixote`, for example. The mapping which states which variables will be replaced by which terms is known as a *substitution*. Relations, like authorship, that exist between terms are expressed through symbols called *predicates*. The name of a predicate sometimes appears with a number after it (*e.g.* `author/2`) denoting its *arity*, the number of terms the predicate relates. Together, the constants, functions and predicates form what is called an *alphabet* and defines the set of symbols which can be used to build logical expressions.

When a predicate is instantiated with a particular set of terms it is called a *literal*. The literal `author(don_quixote,cervantes)`, for example, can be used to express that Cervantes is the author of *Don Quixote*. The *negation* of a literal is also a literal and is denoted by placing the negation symbol "¬" in front of a literal, for example ¬`author(don_quixote,dan_brown)`. This can be used to express the denial of a particular relationship. Literals that have no negation sign are called *positive literals* to distinguish them from *negative literals*. When literals do not contain any variables, as in the examples above, they are called *ground* literals. Positive ground literals are known as *atoms* and are typically used to represent instances when learning relational rules.

Rule conditions are expressed in first-order logic using disjunctions of literals, known as *clauses*. A clause with p positive literals H_i and n negative literals $\neg B_i$ is denoted

$$H_1 \vee \ldots \vee H_p \vee \neg B_1 \vee \ldots \vee \neg B_n.$$

Since the rules of logic state $\neg(X \vee Y) = \neg X \wedge \neg Y$ and define $(X \leftarrow Y) = X \vee \neg Y$ a clause is commonly written as an implication

$$H_1 \vee \ldots \vee H_p \leftarrow (B_1 \wedge \ldots \wedge B_n)$$

and the literals H_i and B_i are respectively known as *head* and *body* literals. Clauses with at most one head literal are called *Horn clauses* and are the most common form of clauses used in rule induction. For convenience, Horn clauses will often be written using the same notation as Prolog programs, that is H `:-` $B_1, \ldots, B_n$. The set of all clauses that can be built up from a given alphabet is called a *language*.

Once a language is specified by choosing the terms and predicates, rule

conditions and instances for relational rule learning can be represented using clauses from that language. In order to classify instances according to rules a matching relation $\in$ between them needs to be specified.

2.2.4. Semantics. Informally, we would like the matching relation to be defined so that the instance `enjoy(don_quixote)` will match the condition

```
enjoy(B):-author(B,A),born(A,spain)
```

if the variable `B` can be substituted with `don_quixote` and there is some term that can be substituted for variable `A` (*e.g.* `cervantes`) so that the body literals in the clause are true.

The truth or falsity of literals and clauses is specified through a *Herbrand interpretation* which, for the purposes of this thesis, can be thought of as the set of ground atoms that are to be considered to represent all the true facts. The truth value for more complex expressions, such as clauses, are derived from the ground facts using the standard truth tables for logical connectives such as $\vee$ and $\wedge$. Details can be found in [Nienhuys-Cheng and de Wolf, 1997, §3.4].

If an expression F, such as a literal or clause, is true under the interpretation I the interpretation is said to be a *model* for the expression. A *contradiction*, such as the *empty clause* $\square$, is an expression that has no models. If every model for F is also a model for G - that is, G is true whenever F is - we say that F *entails* G and write $F \models G$. Consequently, if $F \models \square$ then F has no models and is also a contradiction. Expressions with at least one model cannot entail a contradiction and are said to be *satisfiable*. This is written $F \not\models \square$.

The relations referred to by the predicates `author/2` and `born/2` in the example above are called *background knowledge*. Relations which are part of the background knowledge are defined by a set of clauses and are used when determining whether a condition matches an instance as follows. If e is an atom (*i.e.* positive ground literal), C a clause and B the background knowledge then we define C to match e (with respect to the background B) if and only if B and C together entail e. That is,

$$e \in_B C \text{ iff } B \wedge C \models e.$$

This definition of the matching relation is comparable to what De Raedt [1997, §3.2] calls the *intensional inductive logic programming* framework for learning. As stated earlier, relational rule learning involves learning sets of rules with clauses as conditions. The clausal matching relation just described is what

allows these conditions to be tested against examples and therefore defines the classifying behaviour of sets of clausal rules called *relational classifiers*.

Learning relational classifiers from examples is what is known as a *supervised* learning problem, in contrast to *unsupervised* rule learning where class labels are not available. Unsupervised rule learning problems ask that a learner produce a set of rules that describe frequent patterns within a training dataset. Problems of this type are commonly called *association rule mining* [Zhang and Zhang, 2002] or *rule discovery* [De Raedt and Dehaspe, 1997, Flach and Lachiche, 2001]. Although the work presented in this dissertation may be applicable to rule learning in the unsupervised setting it is not the main focus at this time and so its discussion is deferred to Chapter 6. Relational classifiers are also a less general type of classifier than the sets of clauses, or logic programs, which are used in the general inductive logic programming framework. In general ILP a hypothesis H covers an example e if it, along with the background knowledge, entails the example. That is, if $B \wedge H \models e$ holds. This more general condition allows for *program synthesis* [Flener and Yilmaz, 1999]: the induction of recursively defined relations such as list membership. ILP systems such as MIS [Shapiro, 1983] and HYPER [Bratko, 1999] solve program synthesis tasks by systematically generating and testing entire logic programs against training examples. In theory, this program synthesis is also able to induce relational classifiers. However a simpler strategy called sequential covering takes advantage of the fact that in relational classification tasks no interplay between clauses is required to determine whether an example is matched. As the focus of this thesis is on classification rather than program synthesis its focus is further narrowed to algorithms which adopt this strategy to learn sets of rules as classifiers.

2.3. Sequential Covering Algorithms

Algorithms that learn concepts as sets of rules have been studied for over forty years and the resulting literature on this topic is both broad and deep. Any attempt at surveying its entirety in this dissertation would be misguided. Instead, the purpose of this section is to focus on a widely used family of rule learning algorithms known as *sequential covering* algorithms whose common ancestor is Michalski's AQ algorithm [Michalski, 1969]. These algorithms all share the same general strategy for building a classifier from a set of training examples: while there are uncovered positive examples, find a single rule that covers at least some of these and make it part of the classifier.

One such sequential covering algorithm called PROGOL, originally proposed and implemented by Muggleton [1995], is examined in some detail in this

section. This is done for two reasons. First, the PROGOL algorithm is an efficient and widely used instance of this class of rule learners, and secondly this particular algorithm forms the core of the implementation of the inductive transfer system DEFT introduced in Chapter 4. The particular implementation of PROGOL that is used throughout this thesis is the one found in the ILP system ALEPH [Srinivasan, 2001].

The remainder of this section frames the rule searches performed by ALEPH and other algorithms as an optimisation problem in which the function to optimise is one that assigns a measure of quality to each rule. The biases of rule searching algorithms can be understood as constraints on the candidate solutions, the order in which candidates are evaluated and the evaluation function to optimise. Of these, the evaluation function is considered the most important for the purposes of this thesis and is examined in some detail.

2.3.1. Separate and Conquer Rule Learning. A generic sequential covering algorithm for inducing sets of rules from examples is shown in Algorithm 1. This strategy sometimes goes by the name "separate and conquer" as the inner loop of this procedure repeatedly calls FINDRULE to cover part of the example set and then separates out those examples that were covered from those that were not. By repeating this loop until all positive examples are covered a set of rules is constructed which explains all the positive examples.

Algorithm 1 SEQCOVER - Sequential Covering Algorithm for Learning Sets of Rules

```
1: H = {}
2: while E+ is not empty do
3:     Call FINDRULE(E) to get rule r
4:     Remove examples in E+ covered by r
5:     Add r to H
6: end while
```

There are many slight variations on the basic covering algorithm. The restriction that all the positive examples be covered can be weakened to handle erroneously labelled examples. A post-pruning of the rules and their conditions can be performed based on a "hold-out" set of examples separate from those used to find the rules. While these modifications have some effect on the algorithm and the quality of the rules it induces, its real nature is determined by the workings of the FINDRULE procedure.

The FINDRULE procedure takes a set of examples E as input and returns a single rule r that explains at least some of those examples. The dozens of methods used to implement this procedure for covering algorithms are comprehensively surveyed by Fürnkranz [1999]. To avoid going into too much detail it

is enough to consider the implementations of FindRule as searches of a rule space. All of these different types of search have the following general structure. First, a language for the rules is chosen and the search is initialised with some starting rule. Then, the rule space is explored by repeatedly evaluating and refining rules and, eventually, the search terminates and the best rule is returned.

A particular instance of the FindRule procedure can be characterised by describing how it implements the language, search, and evaluation aspects when solving a learning problem. Section 2.5 below reviews several relational rule learning systems which use inductive transfer to modify one or more of these three aspects. The majority of those systems are built upon variants of the covering algorithm for relational learning. The inductive transfer system introduced in this thesis also has a covering algorithm as its foundation. This algorithm, called Progol, is implemented as part of the ILP system Aleph which is now described.

2.3.2. Aleph and Progol. The Aleph system is the Swiss-army knife of first-order rule learners.[6] Its design and implementation in Prolog is very modular and incorporates ideas and algorithms from a variety of other systems. This generality makes it a valuable "prototype for exploring ideas" [Srinivasan, 2001] in inductive logic programming. Only a small part of the Aleph system was modified and used for the implementation and exploration of inductive transfer for rule learning in this dissertation. Specifically, its slight variation on the original inverse entailment algorithm in Progol [Muggleton, 1995] will act as the main learning algorithm within Deft. Hereafter, when the term Aleph is used it is meant in this restricted sense.

Given a collection of positive and negative ground facts Aleph induces a theory as a set of Horn clauses using the sequential covering strategy described above. Each clause in the theory is constructed from predicates defined in the task's background knowledge. The way in which literals can be combined in the body of a clause is determined by mode and type constraints as well as an upper limit on the variable depth within a clause (set using the Aleph parameter `i`). These two static constraints are used to define what Muggleton [1995, Definition 22] calls the "depth-bounded mode language" for the task. Clauses in the depth-bounded mode language are called the *legal clauses* for the learning task at hand.

Even for relatively simple learning tasks the total number of legal clauses for a problem can be potentially infinite which make complete searches for an

[6] Aleph is an acronym for **A** **L**earning **E**ngine for **P**roposing **H**ypotheses

Algorithm 2 The FINDRULE procedure as it is implemented in ALEPH: a general-to-specific, breadth-first, branch-and-bound optimisation of evaluation score over a refinement lattice of clauses.

```
1: procedure FINDRULE(E)
2:     Select (x, +) ∈ E+ and compute ⊥ = BOTTOM(x)
3:     Initialise best = x and set best_score = SCORE(best, E)
4:     Initialise Q = a FIFO queue containing only the empty rule □
5:     while Q is not empty do
6:         Remove head from the front of Q
7:         for all rules new ∈ REFINE(head, ⊥) do
8:             if BOUND(new, E) ≥ best_score then
9:                 Add new to the back of Q
10:                if ACCEPT(new, E) and SCORE(new, E) > best_score then
11:                    Set best = new and best_score = SCORE(new, E)
12:                end if
13:            end if
14:        end for
15:    end while
16:    return best
17: end procedure
```

optimal clause infeasible. To avoid this impracticality, ALEPH's FINDRULE procedure (shown in Algorithm 2) implements a general-to-specific, breadth-first search which uses several techniques to carefully limit its exploration of the set of legal clauses.

The first and most important of these techniques is the construction of a *bottom clause* by the procedure BOTTOM in line 2. The construction process uses a technique called "saturation", or "elaboration" [Sammut and Banerji, 1986], in which an instance is generalised and conjoined with any related facts that are derivable from the background knowledge. The precise details of this procedure are not relevant to this discussion and the reader is referred to [Muggleton, 1995, §8.1] for more information. All the detail required here is that this procedure will, when passed an instance x, construct the most specific legal clause, denoted $\bot$, that covers that instance. The bottom clause generally consists of tens or hundreds of literals. Since the bottom clause is most specific, any other legal clause that subsumes $\bot$ must necessarily cover the instance x. By only considering legal clauses which subsume the bottom clause, the search is constrained to finding a legal clause which must cover x, greatly reducing the complexity of the search.

In its main loop, the FINDRULE procedure repeatedly removes the foremost clause *head* from its search queue for refinement. The REFINE procedure ensures that only legal specialisations of the clause are constructed and that each

Algorithm 3 The SCORE and BOUND procedures used within the search implemented by FINDRULE

procedure SCORE(r, E)
    $\mathbf{n}$ = the contingency table $\mathbf{n}_E(r)$
    return $f(\mathbf{n}; r)$
end procedure

▷

procedure BOUND(r, E)
    $\mathbf{n}$ = the contingency table $\mathbf{n}_E(r)$
    $\mathbf{n}_{\max} = \begin{bmatrix} n_{++} & 0 \\ n_{-+} & n_{--} + n_{+-} \end{bmatrix}$
    return $f(\mathbf{n}_{\max}; r)$
end procedure

refinement subsumes the bottom clause $\perp$. The most general specialisations of *head* satisfying both these conditions are created efficiently in one of three ways: by adding a single literal from $\perp$ to *head*, by substituting a variable in *head* with a constant from $\perp$, or by binding two variables from *head* with each other. The specialisations are further constrained by limiting the total number of literals, or length, of the clauses returned by REFINE. The maximum allowed clause length is controlled by the ALEPH parameter `clauselength`.

After their construction, each refinement of *head* is evaluated and potentially pruned from the search using an A^*-like admissible pruning heuristic. The evaluation and pruning are handled by the SCORE and BOUND procedures respectively. These are shown in Algorithm 3 and, although relatively simple, some space is devoted to these routines here as they are the only ones that are modified for use in the inductive transfer system DEFT introduced in Chapter 4.

The SCORE and BOUND routines do essentially the same thing. They construct a contingency table for the rule r based on the training data E and then pass both the rule and the table to the *rule evaluation function* f which is discussed further below. The value returned by f is an estimate of the quality of the rule r in the case of SCORE, and an upper bound on the quality of any refinement of r when returned by BOUND. The upper bound is obtained by applying the evaluation function to a modified version of the contingency table in which all of the misclassified negative examples (as counted by n_{-+}) are assumed to be corrected classified and hence added to n_{--}.

The upper bound used here can be justified as follows. Since any refinement of the rule r is necessarily a specialisation it cannot cover any more examples than those covered by r. That is, the counts in the top row of the contingency table for r can only decrease, with the difference being added to the corre-

sponding term in the bottom row. The best refinement of r will therefore be one that decreases the misclassification counts n_{-+} without covering any fewer correctly classified examples. Such a refinement will have the best possible evaluation score according to f provided that f does not penalise a reduction in the false positive rate which does not affect the true positive rate. A rule with a bound that is not an improvement on the best score seen by a search is pruned since such a rule cannot be refined to produce a higher scoring rule.

Rules can only be considered for inclusion in a theory if they are the best ones seen during a search and also meet some user-defined "acceptability" conditions. These are implemented through the ACCEPT procedure which is called in line 10 of the FINDRULE algorithm. In ALEPH, this procedure is used to ensure that the returned rules do not cover too many negative examples. Exactly how many negative examples can be covered is controlled by a `minacc` setting. When set to the value $K \in [0, 1]$, the ACCEPT procedure will only allow a rule r into a theory if its "rule accuracy" (also known as precision), $\frac{n_{++}}{n_{++}+n_{+-}}$, is greater than or equal to K.

The process used by the FINDRULE procedure within ALEPH and other covering algorithms for rule learning can be viewed abstractly as a way of systematically generating rules from a predefined space and then testing them against the training data according to some measure of quality. Whichever rule returns the highest score within the space searched is the one selected and added to the theory. As the measure of rule quality is determined from a (possibly small) sample of classified instances this estimated quality can differ from the same measure applied to the entire instance space. In this sense, rule learning algorithms can be seen as optimising an estimate of true rule quality over a space of candidate rules.

2.3.3. Rule Finding as Estimate Optimisation. Finding rules that explain a set of training examples is often framed as a particular type of search problem called a *optimisation problem* in which a function must be maximised over a set of candidates. Formally, an optimisation problem consists of a discrete *candidate space* C and an *objective function* $f : C \to \mathbb{R}$. The *solution set* S_f to such a problem is the set of all candidates that maximise the objective function. If $f^* = \max_{c \in C} f(c)$ is the *optimal value* of the objective function then

$$S_f = \{c \in C \,:\, f(c) = f^*\}.$$

Any element of the solution set is known as a *solution*. In rule learning the candidate space is the set of rules defined by the language used for the learning

task and the objective function is the evaluation function used to score the rules.

Traditional optimisation problems, such as finding shortest paths and job scheduling, use an objective function which is the true measure of the quality of a solution. The length of an optimal path or schedule is exactly the number of steps or time that solution will require. There are also situations when the values given to candidates are only an estimate of the actual values required. Imagine having to choose the person with the highest reach from a group at a dinner party in order to fetch something. If everyone is sitting down and the choice is made based on sitting height then a person with long legs may be overlooked in favour of someone with a long torso. This may lead to a sub-optimal choice for someone to fetch the wine from the top shelf. This is analogous to the situation in rule learning: the evaluation function is only an *estimate* of the actual value of a rule based on the available training data. The value assigned to a rule on a training set may differ greatly from its value on a large test set.

An optimisation problem that uses an *estimated objective function*, $\bar{f}$ (*e.g.* a person's sitting height or training set evaluation) in place of the actual evaluation, f (the person's reach or test set evaluation), to score candidates constitutes an *estimate optimisation problem.* Provided the solution set $S_{\bar{f}}$ for the estimate optimisation problem has some candidates in common with the solution set S_f for the actual optimisation then it is possible that an optimisation using the estimate will return an actual solution. However, if there are many candidates in $S_{\bar{f}}$ that are not in S_f there is a good chance that an estimated solution will not be an actual solution. It is also worth noting that an algorithm using a poor $\bar{f}$ as an evaluation estimate may still select candidates with good f values if its search of the candidate space is incomplete. This is because candidates with optimal $\bar{f}$ values may not be tested and those with sub-optimal $\bar{f}$ values may be in S_f.

A visualisation of a rule search as an estimate optimisation is shown in Figure 2.3. Ideally, a search of the rule space would return a rule from the set labelled "Actual Best". This set contains the rules that maximise the evaluation function f as it is evaluated over some large and representative test set of examples. The set labelled "Estimated Best" corresponds to the solution set for f_E, the evaluation function computed with respect to the training examples E. A typical algorithm for finding good rules within the space would repeatedly refine a starting rule (as shown by the arrows) evaluating the resulting rules using f_E.

Agreement between an evaluation function and its estimate depends on

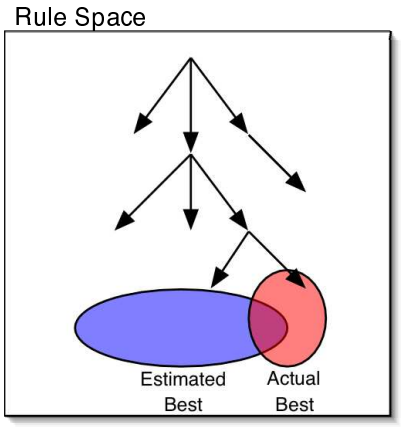


FIGURE 2.2. Rule search as an estimate optimisation problem.

whether the training examples used to estimate the evaluation function is representative of the entire instance space. When a large number of examples of the target concept are made available to the learner, the estimated classification probabilities for candidate rules will be close to their true values. When training data is scarce these estimates may be poor and the resulting scores given to rules by $\bar{f}$ can rank rules very differently to f. The effect of limited data on rule evaluation is taken up again in Section 2.4 below.

2.3.4. Language, Search and Evaluation Bias. The *bias* of a learning algorithm is any extra-evidential decision-making procedure that determines which hypothesis is induced by a learner from a given set of training examples. Formally, Mitchell [1997, §2.7.3] defines the bias of a learning algorithm to be "the set of additional assumptions ... sufficient to justify its inductive inferences as deductive inferences". The framing of rule search as an estimate optimisation problem provides a way of decomposing the additional assumptions required for the SEQCOVER algorithm into three factors: *language*, *search* and *evaluation* bias as shown in Figure 2.3. The language bias is defined by the choice of candidate space while the evaluation bias corresponds to a selection of the objective function f. The search bias covers all the other aspects that are used to order and dynamically prune an exploration of the candidate space. This categorisation is similar to the one given by Fürnkranz [1999] in his review of separate-and-conquer rule learning algorithms with a couple of exceptions. Firstly, what he denotes as "search bias" is a combination of what is here called evaluation bias (which he calls "search heuristics") and search bias. Evaluation bias is considered a separate type of bias here as it is central to much of the theory developed in the next chapter. Secondly, Fürnkranz identifies "overfitting avoidance bias" as a separate category of bias that is primarily used to

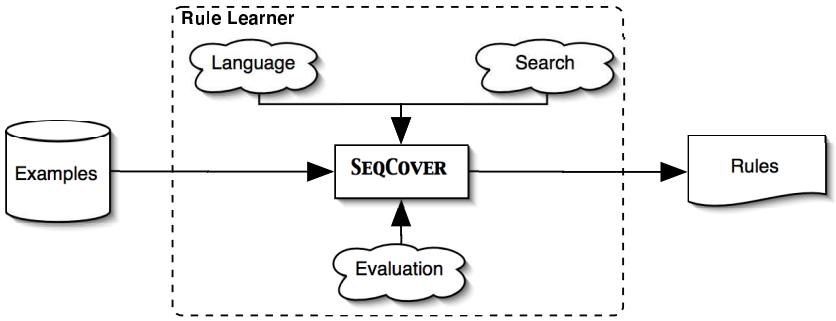


FIGURE 2.3. Rule learning as a search controlled by language, search and evaluation biases.

manage noisy data. While this is an important type of bias it is not directly relevant to the work here. Nédellec et al. [1996] also offer a categorisation of bias similar to Fürnkranz, organising biases into one of "language", "search" and "validation". What is termed evaluation bias here would be a type of search bias that they call an "intermediate validation criteria". Once again, evaluation bias is put into its own category here due to its importance in this work.

Inductive logic programming has been a popular formalism for rule learning due to the amount of control over a learner's language bias that is available through the specification of the background predicates for a learning task. Most ILP systems, including ALEPH, allow background predicates to be defined using arbitrary Prolog programs. This expressiveness provides a great deal of flexibility when it comes to defining a space of candidate rules. Complex predicates implementing ranges, disjuncts or other transformations can be defined as logic programs built from the predicates originally available with the training data. Predicates implementing more sophisticated techniques such as predicate invention [Khan et al., 1998] and numerical regression [Srinivasan and Camacho, 1996, 1999] are also possible. Further structure can be given to the set of rule candidates through the imposition of restrictions on the way literals can be joined. Mode and type restrictions on predicates [Shapiro, 1983] and restrictions on variable depth and clause length are used by many ILP systems including PROGOL [Muggleton, 1995] to prohibit the generation of overly complex or nonsensical clauses. Many other types of constraints for allowable clauses are also available in ALEPH including task specific pruning rules and integrity constraints which further constrain the rules allowed in induced theories. Explicitly defining rule spaces through rule models [Morik, 1993, Kietz and Wrobel, 1992], generative grammars [Cohen, 1994], and clause set or templates [Dehaspe and De Raedt, 1996, Nédellec et al., 1996] has also proved

useful for some problems. While these are all powerful techniques they do require their users to have well-defined ideas regarding the form of the clauses expected to be in the target theories.

The search bias of a learning algorithm is determined by the search strategy used to explore the candidate space defined by the language bias. These strategies fall into three main camps: the top-down or general-to-specific search, the bottom-up or specific-to-general search, and stochastic search techniques. The first of these categories is exemplified by FOIL [Quinlan, 1990], the second by MARVIN [Sammut, 1981] (and later GOLEM [Muggleton and Feng, 1990]) and the third by the probabilistic search methods explored in [Srinivasan, 1999]. Top-down search strategies tend to induce rules with fewer literals than bottom-up systems as the termination condition for the rule search is met well before the entire space is explored. For efficiency reasons, many implementations of rule search use incomplete search techniques such as greedy or beam search. In these cases the order in which rules are added to or removed from the beam will affect the induced theory. As well as the coarse control provided by the selection of a search direction and parameters such as beam size, finer-grained control can be obtained through the use of relational clichés [Silverstein and Pazzani, 1991] and rule deferral [Cohen, 1993] to determine the order in which rules are generated.

2.3.5. Rule Evaluation. According to Fürnkranz [1999, §4.3] the "most influential bias" is the "search heuristic" (here called evaluation bias) as it "estimates the quality of rules found in the search space and ideally guides the search algorithms into the right regions of the hypothesis space". There is no single relationship that best defines this quality in terms of how many positive and negative examples a rule should cover so that the separate and conquer algorithm returns the best possible set of rules. Simply returning a rule that covers no negative examples and the most positives may not be appropriate for all target theories, especially when it is likely that a perfectly discriminating theory cannot be expressed with rules from the candidate set. When predicting the presence of a disease, for example, it may be crucial that future instances of the disease be correctly predicted and occasionally falsely predicting disease when it is not present is acceptable. In this case rules that cover many positive examples and perhaps a few negatives would be more valuable than rules that cover fewer negative examples at the expense of covering fewer positive examples. The situation is reversed when predicting reading recommendations. Predicting someone will like a book when, actually, they will not is far worse than making fewer positive recommendations. For

these reasons the specification of what constitutes a good quality rule is a type of bias as the most appropriate rule evaluation function depends on the learning task[7]. Some space is dedicated to examining evaluation bias here as it is this aspect of a rule learning algorithms that will be modified by the inductive transfer method described in the next chapter.

For the purposes of this thesis there are two main types of rule evaluation functions: purity-based functions and description-based functions.[8] An evaluation function is called *purity-based* if it can be written as a function of the values in rule contingency tables. That is, a purity-based assessment of a rule only depends on the number of correctly and incorrectly classified examples rather than any properties of specific examples or properties of the rule itself. Put formally, the value of purity-based evaluation function $f_E(r)$ for a given rule r and example set E can be written as a function g_p of the rule's contingency table on E

$$f_E(r) = g_p(\mathbf{n})$$

where $\mathbf{n} = \mathbf{n}_E(r)$ is the contingency table for r. Evaluation functions of this type have been the subject of some analysis recently [Vilalta and Oblinger, 2000, Fürnkranz and Flach, 2003]. The conclusions drawn from that research have shown that, for the purposes of ranking rules by their quality, there are essentially two prototypical evaluation functions: *precision* and *cost-weighted difference*. The precision of a rule r is the proportion of positive examples in a training set that were correctly classified as positive by r. That is,

$$prec(r) = \frac{n_{++}}{n_{++} + n_{+-}}.$$

As the name suggests, the cost-weighted difference function has a parameter called cost $c \in [0, 1]$ that trades off the relative importance of covering positive examples and covering negative examples. That is,

$$cwd_c(r) = cn_{++} - (1 - c)n_{+-}.$$

The default evaluation function used by ALEPH is called *coverage* and is defined to be $cov(r) = n_{++} - n_{+-}$. This evaluation function ranks rules the same way as *cwd* when $c = \frac{1}{2}$. Setting the cost in this way means incorrectly classifying a positive or negative example incurs the same penalty.

[7]This asymmetry in the importance of errors is sometimes handled by assigning costs to each type of misclassification [Elkan, 2001]. When a learner attempts to minimise these costs during learning they act as an evaluation bias.

[8]Fürnkranz [1999] makes a similar distinction with the categories "basic heuristics" and "complexity estimates" roughly corresponding to the two here.

Description-based evaluation functions allow the assessment of rule quality to depend on rule features such as the number or type of conditions in a rule. This allows the evaluation function to explicitly encode a preference for shorter rules or rules with particular conditions or structure. Formally, if g_d is some function that ascribes some measure of complexity to rules then a description-based evaluation function has the form

$$f_E(r) = F(g_p(\mathbf{n}), g_d(r))$$

where r, E, g_p and $\mathbf{n}$ are as described above. The function F combines the purity and complexity of a rule so that one may be traded off against the other. As an example, the default evaluation function used by the ILP system PROGOL [Muggleton, 1995] is a description-based evaluation called *compression*. This is defined as $comp(r) = cov(r) - len(r)$ where $cov(r)$ is the purity-based coverage of r just discussed and $len(r)$ is the number of literals in r. This function trades off the coverage of a rule and its complexity, preferring shorter and possibly less accurate rules over longer but more accurate ones. Here, $g_p = cov$, $g_d = len$ and $F(a, b) = a - b$. This and other description-based evaluation functions are generally motivated by the theory of *minimum description length* (MDL) [Rissanen, 1978, Pfahringer, 1995] or *minimum message length* (MML) [Wallace and Georgeff, 1983] to decide how to trade-off complexity and accuracy.[9] In more complex evaluation functions, the exact nature of this trade-off is controlled by a human expert through the specification of the encoding to be used for defining a rule's description length.

Defining a description-based evaluation function can be difficult in general and requires that the expert make many decisions regarding the relative importance of rule purity, the presence of certain literals in the rule, rule length and so on. In Chapter 3 a method is given for converting an arbitrary purity-based function g_p into a description-based function using information gleaned from support tasks related to the primary one being solved. This means the many decisions regarding the relative importance of a rule's description can be replaced by a choice of support task. Because existing description-based evaluation functions are also functions of purity-based functions these can also be modified using the same technique, making it a broadly applicable way to modify evaluation bias. However, at least three types of evaluation heuristic do not fit into the above categories of purity- and description-based evaluation. These are gain heuristics, proof-complexity heuristics and process-oriented heuristics. They are briefly discussed below in order to delimit the scope of this thesis.

[9] "A little inaccuracy sometimes saves tons of explanation" Saki [1968].

Gain heuristics such as information gain [Quinlan, 1990] measure the quality of a rule with respect to the rule it was refined from, meaning rule quality is not strictly a function of its purity and complexity. *Process-oriented heuristics* [Domingos, 1998] use information about the entire history of a rule search to reduce the probability of choosing a candidate that might overfit the training data. This is done by quantifying the probability that the best rule seen during a search fits the data purely by chance and factoring it into the evaluation function. Because of this dependence on search history, process-oriented heuristics are not purity- or description-based in the sense defined here. Heuristics that use *proof-complexity* [Muggleton et al., 1992] as a quality measure take into account the number of resolution steps it takes to determine whether or not a rule covers an example. An examination of whether the present work can be extended to other types of evaluation heuristic is left for future research.

Purity-based and description-based evaluation functions both depend on the contingency table of a rule in order to make their assessments. The relationship between the actual and estimated quality of a rule therefore depends on the reliability of the values in the contingency tables. The reliability of these values when data is limited is the topic of the next section.

2.4. Learning from Limited Data

Obtaining correctly classified examples for a concept learning problem is a crucial but sometimes costly task, especially if experts are required to classify the examples. In some chemical domains for example, datasets with a limited number of training examples are "not unusual" since "data are often sparse, and bioassays expensive" [Srinivasan and King, 1999]. In this section the difficulties of learning from limited training examples are examined and a brief review of approaches to this problem is presented. These approaches invariably involve some selection of strong and correct biases. As stated in the previous section these biases for rule learning algorithms fall into three categories: language, search and evaluation. Much work has been done on mechanisms to express these biases, especially language biases, in rule learning. However these can be difficult for a domain expert to specify without some knowledge of the workings of the learning algorithm. Systems that perform *inductive transfer* allow a bias to be expressed by a domain expert by supplying the system with examples of related tasks. It is then the job of the inductive transfer system to find a good bias for the limited data task knowing that the domain expert believes a bias for the related tasks should also be suitable for the main task. Successful inductive transfer means fewer examples need to be collected and classified for new tasks to achieve good inductive performance. Since the costs

for collecting and classifying the related task examples are not incurred again when they are reused, inductive transfer can be seen as a way of amortising the cost of learning over a number of different tasks.

2.4.1. How Little is Not Enough? In an idealised machine learning problem a learner is expected to induce a hypothesis that is consistent with the training examples, that is, one which classifies them as would the target concept. Each training example can be thought of as a constraint on the set of consistent hypotheses. The subset of the hypothesis space consistent with the training data is known as the *version space* [Mitchell, 1977]. The main problem when learning from small amounts of data is that the version space is under-constrained and can therefore contain very many candidate hypotheses. In this situation a learner must rely heavily on its inductive bias to decide between them [Mitchell, 1980].

As mentioned in Section 2.1.3 above, the belief that the estimate of the misclassification error from the training set is a good approximation of the true error of a hypothesis is known as the inductive learning hypothesis. Central to this hypothesis is the assumption that "a sufficiently large set of training examples" is available to the learner. Research in computational learning theory has quantified the term "sufficiently large" for broad classes of learning problems [Kearns and Vazirani, 1994]. Such results are framed within the PAC (Probably Approximately Correct) setting [Valiant, 1984] and provide lower bounds on the minimum number of examples required to ensure that a hypothesis with low training set error has, with high probability, a low true error. For rule learning, Džeroski et al. [1992] and later Cohen [1995b] provided PAC lower bounds on the number of examples required to learn theories using representations that are restricted subsets of full Horn clause logic.

While the PAC bounds provide a theoretical answer to the question of "how few is limited?" the notion is intended more broadly and practically here with "limited data" referring to any task for which learning with more data would lead to significantly better theories. The imprecision of this definition is intentional and is similar in spirit to the one Good [1965] used to talk of estimation from "effectively small samples". By this he meant "the sample is small for the purpose of estimating some probability, though the sample might be absolutely large". Similarly, "limited data" will be taken to mean that theories of higher quality can be induced from potentially larger datasets even if the number of available training examples is absolutely large. This means that a task is deemed to have limited training examples relative to a particular learning algorithm and its biases. Techniques for learning from limited data all

attempt to modify a learner's bias in some way in order to reduce the number of examples required to induce a good hypothesis.

2.4.2. Contingency Classes. A simple counting argument can be used to show why learning rules from a limited amount of training data is inherently difficult without a good bias. Suppose a training set E for a learning task has $|E^+|$ positive and $|E^-|$ negative examples. A contingency table $\mathbf{n}$ for this example set must satisfy

$$\begin{aligned} |E^+| &= n_{++} + n_{-+} \\ |E^-| &= n_{+-} + n_{--} \end{aligned}$$

regardless of which rule is being tested on the examples. As each entry in a contingency table must be a non-negative integer there are only $K = (|E^+| + 1)(|E^-| + 1)$ possible contingency tables for the given training set. When the overall number of examples in a training set is low there will only be a small number of unique contingency tables. For instance, in the case of five positive and five negative examples there are only 36 unique contingency tables. When the candidate space R has more than K rules there must necessarily be at least one contingency table shared by two or more rules. A given set of training examples E therefore partitions the rule space R into K *contingency classes*, one for each possible contingency table $\mathbf{n}$:

$$\mathcal{C}_{\mathbf{n}}(E) = \{r \in R : \mathbf{n}_E(r) = \mathbf{n}\}.$$

Rules that have the same contingency table will, by definition, be assigned the same quality assessment by any purity-based evaluation function f_E. In fact, many purity-based evaluation functions (such as precision) will map different contingency classes, and all the rules within them, to the same evaluation score.

The true evaluation scores, as assigned by f, vary over the rules within each contingency class. In this case, the search and language biases of the learning algorithm must be relied upon to determine which rules are to be selected. Alternatively, description-based evaluation functions can also exert a preference between rules within a contingency class. If, like the *comp* function, the evaluation function penalises overly long rules, the shortest rules within a class will be selected. If these biases select for rules with high true evaluation score then the learner can be expected to return good theories.

Although the selection of a correct bias for a task is critical for ensuring good theories are learnt when examples are limited, it is a potentially difficult and time-consuming process. The remainder of this section looks at ways in

which a domain expert can specify a bias at a high-level. By giving a learning system extra information in the form of a task that is believed to have similar bias requirements to the one that is to be solved, procedures for automatically discovering such a bias can be used and the extracted bias transferred to the limited data task.

2.4.3. Meta-Learning: Beyond Single Task Learning. Much of concept learning research within machine learning has been focused on developing efficient algorithms that can induce accurate theories from a single batch of training data and characterising the classes of concepts for which such algorithms exist. When training data is limited there is simply not enough information available for a learner to be expected to induce high-quality theories without strong hints in the form of inductive biases. The selection of an appropriate bias is both crucial and necessary within a single task learning framework. Careful selection is crucial because poor choices can make learning difficult, if not impossible, and necessary since it has been shown that there is a "conservation law for generalisation performance" [Schaffer, 1994, Wolpert, 1996]. That is, that there is no single learner or bias that will always return the most accurate hypotheses across all possible learning tasks.[10]

The selection of a bias by an expert is made based on her experience with various algorithms and their performance on a variety of tasks. To reduce the amount of routine decision-making required of the expert, it would be useful to automate part or all of this process. This calls for systems that are able to modify or select their own biases and requires the study of concept learning to move away from learning tasks in isolation. This realisation and shift in research is currently over a decade old and systems that subscribe to this approach and are able to "increase in efficiency through experience" do so under the banner of "meta-learning" [Vilalta and Drissi, 2002, provide a survey]. Several overviews of this now large field are provided in books [Thrun and Pratt, 1997a], workshop proceedings [Caruana et al., 1995], journal special issues [Giraud-Carrier et al., 2004] and literature reviews from doctoral theses [Bensusan, 1999, Morin, 1999, Silver, 2000].

The focus in this thesis in on a particular type of meta-learning that Vilalta and Drissi [2002] call "inductive transfer". This term encompasses a range of others used in the literature including "learning to learn" [Thrun and Pratt, 1997b], "multitask learning" [Caruana, 1997, Silver and Mercer, 1998] and "transfer of learning" [Bensusan, 1999]. Research into inductive transfer

[10]Even if the conservation law does not hold, as it has been argued by Rao et al. [1995], and there is a general bias for learning it is of little comfort to the practitioner as its existence is solidly in the realm of the theoretical.

departs from traditional machine learning by assuming that the learner is situated in an *environment* of tasks that share bias requirements. Within these environments the learner can drawn on information gathered from past or parallel learning experiences as well as examples for the main concept that is to be learnt. If these other experiences are able to help with present and future learning tasks then some learning about learning must have taken place. As Thrun and Pratt [1997b] put it:

> Given a family of tasks, training experience for each of these tasks, and a family of performance measures (*e.g.*, one for each task), an algorithm is said to *learn to learn* if its performance at each task improves with experience *and* with the number of tasks.

They also argue that approaches to learning to learn "appear to be applicable to any application that involves *cheap data* and *expensive data*. This includes systems that must be trained by a customer (where data is often expensive), and that can practice the learning task by itself while still at the factory (where data is cheap)". This is especially true of text-based domains in which large corpora of easily categorised documents, such as newsgroups, can be used to form biases when learning from similar, user-classified documents, such as email [Oblinger et al., 2002].

2.4.4. Inductive Transfer. For the purposes of this thesis, an *environment* for inductive transfer consists of at least two learning tasks. When a learner is required to solve one of these tasks it will be called the *target* task and one or more of the remaining tasks in the environment will be used as *support* tasks. The aim of an *inductive transfer system* is to induce an accurate or otherwise suitable classifier for the target task using the examples available for the target and support tasks. These systems consist of one or more base-level learning components (or *base learners*) and a component that is able to modify the bias of the base learners (the *meta learner*).[11] The base learning components are often standard machine learning algorithms, or slight variants, that are applied to single tasks to produce classifiers. When the system as a whole is exposed to multiple tasks (either sequentially or in parallel) its meta learner can record information concerning the tasks and the performance of its base learners. This is used by the meta learner to combine, select or modify

[11] A noteworthy exception to this is found in the work of Jürgen Schmidhuber [Schmidhuber, 2004, 1997] in which the distinction is blurred between learning at the base-level and learning at the meta-level.

the base learning algorithms and their classifiers in order to improve the performance of the entire system on future or target tasks. An inductive transfer system is considered to have *succeeded* in performing inductive transfer if the modifications made to the base learner's bias result in better classifiers than if the base learner was used without any modification.

One of the simplest forms of inductive transfer is *direct transfer* "in which a learned classifier is simply copied from one [task] to another" [Cohen and Kudenko, 1997]. In this case, a base learner would be applied to the support task examples and the resulting classifier used as the solution to the target task. In the case where the support and target are for concepts with a large overlap in training examples this can be a successful approach, especially if there are many examples available for the support task. Direct transfer lies on the extreme of what is generally meant by inductive transfer since no learning is actually performed on the target task. Therefore, any information available in its examples is not used. More sophisticated approaches to transfer make use of both the support and target examples through a variety of meta learning and base learning algorithms as illustrated in the following examples.

The Multi-Task Learning (MTL) system described in [Caruana, 1997] uses an Artificial Neural Network (ANN) as the base learner. The MTL system learns concepts for the target task and support task in parallel using a shared network architecture. The support task examples influence the learning process since the back-propagation algorithm used by the base learner must find suitable network weights that are shared between the target and support concepts. Adaptations of this idea for sequential rather than parallel multi-task learning have been explored in the consolidation system of Silver and Mercer [1995] and the Task Rehearsal Method (TRM) of Silver [2000].

In their Task Clustering (TC) system Thrun and O'Sullivan [1996] use a nearest-neighbour clustering algorithm with an adjustable distance metric as a base learner. The meta learner in this case simultaneously optimises the weights that parameterise the distance metric using examples from a collection of tasks. Tasks with similar optimal weights are clustered together and deemed to be similar. When a novel target task is encountered the TC system determines the task cluster most similar to the target task and then uses that cluster's optimal distance metric for classifying future target task instances.

2.5. Inductive Transfer in Rule Learning

Having discussed inductive transfer in general, this section will now focus on how these approaches have been used in relational rule learning and ILP systems. Two aspects of relational learning make it a prime candidate for bias

learning techniques such as these. The first is the size of the search spaces encountered. The combinatorial nature of variable bindings mean the number of clauses can grow very rapidly as the size of clauses under consideration increases. As discussed in Section 2.4, this increases the minimum expected number of examples required for good generalisation. Secondly, when the task is individual-centred [Flach and Lachiche, 1999, Lavrač and Flach, 2001] background knowledge needs to be added for each example used. When the examples are graphs, as in molecular domains [Srinivasan et al., 1997, 1994] or finite element models [Dolsak and Muggleton, 1992], the amount of information needed to specify each example can be large and often expensive to obtain. For these two reasons it is more likely that relational learning tasks will have insufficient data when compared to similar, propositional tasks.

While a great deal of research has been carried out on biases for single task learning in ILP there are only a handful of systems that attempt to transfer these biases between tasks. The remainder of this section provides a chronological review of all six of these systems and an analysis in terms of the biases they infer. Wherever possible a system's review will include brief descriptions of any empirical work that assesses its ability to transfer bias. This is to create a picture of the type of environments current inductive transfer approaches to relational learning have been tested upon.

2.5.1. CLINT-CIA (1989). The interactive concept learning system, CLINT-CIA [De Raedt and Bruynooghe, 1989, De Raedt and Bruynhooghe, 1992] is one of the earliest first-order concept learning systems to transfer inductive experience between two or more learning tasks.

The base learner used by CLINT-CIA is CLINT [DeRaedt and Bruynooghe, 1988], an interactive learner developed to work as a learning apprentice. In this paradigm, the learning system is given a set of positive and negative examples as training data but can also repeatedly query a user during its specific-to-general search for clauses. The search is constrained by a sequence of *concept description languages* $L_1, \ldots, L_n$ that define progressively weaker restrictions on allowable clauses. For example, L_1 might only allow clauses where all variables in the body of a clause must also appear in the head, L_2 might loosen this to allow one variable not appearing in the head, and so on. Starting with the first concept description language and an uncovered positive example, CLINT constructs the most specific clause within L_1 that covers the example. If this *starting clause* covers any negative examples CLINT shifts its language bias by moving to the next concept description language and tries again. Once a consistent starting clause is found it is generalised by

systematically dropping single predicates and querying the user about these changes.[12] This process of requesting examples, shifting bias, generalising and asking questions continues until the clause is generalised as much as possible without covering negative examples.

The Constructive Induction by Analogy (CIA) component of CLINT-CIA aims to reduce the number of search steps and queries to the user by allowing the system to improve its generalisation ability through analogies with previously learnt concepts. This transfer mechanism is based on the assumption that "concept-descriptions are often very similar in the sense that they are instantiations of the same second order schema" [De Raedt and Bruynhooghe, 1992, §5.2]. A *second order schema* is a generalisation of a first order clause in which predicate names are replaced with *predicate variables*: terms that can be instantiated using predicate symbols. For example, the predicate variables `P`, `Q` and `R` in the second order schema `P(X) :- R(X,Y), Q(Y)` might be substituted with the predicate symbols `like`, `author`, and `french` to create the clause `like(X) :- author(X,Y), french(Y)` in the book preferences domain.

Second order schemata are created from first order clauses using the inverse of instantiation. At the end of its search, a clause found by CLINT is generalised into a schema by replacing all of its predicate symbols with predicate variables. CLINT-CIA makes use of these second order schemata in two ways. The first is by proposing concepts that are instances of a schema whenever one matches part of a newly learnt concept. These proposed concepts are shown to the user who can give them names if they are deemed to be meaningful. For example, suppose the following definition for grandson has just been learnt:

```
grandson(G,C) :- parent(G,X1), parent(X2,C), equal(X1,X2), male(C).
```

The schema `P(X,Y) :- R(X,Y), Q(Y)` matches this clause and so the user is asked if

```
P(X,Y) :- parent(X,Y), male(Y)
```

is a useful concept. If the user recognizes this as the "son" relation, it can be named as such and stored in the knowledge base for later use. The addition of these extra predicates modifies the language bias used by CLINT and can make future tasks easier to learn.

The second way the second order schemata are used is by modifying the search actions available to CLINT allowing it to guess at the target concept. When a starting clause is found, all the second order schemata are tested against it and if part of the starting clause matches a schema, the match is

[12]The types of queries CLINT can ask of the user are: requests for a new training example, a classification label for an instance proposed by CLINT, and whether a predicate in a justification for an example is irrelevant or not.

proposed to the user as a candidate for the target concept. This means target concepts similar to past ones can be found quickly by analogy instead of the predicate-by-predicate generalisation of the starting clause. It is important to note that both of these methods require the user to decide on the relevance or validity of the concepts derived from second order schemata. This is quite different to the majority of learning to learn approaches and their goal of independent agents within an environment of tasks.

Experiments with CLINT-CIA presented in [De Raedt and Bruynhooghe, 1992] show that its analogical approach to inductive transfer invents several useful intermediary concepts and helps reduce the number of queries made to the user. These results were shown to be the case in two distinct environments. In the *family* environment training examples for 29 different kinship relationships (including "grandparent", "sibling", "sister", "daughter in law", "nephew" and "male cousin") were presented to CLINT-CIA in a sequence for learning with the concepts "parent", "male", "female" and "married" as background knowledge. During this sequence, the CIA component helped "guess" 14 of the 31 required clauses using the second schema method described above and invented eight meaningful predicates using the first method. The remaining 17 clauses were found using the standard predicate-by-predicate generalisation.

Similar results were also obtained in a *chess* environment. Here, the concepts to be learnt were the rules for the movement of chess pieces (Rook, Bishop, Queen, Knight, King, and Pawn) on a board that could contain other pieces of either colour in various arrangements. The background knowledge given to CLINT-CIA consisted of various piece and tile relationships such as "on the same row", "nothing in between", "adjacent", "vertical distance", *etc.* In this environment the CIA component created seven schemata and using these was able to correctly guess 14 of the 21 clauses required to describe all the piece movement concepts.

2.5.2. MOBAL-MAT (1989). Like the CLINT system, MOBAL [Morik, 1993, Wrobel, 1994] fits into the "apprentice" category of learning systems, working with a user to interactively and inductively create a knowledge base.[13] The design principle that distinguishes the two systems is the focus in MOBAL on a balanced interaction between the system and its user. The aim is to avoid, without preventing, unnecessary user inspection and interaction. MOBAL accomplishes this through a uniform representation of facts, rules and other

[13]MOBAL is derived from the earlier BLIP system [Morik, 1989]. As the ideas in MOBAL encompass those in BLIP only MOBAL will be discussed here.

information and the careful modularisation of the system's learning tools so that one can easily provide input to another.

From the perspective of inductive transfer, the tools in MOBAL that are most interesting are its Rule Discovery Tool (RDT), Concept Learning Tool (CLT) and Model Acquisition Tool (MAT). The first two of these can be seen as the base level learner which the latter helps through learning better biases. The use of *rule models* is the mechanism that allows the MAT to modify the constraint bias used by the base level learner.

Rule models are very similar to the second order schemata used by the CLINT-CIA system described above. Like schemata, rule models are second order, making use of predicate variables that can be replaced by predicate symbols from the background knowledge.[14] The main difference between schemata and rule models, however, is in their use. In CLINT-CIA schemata are used to invent new concepts or allow the CLINT search procedure to "jump" to a potential target clause. In contrast, rule models are used by the RDT to define its entire search space - much like the concept description languages used by CLINT - which it then explores in a general-to-specific fashion.

The RDT, given a set of facts and a set of rule models, is expected to return a set of clauses that are: a) each instantiations of one of the given rule models and b) the most general generalisations of the given facts. It finds this set of clauses by systematically generating and testing each of the possible instantiations of the rule models against the facts. The order in which these are tried is a consequence of a natural subsumption-like generality relationship[15] over the rule models. The most general rule models are tried first, followed by the next most general, and so on.

The rules returned by the RDT need only be deductively supported by the facts and the existing knowledge base. This condition is weaker than the usual concept learning requirement that the learnt rules and background knowledge explain the training examples. Even if all the facts are for a particular target concept the RDT can return rules that do not mention that concept. The task of the CLT within MOBAL is to call the RDT and guarantee that the rules it returns characterise the target concept. This is achieved by the CLT partially instantiating the rule models it gives to the RDT so each rule model contains the target concept.

[14]One slight difference is that rule models place an extra restriction on substitutions: two distinct predicate variables in a rule model cannot be instantiated to the same predicate symbol.

[15]Rule model R_1 is *more general than* R_2 if for all predicate variable substitutions Σ there exists a first order substitution of terms σ such that $R_1\Sigma\sigma \subseteq R_2\Sigma$.

The MAT within MOBAL allows the system to construct new rule models thereby shifting its constraint bias. When new rules are entered into its knowledge base the MAT tests to see if they are instances of any of the available rule models. If this is not the case, a new rule model is created by turning the predicate symbols in the rule into predicate variables, much like the CLINT-CIA approach. The newly created model is then added to the rule model hierarchy and used for future inductions by the RDT. An important observation about this process is that since rules can only be created by RDT *from* the existing rule models, any such rule will not give rise to *new* rule models. New rule models can only be created from rules that are entered into the knowledge base by a user.

Both MOBAL and CLINT-CIA require human intervention to when learning or shifting their biases. This is what makes them "learning apprentice" systems rather than traditional concept learners. We now turn our attention to systems that can *automatically* detect and transfer biases between learning tasks.

2.5.3. Learning Relational Clichés (1993). *Relational cliché* is the name given to "potentially useful combinations of predicates" that are "semantically meaningful" [Silverstein and Pazzani, 1991] when searching a space of rule candidates. The ILP system FOCL [Pazzani and Kibler, 1992] performs a greedy search, repeatedly adding a single literal with the highest information gain at each step. This strategy can mistakenly overlook the addition of determinate structural predicates as they provide no information gain. For example, when building a rule for book preferences a predicate `author(Book, Author)` would never be added to a rule since every book has an author. Therefore, regardless of the training data, it will not help discriminate between positive and negative examples. This is unfortunate since the *pair* of literals `author(Book, Author), nation(Author, uk)` may give a very good split between a reader's likes and dislikes but will not be considered due to the greedy avoidance of the `author/2` predicate.

A relational cliché or RC consists of a *pattern* that describes how two literals are to be connected and a set of *restrictions* that limit the ways in which the pattern can be instantiated. The most general RC used in FOCL is the *unconstrained cliché* (UC) which allows any two predicates to bind to one another in any manner. More specific RCs, such as the "partof" cliché or "threshold comparator" cliché describe common logical tests. The latter, for example, has the following pattern `pred(...,A,...) & comp(A, Thresh)` and the restrictions that: argument `A` be a new variable, `Thresh` be a constant with the same type as `A` and `comp` be a comparator predicate such as less-than or

not-equal-to. FOCL tries adding all possible instantiations of its available RCs whenever it reaches a point in its search where no single additional literal will improve the information gain of the rule being built.

In their paper "Learning Relational Clichés", Silverstein and Pazzani [1993] describe a fairly involved method of automatically discovering and organising RCs. To begin with, only the unconstrained cliché can be used by FOCL when learning new concepts. Once a new concept is learnt, all pairs of conjunctions within the new theory are examined to build a set of *cliché instantiation candidates* or CICs. These are the conjunctions which have the maximum information gain over the training data but would not have been found during a literal-by-literal search. Each CIC is then transformed into the RC with the most specific pattern and restrictions that subsume it. Finally, the newly created RCs are added to a cliché hierarchy which is ordered by subsumption and pruned using a heuristic that trades off a RC's coverage and efficiency on the task just attempted.

It is worth noting here that the RCs constructed using this LRC process do not actually extend the range of hypotheses FOCL can induce beyond that which is possible using only the unconstrained cliché. Any conjunction of literals that can be instantiated from a RC can also be instantiated from the UC. The aim of building up and pruning the cliché hierarchy is to reap the benefits of a search with look-ahead while minimising its computational overhead.

2.5.4. Concept Sharing in M-FOCL (1993). The M-FOCL system of Datta and Kibler [1993] is based on "the idea of learning multiple concepts during classification, and biasing reuse of conceptual structures". Based on FOCL [Pazzani and Kibler, 1992], M-FOCL's method of inductive transfer is summed up by the authors as the ability to "dynamically [change] its representational bias by introducing new terms that have been found to be useful in the past". The new terms are created by inspecting the rules induced for a secondary task and are added as extra background knowledge before inducing a classifier on the primary task. This "concept sharing" extends the learner's background knowledge so that rules that would have been previously excluded by the language bias or overlooked by FOCL's greedy, general-to-specific search can be considered for use in the primary task.

Three modes for constructing new predicates from rules induced for a secondary task are presented and tested with M-FOCL: CNR (concept rules), CLR (clause rules) and CJR (conjunct rules). These differ in which structures within the learnt secondary concept are turned into new predicates for

learning on the primary task. In the CNR mode, M-FOCL transfers the entire learnt secondary concept as a single new predicate whereas in CLR and CJR modes M-FOCL creates, respectively, new predicates for each clause or each two-predicate conjunct that appears in two or more clauses. The implementation of the concept sharing methods are better understood through the following example.

Returning to the book preferences scenario discussed earlier, suppose this theory was induced for books a reader might like:

```
like(B) :- nation(B, aus), genre(B, scifi), year(B, 90s).
like(B) :- nation(B, aus), genre(B, horror), year(B, 90s).
like(B) :- nation(B, uk), genre(B, scifi).
```

In its CNR mode, M-FOCL would simply add a renamed version of the `like/1` predicate, defined using the same clauses, to the background knowledge. Future inductions would be able to introduce this new predicate into rules. To see how this might make learning easier, suppose a second reader liked exactly the same type of books as the first with the extra proviso that they be short. This second reader's book preferences could then be represented using a single clause:

```
like(B) :- cnr(B), size(B, short)
```

where `cnr` is the renamed version of the first reader's `like/1` predicate.

When M-FOCL is in CLR mode, each of the above clauses defining the `like/1` predicate would be added to the background knowledge with unique names such as `clr1`, `clr2` and `clr3`. With these extra predicates in the background knowledge, the rule `like(B) :- clr2(B), size(B, short)` would capture a preference for short Australian horror novels written in the 1990s.

The conjunct rule mode, CJR, enables M-FOCL to reuse even smaller parts of previously learnt concepts. Whenever a pair of predicates appears in at least two rules, a new predicate symbol is invented and defined using the conjunct. In the example above there is only one such repeated conjunct, `nation(B, aus), year(B, 90s)`, which appears in the first two of the example rules above. Only repeated conjuncts are named and reused in order to keep the number of invented predicates manageable.

M-FOCL and its various transfer modes were empirically tested on two domains: chess piece movements and poker hands. The chess domain consists of four types of learning tasks, each requiring the learner to induce rules describing the movement of a single chess piece on an empty chessboard. The chess pieces considered are the Knight, Rook, Bishop and Queen. Tasks in the poker hand domain present the learners with examples in the form of a hand

consisting of five cards, each with a suit (hearts, clubs, etc) and a rank (ace, 2, 3, etc). The five target concepts in this domain correspond the following special hands in poker: pair, three of a kind, four of a kind, two pair and full house.

The authors were concerned with how the CNR, CLR and CJR methods affected the generalisation accuracy, concept size and search complexity when compared to FOCL on the same tasks. A variety of training set sizes were tested in both domains, and in the chess domain attribute noise was added to test M-FOCL's robustness to training set errors. In general, the results showed that, when training data is limited, concept sharing can increase generalisation accuracy and reduce the induced concept size, though often at the expense of increasing the search complexity. No single transfer method, however, consistently outperformed any other. On the noise-free chess domain transferring whole concepts (CNR) or clauses (CLR) was more successful in improving accuracy and decreasing concept size than transferring conjuncts (CJR). When noise was added the CLR and CJR approaches were better than CNR which did not improve on the accuracy of the base learner FOCL. On the poker domain, CNR had a lower accuracy than FOCL on average while the CLR and CJR methods dramatically reduced learnt concept size while improving accuracy.

2.5.5. Repeat Learning in Progol (1998). A similar but more sophisticated type of representational inductive transfer is the Repeat Learning approach proposed by Khan et al. [1998]. This framework assumes that if a learner's hypothesis language is less expressive than that needed to succinctly describe the target concepts presented to it, the learner should be able to shift representation languages by inventing predicates. The authors implement such a predicate inventing learner using the constraint solving mechanism of PROGOL 4.4. This implementation will be referred to as RL-PROGOL in the remainder of this thesis.

Constraint solving is a type of "lazy-evaluation" [Srinivasan and Camacho, 1996] in which specified predicates collect values substituted into their input arguments during training. When such a predicate is used in a rule during testing and is required to give a value for their output arguments, all the collected input values are passed to its definition for processing. This technique extends the type of background predicates that can be used in rules, allowing predicates that compute the maximum value or perform linear regression on the set of input constants.

RL-Progol uses constraint solving to invent predicates. This is done by introducing a "lazy" literal invent(P, X) into a bottom clause wherever its mode declaration will allow it. This is defined in such a way that whenever a clause containing such a literal is evaluated against a training example, all the substitutions for the input variable X such that the clause covers that example are collected and recorded to a Prolog database. Once this is done for all the examples, the constants in the database are saved as a model for an invented predicate sym_i. During testing, the literal invent(sym_i, X) is interpreted as $sym_i(X)$ with the model saved for it during the training phase. The sym_i predicates invented during a support task run are saved along with their models and added as extra background knowledge in subsequent learning tasks, expanding the representation language of the learner.

The predicates invented and transfered by RL-Progol aid learning from limited data by reducing the number of rules required to express a concept. This can help avoid the "small disjunct" problem [Holte et al., 1989] where rules covering only a small portion of the instance space can be under-represented in a training sample. Taking the book preferences domain as an example again, consider a task with a target concept "the reader likes a book if it is Australian romance, science-fiction or horror". A training set for this task might have the following positive examples and associated background information:

```
like(a). nation(a,aus). genre(a,scifi). year(a,90s). size(a,short).
like(b). nation(b,aus). genre(b,romance). year(b,90s). size(b,long).
like(c). nation(c,aus). genre(c,horror). year(c,80s). size(c,long).
```

When evaluating the rule `like(B) :- genre(B, G), invent(P, G)` the variable `G` will be bound to the constants `scifi`, `romance` and `horror` when it is tested against the first, second and third example respectively. These constants are recorded by the lazy `invent/2` predicate and used to create a new predicate p1/1 with model $\{p1(scifi), p1(romance), p1(horror)\}$. When added as extra background knowledge for some new learning task, this predicate allows the target concept

```
like(B) :- size(B,short), genre(B,scifi).
like(B) :- size(B,short), genre(B,romance).
like(B) :- size(B,short), genre(B,horror).
```

to be expressed as a single rule: `like(B) :- size(B,short), genre(B,G), p1(G).` Even if the training set for this new task is small and only mentions short romance and science-fiction books, the predicate `p1/1` enables an inductive leap to be made that includes short horror books. Whether this new inductive leap is valid or not depends on the relatedness of the two learning tasks.

In [Khan et al., 1998], the Repeat Learning approach to inductive transfer is empirically tested on a chess movement domain similar to the one used for M-FOCL but restricted to Knight and King movement tasks only. These same tasks are used to compare RL-PROGOL and DEFT in experiments in Chapter 5 and so are only briefly discussed here. The King and Knight pieces both have an eight-way symmetry in their movement requiring eight separate rules to describe in terms of rank and file differences on the chess board. In addition, the differences used only take on values from -2, -1, 0, 1 or 2 (e.g., the King can move one rank forward and change its file by -1, 0 or 1). RL-PROGOL invents predicates that express this disjunction allowing the number of rules required to express the pieces' movement to three for the King and five for the Knight. This in turn improves the generalisation accuracy on small datasets when compared to learning without the new predicates.

2.5.6. XFOIL-CLUSE (1999). In her Ph.D. thesis [Morin, 1999] and conference paper [Morin and Matwin, 2000] Morin introduces the CLUSE system for inductive transfer which builds on ideas from the Learning Relational Clichés system described in Section 2.5.3 above. In contrast to the LRC approach, which derives clichés from conjunctions in previously learnt concepts, CLUSE uses a modification of relative least general generalisation (RLGG) [Plotkin, 1971] called *contextual least general generalisation* (CLGG) to generalise simple relational structures, called "chains"[16] that are shared between training examples. The resulting conjunction of predicates are called *domain dependent clichés* (DDCs). The DDCs are further generalised into *domain independent clichés* (DICs) by replacing all predicate symbols in a DDC with predicate variables. Like the clichés in LRC, DDCs and DICs could be tested and pruned based on their utility. As their names suggest, domain *dependent* clichés are shared between tasks that share the same domain language (predicate symbols, functions, constants) whereas domain *independent* clichés can be shared between tasks with completely different language elements. Both DDCs and DICs are used for the same reason relational clichés are used in LRC: to avoid local minima when searching.

The base learner that performs the rule search in Morin's work is called XFOIL. This FOIL [Quinlan, 1990] variant performs roughly the same greedy, information gain guided, general-to-specific search as its predecessor but can also make use of clichés discovered by CLUSE. Clichés are used whenever the

[16]Example chains are conjunctions of literals containing exactly one literal with arity two, the rest of the chain being property predicates describing the terms within that literal. These are similar to, but less general than, the "simple clauses" used in LIME [McCreath and Sharma, 1998]

search reaches a point where adding a single predicate will not increase the rule's gain. If DDCs are available and applicable to the current domain each one is tried by adding all its predicates to the rule and computing the gain for the result. Also tried are rule extensions by conjunctions of predicates derived from DICs. These are done by substituting predicate symbols for predicate variables in any available DICs. If any of these rule refinements improve the information gain of the rule the refinement with the largest gain is applied and the search continues. This ability of clichés constructed by CLUSE to modify the search actions performed by XFOIL is an instance of transferring search bias.

The XFOIL-CLUSE approach was tested on and across two domains: a *blocks* domain where the learner must classify scenes similar to those found in Bongard problems [Bongard, 1970], and a *mesh* domain, first investigated in [Dolsak and Muggleton, 1992] where the learner must classify edges found in finite element models for computer aided design. The experiments on the blocks domain were primarily to test the hypothesis that DDCs and DICs provide an appropriate look-ahead to XFOIL myopic search. The domain involved three tasks: two primary blocks tasks B1 and B2, and a secondary task B3 that is used to generate clichés. All three tasks had different target concepts that described scenes that contain structures like "a circle above a rectangle above a black right-angled or isosceles triangle". Without clichés XFOIL was unable to learn any rules for the tasks B1 and B2. In both cases this was because the relational predicate `above` was needed to express the target concepts but did not alone discriminate between positive and negative examples. DDCs constructed from examples in the task B3 included the cliché `above(X,Y), circle(X), rectangle(Y)` which, when added to a rule body, correctly split some positive and negative examples and allowed the target concepts in B1 and B2 to be learnt. Similarly, the DIC derived from the above DDC also allowed XFOIL to consider the addition of the `above` predicate and therefore find discriminating rules.

The mesh domain was not originally conceived of as an environment of related learning tasks. Normally, the target relation, `mesh(Edge, Num)`, to be learned is one that specifies how many elements an edge on a model should be divided into given the structure of the model around the edge.[17] The model structure is described through background predicates which specify whether a given edge is "fixed", "loaded", "short", "free" or whether two edges in a model

[17]This is an important design problem when testing physical properties of models. Too many elements in a model means computation can be slow while too few result in poor approximations.

are "neighbours" or "opposite" in relation to one another. In order to test CLUSE, Morin derived from the mesh dataset a collection of six distinct target concepts, $mesh_1, \ldots, mesh_6$, where each $mesh_i$ treated examples satisfying `mesh(Edge,`i`)` as part of the concept and all other examples as negative. DDCs were derived from training data for each concept and used when learning each of the other concepts. The 3-fold cross-validation accuracy of XFOIL on the resulting 36 trials showed that the use of DDCs between the mesh concepts improved its average accuracy on 17 of the trials when compared to XFOIL without DDCs. Although generalisation accuracy increased by up to 16% when DDCs were used, the cost incurred by using them in this domain was a doubling of CPU time required for learning. Through the use of pruning DDCs this penalty was reduced to a 20% increase in CPU time with comparable accuracy improvements.

Experiments were also carried out on the mesh concepts to test the utility of cross-domain transfer using DICs constructed using examples from the B3 blocks task. Once again, considerable improvements in accuracy were noted on half the mesh tasks but these came at a cost: a 14-fold increase in search time. This was mainly due to the large number of possible instantiations of DICs in the mesh domain that had to be tried.

2.5.7. Summary. All of the systems reviewed above perform inductive transfer for systems that use a covering strategy for rule learning. The main differences between them are the mechanisms they use to extract a bias from a support task and how it is then applied to the target task. Using the characterisation of rule search as in Section 2.3.4 above, these systems can be categorised in terms of how they modify the language, search, and evaluation bias of the base-level learner. The summary in Table 2.2 lists all six inductive transfer systems along with a short description of the mechanisms each uses to modify one or more of those three biases.

TABLE 2.2. Inductive transfer systems for relational rule learning and the bias each modifies.

	Language	**Search**	**Evaluation**
CLINT-CIA	Pred. Inv.	2nd-order Schema	
MOBAL-MAT	Rule Models		
FOCL-LRC		Clichés	
XFOIL-CLUSE		Clichés	
PROGOL-RL	Pred. Inv.		
M-FOCL	Pred. Inv. (CNR)	Macros (CLR/CJR)	
DEFT			DFTs

The first two systems in the table are considered mainly for the techniques they employ to transfer bias as they are not able to learn concepts without human interaction. Both CLINT-CIA and MOBAL-MAT make use of second-order expressions - schema and rule models - to modify the way in which their base learning algorithms search for rules and the space they search within. In MOBAL-MAT, rule models are used to constrain the rule language, restricting candidate rules to those which are instantiations of the learner's current set of rule models. The MAT component of MOBAL can construct new rule models when expert provided rules are provided for one task and then use the new models as constraints on subsequent tasks.

While the second-order schema used by CLINT-CIA are similar to rule models in that they derived from first-order rules, however, they are used to modify both the search and languages biases. If part of a starting clause constructed from an example matches a schema, it is immediately proposed to the expert as a candidate to add to the hypothesis. This short-cuts the usual literal-by-literal removal of conditions from the starting clause. During such a search, CLINT-CIA can also invent new concepts for use in later tasks by proposing rules to the user for naming. When added to the learner's knowledge base these invented predicates modify its language bias, potentially making it easier to express complex concepts with simpler rules.

Predicate invention is also used by two of the systems that can learn concepts and modify language bias without human intervention. Both PROGOL-RL and M-FOCL automatically define new predicates during or after learning on a secondary task. The repeat learning approach used by PROGOL-RL is more a "data-driven" form of constructive induction (models for new predicates are drawn from tests against examples) when compared to M-FOCL's "hypothesis-driven" approach (predicates are defined using part or all of a learnt concept).[18] Except when it is in CNR mode (where whole concepts are reused), the predicates invented by M-FOCL do not change the semantics of concepts that can be expressed by single rules. When part or whole rules are used to define new predicates in M-FOCL's CJR and CLR their addition during a search acts like a "macro", adding several literals at a time. Constructing macro-refinements for better search is also the motivation for the inductive transfer approaches used by FOCL-LRC and XFOIL-CLUSE systems. The transfer of relational clichés performed by both these systems modifies the refinement bias of the hill-climbing search used by their base learners. This can

[18]Kramer [1995] discusses data- and hypothesis-driven approaches to constructive induction and predicate invention for concept learning.

help the searches avoid local minima or plateaus when attempting to maximise the evaluation function.

The last row and column of Table 2.2 show where the system proposed in this dissertation fits into the existing research. Unlike any of the other systems, DEFT only modifies the base-level learner's evaluation bias. This is done through the construction and transfer of a data-structure called a DFT which captures statistical properties of rule performance on a support task. The details of the theory and implementation are provided in the next two chapters.

2.6. Conclusions

The aim of this chapter was to introduce and explore the topic of rule learning from limited data and investigate some of the existing solutions that have been proposed in the literature. The above discussion has focused on this problem for a class of rule learning algorithms that use a "separate and conquer" strategy for building sets of rules. An analysis of the core of this strategy - repeatedly finding a single, high quality rule - emphasised the importance of evaluation. As argued in Section 2.3.3, evaluation is central to the view of rule search as an estimate optimisation problem. When data is limited, much of the poor performance of rule learning systems can be accounted for by unreliable estimates of rule quality. Of the three aspects of rule search described in Section 2.3.4, the review of inductive transfer systems for rule learning in Section 2.5 shows that only language and search have been explored as biases that can be modified and transferred in rule learning. This observation points to the possibility of inductive transfer systems for rule learning which modify evaluation bias. The next two chapters introduce an inductive transfer system called DEFT which is able to modify a base learner's evaluation bias. The aim of the system is to use information obtained from tasks within an environment to improve evaluation estimates on other tasks where training examples are scarce.

"[The] task is to define the relation of confirmation ... between evidence and hypothesis in terms of anything that ... may reasonably be supposed to be at hand when a question of inductive validity arises. This will include, amongst other things, some knowledge of past predictions and their successes and failures"

- Nelson Goodman [1983, pg. 85]

CHAPTER 3

Similarity-Based Transfer of Evaluation Bias

All the approaches to inductive transfer reviewed in the last chapter modify either the learner's search bias or representation bias. Unlike those approaches, the transfer system described in this chapter is designed to modify the base learner's evaluation bias to improve estimates of rule quality. By testing rules on examples from the support task a learner can gain "some knowledge of past predictions and their successes and failures" and use this to better assess rules on the target task.

The remainder of this chapter defines what it means for rules and tasks to be similar to one another and describes a general method for taking them into account during evaluation. It is organised as follows. Similarity-based inductive transfer is motivated through the use of an example and a discussion presented in Section 3.1. Section 3.2 then reviews some Bayesian estimation techniques used in machine learning. These techniques are extended to modify rule classification probability estimates through the use of priors. Section 3.3 introduces the notion of rule similarity as a general equivalence relation and describes how classification priors can be defined for a rule by averaging the performance of similar rules on a support task. Section 3.4 uses rule similarity to define what it means for two learning tasks to be similar and goes on to state and prove the main theorem of the chapter: priors calculated from support tasks can guarantee better classification estimates on a target task provided some natural conditions regarding the similarity of the two tasks are satisfied. Section 3.5 addresses some difficulties with defining general rule similarity relation and computing priors using them. This is done by defining a special type of rule similarity called description similarity which opens the way for a computationally efficient way to create classification priors. Finally, in Section 3.6, properties of the new transfer technique are discussed in light of related research from neural network research and computational learning theory.

3.1. A Motivating Example

The purpose of this section is to introduce a set of learning tasks in a very simple domain to highlight the problem of learning from limited data and to

	size	
genre	small	large
scifi	?	+
horror	-	?
romance	-	?

Tina

	size	
genre	small	large
scifi	+	-
horror	+	-
romance	+	-

Scott

	size	
genre	small	large
scifi	-	-
horror	+	+
romance	-	-

Harry

FIGURE 3.1. The reading preferences of Tina, Scott and Harry in the Book World. Each of the six books lie at the intersection of genre and a size. A '+', '-' and '?' indicate that the reader enjoyed, disliked or has not read the book, respectively.

show how this problem can be solved through the modification of a learner's evaluation bias. This example will be reused throughout this chapter to clarify some of the details of the various definitions and theorems.

3.1.1. The Book World. Three friends, Tina, Scott and Harry, all reside in a very contrived world in which only six books exist. Each book can be uniquely described by its genre, either science-fiction, horror or romance, and its size, either small or large. Both Scott and Harry are avid readers, each having read all six books. The somewhat morbid Harry enjoyed both the small and large horror novels and disliked the four others while Scott's diminished attention span meant he liked reading all three short books but none of the long ones. Tina has only read half the available books. She enjoyed the large science-fiction book but disliked the small horror and small romance novels. The reading preferences of all three friends are summarised in Figure 3.1.

Both Scott and Harry's favourite books can each be summarised with a single rule: Harry will enjoy a book if it is the horror genre whereas Scott will enjoy a book if it is small. However, proposing a single rule which captures Tina's preferences is difficult given the data that is available as there are three rules involving genre and size that are consistent with Tina's tastes, namely:

GS: Enjoy if book's Genre is Science-fiction

SL: Enjoy if book's Size is Large

SLGS: Enjoy if book's Size is Large and its Genre is Science-fiction

Tina's mother wants to infer a rule about Tina's taste in books to buy her a birthday present she will like. An appropriate evaluation function for this task

will therefore measure a rule's quality by its ability to predict correctly. There is no purely evidential way of deciding which of these rules is best as all three have a classification probability matrix

$$\begin{bmatrix} \frac{1}{3} & 0 \\ 0 & \frac{2}{3} \end{bmatrix}.$$

Any evaluation function which only uses CPMs to score a rule will assign the same value to GS, SL and SLGS. If Tina's likes and dislikes for all six books were known this situation would not occur as only one of these rules would have zero misclassifications. However, the estimates of the true classification probabilities from only three examples are too rough to confidently decide on a single rule.

How would the method for inductive transfer proposed in this chapter help in the above situation? In a nutshell, it would modify the estimated CPMs for rules regarding Tina's taste in books by letting Tina's mother express a bias in relation to the preferences of Harry or Scott. For example, she may believe that the best rule to describe Tina's tastes will be similar to the best rule for describing Harry's. In this case, the inductive transfer system needs to interpret what is meant by the term "similar". In what sense is the rule "enjoy if genre is science-fiction" similar to the rule "enjoy if genre is horror"? One might argue that there are elements in common between the two genres in that they both explore the human condition in unusual circumstances. However, there is a simpler and purely syntactic relationship and that is both rules consider a book's genre unlike the rule "enjoy if size is large". This notion of similarity carves up the set of rules for the Book World domain into classes containing the rules that mention genre or the rules that mention size. When it comes to estimating the classification performance of a rule for Tina's tastes the performance on Harry's examples of the rules from the same class can be taken into account. For example, the inductive transfer system might use this information to increase the true positive rate of the GS rule since it is in the same similarity class as "enjoy if genre is horror" which performs well on Harry's examples. Also, the false negative rate for the SL rule might be increased as rules like it are poor predictors on the examples of Harry's taste in books. These modifications would be reversed if Tina's mother believed that Scott's tastes formed a better support task. With better CPM estimates the evaluation function used to score the rules will have a better chance of picking the one that best describes Tina's tastes.

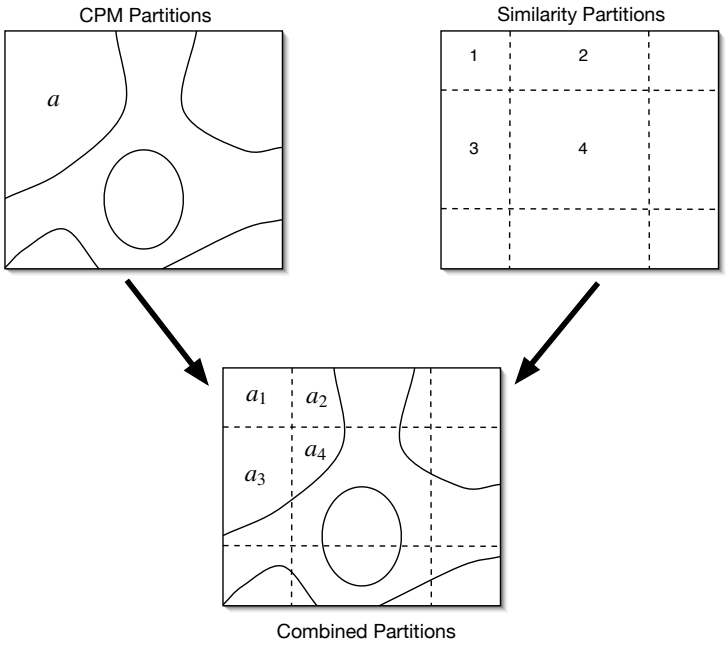


FIGURE 3.2. The rule space is partitioned according to CPM and similarity. The intersection of these results in a more refined partition.

3.1.2. Estimation, Similarity and Transfer. The above example illustrates a couple of key ideas about the inductive transfer of evaluation bias that are explored in detail in the remainder of this chapter. The first is that the mechanism for implementing an evaluation bias is through the modification of classification probabilities. The estimates of these probabilities that are made using examples of the target task are pushed up or down depending on the performance of rules on the support task. This manipulation of classification estimates is given a stronger theoretical background in Section 3.2 below. The second key idea is that similarity between two learning tasks can be interpreted in terms of the performance of similar rules on the two tasks. Statements such as "the books Tina and Harry enjoy are similar" can be interpreted as saying something about the predictive power of classes of rules on both tasks. These classes are defined in terms of properties of the rules such as "the rule tests for genre" and "the rule tests for size". This notion of similarity classes is formalised in Section 3.3. Inductive transfer from a support task to a target task is implemented by joining together these two ideas. The estimates of a rule's classification probabilities for the target task can be modified by values derived from the classification probabilities of similar rules on the support task.

Figure 3.2 gives some idea of why this approach might be of value in general. The CPM partitions are sets of rules which have identical CPMs on the training

data. In the context of the earlier example the set denoted a could contain the three rules SL, GS and SLGS. As argued in the last chapter, when training data is limited there are generally many rules per CPM class. The similarity classes shown on the right of the figure partition the same rule space in a different way to the CPM partitions. In terms of the example once again, the similarity class labelled '1' might contain rules that test for a book's genre whereas the class labelled '2' might be for all the rules that do not. Overlaying the CPM and similarity classes results in a more refined partition of the rule space. If the CPM estimates for rules in similarity class 1 are modified in a different manner to those in similarity class 2 then rules in the new class a_1 will have different evaluation scores than the rules in class a_2.

One way in which CPM estimates for rules in different similarity classes can be modified independently is by assuming different prior values for the estimates in each class. These priors can then be updated using the information in the real CPM estimates. This approach is the topic of the next section.

3.2. Classification Priors

As argued in the previous chapter, one of the fundamental difficulties with learning from small training sets is the poor estimation of the classification probabilities used to assess rule quality. Evaluation functions that use poorly estimated classification probabilities or contingency matrices will rank candidates badly, assigning higher scores to candidates that might receive low true scores when assessed over the entire instance space.

When an appropriate bias for a learning task is selected the problems associated with limited training data can be lessened. The inductive transfer techniques reviewed in the last chapter show that language and search biases can be learnt and transferred between tasks. However, none of these methods directly address the problem of improving the classification probability estimates. Instead, rules with low true evaluation scores are avoided by restricting or expanding the allowable candidate rules, or changing the order in which they are searched. In contrast, directly improving the classification probability estimates is the starting point for the similarity-based approach which draws on well-known Bayesian techniques for estimation of probabilities from small samples.

In the earlier example, the classification probability estimates for a rule were modified by nudging estimates up or down based on the performance of similar rules. This tweaking of estimates can be framed as Bayesian estimation which advocates the use of prior distributions over the possible values an

estimate can take before any data is examined. These methods are reviewed and are used to encode evaluation bias when learning from limited data.

3.2.1. Bayesian Estimation. Estimating probabilities from samples is an important and well-studied area of statistics [Bishop et al., 1975, Walpole and Myers, 1978]. The methods used for this type of estimation fall into one of two camps: frequentist, where estimates are calculated directly from the sample; and Bayesian, where the sample is used to update an *a priori* value for the quantity being estimated [Good, 1965, Jaynes, 2003, Jeffrey, 2004]. Their differences are best grasped through the following simple example.

If a biased coin is flipped N times and comes up heads n of those, the frequentist (or maximum likelihood) estimate for the chance of a head when the same coin is flipped in the future would be $\frac{n}{N}$. When N is large the central limit theorem ensures that this estimate approaches its true value. When the sample is small, however, the estimate can be unreliable. For instance, imagine a coin which is flipped once and comes up tails. A frequentist would estimate the probability of heads in the future to be zero, that is that there is no chance of observing a head. Arguably, a more palatable estimate could be obtained by assuming that there is an even chance of heads and tails before any throw is made and updating this initial guess depending on the result of the coin toss. In the case of a single tail being thrown, the updated estimate might assign odds of 2:1 (or at least something non-zero) against the next throw landing heads. Even assuming that the probability the coin will land heads is uniformly distributed over the interval $[0, 1]$ results in a non-zero estimate for heads after a single tail is observed.

This second estimate epitomises the Bayesian approach in which the "essential defining property" is, according to Good [1965, §2.1], that it is "meaningful to talk about the probability $\Pr(H|E)$ of a hypothesis H, given evidence E". In the case of the coin tossing example, the hypotheses are values for the true probability of the coin landing heads. If such a distribution can be computed, the value to which it assigns the maximum likelihood can be used as an estimate that is informed by the prior assumptions. There is no question that the posterior probability $\Pr(H|E)$ can be computed as it is consequence of the definition of conditional probability known as Bayes' identity:

$$\Pr(H|E) = \frac{\Pr(H)\Pr(E|H)}{\Pr(E)}.$$

This states it is proportional to the product of $\Pr(E|H)$, the probability of the evidence if H is true, and $\Pr(H)$, the prior probability for H.

The only slightly controversial aspect of this type of estimation (and one best left to philosophers of knowledge) is the assumption that the distribution $\Pr(H)$ can be known in advance of any evidence. In practice, priors are used to encode any extra knowledge one may have about a random variable that we arrive at analytically (*e.g.*, the coin is symmetric) or from past experience (*e.g.*, past coins have been fair). A choice of a prior distribution determines a preference for certain hypotheses over others. When such a preference cannot be justified, we can encode our lack of knowledge by assuming a uniform distribution over the hypotheses.

In this sense, as Jaynes [2003] puts it, "Bayesian methods are - or at least may be - speculative. If the extra hypotheses are true, then we expect that Bayesian results will improve on maximum entropy; if they are false Bayesian inferences will likely be worse". These beliefs may turn out to be wrong, but when they are correct they can substantially improve the quality of an estimate.

When evaluating a rule, the classification probabilities for that rule are quantities that require careful estimation. Many evaluation functions use one or more of the values found in a rule's classification probability matrix to assign it a score. When training data is limited these probability estimates can be poor resulting in inaccurate rule evaluation. One way, therefore, of achieving better rule evaluation is to improve the estimates they are based upon. As discussed above, prior distributions over estimates appears to be a natural place to add any extra available knowledge about the predictive quality of rules. The application of the Bayesian approach to the problem of classification probability estimation is one of the key theoretical steps towards the inductive transfer of evaluation bias presented in this chapter.

3.2.2. Priors for Classification Probability Matrices. In the introductory example for this chapter, the idea of modifying the true positive and false positive rates for a rule was proposed as a way of breaking ties between them, possibly by encoding extra information about the expected performance of each rule. This fiddling of classification rates can be seen as a special case of modifying the values in a rule's classification probability matrix. The Bayesian approach of using prior distributions for the classification probabilities provides a principled way of doing this and the result can be seen as an attempt to improve the probability estimates that are derived from the training examples.

To recap part of Section 2.1.2 of the preceding chapter, a rule's classification probability matrix (CPM) is defined to be the relative frequencies of the

classification counts appearing in its contingency table. That is, if

$$\mathbf{n} = \begin{bmatrix} n_{++} & n_{+-} \\ n_{-+} & n_{--} \end{bmatrix}$$

is a contingency table for the evaluation of rules against $N = \sum_{i,j} n_{ij}$ examples then its corresponding CPM is

$$\mathbf{p} = \begin{bmatrix} p_{++} & p_{+-} \\ p_{-+} & p_{--} \end{bmatrix}$$

where each $p_{ij} = \frac{n_{ij}}{N}$ is a frequentist estimate of the probability that the rule will classify an example with label j as having label i. In what follows, this CPM estimate will be known as the *raw CPM estimate*. Like all frequentist estimates the raw CPM estimate can be unreliable when the total number of examples N is small and the purpose of this section is to examine how they might be improved.

The classification counts in the matrix $\mathbf{n}$ can be thought of as frequency counts for what is known as a *multinomial sample*. Each time an example is tested against the rule the resulting predicted and actual classification of the example must fall into exactly one of the four classification categories: true positive, false positive, false negative, true negative. A natural family of prior distributions for the frequencies $\mathbf{p} = (p_{ij})$ for a multinomial sample is the Dirichlet distribution which has a density function

$$D(\mathbf{p}; \mathbf{m}) = \Gamma(M) \prod_{i,j} \frac{p_{ij}^{m_{ij}-1}}{\Gamma(M)}$$

where the $\mathbf{m} = (m_{ij})$ are the distribution's parameters, $M = \sum_{i,j} m_{ij}$ and Γ is the gamma function that is commonly found in these sorts of distributions. Although this is a complicated prior distribution to define, the maximum likelihood value of its posterior distribution fortunately has a simple form [Good, 1965, Bishop et al., 1975]. If the testing of a rule on N examples results in the classification counts $\mathbf{n} = (n_{ij})$ then the most probable values $\mathbf{p}^* = (p^*_{ij})$ for the classification probabilities assuming a Dirichlet prior with parameter $\mathbf{m}$ is

$$\mathbf{p}^* = \frac{\mathbf{n} + \mathbf{m}}{N + M}.$$

This posterior estimate of classification probabilities is intuitively appealing. The parameters $\mathbf{m}$ of the Dirichlet distribution can be seen as M extra sample counts which are added to the original N. Denoting the combination of these counts by $\mathbf{n}^* = \mathbf{n} + \mathbf{m}$ and the new total number of examples by $N^* = N + M$, the posterior classification probabilities are just the frequen-

tist estimates $\frac{\mathbf{n}^*}{N^*}$ of the combined counts. Viewing the Dirichlet parameters $\mathbf{m}$ as counts for M extra samples, they can be assumed to have been drawn independently with probabilities

$$\mathbf{q} = \begin{bmatrix} q_{++} & q_{+-} \\ q_{-+} & q_{--} \end{bmatrix} = \frac{1}{M} \begin{bmatrix} m_{++} & m_{+-} \\ m_{-+} & m_{--} \end{bmatrix}.$$

Collectively, these probabilities will be called a *CPM prior* and the parameter M the number of *virtual examples*.[1] In this case, the posterior probabilities are just a linear combination of the raw CPM $\mathbf{p}$ and the CPM prior $\mathbf{q}$. Specifically, each posterior probability

$$p^*_{ij} = \frac{N}{N+M} p_{ij} + \frac{M}{N+M} q_{ij}.$$

The total number of virtual examples M can be thought of here as a parameter that determines the relative importance of priors q_{ij} compared to the estimates p_{ij}. As M becomes much larger than the number of actual examples N the posterior estimates will be dragged towards the priors. Conversely, when $M = 0$ the posterior estimates have exactly the same value as their frequentist counterparts.

3.2.3. Class Skew. The left and right column totals in any CPM give the proportions of positive and negative class labels in the training examples from which the CPM was derived. When a raw CPM is combined with a CPM prior the column totals of the raw and posterior CPMs can differ. This difference in class distributions is known as *skew* and has been known to cause problems when it occurs between training and test sets in traditional, single-task learning [Weiss and Provost, 2001, §2.3]. A similar problem is faced in Section 3.3 below when CPM priors are constructed from support tasks which may have very different class distributions to the target task from which the raw CPM estimates are derived. This may or may not be a problem for inductive transfer depending on whether the target task's or support task's class distribution is believed to be more appropriate. In either case, it is worthwhile briefly outlining how skew can be corrected for here before delving into the details of how priors are constructed.

For a given CPM $\mathbf{p} = (p_{ij})$, the positive column total $\pi^+(\mathbf{p}) = p_{++} + p_{-+}$ and the negative column total $\pi^-(\mathbf{p}) = p_{+-} + p_{--}$ can be used to define a skew correction between two CPMs as follows.

[1] In the remainder of this thesis the distinction between a prior probability distribution $D(\mathbf{p}; \mathbf{m})$ and its parameters $\mathbf{m} = M\mathbf{q}$ will be deliberately blurred by calling $\mathbf{q}$ "priors" when there is an implied or irrelevant value for M.

DEFINITION 3.1. If $\mathbf{p}_1$ and $\mathbf{p}_2$ are two classification probability matrices, a *skew correction matrix* for a *source* $\mathbf{p}_2$ with respect to a *destination* $\mathbf{p}_1$ is defined to be

$$\sigma_{\mathbf{p}_1}(\mathbf{p}_2) = \begin{bmatrix} \frac{\pi^+(\mathbf{p}_1)}{\pi^+(\mathbf{p}_2)} & 0 \\ 0 & \frac{\pi^-(\mathbf{p}_1)}{\pi^-(\mathbf{p}_2)} \end{bmatrix}.$$

The subscript will often be dropped when the destination CPM is implied by context.

Skew correction matrices are defined so that the matrix multiplication of the source CPM by the skew correction will result in a new CPM that has the same column totals as the destination CPM. This is easily verified by carrying out the multiplication $\mathbf{p}_2\sigma_{\mathbf{p}_1}(\mathbf{p}_2)$. Using a raw CPM estimate $\mathbf{p}$ as the destination and a CPM prior $\mathbf{q}$ as a source, a skew corrected posterior CPM can be defined as follows so as to have the same column totals as $\mathbf{p}$.

DEFINITION 3.2. Given a raw CPM $\mathbf{p}$ and a CPM prior $\mathbf{q}$, the *skew corrected posterior CPM estimate* is defined to be

$$\mathbf{p}^* = \frac{N}{N+M}\mathbf{p} + \frac{M}{N+M}\mathbf{q}\sigma_{\mathbf{p}}(\mathbf{q}).$$

Unless stated otherwise, all classification prior matrices in the remainder of this chapter will be assumed to be skew corrected for the target task.

3.2.4. An Example Application. Blending guessed values with estimates derived from observed data using the methods described above is precisely the type of tweaking mechanism required to modify classification probabilities in light of extra knowledge that might be available during an inference task. In the opening example, one way of breaking the three-way tie for best rule was to assume that rules for Tina's tastes were most like Harry's and so those with tests for genre would be better predictors than those without. This assumption could be implemented using the prior CPMs shown in the second column of Table 3.1. The first column shows each rule's raw CPM estimates and the third column shows the CPMs resulting from the combination of the priors with the raw values when using three virtual examples, that is when $M = N = 3$. The priors, shown in the second column, are chosen to reflect the assumption that rules testing genre are more likely to be better predictors than those that do not test for genre. In each case, skew correction matrix is given by

$$\begin{bmatrix} \frac{1/3}{2/3} & 0 \\ 0 & \frac{2/3}{1/3} \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & 0 \\ 0 & 2 \end{bmatrix}.$$

TABLE 3.1. Example posterior CPM estimates for the rules GS, SL and SLGS from the introductory example. The parameter M used here was set to the number of examples $M = N = 3$ and the prior CPMs were skew corrected for the raw estimates.

	$\mathbf{p}$	$\mathbf{q}$	$\mathbf{p}^*$
GS	$\begin{bmatrix} \frac{1}{3} & 0 \\ 0 & \frac{2}{3} \end{bmatrix}$	$\begin{bmatrix} \frac{2}{3} & 0 \\ 0 & \frac{1}{3} \end{bmatrix}$	$\begin{bmatrix} \frac{1}{3} & 0 \\ 0 & \frac{2}{3} \end{bmatrix}$
SL	$\begin{bmatrix} \frac{1}{3} & 0 \\ 0 & \frac{2}{3} \end{bmatrix}$	$\begin{bmatrix} \frac{2}{3} & \frac{1}{3} \\ 0 & 0 \end{bmatrix}$	$\begin{bmatrix} \frac{1}{3} & \frac{1}{3} \\ 0 & \frac{1}{3} \end{bmatrix}$
SLGS	$\begin{bmatrix} \frac{1}{3} & 0 \\ 0 & \frac{2}{3} \end{bmatrix}$	$\begin{bmatrix} \frac{1}{3} & 0 \\ \frac{1}{3} & \frac{1}{3} \end{bmatrix}$	$\begin{bmatrix} \frac{1}{4} & 0 \\ \frac{1}{12} & \frac{2}{3} \end{bmatrix}$

Using this process the prior for the rule GS increases the estimated value of its true positive rate while the the priors for SL and SLGS respectively increase their false positive and false negative rates. It is important to note the reason that these three rules which have the same raw CPM estimates have different posterior CPMs. This difference only comes about because each rule was assigned a different prior. Were all the priors the same, all the posteriors would be the same. In order to effectively improve evaluation estimates a function is required that assigns possibly different prior CPMs to each rule being evaluated.

3.2.5. Prior CPM Functions. A *CPM function* is a name given to any procedure that when given a rule returns a two-by-two matrix with entries that sum to one. The evaluation of a rule's CPM on a set of training examples is one such procedure that fits this definition. So too is any procedure that returns a prior CPM when given a rule. The results of these functions can be combined for each rule using the method described for creating posterior CPM as follows.

DEFINITION 3.3. Let $r \in R$ be a rule and $\mathbf{p}_E(r)$ be the CPM for r evaluated on examples E. If $\mathbf{q}(r)$ is a prior for r, then the *posterior CPM* for the rule r is given by

$$\mathbf{p}^*_E(r) = \frac{N}{N+M}\mathbf{p}_E(r) + \frac{M}{N+M}\mathbf{q}(r)$$

where N is the total number of examples in E and M is the transfer parameter.

The question now is how to choose a classification prior function. One possibility is that the function $\mathbf{q}(r)$ be defined by an expert to best reflect any extra information regarding the candidate rules and the learning task at hand. For example, the prior CPM function below encodes an evaluation bias

that will prefer short rules over longer ones. Letting n denote the number of conditions in a rule, this function is

$$\mathbf{q}(r) = \frac{1}{2(n+1)} \begin{bmatrix} 1 & n \\ n & 1 \end{bmatrix}.$$

Using this function, longer rules will receive priors resulting in larger false positive and false negative rates.

Although hand-crafting evaluation biases through the use of prior CPM function is possible, this approach places an extra burden on anyone wishing to use a rule learning algorithm that uses adaptive classification priors. An alternative is to take a lead from Goodman's suggestion at the head of this chapter. By assuming that learning takes place in an environment of tasks, "past predictions and their successes and failures" can be used to learn the function which is to assign priors to rules. In terms of the motivating example, the question to be answered in the next section is: how can knowledge of Harry's tastes in books and the statement "Tina's tastes are similar to Harry's" be turned into a prior CPM function?

3.3. Rule Similarity and Inductive Transfer

The previous section described a mechanism with which classification probability estimates from limited training examples could be modified in order to implement an evaluation bias. This is done by using a function that assigns a CPM prior to each rule to be evaluated. A prior CPM can be updated using the available training examples to obtain posterior estimates of classification probabilities for that rule. Although prior functions can be constructed by hand, requiring this to be done for each learning task would place the same burden on an expert as selecting a language bias or search strategy. As argued earlier, this may not be easy for a domain expert who is not a machine learning expert.

The purpose of this section is to describe a method for creating CPM prior functions using examples from a support task and a definition of what it means for rules to be similar. If an domain expert believes that the rules similar to those that perform well on one task will also perform well on another then this extra information can be used to encode a CPM prior function that will express that belief.

3.3.1. Rule Similarity. Similarity is a relation that asserts that two objects are in some sense the same. In common usage the term usually implies

some sliding scale. Stringed instruments are all similar in a sense, but an electric guitar and an acoustic guitar would be regarded as more similar than an acoustic guitar and a harp for example. In mathematics similarity often does not have these shades. For example, two triangles are said to be similar if they have the same internal angles. In this case, two triangles are either similar or they are not.

For theoretical discussion it is easier to conceive of rule similarity as this second, discrete form of similarity. As a general notion, this kind of similarity can be formalised as an *equivalence relation*, that is a binary relation $\sim$ that is reflexive, symmetric and transitive. Hereafter, when two rules are said to be similar it will be with respect to some relation that satisfies the following definition.

DEFINITION 3.4. A *similarity relation* $\sim \subseteq R \times R$ over the set of rules R is a binary relation satisfying the following three conditions: *reflexivity* - every rule must be similar to itself ($r \sim r$) ; *symmetry* - if one rule is similar to another, the other must be similar to the first ($r_1 \sim r_2$ implies $r_2 \sim r_1$) ; *transitivity* - if a first rule is similar to a second, and the second is similar to a third then the first must be similar to the third ($r_1 \sim r_2$ and $r_2 \sim r_3$ implies $r_1 \sim r_3$).

One property of equivalence relations that will be used extensively is that they partition the set of objects they are defined over into equivalence classes. A similarity relation over rules therefore partitions a rule space into sets of similar rules.

DEFINITION 3.5. Given a set of rules R and a similarity relation $\sim$, each rule $r \in R$ is a member of exactly one *similarity class* $[r]$, which is defined to be

$$[r] = \{r' \in R \,:\, r' \sim r\},$$

that is, all the rules that are similar to r. This notation for similarity classes can also be used to denote the function $[\cdot] : R \to 2^R$ that maps rules to subsets of the rule space.

Thinking of a similarity relation as inducing both a partition of classes and as a mapping is useful when the the idea is extended to take into account target and support tasks that may use incompatible sets of candidate rules. This possibility is discussed below after an example of a similarity relation is defined for the Book World tasks.

3.3.2. An Example Similarity Relation. In the Book World example, there are 12 possible rules each uniquely determined by the book features they

test. These can be partitioned into four similarity classes C_0, C_s, C_g and C_{sg} based on whether a rule performs no test, a size test only, a genre test only or both tests respectively. While any other partition of the rules would have also served as similarity classes these four are arguably quite natural. The similarity relation corresponding to these classes define two rules to be similar if and only if they test the same book features.

Table 3.2 lists all 12 rules along with the similarity class each belongs to and their CPM estimates on the examples for Tina, Scott and Harry's reading preferences. The rules SL, GS and SLGS are highlighted in bold as the only three rules with no misclassifications on Tina's examples. Notice that each of these fall into a different similarity class, namely $C_s = [SL]$, $C_g = [GS]$ and $C_{sg} = [SLGS]$.

3.3.3. Similarity between Tasks with Different Representations. The representation language used to describe instances and rules in the Book World domain is the same across all three learning tasks. This means that a rule used to test whether Harry likes a book can be applied to examples of Scott's reading preferences and, importantly, the sets of similar rules on one task are identical to the similarity classes on another. This is not always the case in general as support and target tasks may be created at different times by different researchers. Extra instance features may be available on a target task that were not available on a support task, features that are deemed irrelevant might be removed from a task or feature names may change. For a theory of similarity-based transfer to be useful it must be applicable in cases such as these provided some similarity relation still exists between the tasks. Even if tasks share no common features, more abstract similarity relations can still be defined in terms of the number of conditions in a rule or the number of repeated conditions. In first-order rules, a similarity relation might test for whether constants or functions are present or check the number of distinct variables. This section briefly introduces some concepts relating to this more general take on similarity which are necessary to define how transfer can take place in these situations.

If the sets of rules used for support and target tasks are different, a similarity relation can still be defined over the union of the rules for both tasks. Figure 3.3 illustrates this situation between two tasks, one with rule space R_1 and the other with rules R_2. In this case, a similarity relation would need to be defined over $R = R_1 \cup R_2$ for any transfer to be possible. If there are rules in both spaces which are deemed similar by such a relation, a similarity map can be defined which, given a rule in R_1 will state which rules are similar to

TABLE 3.2. The rule space for the Book World tasks. Each row contains an identifying name for a rule, the test that rule performs, the similarity class it belongs to and its CPM on each of the three Book World tasks. The three rules highlighted in **bold** are those that best discriminate positive from negative examples of Tina's preferences.

Tests				**CPM Estimates**		
ID	**Size**	**Genre**	**Class**	**Tina**	**Scott**	**Harry**
T	-	-	C_0	$\begin{bmatrix} .33 & .67 \\ 0 & 0 \end{bmatrix}$	$\begin{bmatrix} .50 & .50 \\ 0 & 0 \end{bmatrix}$	$\begin{bmatrix} .33 & .67 \\ 0 & 0 \end{bmatrix}$
SS	small	-	C_s	$\begin{bmatrix} 0 & .67 \\ .33 & 0 \end{bmatrix}$	$\begin{bmatrix} .50 & 0 \\ 0 & .50 \end{bmatrix}$	$\begin{bmatrix} .17 & .33 \\ .17 & .33 \end{bmatrix}$
SL	**large**	**-**	C_s	$\begin{bmatrix} .33 & 0 \\ 0 & .67 \end{bmatrix}$	$\begin{bmatrix} 0 & .50 \\ .50 & 0 \end{bmatrix}$	$\begin{bmatrix} .17 & .33 \\ .17 & .33 \end{bmatrix}$
GS	**-**	**scifi**	C_g	$\begin{bmatrix} .33 & 0 \\ 0 & .67 \end{bmatrix}$	$\begin{bmatrix} .17 & .17 \\ .33 & .33 \end{bmatrix}$	$\begin{bmatrix} 0 & .33 \\ .33 & .33 \end{bmatrix}$
GH	-	horror	C_g	$\begin{bmatrix} 0 & .33 \\ .33 & .33 \end{bmatrix}$	$\begin{bmatrix} .17 & .17 \\ .33 & .33 \end{bmatrix}$	$\begin{bmatrix} .33 & 0 \\ 0 & .67 \end{bmatrix}$
GR	-	romance	C_g	$\begin{bmatrix} 0 & .33 \\ .33 & .33 \end{bmatrix}$	$\begin{bmatrix} .17 & .17 \\ .33 & .33 \end{bmatrix}$	$\begin{bmatrix} 0 & .33 \\ .33 & .33 \end{bmatrix}$
SSGS	small	scifi	C_{sg}	$\begin{bmatrix} 0 & 0 \\ .33 & .67 \end{bmatrix}$	$\begin{bmatrix} .17 & 0 \\ .33 & .50 \end{bmatrix}$	$\begin{bmatrix} 0 & .17 \\ .33 & .50 \end{bmatrix}$
SSGH	small	horror	C_{sg}	$\begin{bmatrix} 0 & .33 \\ .33 & .33 \end{bmatrix}$	$\begin{bmatrix} .17 & 0 \\ .33 & .50 \end{bmatrix}$	$\begin{bmatrix} .17 & 0 \\ .17 & .67 \end{bmatrix}$
SSGR	small	romance	C_{sg}	$\begin{bmatrix} 0 & .33 \\ .33 & .33 \end{bmatrix}$	$\begin{bmatrix} .17 & 0 \\ .33 & .50 \end{bmatrix}$	$\begin{bmatrix} 0 & .17 \\ .33 & .50 \end{bmatrix}$
SLGS	**large**	**scifi**	C_{sg}	$\begin{bmatrix} .33 & 0 \\ 0 & .67 \end{bmatrix}$	$\begin{bmatrix} 0 & .17 \\ .33 & .50 \end{bmatrix}$	$\begin{bmatrix} 0 & .17 \\ .33 & .50 \end{bmatrix}$
SLGH	large	horror	C_{sg}	$\begin{bmatrix} 0 & 0 \\ .33 & .67 \end{bmatrix}$	$\begin{bmatrix} 0 & .17 \\ .33 & .50 \end{bmatrix}$	$\begin{bmatrix} .17 & 0 \\ .17 & .67 \end{bmatrix}$
SLGR	large	romance	C_{sg}	$\begin{bmatrix} 0 & 0 \\ .33 & .67 \end{bmatrix}$	$\begin{bmatrix} 0 & .17 \\ .33 & .50 \end{bmatrix}$	$\begin{bmatrix} 0 & .17 \\ .33 & .50 \end{bmatrix}$

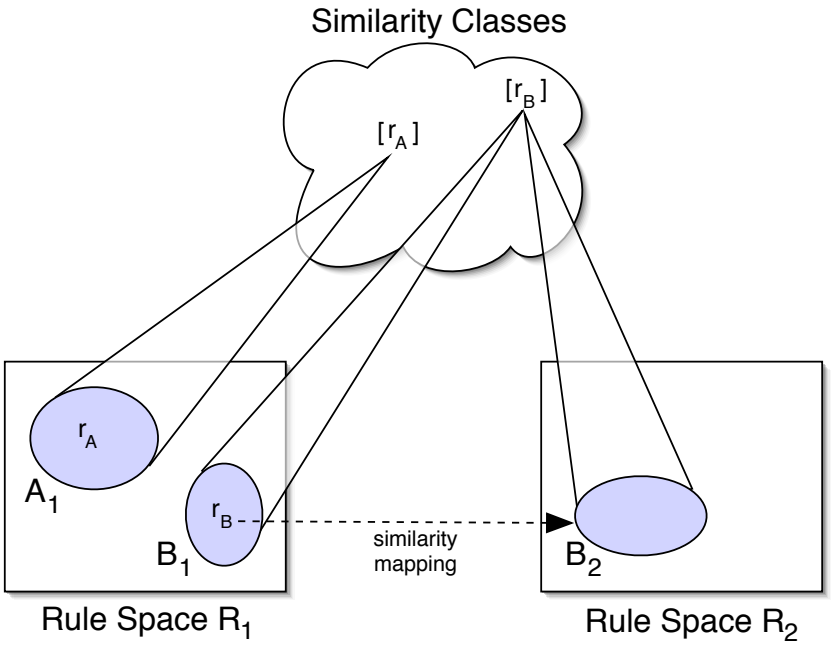


FIGURE 3.3. A mapping between two rule spaces via similarity classes. The rules in sets B_1 and B_2 are all in similarity class $[r_B]$. The rules in set A_1 are in the similarity class $[r_A]$ which has no corresponding rules in R_2.

it in R_2. For example, the rule $r_B \in B_1$ is similar to all of the rules in B_2 as they all share the similarity class $[r_B]$. In the case of $r_A \in A_1$ however, a corresponding set of rules in R_2 does not exist.

The two cases discussed in the above example are important for a general definition of similarity-based transfer and so will be made explicit through the following definition.

DEFINITION 3.6. Let R_1 and R_2 be sets of rules and $R = R_1 \cup R_2$. If$\sim$ is a similarity relation over R then the *kernel* $\ker(R_1, R_2)$ and the *domain* $\mathrm{dom}(R_1, R_2)$ for the relation are defined by

$$\ker(R_1, R_2) = \{r \in R_1 : [r] \cap R_2 = \emptyset\}$$
$$\mathrm{dom}(R_1, R_2) = \{r \in R_1 : [r] \cap R_2 \neq \emptyset\}.$$

The names 'kernel' and 'domain' were chosen deliberately to emphasise the idea of a similarity relation inducing a mapping from one set of rules to another. If the mapping is seen to be from R_1 to R_2 then the kernel contains those rules which do not have any corresponding rules in the image of the map whereas the domain contains all the rules which do. Since every rule must either have an image under this map or not, it should be clear that the rule

space being mapped from is completely covered by the union of the kernel and the domain.

3.3.4. Similarity-based Transfer. When a rule for a target task is in the domain of a similarity mapping there will exist similar rules to it in the support task. By examining the performance of these similar rules on the support task a CPM prior can be constructed that says something about the performance of rules like them. If an expert believes that similar rules perform similarly on related tasks this information may improve the original rule's estimate on the target task.

The key to doing this is to formalise what is meant by "similar rules performing similarly" and the method used here is defined probabilistically. Given a set of rules for a support task, it is possible to ask "what is the probability that a rule chosen at random from this set will correctly classify or misclassify a positive or negative example of the support task?". If similar rules on the target task do indeed perform similarly, these probabilities will make good choices for values in prior CPMs. Rules for the target task that belong to different similarity classes will have different prior CPMs and so the similarity relation can be used to define a prior CPM function for the target task. This approach is formalised in the following definition.

DEFINITION 3.7. Let T and S be a target and support task respectively and let R_S and R_T be the sets of candidate rules for the respective tasks. If $R = R_S \cup R_T$ has a similarity relation $\sim \subseteq R \times R$ defined over it then the *similarity-based prior CPM function* $\mathbf{q} = (q_{ij})$ is defined for each $r \in \mathrm{dom}(R_T, R_S)$ by its entries

$$q_{ij}(r) = \Pr_{\substack{(x,y)\in S \\ r'\in R_S}}(r'(x) = i, y = j | r' \sim r).$$

It is important to note that the above definition assumes that there is some distribution over examples and rules in R_S. The distribution over examples is not problematic as the examples can be assumed to be drawn identically and independently in both the target and support task. Also, any skew in the distribution of class values between the two tasks can be corrected for using the method described in Section 3.2.3 above. The interpretation of the distribution over rules in R_S is slightly more subtle and its discussion is left for Section 3.3.5 below.

Although the above definition is in terms of the probabilities of drawing and classifying examples according to a set of rules, each value in a prior CPM can also be expressed as an average. In particular, the value a prior CPM

function takes on a given rule can be written as the average CPM over all similar rules on the support task.

PROPOSITION 3.8. *For any given example set E, rule space R and similarity relation $\sim$, the prior CPM function $\mathbf{q}_E$ for E satisfies*

$$\mathbf{q}_E(r) = \mathop{\mathbb{E}}_{r' \in R} [\mathbf{p}_E(r') \mid r' \sim r]$$

where $\mathbf{p}_E$ is the raw CPM function for E.

PROOF. The proof follows from the definition of conditional probability and conditional expectation:

$$\begin{aligned} \Pr(r'(x) = i, y = j | r' \sim r) &= \frac{\Pr(r'(x) = i, y = j, r' \sim r)}{\Pr(r' \sim r)} \\ &= \frac{1}{\Pr(r' \sim r)} \int_{r' \sim r} \Pr(r'(x) = i, y = j) \\ &= \mathop{\mathbb{E}}_{r' \in R} [p_{ij}(r') \mid r' \sim r]\,. \end{aligned}$$

□

In the case where R is finite and there is an assumed uniform distribution over rules, each rule will have probability $\frac{1}{|R|}$ and so each similarity class $[r]$ will have probability $\Pr(r' \sim [r]) = \frac{|[r]|}{|R|}$ since $r' \sim r$ is equivalent to $r' \in [r]$. The expectation for this distribution is therefore a familiar looking average:

$$\mathbf{q}_E(r) = \frac{1}{|[r]|} \sum_{r' \in [r]} \mathbf{p}_E(r').$$

For this expected CPM to be used as a prior for CPM computed from training examples it is necessary that the matrices $\mathbf{q}_E(r)$ computed in the above manner themselves be CPMs. The following proposition ensures this.

PROPOSITION 3.9. *Each $\mathbf{q}_E(r)$ is a classification probability matrix for each r. That is, then entries $q_{ij}(r)$ in $\mathbf{q}_E(r)$ satisfy $\sum_{ij} q_{ij}(r) = 1$.*

PROOF. From Proposition 3.8 each $q_{ij}(r) = \mathbb{E}_R [p_{ij}(r') | r' \sim r]$. Summing these over the indices i and j gives

$$\begin{aligned} \sum_{i,j} q_{ij}(r) &= \sum_{i,j} \frac{1}{\Pr(r' \in [r])} \int_{r' \in [r]} p_{ij}(r') \\ &= \frac{1}{\Pr(r' \in [r])} \int_{r' \in [r]} \sum_{i,j} p_{ij}(r') \\ &= \frac{1}{\Pr(r' \in [r])} \int_{r' \in [r]} 1 \\ &= 1. \end{aligned}$$

Thus, $\mathbf{q}_E(r)$ is a CPM. □

Since the function defined above is guaranteed to return a CPM for every rule it is given that is in the domain of the similarity relation between a support and a target task, each CPM returned by this function can be combined with a raw CPM estimate and result in a valid posterior CPM. For those rules not in the similarity relation's domain, the posterior estimate for a rule's CPM can be taken to be the original raw CPM estimate.

DEFINITION 3.10. Let R_T be a set of rules for a target task T and R_S be a set of rules for a support concept S. If $\mathbf{q}_S$ is the similarity-based prior function for S then the *similarity-based posterior function* for task A using task B as support is defined to be

$$\mathbf{p}^*_{A,B}(r) = \begin{cases} \mathbf{p}_A(r) & \text{if } r \in \ker(R_A, R_B) \\ \frac{N}{N+M}\mathbf{p}_A(r) + \frac{M}{N+M}\mathbf{q}_B(r) & \text{otherwise.} \end{cases}$$

3.3.5. Rule Probability and Admissibility. A similarity-based prior CPM requires that some distribution be defined over the set of candidate rules in order to give meaning to the $\Pr(r' \in [r])$ term in the definition. There are two extreme approaches to doing this. The first way is to assume as little as possible about the rule distribution. For example, if the candidate space is finite, assume that all rules are equally probable. Alternatively, Occam's razor could be invoked and distribution that exponentially decays with the rule size could be used for infinite rule spaces. As will be shown in an example below, these sort of uninformative distributions are not ideal.

The other extreme is to let the rule distribution represent *a posteriori* probabilities. That is, the probability of each rule is the chance it will appear in the induced theory given all the examples for the task. This is also not satisfactory as it requires too much extra knowledge about the learning algorithm being used and its search and restriction biases. When the priors created from this bias are transferred to a new problem there is no guarantee the same algorithm or biases will be in use.

The compromise used here and in the implementation described in the next chapter is to choose a rule distribution that makes a very weak assumption about the learning process and also takes into account the available examples. The assumption is that the distribution should only assign non-zero probability to those rules that cover at least one positive example. Rules which do not satisfy what will be called the *admissibility* condition are rarely, if ever, considered for inclusion in induced theories regardless of evaluation function or search procedure. This is because of the asymmetry between the covering

of positive and negative examples by sets of rules. An instance is classified positive if any rule in a theory covers it while instances are only classified negative if no rule covers it. Admissibility takes this asymmetry into account by discounting rules that only cover negative examples during the construction of a classification prior.

DEFINITION 3.11. A rule r is said to be *admissible for the examples* E if it covers at least one positive example in E. That is, there must exist an example $(x, +) \in E$ such that $r(x) = +$. Given a set of rules R, the notation R_E will be used for the set of all rules in R which are admissible for E.

Why is a distribution that supports only admissible rules better than a completely uniform one? When a domain expert claims that the performance of rules on a target task will be related to the performance of similar rules on a support task it is unlikely he or she has in mind rules that are irrelevant for the tasks. Not taking this into account can have counter-intuitive effects as the following example shows.

Consider constructing a prior CPM function using the similarity classes for the Book World and the examples for Scott's reading preferences. Using the CPMs shown in Scott's column in Table 3.3 the prior CPM for rules in the similarity class C_s is given by

$$\begin{aligned}
\mathbf{q}_S(SL) = \mathbf{q}_S(SS) &= \frac{1}{|C_s|} \sum_{r' \in C_s} \mathbf{p}_S(r') \\
&= \frac{1}{2} \left(\begin{bmatrix} .50 & 0 \\ 0 & .50 \end{bmatrix} + \begin{bmatrix} 0 & .50 \\ .50 & 0 \end{bmatrix} \right) \\
&= \begin{bmatrix} .25 & .25 \\ .25 & .25 \end{bmatrix}.
\end{aligned}$$

Notably, each entry q_{ij} in this matrix is the product of its column's total $q_{i\cdot} = \sum_j q_{ij}$ and its row's total $q_{\cdot j} = \sum_i q_{ij}$. This means, for rules within the class C_s, the predicted and actual class labels of instances are independent. That is, the prior values impart no information regarding the true and predicted class labels. When a uniform distribution is assumed this independence will occur whenever a similarity class contains rules and their complements. Not allowing this type of similarity class is a strong restriction and one that can be avoided by using the admissibility condition.

The effects of using the admissibility condition is tested empirically in Chapter 5 and found to have a large impact on positive transfer effects when compared to inductive transfer that does not use it. For this reason and those

given above, the admissibility condition will be a default assumption when discussing transfer.

3.3.6. An Example of Transfer. All of the concepts introduced in this section can be applied to the Book World example to show how inductive transfer can be used to construct a prior CPM function. This is then used to compute the posterior CPM estimates for Tina's reading preferences when using Scott and Harry's examples as support.

The top row of Table 3.3 shows the raw CPM estimates for the three rules SL, GS and SLGS on the examples of Tina's preferences. The next two rows show the prior CPMs for each rule constructed from the admissible rules for Scott's examples and the posterior CPM estimate from the combination of the raw CPMs with these priors. Similarly, the forth and fifth row show the prior and posterior CPMs for each rule when Harry's examples are used as the support task.

When Scott's examples are used as support, the SL rule receives a posterior CPM with no misclassifications while the other rules are assigned CPMs with increased false positive or false negative scores. In contrast, it is rule GS that is assigned a perfect CPM when Harry's examples are used as support while the others have an increased proportion of misclassifications.

The bottom two rows of Table 3.3 show the true CPMs of each of the four rules in the case when Tina's reading preferences are described by the SL rule and in the case that they are described by the GS rule. In the latter case, it can be seen that the true CPMs for rules T and GS are estimated perfectly by the posterior CPM function $\mathbf{p}^*_{T,H}$ constructed using Harry's examples as a support task. While the posterior CPMs for the other two rules, SL and SLGS, are different from the true CPMs for those rules the posterior estimates are much closer to the true CPMs than the raw estimates.

If the rule GS best describes Tina's reading preferences then the use of Harry's examples as support improves the estimates of all four of the rules admissible for Tina's examples. In this situation, assuming that Harry and Tina's preferences are similar was able to be turned into a useful evaluation bias using the similarity-based method described above. Can this type of estimate improvement be formalised? If so, when can estimate improvements like this be guaranteed? That is, what relationships are required between the support and target task for this type of positive transfer to occur? Answering these questions is the focus of the next section.

TABLE 3.3. Raw, prior and posterior CPMs for rules that are admissible for Tina's reading preferences (T). Priors are created from the examples and admissible rules for the tasks for Scott (S) and Harry (H). Support task priors are combined with the raw estimates using $M = 3$ and skewed to match the target task's class distribution. The bottom two rows show the performance of each of the rules when the target concept for Tina is described by SL and GS.

	Rule $\in$ Similarity Class			
	T$\in C_0$	SL$\in C_s$	GS$\in C_g$	SLGS$\in C_{sg}$
$\mathbf{p}_T$	$\begin{bmatrix} .33 & .67 \\ 0 & 0 \end{bmatrix}$	$\begin{bmatrix} .33 & 0 \\ 0 & .67 \end{bmatrix}$	$\begin{bmatrix} .33 & 0 \\ 0 & .67 \end{bmatrix}$	$\begin{bmatrix} .33 & 0 \\ 0 & .67 \end{bmatrix}$
$\mathbf{q}_S$	$\begin{bmatrix} .33 & .67 \\ 0 & 0 \end{bmatrix}$	$\begin{bmatrix} .33 & 0 \\ 0 & .67 \end{bmatrix}$	$\begin{bmatrix} .11 & .22 \\ .22 & .44 \end{bmatrix}$	$\begin{bmatrix} .11 & 0 \\ .22 & .67 \end{bmatrix}$
$\mathbf{p}^*_{T,S}$	$\begin{bmatrix} .33 & .67 \\ 0 & 0 \end{bmatrix}$	$\begin{bmatrix} .33 & .0 \\ 0 & .67 \end{bmatrix}$	$\begin{bmatrix} .22 & .11 \\ .11 & .56 \end{bmatrix}$	$\begin{bmatrix} .22 & 0 \\ .11 & .67 \end{bmatrix}$
$\mathbf{q}_H$	$\begin{bmatrix} .33 & .67 \\ 0 & 0 \end{bmatrix}$	$\begin{bmatrix} .17 & .33 \\ .17 & .33 \end{bmatrix}$	$\begin{bmatrix} .33 & 0 \\ 0 & .67 \end{bmatrix}$	$\begin{bmatrix} .17 & 0 \\ .17 & .67 \end{bmatrix}$
$\mathbf{p}^*_{T,H}$	$\begin{bmatrix} .33 & .67 \\ 0 & 0 \end{bmatrix}$	$\begin{bmatrix} .25 & .17 \\ .08 & .50 \end{bmatrix}$	$\begin{bmatrix} .33 & 0 \\ 0 & .67 \end{bmatrix}$	$\begin{bmatrix} .25 & 0 \\ .08 & .67 \end{bmatrix}$
SL	$\begin{bmatrix} .50 & .50 \\ 0 & 0 \end{bmatrix}$	$\begin{bmatrix} .50 & 0 \\ 0 & .50 \end{bmatrix}$	$\begin{bmatrix} .17 & .17 \\ .33 & .33 \end{bmatrix}$	$\begin{bmatrix} .17 & 0 \\ .33 & .50 \end{bmatrix}$
GS	$\begin{bmatrix} .33 & .67 \\ 0 & 0 \end{bmatrix}$	$\begin{bmatrix} .17 & .33 \\ .17 & .33 \end{bmatrix}$	$\begin{bmatrix} .33 & 0 \\ 0 & .67 \end{bmatrix}$	$\begin{bmatrix} .17 & 0 \\ .17 & .67 \end{bmatrix}$

3.4. Sufficient Conditions for Successful Transfer

The purpose of this section is to formalise and answer the following question about similarity-based transfer. When will the use of similarity-based priors improve the quality of the classification probability estimates? To do this three new concepts are required. The first, *CPM function distance*, allows the term "improve" to be defined in terms of how much two CPM functions differ, on average, over a particular set of rules. The second concept, *task similarity*, uses this distance to formalise what it means to say that two tasks have similar rules. Finally, the third concept, *regularity*, defines how well the similarity relation partitions the rule space and captures features of rules that correlate well with good prediction.

With these concepts in place the transfer theorem states that using prior CPMs derived from a support task will reduce the distance between the raw

CPM estimates and their true values on a target task whenever the support and target tasks are similar and the similarity relation captures salient features about the rules. The actual proof of the transfer theorem does not rely on very sophisticated mathematics. It is a result of the definition of posterior CPMs, some simple properties of expectations and the triangle inequality. What is important here is that it lends some credence to the definitions of similarity and regularity of tasks that are introduced to define the sufficient conditions of the theorem.

3.4.1. CPM Function Distance and Error. It is difficult to make broad generalisations about the large variety of evaluation functions that can be used by a learner. Evaluation functions can include misclassification costs, penalties for complex rules or prefer precise rules over general ones. To prove anything about whether the use of prior CPMs improves the rankings of rules on training data would require a case by case analysis of classes of evaluation functions.

Another approach is to show that CPMs which take into account priors based on a support task will, on average, be better estimates of the true CPM than evaluations on the training data alone. In the best case, the augmented CPMs will match the true CPMs exactly, in which case the ranking on the training data will match the true ranking of the rules. In general, it can be expected that if the average CPM improves there may be a corresponding improvement in rule rankings for a large number of evaluation functions.

To answer the above question a measure of how close one CPM is to another is required. Keeping with the general approach taken so far, we will assume that there is some *norm* $\|\cdot\|$ defined over the set of possible CPMs. One simple choice would be to treat the CPM as a 4-dimensional vector and use the Euclidean distance as the norm. That is, let $\|\mathbf{p}\| = \sqrt{\sum_{ij} p_{ij}^2}$.

DEFINITION 3.12. The distance $\Delta_R(\mathbf{f}, \mathbf{g})$ between two CPM functions $\mathbf{f}$ and $\mathbf{g}$ over the set of rules R is defined to be

$$\Delta_R(\mathbf{f}, \mathbf{g}) = \mathop{\mathbb{E}}_{r \in R} \|\mathbf{f}(r)\sigma_{\mathbf{h}}(\mathbf{f}) - \mathbf{g}(r)\sigma_{\mathbf{h}}(\mathbf{g})\|$$

where $\mathbf{h}$ is a CPM with column totals both equal to $\frac{1}{2}$. The $\sigma_{\mathbf{h}}$ function is used to skew the original CPM functions $\mathbf{f}$ and $\mathbf{g}$ so that they both have column totals of $\frac{1}{2}$.

The reason the CPM values for each rule are skew corrected is to ensure that CPMs for different tasks can be compared. For example, the CPMs for the rule T on the tasks SL and GS shown in Table 3.3 both have zero entries

in their bottom rows as the rule T predicts all instances to be positive. The only reason the values in the top row are different is because of the different class distributions for SL and GS.

There are several properties of this distance measure that are needed for the proof of the transfer theorem. The first of these is that the CPM distance function is what is known as a *pseudo-metric*. The CPM distance is clearly non-negative and symmetric as a consequence of the properties of the norm it is based upon. The triangle inequality also holds due to the same property of the norm. The CPM distance is not quite a metric however since $\Delta_R(\mathbf{f},\mathbf{g}) = 0$ only implies that $\mathbf{f}$ and $\mathbf{g}$ are equal up to a skew correction. Two further properties of the CPM distance are required and are established in the following proposition.

PROPOSITION 3.13. *Let R be a set of rules and let ρ be a probability measure over R. Then for any CPM functions $\mathbf{f}$ and $\mathbf{g}$ and any $R_1, R_2 \subseteq R$ such that $R = R_1 \cup R_2$, the following properties of the average CPM distance will hold:*

(1) $\Delta_R(\mathbf{f},\mathbf{g}) = \frac{\Pr(R_1)}{\Pr(R)}\Delta_{R_1}(\mathbf{f},\mathbf{g}) + \frac{\Pr(R_2)}{\Pr(R)}\Delta_{R_2}(\mathbf{f},\mathbf{g}) - \frac{\Pr(I)}{\Pr(R)}\Delta_I(\mathbf{f},\mathbf{g})$ *where* $I = R_1 \cap R_2$.

(2) $\Delta_R(\alpha\mathbf{f}_1 + \beta\mathbf{f}_2, \alpha\mathbf{g}_1 + \beta\mathbf{g}_2) \leq \alpha\Delta_R(\mathbf{f}_1,\mathbf{g}_1) + \beta\Delta_R(\mathbf{f}_2\mathbf{g}_2)$ *where α and β be are any non-negative numbers.*

PROOF. Using the definition of Δ the right hand side of the first property can be expanded to

$$\begin{aligned}
& \frac{\Pr(R_1)}{\Pr(R)} \mathop{\mathbb{E}}_{R_1} \|\mathbf{f}-\mathbf{g}\| + \frac{\Pr(R_2)}{\Pr(R)} \mathop{\mathbb{E}}_{R_2} \|\mathbf{f}-\mathbf{g}\| - \frac{\Pr(I)}{\Pr(R)} \|\mathbf{f}-\mathbf{g}\| \\
= \;& \frac{1}{\Pr(R)} \int_{R_1} \|\mathbf{f}-\mathbf{g}\| + \frac{1}{\Pr(R)} \int_{R_2} \|\mathbf{f}-\mathbf{g}\| - \frac{1}{\Pr(R)} \int_{I} \|\mathbf{f}-\mathbf{g}\| \\
= \;& \frac{1}{\Pr(R)} \int_{R} \|\mathbf{f}-\mathbf{g}\| \\
= \;& \mathop{\mathbb{E}}_{R} \|\mathbf{f}-\mathbf{g}\| \\
= \;& \Delta_R(\mathbf{f},\mathbf{g})
\end{aligned}$$

since R_1 and R_2 cover R. The second property can be seen to hold by writing

$$\begin{aligned}
\Delta_R(\alpha\mathbf{f}_1 + \beta\mathbf{f}_2, \alpha\mathbf{g}_1 + \beta\mathbf{g}_2) &= \mathop{\mathbb{E}}_{R} \|(\alpha\mathbf{f}_1 + \beta\mathbf{f}_2) - (\alpha\mathbf{g}_1 + \beta\mathbf{g}_2)\| \\
&= \mathop{\mathbb{E}}_{R} \|(\alpha\mathbf{f}_1 - \alpha\mathbf{g}_1) + (\beta\mathbf{f}_2 - \beta\mathbf{g}_2)\| \\
&\leq \mathop{\mathbb{E}}_{R} \|(\alpha\mathbf{f}_1 - \alpha\mathbf{g}_1)\| + \mathop{\mathbb{E}}_{R} \|(\beta\mathbf{f}_2 - \beta\mathbf{g}_2)\| \\
&= \alpha\Delta_R(\mathbf{f}_1,\mathbf{g}_1) + \beta\Delta_R(\mathbf{f}_2,\mathbf{g}_2)
\end{aligned}$$

by the triangle inequality and the linearity of $\|\cdot\|$. □

A corollary of the above proposition is that for disjoint R_1 and R_2 the distance $\Delta_R(\mathbf{f}, \mathbf{g})$ is just the sum of the R_1 and R_2 weighted distances since the term involving their intersection is zero.

The distance between a true CPM and its estimates is a natural basis for the error of a CPM estimate function over a task. It measures by how much, on average, the classification probability estimates differ from their true values over the rule space for the task in question.

DEFINITION 3.14. The *CPM error* $\mathrm{err}_T(\mathbf{p})$ of a CPM estimate $\mathbf{p}$ for the concept T is the average error over all admissible rules for T of $\mathbf{p}$ and the true CPM $\mathbf{p}_T$. That is,

$$\mathrm{err}_T(\mathbf{p}) = \Delta_{R_T}(\mathbf{p}, \mathbf{p}_T).$$

3.4.2. CPM Error on the Book World Example. The above definition can be used to compute the CPM distance between the example estimates from the Book World example. To make the computation concrete, the Euclidean norm

$$\|\mathbf{p}\| = \sqrt{\sum_{i,j} p_{ij}^2}$$

will be used. Assuming that Tina's preferences are described by the rule GS in Table 3.3 the true CPM estimates can be denoted $\mathbf{p}_{GS}(r)$ and compared to the raw CPM estimates $\mathbf{p}_T(r)$. In this case the CPM function distance over the six admissible rules for the concept GS, namely $R = \{T, SL, SS, GS, SSGS, SLGS\}$ for the target task is given by

$$\begin{aligned}
\Delta_R(\mathbf{p}_{GS}, \mathbf{p}_T) &= \frac{1}{6}(2\left\|\begin{array}{cc} -.25 & .25 \\ .25 & -.25 \end{array}\right\| + 2\left\|\begin{array}{cc} -.25 & 0 \\ .25 & 0 \end{array}\right\|) \\
&= \frac{1}{6}(1.0 + 0.708) \\
&= 0.285.
\end{aligned}$$

Note that the CPMs are skewed to have column totals of a half before their difference is taken. In comparison, the distance between the posterior estimates and the true estimates is given by

$$\begin{aligned}
\Delta_R(\mathbf{p}_{GS}, \mathbf{p}^*_{T,H}) &= \frac{1}{6}(2\left\|\begin{array}{cc} -.125 & .125 \\ .125 & -.125 \end{array}\right\| + 2\left\|\begin{array}{cc} -.125 & 0 \\ .125 & 0 \end{array}\right\|) \\
&= \frac{1}{6}(0.5 + 0.354) \\
&= 0.142
\end{aligned}$$

which is half the error of the raw estimate.

The question of when this estimate improvement occurs in general can be framed in terms of CPM error. Suppose that A and B are a target concept and a support concept respectively. A rule learning algorithm is given access to some limited subset of examples $\bar{A} \subseteq A$ as the target task and all the examples for B as the support task. Under what conditions will the posterior CPM estimate $\mathbf{p}^*_{\bar{A},B}$ be better, on average, than the raw CPM estimate $\mathbf{p}_{\bar{A}}$? That is, when will

$$\mathrm{err}_A(\mathbf{p}^*_{\bar{A},B}) \leq \mathrm{err}_A(\mathbf{p}_{\bar{A}})?$$

Two new measurements, similarity and regularity, are required to answer this question.

3.4.3. Similarity and Regularity. The conditions required to guarantee an improvement of posterior CPM estimates over raw CPM estimates are stated in terms of the similarity of the support and target tasks and the regularity of the target task. To be more precise, a bound is required on the *dissimilarity* of the two tasks and the *irregularity* of rules within each similarity class when evaluated on the active task.

The dissimilarity of two tasks is defined with respect to a similarity relation between rules for the tasks. It is essentially a measure of how much the tasks' CPM priors vary across all of the similarity classes.

Definition 3.15. Given a similarity relation $\sim$ defining similarity classes $[r]$ over a rule set $R \subseteq \mathrm{dom}(R_A, R_B) \cup \mathrm{dom}(R_B, R_A)$ the *dissimilarity* of tasks A and B is defined to be

$$\delta_R(A, B) = \Delta_R(\mathbf{q}_A, \mathbf{q}_B)$$

where R_A and R_B are the rule spaces for task A and B respectively. This quantity is just the average difference between the similarity-based prior functions $\mathbf{q}_A$ and $\mathbf{q}_B$ taken over all the similarity classes for R.

The reason the rule set R must be in the domain of either the mapping from R_A to R_B or *vice versa* is to ensure that both prior CPM functions are defined for every rule in R.

In terms of the Book World example, the dissimilarity between two tasks is computed by comparing the prior CPMs for each task over the four similarity classes C_0, C_s, C_g and C_{gs}. Using Table 3.3 as reference again, the dissimilarity between the learning tasks for Scott and Harry's preferences can be determined by noticing that of the four rules used to form the columns of the table each fall in a different similarity class. Averaging the differences of these CPMs over all the admissible rules for Harry's examples yields a value of 0.294. This

is also the average CPM difference between Tina and Scott's tasks assuming that the best rule to describe Tina's preferences is GS whereas the dissimilarity between Tina and Harry's tasks under this assumption is zero. These values sit well with the fact that GS and GH (the rule describing Harry's examples) are both in the same similarity class but GS and SS (the rule describing Scott's examples) are not.

Dissimilarity describes how different the the average CPM values in each similarity class are between two tasks. However, knowing that the dissimilarity between two tasks is low is not enough to ensure that incorporating these values as priors will reduce CPM error. It is possible that rules within a similarity class could have wildly different true CPM values on a support and target task but exactly the same mean. What is needed is some measure of the variance of CPM values for a task. This variance will be called the *irregularity* of a task and is defined as follows.

DEFINITION 3.16. The *irregularity* $\gamma_R(A)$ of a similarity relation $\sim$ with respect to the task A over the rules in R is a measure of how much rules within each similarity class differ from the similarity-based average for that class. Specifically,

$$\gamma_R(A) = \Delta_R(\mathbf{p}_A, \mathbf{q}_A).$$

The irregularity of all three tasks in the Book World domain is zero which means that within each similarity class true rule CPMs are all identical. This can be better understood by considering the CPM for the admissible rules for Harry's task. For this task, the class C_0 only contains the rule T and so trivially has zero variance. The class C_s has two admissible rules, each of which cover one positive and two negative examples. Similarly, the class C_{sg} contains three admissible rules, each covering a single positive example. The only admissible rule in the class C_g is GH and so this class also has zero variance.

Dissimilarity and irregularity together can be used to compare the CPM priors for one task with the actual CPMs of another task. This comparison is needed to prove the central theorem of this chapter and so it is stated below in full.

LEMMA 3.17. *Given a target task A and a support task B, if $R \subseteq R_A$ is a subset of the admissible rules for A then the CPM function distance between $\mathbf{q}_B$ and $\mathbf{p}_A$ satisfies*

$$\Delta_R(\mathbf{q}_B, \mathbf{p}_A) \leq \delta_R(A, B) + \gamma_R(A).$$

PROOF. The triangle inequality for Δ implies that

$$\Delta_R(\mathbf{q}_B, \mathbf{p}_A) \leq \Delta_R(\mathbf{q}_B, \mathbf{q}_A) + \Delta_R(\mathbf{q}_A, \mathbf{p}_A) = \delta_R(A, B) + \gamma_R(A)$$

by the definitions of the dissimilarity and irregularity. □

3.4.4. The Transfer Theorem. Enough conceptual machinery is available to state and prove the main theorem of this chapter. In simple terms, this theorem states the conditions which guarantee when using similarity-based classification probability estimates will be closer to the true probabilities than raw estimates. These conditions require that the dissimilarity between the support and target task and irregularity of the target task be less than the error of the raw estimates. This is stated more formally below.

THEOREM 3.18. *Let A be a target concept and B be a support concept with sets of rules R_A and R_B respectively with some similarity relation defined over their union. Suppose that $\bar{A}$ is some subset of the examples of A and let $\mathbf{p}_{\bar{A}}$ be the raw CPM estimate function for concept A based on the examples in $\bar{A}$. Let $\mathbf{p}^* = \mathbf{p}_{\bar{A},B}$ be the posterior CPM function for concept A using examples $\bar{A}$ and the set of examples B as a support task. The average CPM errors on concept A for these two CPM functions satisfy*

$$\text{err}_A(\mathbf{p}^*) \leq \text{err}_A(\mathbf{p}_{\bar{A}}) - \frac{M}{N+M}\frac{\Pr(D)}{\Pr(R_A)}\epsilon \tag{3.1}$$

where $\epsilon \geq 0$ is the difference between the raw CPM error over the rules in $D = \text{dom}(R_A, R_B)$ and the sum of the dissimilarity between A and B and irregularity of A. That is,

$$\epsilon = \Delta_D(\mathbf{p}_{\bar{A}}, \mathbf{p}_A) - (\delta_D(A, B) + \gamma_D(A)).$$

PROOF. Let $K = \ker(R_A, R_B)$ be the kernel of the prior CPM function $\mathbf{q}_B$. By definition, the intersection of the kernel and domain D is empty and the rule space R_A for the target task satisfies $R_A = K \cup D$. The average CPM error of $\mathbf{p}^*$ is the average CPM distance between that estimate and the true CPM function $\mathbf{p}_A$ over the admissible rules for A. Since the posterior CPM function is defined to be $\mathbf{p}_{\bar{A}}$ on the kernel of the prior function the application of Proposition 3.13 implies that

$$\text{err}_A(\mathbf{p}^*) = \frac{\Pr(D)}{\Pr(R_A)}\Delta_D(\mathbf{p}^*, \mathbf{p}_A) + \frac{\Pr(K)}{\Pr(R_A)}\Delta_K(\mathbf{p}_{\bar{A}}, \mathbf{p}_A). \tag{3.2}$$

For the rules in the domain D of the prior CPM function the definition of $\mathbf{p}^*$ means that the first term in the above equation can be expanded to

$$\begin{aligned}
\Delta_D(\mathbf{p}^*, \mathbf{p}_A) &= \Delta_D\left(\frac{N}{N+M}\mathbf{p}_{\bar{A}} + \frac{M}{N+M}\mathbf{q}_B, \mathbf{p}_A\right) \\
&= \Delta_D\left(\frac{N}{N+M}\mathbf{p}_{\bar{A}} + \frac{M}{N+M}\mathbf{q}_B, \frac{N}{N+M}\mathbf{p}_A + \frac{M}{N+M}\mathbf{p}_A\right) \\
&\leq \frac{N}{N+M}\Delta_D(\mathbf{p}_{\bar{A}}, \mathbf{p}_A) + \frac{M}{N+M}\Delta_D(\mathbf{q}_B, \mathbf{p}_A)
\end{aligned}$$

with this final inequality due to the linearity property of Lemma 3.13. Substituting this back into equation 3.2 and using the fact that

$$(\frac{N}{N+M} + \frac{M}{N+M})\Delta_K(\mathbf{p}_{\bar{A}}, \mathbf{p}_A) = \Delta_K(\mathbf{p}_{\bar{A}}, \mathbf{p}_A)$$

some rearrangement of the terms yields

$$\begin{aligned}
\mathrm{err}_A(\mathbf{p}^*) &\leq \frac{\Pr(D)}{\Pr(R_A)}\left[\frac{N}{N+M}\Delta_D(\mathbf{p}_{\bar{A}}, \mathbf{p}_A) + \frac{M}{N+M}\Delta_D(\mathbf{q}_B, \mathbf{p}_A)\right] \\
&\quad + \frac{\Pr(K)}{\Pr(R_A)}\left[\frac{N}{N+M}\Delta_K(\mathbf{p}_{\bar{A}}, \mathbf{p}_A) + \frac{M}{N+M}\Delta_K(\mathbf{p}_{\bar{A}}, \mathbf{p}_A)\right] \\
&= \frac{N}{N+M}\left[\frac{\Pr(D)}{\Pr(R_A)}\Delta_D(\mathbf{p}_{\bar{A}}, \mathbf{p}_A) + \frac{\Pr(K)}{\Pr(R_A)}\Delta_K(\mathbf{p}_{\bar{A}}, \mathbf{p}_A)\right] \\
&\quad + \frac{M}{N+M}\left[\frac{\Pr(D)}{\Pr(R_A)}\Delta_D(\mathbf{q}_B, \mathbf{p}_A) + \frac{\Pr(K)}{\Pr(R_A)}\Delta_K(\mathbf{p}_{\bar{A}}, \mathbf{p}_A)\right]. \qquad (3.3)
\end{aligned}$$

The term multiplied by $\frac{N}{N+M}$ in the final sum is equal to $\Delta_R(\mathbf{p}_{\bar{A}}, \mathbf{p}_A)$ by the application of Lemma 3.13 again, which, by definition, is $\mathrm{err}_A(\mathbf{p}_{\bar{A}})$. Inside the term multiplied by $\frac{M}{N+M}$ the value of$\Delta_D(\mathbf{q}_B, \mathbf{p}_A)$ cannot be greater than the sum of the dissimilarity of A and B and irregularity of B over I by Lemma 3.17. The conditions of this theorem require that this same sum be bounded by $\Delta_D(\mathbf{p}_{\bar{A}}, \mathbf{p}_A)$, which together implies

$$\Delta_D(\mathbf{q}_B, \mathbf{p}_A) \leq \delta_D(A, B) + \gamma_D(A) = \Delta_D(\mathbf{p}_{\bar{A}}, \mathbf{p}_A) - \epsilon.$$

Substituting these two results back into the terms in 3.3 gives

$$\begin{aligned}
\mathrm{err}_A(\mathbf{p}^*) &\leq \frac{N}{N+M}\,\mathrm{err}_A(\mathbf{p}_{\bar{A}}) \\
&\quad + \frac{M}{N+M}\left[\frac{\Pr(D)}{\Pr(R_A)}\left(\Delta_D(\mathbf{p}_{\bar{A}}, \mathbf{p}_A) - \epsilon\right) + \frac{\Pr(K)}{\Pr(R_A)}\Delta_K(\mathbf{p}_{\bar{A}}, \mathbf{p}_A)\right] \\
&= \frac{N}{N+M}\,\mathrm{err}_A(\mathbf{p}_{\bar{A}}) + \frac{M}{N+M}\,\mathrm{err}_A(\mathbf{p}_{\bar{A}}) - \frac{M}{N+M}\frac{\Pr(D)}{\Pr(R_A)}\epsilon \\
&= \mathrm{err}_A(\mathbf{p}_{\bar{A}}) - \frac{M}{N+M}\frac{\Pr(D)}{\Pr(R_A)}\epsilon
\end{aligned}$$

where the second to last step is once again due to Lemma 3.13. Since $\epsilon \geq 0$ this proves that the posterior CPM estimate has a lower average CPM error than the raw estimate. □

The amount that the posterior estimate will reduce the CPM error compared to the raw CPM estimates is given the term

$$\frac{M}{N+M}\frac{\Pr(D)}{\Pr(R_A)}\epsilon.$$

The maximum value this difference can take is ϵ. This occurs when M is very large relative to N and the domain of the similarity map from R_A to R_B is the whole of R_A. Conversely, if M is chosen to be small relative to N or there are a small proportion of rules in R_A that are similar to rules in R_B then the guaranteed reduction in average CPM error will be limited.

It is trivial to check that conditions and implications of the theorem hold for the calculations on the Book World example. As determined above, the dissimilarity between Harry's reading preferences and Tina's preferences, assuming they are actually described by the rule GS, is zero. So too is the irregularity of the rules on Tina's learning task. The CPM error of the raw estimates based on the three examples of Tina's preferences is approximately 0.285. This is greater than zero and therefore greater than the sum of the dissimilarity and irregularity just described. The implication of the theorem guarantees that the posterior CPM estimates using Harry's example as support must be less than the raw CPM estimates regardless of the choice of M parameter. This can be checked explicitly for the case when $M = 3$ as the posterior error is 0.142 compared to the raw error of 0.285.

3.4.5. Discussion. The main purpose of the transfer theorem is to show that the dissimilarity and irregularity measures for tasks are meaningful when using those tasks for similarity-based transfer. The theorem shows that low values for these quantities will ensure that the posterior estimates will outperform a raw estimate in terms of reducing average CPM error. It is important to note that average CPM error is a fairly rough measure of improvement. A posterior CPM error that is lower than a raw CPM error does not necessarily mean that the CPM error for any particular rule is better. This makes it difficult to say anything concrete about whether using such an estimate in a learning system will result in a theory containing rules with higher accuracies. To properly analyse when more accurate rules will be induced by a system much more information is required about the evaluation function used and the

other types of bias in the system. The analysis of CPM error in the above theorem was a way to say something weaker about CPM estimates that applies to a broader range of learning algorithms.

There are several other practical short-comings to the theory developed in this chapter. The first and most important is that the key measurements, dissimilarity and irregularity, are defined in relation to the true CPM values for the target and support tasks. In practice these are not known ahead of time so it is not possible to use these measurements to select support tasks that will guarantee estimation improvements. It may be possible however to estimate these measurements from available training data and use these quantities to guide support task selection. As the focus of this dissertation is on the inductive transfer and not automatic bias selection this avenue of investigation is left as future work.

The other practical hurdle to using posterior CPM estimates is that they can potentially be very expensive to compute. In the Book World example there are only 12 possible rules spanning four similarity classes which means that prior CPMs can be computed with little computational expense. On more practical problems the rule spaces under consideration may contain hundreds of thousands of rules. To compute a single prior CPM for a rule every rule similar to it must be determined and tested on the support task. Task irregularity is also an issue since, as the transfer theorem shows, this is part of what determines whether a posterior estimate is of any value. For task irregularity to be low when rule spaces are large and complex a similarity relation will need to define many classes. This potentially places the burden of defining a complex similarity relation on the user of this system. To rectify some of these practical problems, a refinement of posterior CPM estimation is proposed in the next section.

3.5. Description-based Transfer

The previous section described a mechanism based on rule similarity that can be used to transfer an evaluation bias, in the form of a CPM prior function, from a support task to a target task. The earlier discussion used a very general definition of similarity as an equivalence relation over rules. In philosophy and psychology, it has been argued that "for similarity to be a useful construct, one must be able to specify the ways or respects in which two things are similar" [Medin et al., 1993] and that similarity without reference to these respects is "invidious, insidious, a pretender, an impostor, a quack" [Goodman, 1972].

This section attempts to turn the theory developed above into a more useful construct through the use of a less abstract type of similarity relation which

compares *descriptions* of rules. A rule description is a vector of attribute values which capture syntactic features of a rule. The rule features, or *descriptors*, are the respects by which rules are compared. Assuming that descriptors are independent this decomposition of the similarity relation allows for a much more tractable approach to computing prior CPM functions. It also paves the way for the descriptor templating system described in the next chapter. This allows a user of such a system to quickly and easily define a similarity relation.

3.5.1. Rule Descriptions. A descriptor is an attribute that provides some information about properties of a rule such as its length, the tests it performs or the constants it contains. The features used to define similarity classes for the Book World example are instances of descriptors. One of these features determines whether or not a rule has a condition that tests the genre of a book. The other feature checks for whether the size of a book is tested by a rule. Both of these fit the following definition of a descriptor, as does any function that maps a rule to a unique value.

DEFINITION 3.19. If R is a set of rules a *descriptor* is any function $d : R \to V_d$. The set V_d are called *descriptor values* for d. A collection of descriptors $D = \{d_1, \ldots, d_K\}$ with the same domain R is called a *descriptor set* and can be used to construct a *description function* $\mathbf{d} : R \to V_{d_1} \times \cdots \times V_{d_K}$ which maps each rule $r \in R$ to its *description* $\mathbf{d}(r) = (d_1(r), \ldots, d_K(r))$ which is, essentially, an attribute-value vector for rules.

When two or more rules share a common set of features we can say they are similar with respect to those features. Given a set of descriptors D and its associated description function $\mathbf{d}$ we can define a *description similarity* relation $r_1 \sim_{\mathbf{d}} r_2$ that holds for rules $r_1, r_2 \in R$ if and only if $\mathbf{d}(r_1) = \mathbf{d}(r_2)$. This relation is a similarity relation since reflexivity, symmetry and transitivity are all inherited from the equality relation used to compare descriptions. The equivalence classes that arise from this definition of description similarity can be identified by the description shared by its members.

The use of rule features is not a new idea. The evaluation functions discussed in Section 2.3.5 of the previous chapter take features of rules, such as their length and number of free variables, into account when making an assessment of rule quality[2]. The term "descriptors" for rule features was borrowed from Bensusan [1999, §4.5] who uses it to denote features of decision trees such as the "number of nodes per attribute", "maximum tree depth", "tree shape", "homogeneity" and "imbalance". The definition of descriptors proposed here is

[2]For more examples see the discussion of complexity estimates in [Fürnkranz, 1999, §4.3.2]

TABLE 3.4. The description classes defined by the Book World example description function given in the text, their corresponding similarity task and the rules they contain.

Description	Similarity Class	Rules
$[n, n]$	C_0	T
$[y, n]$	C_s	SS, SL
$[n, y]$	C_g	GS, GH, GR
$[y, y]$	C_{sg}	SSGS, SSGH, SSGR, SLGS, SLGH, SLGR

intended to capture the general idea of extracting pertinent features from the rule or model under consideration.

3.5.2. Description Examples. The similarity classes used in the Book World example can be viewed as descriptions by defining two boolean descriptors which map rules onto the values y or n. The first descriptor d_s will return the value y if the rule it is applied to contains a condition which tests the size of a book and will return n otherwise. Similarly, the second descriptor d_g will return y or n depending on whether or not a rule tests the genre attribute of a book. The rule description function in this case can be defined to be $\mathbf{d}(r) = [d_s(r), d_g(r)]$. Each of the four possible descriptions returned by this function correspond to one of the similarity classes described in Section 3.3.2 above. Table 3.4 lists all four descriptions along with their corresponding similarity class and the rules contained within them.

Defining similarity classes in this way has several advantages. Firstly, creating tests for two conditions separately is easier than attempting to define a function that classifies a rule into one of the four possible description classes. Each time a new binary descriptor is added the number of possible similarity classes is doubled. This means a fine-grained collection of similarity classes can be defined with a relatively small number of descriptors. Secondly, similarity classes are easier to understand as an attribute vector than as abstract equivalence classes. Finally, and most importantly from an implementation perspective, descriptions (unlike general similarity classes) can be decomposed into their constituent descriptors which allows for an efficient method of computing CPM priors.

3.5.3. Decomposing Description Similarity. The following derivation underlies an efficient approach to computing CPM priors. It begins by applying Definition 3.7 to the description-based similarity relation just described:

$$q_{ij}(r) = \Pr_{\substack{(x,y)\in E \\ r'\in R}}(r(x) = i, y = j | r' \in \mathbf{d}(r)).$$

Using $\Pr(i,j|\mathbf{d}(r))$ as shorthand for this probability, Bayes' identity allows each classification prior to be expressed as

$$q_{ij}(r) = \Pr(i,j|\mathbf{d}(r)) = \frac{\Pr(i,j)}{\Pr(\mathbf{d}(r))}\Pr(\mathbf{d}(r)|i,j).$$

The description $\mathbf{d}(r) = (d_1(r), \ldots, d_n(r))$ in the above equation is a vector of descriptor values. By naïvely assuming that these values are all independent, the final term on the right hand side can be expanded as the product of the probabilities $\Pr(d_k(r)|i,j)$ resulting in what will be called the *naïve CPM prior*.

DEFINITION 3.20. The *naïve classification probability matrix* for the rule r given a descriptor set $D = \{d_k\}_{k=1}^n$ is the classification probability matrix $\mathbf{q}(r)$ with entries

$$q_{ij}(r) = \frac{\Pr(i,j)}{\Pr(\mathbf{d}(r))}\prod_{k=1}^{n}\Pr(d_k(r)|i,j). \tag{3.4}$$

All the probabilities in this equation can be derived from the probabilities $\Pr(i,j,d_k(r))$ which will be called the *ij descriptor frequency of d_k for the value $v = d_k(r)$* and will be denoted $\phi_{ij}[d_k, v]$. The values for $\Pr(i,j)$ will be called the *class priors*. The efficient estimation of these probabilities is the topic of the next chapter.

Simplifications like this one are not uncommon in machine learning. Even when such a naïve Bayes assumption is known not to hold techniques like this can still lead to reasonable results [Domingos and Pazzani, 1997]. Decomposing a description function into n descriptor functions in this way can be seen as creating n different similarity partitions of the rule space. Information regarding CPM priors is computed for each of these partitions and then combined by overlaying the partitions. The CPM prior at each intersection of these overlaid classes is then calculated as the product of the descriptor frequencies for the separate layers.

In terms of the theory developed earlier in this chapter this decomposition has some consequences. The transfer theorem will hold for each of the descriptors considered alone as a similarity relation. If the descriptors really are independent the naïve prior will be identical to the original one and so the theorem will hold for their combination. However, it is less clear whether any guarantees about reduction in CPM error can be given for the naïve CPM priors when the independence assumption does not hold. The impact of this assumption and decomposition is examined empirically in Chapter 5. Modifying the theory developed in this chapter to take into account description decomposition is left as an avenue for future research.

3.6. Discussion and Related Work

The transfer of evaluation bias based presented in this chapter is, to the author's knowledge, a previously unexamined form of inductive transfer. In its general form this new approach allows for the explicit inclusion of expert knowledge at the level of hypothesis evaluation. This is achieved through a Bayesian method of combining raw classification probability estimates with prior values based on extra-evidential properties of the hypothesis being evaluated. Constructing these priors by hand would be difficult in general so the method proposed here includes a method for defining these based on examples from a support task and a relation that defines the respects in which hypotheses are thought to be similar.

As a whole, this approach is novel but its development has been strongly influenced by other work which falls under the overlapping banners of "learning to learn", "meta-learning", "bias learning", "multitask learning", "transfer learning" and "empirical Bayes". It has also drawn upon research that advocates the use of Bayesian methods for single-task learning. This final section compares similarity-based transfer of evaluation bias to the relevant parts of the existing work before closing with a discussion of its parameters.

3.6.1. Classification Priors. The use of Bayesian priors when estimating classification probabilities is not without precedent. The definitions and nomenclature used here were deliberately chosen to reflect those used by Cestnik [1990] in his paper proposing the use of the m-estimate for avoiding zero estimates of probabilities in classifiers. In that paper, $m-$estimates of precision are given by $p^* = \frac{n_{++}+mq}{n_{++}+n_{+-}+m}$ where the prior q is taken to be the proportion of positive examples in the training set and the value m is a parameter analogous to the M used here. This now commonly used technique has been recently generalised by Fürnkranz and Flach [2003] as part of their study of evaluation metrics. The prior used in the generalised estimate is set by an expert to reflect knowledge about class distribution or misclassification costs.

The main difference between the m-estimate and the CPM estimates introduced in this chapter is that the original and generalised m-estimates are used as substitute evaluation metrics for the precision metric. In contrast, CPM estimates can be used as input to any standard evaluation. With an appropriate choice of priors q_{ij} for a CPM estimate the m-estimate can be derived by using the posterior estimates p^*_{++} and p^*_{+-} to compute a classifier's precision. It is also important to note that, unlike the priors for CPM estimation which can vary from rule to rule in order to break ties, the m-estimate prior is kept fixed when it is used for evaluation.

Other Bayesian approaches to rule learning, such as LIME [McCreath and Sharma, 1998, McCreath, 1999] and the positive-only learning used in the ILP system PROGOL [Muggleton, 1996], use priors that vary from rule to rule. These systems randomly generate unclassified instances to obtain better estimates of the cover or generality of a rule. These estimates are then used as part of the system's evaluation heuristics to penalise over-general rules, forcing specialisation in the absence of negative examples. These differ from the present work in that they focus on learning from single tasks and therefore do not make use of support tasks. An interesting avenue for future research would be to investigate whether CPM priors learnt on a support task with both positive and negative example would allow for positive-only learning on a target task with no negative examples. Small general rules for these tasks will inadvertently have large priors for their false positive rate while more specific ones will have smaller prior false positive rates thereby biasing a learner to specialise.

Although not directly related, Cussens [1998] has investigated the use of *rule* priors in a first-order context with applications to part-of-speech tagging. In his approach the rule priors are explicitly provided by an expert through a Prolog program. The training examples are then used to determine the posterior probabilities and clauses are selected using a MAP algorithm. Priors for rule classification probabilities are not the same as priors over rules although the links between the two deserve further investigation.

At first glance, modifying CPM estimates with a prior may bear a resemblance to techniques such as boosting [Freund and Schapire, 1995] or example weighting for cost-sensitive learning [Zadrozny et al., 2003]. Although these methods and the use of CPM priors modify the way a learner evaluates models against training instances the approaches are not equivalent. To see this it is enough to notice that a CPM prior may introduce a non-zero estimate for p_{+-} for some rule even though that rule does not falsely predict a positive label for any of the negative training instances. Therefore, the effect of that prior on the rule's evaluation cannot be achieved by a reweighting of the training examples.

3.6.2. Empirical Bayes. In statistics, choosing values for model priors based on data is known as "empirical Bayes" [Carlin and Louis, 2000, Robbins, 1956]. In standard Bayesian inference the probability $\Pr(\theta|x)$ for a model θ given some observations x is dependant on the prior probability $\Pr(\theta)$. In empirical Bayes it is assumed that this prior over θ is controlled by some

hyper-parameter η and that an approximation for the distribution over η can be estimated from the observations or any other available data.

In the method proposed in this chapter, observations of a rule r's classifications on a training set are used to infer the mean $\mathbf{p}^*(r)$ of most likely multinomial distribution over classifications. This distribution is controlled by the hyper-parameters $\mathbf{q}(r)$ and M where values for the former have been estimated using the performance of rules similar to r on other classification tasks. What makes the method proposed here more than a straight-forward application of empirical Bayes is the use of descriptors to aggregate and transfer estimates between similar rules.

The application of empirical Bayes to the estimation of classification probabilities has also been explored by Latinne et al. [2001]. In their work an iterative EM algorithm is used to estimate classification priors from a set of unclassified instances. These priors are then used to compute the posterior probabilities for the class labels over the same set of instances. The efficacy of this approach is demonstrated on a remote sensing task using logistic regression. While this approach does not do inductive transfer two recently proposed algorithms by Marx et al. [2007] use a similar technique but with priors learnt from related support tasks rather than the test instances. In the same vein, Raina et al. [2006] have described an algorithm for text classification that estimates covariance parameters between words based on data from similar classification tasks. These estimates can be used to relax a commonly used independence assumption regarding word appearances used in text classification problems.

Several other approaches to inductive transfer and multitask learning can be viewed as a type of empirical Bayes. For example, the bias learning methods of Baxter [2000], Heskes [2000], Pfahringer et al. [2000], Oblinger et al. [2002] all use training data from support tasks to determine which part of a large hypothesis space is most likely to contain useful hypotheses for a target task. All of these approaches enforce a language bias on the hypothesis space when learning a novel new task in contrast to the evaluation bias imposed by the similarity-based transfer approach presented here.

3.6.3. Multitask Learning. The approach to inductive transfer proposed here can be seen as a symbolic analogue of the neural network-based multitask learning systems of Silver and Mercer [1998, 1995], Silver [2000], Caruana [1997] and Baxter [1995]. In particular, the use of transfer and consolidation phases closely resembles those of the Task Rehearsal Method (TRM)

[Silver, 2000]. This also uses the idea of "virtual examples" to modify the behaviour of learning on the target task. The main difference between his method and the one proposed here is that the neural network models learnt from earlier tasks are kept by the TRM. Randomly chosen input values are fed through the stored networks to create extra examples that can be learned from in parallel to those for the target task. In contrast, earlier task solutions are not remembered in similarity-based transfer and the virtual examples are actually just changes to the contingency table.

In the symbolic realm, Caruana [1997] suggests how his multitask learning (MTL) approach can be applied to the top-down induction of decision trees. Both that approach and the one described here change how the base learner searches the hypothesis space by modifying the way it assesses the quality of its decisions. Caruana's proposal modifies the gain heuristic that decides which split to add to a decision tree by making the decision based on examples from target and support tasks. This is more akin to direct transfer than the similarity-based approach advocated here. At present, description-based transfer is not applicable to gain heuristics so a straight-forward implementation for decision tree learning would try all possible splits, construct new trees for each and decide which is best based on features of the resulting trees in their entirety.

Zhang and Zhang [2002, Chapter 7] describe a method for improving evaluation for association rule mining from small databases by making use of secondary databases. When computing the support, recall and precision of a rule for the small database these assessments are also made relative to the larger, secondary database. Unlike similarity-based transfer, no attempt is made to consider the evaluation of rules other than the one under consideration making their approach a form of direct transfer.

3.6.4. Task Similarity. "What does similar mean?" is a central question in all forms of inductive transfer. In the theory proposed here, similarity between tasks is defined relative to a similarity relation over the candidate hypotheses for the tasks. In this work, two tasks are similar if the average CPMs for each similarity class do not differ too greatly. The transfer theorem shows that if two tasks are similar and regular enough then the estimates using classification priors derived from the support task will improve on the estimates made on the target task without those priors.

The use of similarity classes to partition a hypothesis space is also the basis of recent work by Ben-David and Schuller [2003]. In their framework, a family of instance transforming functions $\mathcal{F}$ is assumed. It is further assumed

that this collection of transformations is a group under composition and that the hypothesis space is closed under actions of the group. That is, if for every hypothesis h in some hypothesis space H and for every transformation function $f_i \in \mathcal{F}$ the hypothesis $h \circ f_i \in H$. This notion captures the idea of concept invariants such as rotations and scaling when learning from sets of images. The group of instance transformations induce an equivalence relation (and therefore a partition) over the set of hypotheses. Namely, two hypotheses h' and h are equivalent whenever $h' = h \circ f$. This use of equivalence classes is analogous to the one used in similarity-based transfer. The primary difference between the two approaches is how these similarity classes are used. Given a set of tasks $\{T_i\}$, Ben-David and Schuller's meta learner finds a single equivalence class that contains hypotheses for each task which, together, have the lowest average classification error. Then, when a new task is presented, it is assumed that its concept must also come from the best class found in the previous step. This approach therefore transfers a language bias between related tasks in contrast to the evaluation bias that is transferred by the approach presented here.

As part of his Ph.D. thesis, Silver [2000] investigated a number of measures of "relatedness" for multitask learning with his ηMTL system - an extension of the original multitask learning (MTL) [Caruana, 1997]. Whereas normal MTL has a single learning rate η for all output nodes for a shared neural network, Silver's version uses a different η_k for each output node corresponding to a support task T_k. This learning rate determines the amount of influence the task will have during the training of the network. Silver proposes that the learning rate for each task be ηR_k where η is some base learning rate and R_k is some measure of relatedness between the support task T_k and primary task T_0. The suggested R_k measures take into the account of the training errors of the support tasks learnt in parallel, their structural similarity to the target task such as the Euclidean distance or cosine of the angle between the weights from the common feature layer to the output nodes taken as a vector v_k for the target task and support task. All these measures are defined to address multitask learning using artificial neural networks and it is difficult to see how they might be generalised to measure relatedness in other learning contexts.

More recently, Juba [2006] has suggested a general and intuitively appealing definition of task similarity based on Kolmogorov complexity [Li and Vitanyi, 1997]. His work provides PAC sample complexity bounds of $O(K(\mathbf{h})/n)$ for fixed ϵ and δ where $K(\mathbf{h})$ is the joint Kolmogorov complexity of the n hypotheses $\mathbf{h} = (h_1, \ldots, h_n)$. This refinement of Baxter's $O(1/n)$ sample complexity bound [Baxter, 2000] indicates that the number of examples required for good

generalisation decreases with the number of tasks being learnt simultaneously and the combined compressibility of the tasks when taken together. Ties between complexity-based similarity and description-based similarity would be an interesting avenue for future research.

3.6.5. Transfer parameters. In the definition of inductive transfer presented above, the prior function $\mathbf{q}$ depends on three parameters: the support task examples, the set of rules evaluated on those examples, and the choice of similarity relation. The appropriateness of a prior built in this manner will depend entirely on those choices. As discussed briefly in the introduction, Turney [2000] would call the cost of setting these parameters and choosing support tasks the "HCI (human-computer interaction) cost of Parameter Setting" and the "HCI cost of Incorporating Domain Knowledge" respectively. Ideally, these should have a lower HCI cost than explicitly creating a prior function by hand or selecting another kind of bias manually. To achieve this, it will be useful to determine some good default settings for the transfer parameters wherever possible.

The M parameter sets the importance of the prior CPM $\mathbf{q}$ relative to the CPM $\mathbf{p}$ based on a training set of size N. The ratio $M : N$ determines the strength of the prior's influence on the evaluation of a candidate rule or hypothesis. If M and N increase proportionately the influence of the prior will remain constant. This suggests that a default choice of M should be an increasing function of the training set size N. One such function is $M = \sqrt{N}$. Some theoretical justification can be given for this choice with reference to Section 3.2. The values in the virtual contingency table $\mathbf{m}$ of size M are all parameters for the Dirichlet distribution of prior values in $\mathbf{p}$. When there are N training examples used to estimate $\mathbf{p}$ the maximum likelihood estimate for M is given by $\sqrt{N}$. The details of this derivation are beyond the scope of this thesis and the interested reader is referred to [Bishop et al., 1975, pg. 407]. The use of $\sqrt{N}$ as a default setting for the parameter M is tested empirically in Chapter 5.

In our theories, we rightly search for unification, but real life is both complicated and short, and we make no mockery of honest ad hockery

- I. J. Good [1965, pg. 56]

CHAPTER 4

DEFT: An Implementation of Description-Based Transfer

As described in the last chapter description-based inductive transfer is a bias learning technique which modifies a rule learning algorithm's evaluation function. Examples from support tasks are used to calculate the expected classification probability matrix for a rule based on the classification matrices of similar rules evaluated on the support task. This expected CPM is then combined with the standard evaluation process used by the base learner to improve the quality of the evaluation estimates and, in turn, improve the generalisation performance of the base learner when applied to a target task. To test this claim empirically, as done in the next chapter, an implementation of this approach is required. The aim of this chapter is to describe how this approach can be realised for tasks which use a first-order clausal representation for rules. To implement the description-based transfer technique as a system that can be applied to real learning tasks several complications and inefficiencies must be overcome through the application of some "honest ad hockery". The resulting system is called DEFT, Description-based Evaluation Function Transfer.

Section 4.1 provides a high-level view of all of the components and algorithms used in DEFT and how the final system is to be used in practice. Critical to all of the components is the efficient computation of rule descriptions. To this end, Section 4.2 introduces the concept of *descriptor templates*. As the name suggests, these are a kind of "meta-descriptor" from which specific instances of descriptors can be derived. These not only make it easier to specify descriptor sets but allow for rule descriptions to be computed efficiently. The next two sections describe the two central algorithms within DEFT. Section 4.3 presents the BUILDDFT algorithm which collects estimates of descriptor frequencies by sampling and evaluating clauses on a support task. The resulting Descriptor Frequency Table (DFT) is used by the CALCPRIOR algorithm when DEFT is required to compute classification priors for rules on a target task. The details of this second algorithm are given in Section 4.4. Borrowing terms from Silver [2000], the first algorithm is a method for *consolidating* support task knowledge while the second is for the *transfer* of that consolidated knowledge. An example of consolidation and transfer using DEFT is provided in Section 4.5. It makes use of five artificial learning tasks similar to the reading

preferences examples used in earlier chapters. These tasks are also reused in the experimental evaluation of DEFT in the next chapter.

4.1. System Overview

As with all the systems reviewed in Section 2.5, the one described here consists of a base-level learning component - a slight modification of ALEPH - and a meta-level learning component, DEFT. An overview of how these components interact is shown in Figure 4.1. The base-level learning algorithm ALEPH takes as input a set of target examples and outputs a set of rules. The specific rules chosen for inclusion in the output are a result of the language, search and evaluation biases of the SEQCOVER algorithm. The language and search biases are set, as usual, through the choice of background predicates and the selection of the various search parameters as described in Section 2.3.2. The evaluation bias, however, is a combination of the choice of evaluation function for the given task and the bias imparted by DEFT. When a rule is evaluated by the FINDRULE procedure within SEQCOVER the resulting contingency table is augmented with a classification prior for the rule that is determined by the CALCPRIOR procedure described in Section 4.4 below. The prior used depends on what is called a *Description Frequency Table* or DFT and the descriptors that were used to construct it. Both of these terms are defined in Section 4.3 along with the BUILDDFT algorithm. As shown in the figure, the DFT output by BUILDDFT depends on the choice of support task that is given as input to DEFT. This, in turn, is what affects the evaluation bias of the base-level learner.

This combination of DEFT and ALEPH is intended to be used as follows. A domain expert wishes to induce a set of rules for a target task with a limited number of training examples. Examples of other, related tasks are available and the domain expert selects one of these as the support task. The support examples are passed in to the BUILDDFT procedure along with a set of descriptors defining the rule features believed to capture the similarities between the target and support tasks. The BUILDDFT procedure then constructs a DFT from these inputs which can be saved to file. To induce rules for the target task its examples and background knowledge are given to ALEPH and a previously constructed DFT is passed to DEFT. The induction of rules by ALEPH proceeds as normal with the evaluation function used to assess each rule modified using the CALCPRIOR procedure. An example use of DEFT in this manner is provided in Section 4.5 below.

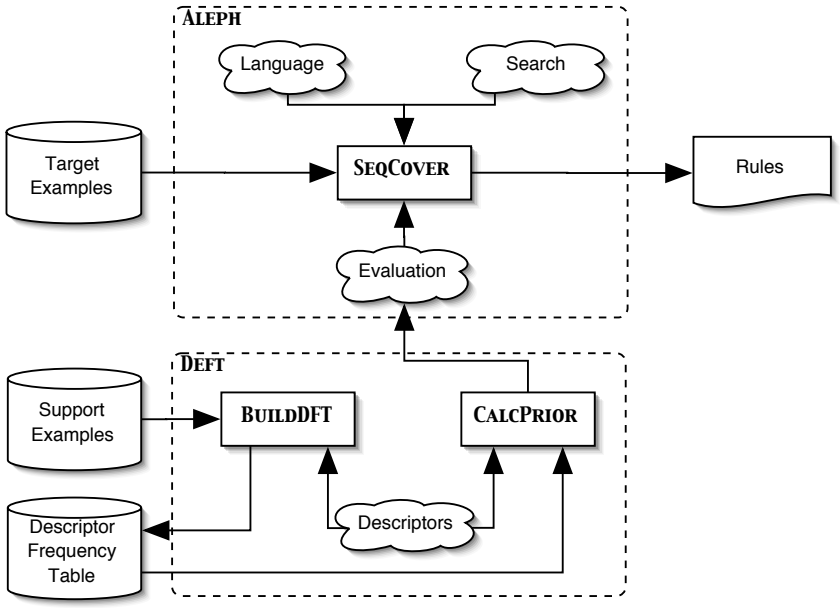


FIGURE 4.1. The base level learner with detail of the DEFT learner.

4.1.1. Requirements. There are two main design requirements for the implementation of DEFT: speed and ease of use. The first requirement is to ensure that using description-based transfer does not add too much computational overhead while the aim of the second is to make using DEFT more appealing than constructing a declarative or preferential bias by hand.

Using a direct implementation of description-based transfer, as described in the previous chapter, would be cumbersome. Defining a similarity relation for rules requires a descriptor set to be specified in advance. To define descriptors similar to those used in the examples of the previous chapter requires prior knowledge of the predicates and constants that are used in the legal clauses for a task. Descriptor templates, introduced in Section 4.2 below, provide a way for descriptors to be generated dynamically during transfer, removing the burden of descriptor definition from the user.

Making description-based transfer efficient is the more difficult requirement to meet. In general, the set of similar rules to average over may be very large or infinite, making the computation of an exact prior costly or impossible. As this procedure needs to be carried out every time a rule is evaluated it is crucial that its implementation be efficient. To achieve this, the computation of exact priors are given up for their estimates. The main concession made in the name of efficiency is to assume that all descriptors used for transfer are independent of one another as discussed in the previous chapter. That is, when two descriptors are applied to a rule, the value of one has no bearing on

the value of the other. While this is not true in general, this assumption allows the computation of classification priors to be reduced to the estimation of how frequently particular descriptor values and class labels co-occur. As described in Section 4.3.2, this can be achieved by repeatedly sampling and evaluating clauses from the support task.

4.1.2. The Meta Learner: DEFT. The computation of classification priors for use by the base learner is carried out by the meta learning component DEFT. This computation is split into two phases. In the *consolidation* phase the BUILDDFT procedure is run over the support examples to collect statistics about rule descriptions and performance which is stored in a descriptor frequency table or DFT. In the *transfer* phase the base learner within DEFT is run on the target examples. Whenever the base learner evaluates a rule on the target examples, the CALCPRIOR procedure is called to compute a classification prior for that rule based on the information stored in the DFT built from the support task.

The two main algorithms in the DEFT system are BUILDDFT and CALCPRIOR. First, a support task is used to estimate the frequencies $\phi_{ij}[d, v]$ for all the appropriate descriptor and value pairs $[d, v]$. This is performed by the BUILDDFT algorithm. Once these are collected and stored in a DFT the second algorithm CALCPRIOR uses them to estimate the probabilities $\Pr(i, j)$ and $\Pr(d(r)|i, j)$ whenever a prior $\mathbf{q}(r)$ is required for the rule r.

There are several advantages to the use of support task summaries such as DFTs. Firstly, a summary generally requires less storage space than the examples used to generate it. Secondly, the summary allows for the fast computation of CPM priors. Thirdly, the summaries can be used to modify the evaluation bias on different target tasks, removing the need to consult the original support task examples. Finally, the DFTs provide an insight into how tasks are related through the comparison of their descriptors and relative frequencies.

The efficient computation of rule descriptions is crucial to both the building of DFTs and their use. Each time a rule needs to be evaluated by the base learner, DEFT is required to compute that rule's description and use it to construct a prior. When a DFT is built, DEFT repeatedly tests rules on the support task and counts the coincidence of each descriptor-value pair and classification type. To do this, a descriptor must be computed for each rule that is tested. Because this procedure is such a central one, Section 4.2 is dedicated to computing descriptions efficiently.

4.1.3. The Base Learner: A Modification of ALEPH. The base-level component used in the implementation is a slight modification of the

Algorithm 4 The description-based SCORE procedure.

1: **procedure** DBSCORE(r, E, Φ, M)
2: Compute $\mathbf{n}_E(r)$ the contingency matrix for r on E
3: Let $\mathbf{q} = \text{CALCPRIOR}(r, \Phi, E)$, the prior for r
4: Let $\mathbf{n}^* = \mathbf{n}_E(r) + M\mathbf{q}$
5: **return** $f(\mathbf{n}^*; r)$
6: **end procedure**

Algorithm 5 The description-based BOUND procedure.

1: **procedure** DBBOUND(r, E, M)
2: $\mathbf{n} = \text{EVALUATE}(r, E)$, the contingency table for r on E
3: $\mathbf{n}_{\max} = \begin{bmatrix} n_{++} & 0 \\ n_{-+} & n_{--} + n_{+-} \end{bmatrix}$
4: $\mathbf{m}_{\max} = \begin{bmatrix} M & 0 \\ 0 & 0 \end{bmatrix}$
5: $\mathbf{n}^*_{\max} = \mathbf{n}_{\max} + \mathbf{m}_{\max}$
6: **return** $f(\mathbf{n}^*_{\max}; r)$
7: **end procedure**

ALEPH system (see Section 2.3.2). These are modifications to the evaluation routines SCORE and BOUND which are called from its FINDRULE procedure (Algorithm 2). The changes to the evaluation and pruning of rules carried out during a search are to allow the combination of description-based prior CPMs with those obtained from the target task's training examples. These modifications are shown in Algorithms 4 and 5.

The DBSCORE procedure[1] differs from the original through the addition of lines 3 and 4. These two extra steps first compute the CPM prior for the given rule using the CALCPRIOR algorithm introduced below. The result is then used to compute the contingency table $M\mathbf{q}$ of M virtual examples which is then added to the normal rule evaluation $\mathbf{n}_E(r)$ of the rule r on the target training examples E. This combined contingency table is passed into the evaluation function used by ALEPH to assess rule quality.

The DBBOUND procedure does not need to use the CALCPRIOR procedure to compute a virtual contingency table as all that is required from the returned bound is that it be larger than every value returned by DBSCORE for any rule that is a refinement of the current one. The original BOUND procedure achieves this by computing the contingency table shown in line 3 and passing it to the evaluation function. A justification for this is given in Section 2.3.2 above. The only difference here is that the virtual contingency tables of the refinements of the input rule r must also be taken into account. This is done by assuming

[1]The "DB" stands for Description-Based and is used as a prefix to distinguish these modifications from the original SCORE and BOUND procedure.

that the evaluation function f will only increase (or at least not decrease) by adding extra covered positive examples to the contingency table it is passed as input. As only M virtual examples are added to the actual contingency table by DBSCORE an upper bound on the quality of refinements of r can be found by assuming the best virtual contingency table is the one shown in line 4. While this is extremely unlikely in practice this procedure does provide a valid, albeit weak, upper bound on the evaluation of r's refinements.

4.1.4. Other Details. The actual implementation of DEFT was written as an extension to a modified version of ALEPH 4 [Srinivasan, 2001] in Yap Prolog 4.4.4 [Costa et al., 2005]. The extensions and modifications come to a total of around 1000 lines of Prolog code and are freely available at `http://threewordslong.com/research/deft/`. Even though the code for DEFT is written in Prolog most of the algorithms described in this chapter are presented in procedural pseudo-code for the sake of exposition. The exception to this is with the code for implementing descriptors as this makes use of backtracking and is therefore better explained using declarative code.

4.2. Computing Clause Descriptions

As described in the overview above, rule descriptions are required in both the consolidation and transfer phases of the DEFT system. Since both of these phases require hundreds or thousands of descriptions to be computed it is necessary that this be done efficiently. A second requirement is that it must be easy to specify the descriptors that are to be used for a transfer between two tasks. If too much time and thought must be spent on carefully selecting descriptors, most of the advantage of inductive transfer will be lost and the user may as well hand-craft a bias. Descriptor templates are introduced and implemented in this section to address both of these requirements.

4.2.1. Descriptor Types and Sparse Descriptions. Descriptors are used for the comparison of rules to determine whether or not they are similar. If two rules share the same values for all of the descriptors under consideration they are deemed to be similar with respect to those descriptors. The choice of descriptors therefore plays an important role in this form of inductive transfer and the scope for this choice is very large. For example, rules represented as clauses can be compared based on the number of free variables they have, the length of the longest variable chain, the number of unique constants used, whether two particular literals share a variable, and so on. To begin with, however, only three types of descriptors - called the *standard descriptors* - will be considered:

(1) The *num_lits* descriptor. Determines the length of a clause, *i.e.* the total number of literals in the head and body of the clause. The values returned by such a descriptor can be any non-negative integer.
(2) The *has_pred* descriptor. Tests whether a certain predicate is present or absent as a literal in the clause. This type of descriptor returns either *true* or *false*.
(3) The *has_arg* descriptor. Test whether a particular argument in a certain predicate is equal to a given value. This type of descriptor also returns *true* or *false*.

These three types of descriptor were chosen primarily for their simplicity and ease of implementation but also because the tests they perform capture a shallow but natural sense of clause similarity. Arguably, a human expert comparing two clauses would look first at their length, literals and constants to determine if they are the same or similar. All of the experiments carried out in the next chapter use at least one of the above descriptor types.

A useful observation can be made regarding the standard descriptors and all of the other examples mentioned above: for any short clause most of the descriptors being used to build a description will take on some default value. As an example consider an instance `has_pred(p/1)` of the *has_pred* descriptor which tests for the presence of the predicate `p/1` in a clause. If there are nine other predicates that can be used to construct clauses for a particular task the descriptor `has_pred(p/1)` might only take the value *true* on roughly one-tenth of the clauses it is tested on. In this sense the value *false* can be seen as the default value for this descriptor and, by a similar argument, all of the others of the same type.

Two important optimisations can be derived from the observation that descriptors will, more often than not, take on a default value. The first is that the size of description vectors can be reduced by storing them *sparsely*. This is done by only recording the descriptor-value pairs for descriptors which take on their non-default values.

DEFINITION 4.1. A *sparse description* $\mathbf{d}$ is a set of *descriptor-value pairs* $[d, v]$ consisting of a descriptor d and a value $v \in V_d$.

A sparse description can be interpreted as a description vector for a descriptor set D as defined in Definition 3.19 of the previous chapter. This is done by associating a default value $v_0 \in V_d$ with each descriptor $d \in D$. The value for $d \in D$ in $\mathbf{d}$ as a description vector is then v if there exists a pair $[d, v] \in \mathbf{d}$ or the default value v_0 for d otherwise.

The other main advantage to using default values is that descriptions can be computed *lazily* since only those descriptor-value pairs with non-default values actually need to be computed. Any descriptor not appearing in a description can be assumed to take on its default value.

4.2.2. Descriptor Templates. Sparse descriptions for clauses can be computed lazily using descriptor *templates*: functions of clauses that return sets of non-default descriptor-value pairs. Each of the three types of descriptors described above - *has_pred*, *has_arg* and *num_lits* - can be implemented using templates. For example, when given a clause an implementation of a template for has_pred descriptors can scan it creating the pair [`has_ pred(p/n)`, *true*] for every predicate p/n that appears in the clause. Any other *has_pred* descriptor is assumed to return the value *false* on the given clause.

DEFINITION 4.2. A *descriptor template* τ is a function which maps rules to sparse descriptions. When applied to a rule r a template τ returns a sparse description $\mathbf{d} = \tau(r)$. Each element $[d, v] \in \mathbf{d}$ is a descriptor-value pair where $v = d(r)$ is a non-default value for d. The descriptor d is said to be an *instance* of the template τ.

As well as having efficient implementations, the use of templates means only the descriptor types need to be specified in advance rather than an entire set of descriptors. This reduces the amount of effort an expert needs to expend when setting up an environment for inductive transfer.

Given a collection of descriptor templates, a description of a rule can be constructed by applying all the templates to the rule and taking the union of the resulting sets of descriptor-value pairs. This procedure is detailed in Algorithm 6 and shows how descriptions are computed within DEFT. There is an implicit assumption that the templates in $\mathcal{T}$ are disjoint in the sense that every descriptor which can be constructed from a template in $\mathcal{T}$ is an instance of exactly one of those templates. This requirement is very easy to ensure in practice and holds for all of the standard descriptor types listed above.

4.2.3. Implementation. Descriptor templates for the standard descriptors are, like the rest of DEFT, implemented in Prolog. Each template is defined through the predicate `value/3` which asserts a relationship between a clause, a descriptor, and a value. The relation holds if and only if the descriptor would return the specified value when applied to the clause. Due to Prolog's backtracking behaviour, the `value/3` relation can be used to create a sparse description by finding all descriptors which return non-default values for a clause. When required, the default value for a descriptor can be

Algorithm 6 The DESCRIBE procedure. Given a clause r and a set of descriptor templates $\mathcal{T}$, this procedure returns a sparse description vector for r using descriptors belonging to $\mathcal{T}$.

```
1: procedure DESCRIBE(r, T)
2:     Let d be an empty set
3:     for all descriptor templates τ ∈ T do
4:         Let d = d ∪ τ(r)
5:     end for
6:     return the sparse description d
7: end procedure
```

determined with a call to `default/2` which associates each descriptor with its corresponding default value.

The Prolog code in Appendix A is the complete, working code which defines the `value/3` and `default/2` predicates for the standard descriptors and the DESCRIBE procedure of Algorithm 6. Apart from the predicates `body_lits/2` and `between/3` all of the code used is standard Prolog. As the name suggests, the `body_lits/2` predicate takes a clause as input and returns a list containing its body literals. The `between/3` predicate asserts that its first argument is an integer that is greater than its second argument and less than or equal to its third.

The descriptor templates given in the Appendix are general enough to apply to new inductive transfer tasks. However, if more is known about a specific pair of support and target tasks new descriptor templates can be written to define similarities between their rules in terms of task-specific predicates. It is relatively straight-forward to write new descriptor templates as they are similar in complexity to writing custom search constraints in ALEPH. One caveat when writing custom descriptors is that all clauses for the `value/3` predicate must be written without cuts so that a call to `value(Desc, Clause, Val)` for a bound value of `Clause` will return all possible bindings for `Desc` and `Val`. Taken as pairs, the collection of all these bindings forms a sparse description of the clause bound to `Clause`. The implementation of DESCRIBE as the predicate `describe/2` in the code does exactly this. Examples of template expansion are given in Section 4.5 below.

4.2.4. Analysis and Discussion. The above discussion demonstrates the value in using sparse descriptions and templates to represent and compute descriptions respectively. However, since the DESCRIBE procedure may be called thousands of times during the consolidation and transfer phases of DEFT, it is worth looking at the time and space complexity of the templates more closely.

When applied to a clause with n literals the implementation of the *num_lits* template iterates through the literals to construct a list and then takes its length. Its time complexity is therefore $\Theta(n)$, although if the data structure used to represent clauses were modified to cache its length this could be reduced to a constant-time lookup. As each clause only has one length, this template will always return a sparse description with exactly one descriptor-value pair making its space complexity constant.

As the *has_pred* descriptor template also converts the n body literals in a clause to a list, it also has time complexity $\Theta(n)$. Unlike the *num_lits* template, however, this cannot be reduced to constant time since all n literals may be instances of n different predicates, requiring a *has_pred* descriptor to be created for each. In this worst case, the sparse description created using the *has_pred* template will contain n distinct descriptor-value pairs implying that its space complexity is $O(n)$.

Finally, the *has_arg* template has a time and space complexity of $O(kn)$ where n is the number of literals in the clause being described and k the maximum arity over all of those literals. The case that forces this upper bound is when all n literals are for different predicates and all k arguments of each are bound to non-variable terms. In this situation, kn *has_arg* descriptors will be created. Even when the predicates are not all different, a large number of descriptors can be created when the literals' arguments contain different terms. As this is the most expensive of the three templates it dominates the cost of constructing descriptions using the DESCRIBE procedure.

Compared to the cost of testing a clause against the all the training examples for a task constructing a description is relatively cheap. If necessary, the running time could be further reduced by caching descriptions with clauses. Each time a clause is refined during a search the description could be updated using incremental templates which analyse the literal added or substitution used to specialise the corresponding clause. This optimisation is left as future work.

Describing clauses plays an integral part of DEFT and the use of descriptor templates ensures that these can be represented and computed efficiently. Computing clause descriptions using descriptor templates is both time and space efficient, not using more than linear space and time with respect to clause length. It is also a convenient way of specifying a descriptor set for performing inductive transfer between two tasks.

4.3. Consolidation: Descriptor Frequency Tables

The aim of the consolidation phase in DEFT is to estimate the descriptor frequencies introduced in Definition 3.20 using examples from a support task. As described in the previous chapter, these estimates are of the probability that a randomly selected rule and example will result in a particular classification and have a particular descriptor value.

Ideally, these probabilities would be computed by testing every candidate rule against every example available for the support task and simply counting the co-occurrence of descriptor-value pairs and classifications. While example sets are usually of a manageable size, sets of clauses for relational learning tasks are generally huge, ruling out an exhaustive computation. The alternative is to sample an appropriate number of rules from the rule space and use their evaluation on the support task to estimate the required probabilities.

A similar problem is encountered when searching [Srinivasan, 1999] and the observation made there is that the best clauses are often not required, merely "good" ones - that is, clauses that fall in the top 1% of all clauses when ranked by evaluation score. This form of goal softening is known as "ordinal optimisation" Ho et al. [1992] and is a form of "satisficing" [Simon, 1999] or goal softening which has been advocated by other ILP researchers [Sebag and Rouveirol, 2000].

4.3.1. The BUILDDFT algorithm. As described in the overview, all that is required by the CALCPRIOR algorithm is that the DFT given to it have reasonable *estimates* of the descriptor frequencies $\phi_{ij}[d, v]$. Provided the number of rules sampled by BUILDDFT is large enough and representative of the entire set, the central limit theorem implies that the descriptor frequencies will tend towards the values they would have taken had the entire set been used. The sampling and evaluation steps are combined in the BUILDDFT procedure shown in Algorithm 7.

The outer two loops of the algorithm control the sampling by selecting an example, constructing a bottom from it and then repeatedly sampling from the bottom clause. The details of the SAMPLE procedure which is called in line 8 are discussed below. Once a clause is selected it is evaluated and the resulting contingency table is added to entries in the hash-table $\mathbf{s}$ for each descriptor-value pair in its description. The contingency table is also added to a total $\mathbf{t}$ which, together with the values in $\mathbf{s}$, can be used to compute the frequencies for any descriptor-value pair. The details of the frequency extraction are described below.

Algorithm 7 The BUILDDFT procedure. Given descriptor templates $\mathcal{T}$ and examples $E = E^+ \cup E^-$ sample S_{exs} examples and S_{cls} clauses per example to construct a DFT from E.

```
1: procedure BUILDDFT(T, E, S_exs, S_cls)
2:     Initialise s as an empty hashtable
3:     Initialise t as 2 × 2 zero matrix
4:     repeat S_exs times
5:         Choose an e ∈ E+ randomly and replace
6:         Construct ⊥ the bottom clause for e
7:         repeat S_cls times
8:             Sample a clause r using SAMPLE(⊥)
9:             Evaluate the contingency table n_E(r) for r
10:            Compute the sparse description d(r) using DESCRIBE(r, T)
11:            for all descriptor-value pairs [d_k, v_k] ∈ d(r) do
12:                Initialise s[d_k, v_k] to a 2 × 2 zero matrix if it does not exist
13:                Add n_E(r) to s[d_k, v_k]
14:            end for
15:            Add n_E(r) to t
16:        end
17:    end
18:    return Φ = (s, t) as the resulting DFT
19: end procedure
```

4.3.2. Clause Sampling. The random clause selection procedure SAMPLE uses the same ideas and code developed by Srinivasan [1999] for the stochastic search methods available in ALEPH. As this procedure is central to the construction of DFTs it is given a brief exposition here.

The first step is to estimate for each $l = 1, \ldots, L$ the number n_l of clauses of length l that are legal clauses and subsume $\perp$. This is done by randomly selecting, without replacement, $l-1$ literals from the body of $\perp$ and forming a clause c using these and the head literal from $\perp$. Clause c can then be tested against the mode and type restrictions for legal clauses. By repeating this process and counting the number of legal and illegal clauses that arise, a Monte Carlo estimate p_l of the proportion of legal clauses of length l which subsume $\perp$ can be derived. The number of repetitions of this process is set to 100 by default within ALEPH. This proportion is then multiplied by $\frac{(|\perp|-1)!}{(l-1)!}$ (the number of subsets of body literals in $\perp$ with size $l-1$) to obtain the estimate n_l. The values for $n_1, \ldots, n_L$ imply there are approximately $N = \sum_1^L n_L$ legal clauses of size L or less. To draw one of these at random an integer n is first chosen uniformly from $0, \ldots, N$. The aim now is to generate the n^{th} legal clause subsuming $\perp$. One way to do this is to let l and k be the smallest integers such that $n = k + \sum_0^l n_i$ and then deterministically generate legal clauses of length l using, for instance, the standard refinement actions and a depth-first search.

The k^{th} clause generated in this way which has not already been sampled is the one returned. Further details regarding this algorithm including ways to improve its efficiency can be found in [Srinivasan, 1999, §3.1.1] or the ALEPH source code [Srinivasan, 2001].

4.3.3. Estimating Descriptor Frequencies. Provided enough samples are taken, the counts and totals recorded in a DFT contain enough information to obtain a good estimate of the descriptor frequencies $\phi_{ij}[d, v]$ for the support task. These estimates will be computed and cached once a DFT is built and later used to construct priors during the transfer phase.

The sum of the entries in the matrix $\mathbf{t}$ will be exactly $|E|.|R|$, the number of examples and rules that were tested against each other. This total will be denoted $t = \sum_{ij} t_{ij}$ and is used to compute the estimates $\phi_{ij} = \frac{t_{ij}}{t}$ of the classification probabilities $\Pr(i, j)$ on the support task.

As discussed in the overview, the probabilities $\Pr(i, j, d = v)$ are required to compute a CPM prior for a clause. For the descriptor-value pairs recorded in Φ, these probabilities are estimated by the descriptor frequencies $\phi_{ij}[d, v] = \frac{s_{ij}[d,v]+\phi_{ij}}{t_{ij}+1}$. The additional pseudo-Bayes terms ϕ_{ij} in the numerator and 1 in the denominator are to ensure these estimates are non-zero[2].

The descriptor-value pairs for default values do not have their counts recorded in Φ directly but these can be derived from the counts in $\mathbf{s}$ and the totals $\mathbf{t}$ as follows. Because descriptors partition any rule space, the counts for the default value v_0 of a descriptor d must be the remainder after all the other counts are subtracted from the totals table. That is $\mathbf{s}[d, v_0] = \mathbf{t} - \sum_v \mathbf{s}[d, v]$. These counts are then used in the same way as the others to compute $\phi_{ij}[d, v_0]$. The procedure that uses these descriptor frequencies to compute priors is given in the Section 4.4.

4.3.4. Analysis. The running time of the BUILDDFT procedure is controlled by the total number of rules that are generated, described, and tested against the support task examples E. As the main loop of the procedure samples S_{cls} clauses for each of the S_{exs} examples that are selected, a total of $S_{exs} \times S_{cls}$ rules are generated. If D is an upper bound on the time it takes to describe a single rule and V an upper bound on the time to evaluate a single rule on a single example then the time complexity of building a DFT is $O(S_{exs}S_{cls}(D + |E|V))$.

As shown above, rules can be described quickly when using descriptor templates. In this case the upper bound on the description time becomes a function

[2]This is a standard trick when estimating probabilities. Cestnik [1990] provides a discussion of this technique.

of the number of descriptor templates and the arity and length of the most complex rule in the search space. This will also control V since the time it takes to evaluate a clause is a function of its complexity [Giordana and Saitta, 2000, Sebag and Rouveirol, 2000]. This maximum clause complexity can be assumed to be fixed for a large variety of tasks and, as the number of support examples will be much larger than the number of descriptor templates, the complexity of the BUILDDFT procedure can be reduced to $O(S_{exs}S_{cls}|E|)$.

Since the running time for BUILDDFT is controlled by the number of clause samples performed, an important question is how many samples are required to get good estimates for the descriptor frequencies? Treating each frequency as an estimate of a random variable we can use estimation theory [Walpole and Myers, 1978, §6.5] to derive a bound on the number of samples required. Specifically, at least

$$t = \frac{z^2_{\alpha/2}}{4\epsilon^2}$$

samples if we want the frequency to be within ϵ of its true value with confidence $1 - \alpha$. Here, z is the standard normal distribution. For values $\epsilon = 0.01$ and $\alpha = 0.01$ this equation suggests we need more than $t = 16590$ samples. The number of examples for a support task is fixed and so the total number of samples made of the frequencies in the columns of each $\phi[d, v]$ will be controlled entirely by the S_{exs} and S_{cls} parameters. That is, if there are $|E^+|$ positive examples, estimating the frequencies $\phi_{i+}[d, v]$ will require $S_{exs}S_{cls} = \frac{t}{|E^+|}$ samples. The product $S_{exs}S_{cls}$ should therefore be chosen so that it is greater than $\frac{t}{\min(|E^+|,|E^-|)}$ to ensure the frequencies $\phi_{i+}[d, v]$ and $\phi_{i-}[d, v]$ are all estimated well.

4.4. Transfer: Calculating Classification Priors

Once a DFT is built during a consolidation phase, it can be later used to generate rule classification priors using the simplifying assumption described in the overview. This redefines a rule classification prior as a product of descriptor frequencies estimated on a support task.

In this section an efficient method for constructing rule classification priors is presented which, like the algorithm for building DFTs, takes advantage of the sparse descriptions for rules that can be quickly generated using descriptor templates.

4.4.1. Using Descriptor Frequencies. In Section 3.5 of the previous chapter a formula for the expected CPM priors was given in Definition 3.20. This used a naïve Bayes assumption regarding the independence of descriptors which allows classification priors $q_{ij}(r)$ for a rule r to be written as the product

of the probabilities $\Pr(d(r)|i,j)$. These probabilities can be estimated from a DFT Φ as

$$\Pr(d(r)|i,j) = \frac{\Pr(i,j,d(r))}{\Pr(i,j)} \approx \frac{\phi_{ij}[d,d(r)]}{\phi_{ij}}.$$

The values for $q_{ij}(r)$ can therefore be written as

$$q_{ij}(r) \approx \frac{\phi_{ij}}{K(r)} \prod_{d\in\Phi} \frac{\phi_{ij}[d,d(r)]}{\phi_{ij}} \tag{4.5}$$

where $K(r)$ is a normalising constant, dependent on the description of r, and equal to $\sum_{ij} q_{ij}(r)$. The product is taken over all the descriptors d that are present in the DFT Φ. The number of multiplications required to compute the above product depends on the number of descriptors in the DFT. However, since many of the descriptors will take on their default values for any given rule, much of the product can be pre-computed once and modified using only those values present in a sparse description of a rule. This is the motivation for the following optimisation.

4.4.2. Default and Update Tables. Suppose $\mathbf{d}(r)$ is a sparse rule description computed from the same descriptor templates used to construct the DFT Φ. In this case, the product in equation 4.5 can be split into two parts:

$$\left(\prod_{[d,v]\in\mathbf{d}(r)} \frac{\phi_{ij}[d,v]}{\phi_{ij}}\right)\left(\prod_{d\notin\mathbf{d}(r)} \frac{\phi_{ij}[d,v_0]}{\phi_{ij}}\right)$$

where $d \notin \mathbf{d}(r)$ are the descriptors which are not present in the sparse description for r and are therefore assumed to take on their default values v_0.

This itself does not reduce the total number of multiplications however the multiplicands in the second product above only depend on the default values of the DFT and not the rule under consideration. If the product of all these values are pre-computed they can be updated according the the descriptor-values in the rule. These pre-computed products and updating terms are known as default and update tables respectively.

Definition 4.3. The *default table* $\mathbf{z}$ for the DFT Φ is a 2×2 matrix with values

$$z_{ij} = \phi_{ij} \prod_{d\in\Phi} \frac{\phi_{ij}[d,v_0]}{\phi_{ij}}$$

where each $[d,v_0]$ is the default descriptor-value pair for d. The product is taken over all the descriptors present in the DFT.

The values in default tables can be computed and stored in a DFT. Once there they can be transformed into rule priors by multiplication with an update table for a rule.

DEFINITION 4.4. An *update table* $\mathbf{u}(r)$ for the rule r and the DFT Φ contains the values

$$u_{ij}(r) = \prod_{[d,v]\in\mathbf{d}(r)} u_{ij}[d,v]$$

where each of the $u_{ij}[d,v] = \frac{\phi_{ij}[d,v]}{\phi_{ij}[d,v_0]}$ is called an *update term* for the descriptor-value pair $[d,v]$.

Update terms, like default tables, can be pre-computed and cached within a DFT. This is done once a DFT is constructed by iterating through all of its descriptor-values pairs and computing the update term for each using the relevant descriptor frequencies.

The following proposition states that the expected CPM prior can be derived from the default and update tables for a DFT.

PROPOSITION 4.5. *For a given rule r and DFT Φ, the normalised product of the values in the default table $\mathbf{z}$ and the values in the update table $\mathbf{u}(r)$ for r is equal to the prior $\mathbf{q}(r)$.*

PROOF. The product in question is

$$\begin{aligned}
z_{ij}u_{ij}(r) &= \left(\phi_{ij}\prod_{[d,v_0]\in\Phi}\frac{\phi_{ij}[d,v_0]}{\phi_{ij}}\right)\left(\prod_{[d,v]\in\mathbf{d}(r)}\frac{\phi_{ij}[d,v]}{\phi_{ij}[d,v_0]}\right)\\
&= \left(\phi_{ij}\prod_{[d,v_0]\notin\mathbf{d}(\mathbf{r})}\frac{\phi_{ij}[d,v_0]}{\phi_{ij}}\right)\left(\prod_{[d,v]\in\mathbf{d}(r)}\frac{\phi_{ij}[d,v]}{\phi_{ij}}\right)\\
&= \phi_{ij}\prod_{d\in\Phi}\frac{\phi_{ij}[d,d(r)]}{\phi_{ij}}\\
&= K(r)^{-1}q_{ij}(r)
\end{aligned}$$

where $K(r)$ is the normalising constant $\sum_{ij} q_{ij}(r)$. □

This equivalence is used to implement an efficient method for computing rule priors.

4.4.3. The CALCPRIOR Algorithm. The procedure presented in Algorithm 8 implements the use of update tables to modify a pre-computed default table for a DFT. The CALCPRIOR procedure does this by first looking up the default table for the DFT it is passed. The sparse description of the rule is then computed and each of its descriptor-value pairs is iterated over. The original default table values are progressively modified through multiplication with the update term for each descriptor-value pair. Finally, the resulting classification prior is skewed using the procedure described in Section 3.2.3 of the previous

chapter so as to have the same class distribution as the input examples and is then returned.

Algorithm 8 Compute the classification prior with the same class distribution as E for a rule r based on a DFT Φ.

```
1: procedure CalcPrior(r,Φ,E)
2:     Let z be the default table for Φ
3:     Let T be the templates used to construct Φ
4:     Let d be the description returned by Describe(r, T)
5:     for all prediction labels i ∈ {+, −} do
6:         for all actual labels j ∈ {+, −} do
7:             Initialise q_ij = z_ij
8:             for all descriptor-value pairs [d, v] ∈ d(r) do
9:                 Multiply q_ij by the update value u_ij[d, v]
10:            end for
11:        end for
12:    end for
13:    Normalise the q_ij so ∑_ij q_ij = 1
14:    Correct the skew of q so it is compatible with E
15:    return prior matrix q
16: end procedure
```

4.4.4. Analysis. The two outer loops for the CalcPrior procedure iterate through the four classification pairs that index the entries in a CPM. For each classification pair the innermost loop is called to update the default table using the update terms for the descriptor-value pairs present in the sparse description of the rule it is passed. The running time of the procedure is therefore dependent on the size of these sparse descriptions. As argued earlier, the size of these descriptions depends only on the complexity of the rule in question and the number of descriptor templates used to construct it. If C is some measure of the complexity of the clause that is passed to CalcPrior then the running time of this procedure is $O(C)$. When the standard descriptors are used this measure could be $C = kn$ where k is the maximum arity of the n literals in the clause.

The steps before and after the main loop do not have any great effect on the running time of the algorithm. The default table is computed once for the DFT being used and can afterwards be cached and looked up. This means its construction time is amortized over the thousands of calls to CalcPrior during a typical search. The normalisation and skew procedures used after the main loop run in constant time.

4.4.5. Descriptor Power. The decomposition of the naïve classification prior into a product of descriptor frequencies allows the default table and update terms to be pre-computed for a DFT. These can then be used to efficiently calculate classification priors from a rule description using the CalcPrior algorithm. Since two DFTs with the same default table and update terms must, by Proposition 4.5, compute the same priors for any given rule, these pre-computed values therefore also characterise DFTs. In the experimental work reported in the next chapter, it was necessary to compare DFTs that were constructed from different sets of examples. For this reason, a quantity called *descriptor power* is introduced here that summarises each of the update terms for descriptor-value pairs within a DFT. This power measures the influence each non-default descriptor value has on classification priors when it appears in the description of a rule.

Definition 4.6. Given the update terms $u_{ij}[d, v]$ for the descriptor value pair $[d, v]$, the *power* β for that pair is

$$\beta[d, v] = \log_2(u_{++}[d, v]) - \log_2(u_{+-}[d, v])$$

where $\log_2(x)$ is the base-2 logarithm of x.

By definition of the update terms u_{ij}, all default descriptor values necessarily have a power of zero. In essence, the power of a descriptor-value pair measures how much more likely it is for a positive example to be correctly classified when the descriptor d takes on the value v instead of its default value. This can be made more precise by expanding and manipulating the above definition as follows.

$$\begin{aligned}
\beta[d, v] &= \log_2\left(\frac{\Pr(+, +, d = v)}{\Pr(+, +, d = v_0)}\right) - \log_2\left(\frac{\Pr(+, -, d = v)}{\Pr(+, -, d = v_0)}\right) \\
&= \log_2\left(\frac{\Pr(+, +, d = v)}{\Pr(+, -, d = v)}\right) - \log_2\left(\frac{\Pr(+, +, d = v_0)}{\Pr(+, -, d = v_0)}\right) \\
&= \log_2\left(\frac{\Pr(j = +, i = + | d = v)}{\Pr(j = -, i = + | d = v)}\right) - \log_2\left(\frac{\Pr(j = +, i = + | d = v_0)}{\Pr(j = -, i = + | d = v_0)}\right) \\
&= w(\text{TP:FP}|v) - w(\text{TP:FP}|v_0)
\end{aligned}$$

where the term $w(A : B|C) = \log_2(\Pr(A|C)/\Pr(B|C))$ is the *weight of evidence* [Good, 1965, Jaynes, 2003, §4.8.1] for the event A over the event B given C.[3] An increase of one unit of evidence represents a doubling of the odds of A over B. Here, TP and FP are the events corresponding to a correct and

[3]This is also known as the *log-odds* of A against B given C.

incorrect classification of a positive example respectively. This means that the power is measuring how much the evidence for true positives over false positives changes when the non-default value v is present in a description.

A descriptor-value pair with a high power indicates that rules for which the descriptor takes on that value will, on average, be more likely to classify positive examples correctly. By tabulating all the descriptor powers for a DFT those rule features that correlate with the correct prediction of positive examples can be quickly assessed. These tabulations are used throughout the next chapter to empirically compare tasks. An example tabulation is provided in the next section.

4.5. An Example Application of DEFT

In this section, the consolidation and transfer algorithms described above are applied to tasks drawn from an environment of five concepts. Following the theme used throughout this thesis, the concepts all express reading preferences for books which are described using attributes such as size and genre. These concepts, described in Section 4.5.1 below, are also used in several of the experiments described in the next chapter.

4.5.1. Example Tasks. The five example concepts are shown in Figure 4.2 as Horn clause theories, each consisting of two clauses. A total of four predicates - `genre/2`, `nation/2`, `size/2` and `year/2` - are used to express book properties. The first argument of each predicate is used to refer to a book while the second argument holds the value of the property for that book. Each predicate can take one of three distinct values in its second argument. The `genre/2` predicate can take the values `horror`, `romance` or `scifi`; the `nation/2` predicate uses the values `aus` (Australian), `uk` (United Kingdom) and `us` (U.S.A.); the `size/2` predicate takes values `small`, `medium` and `large`; and the `year/2` predicate has values corresponding to the decades `80s`, `90s`, and `00s`. Each book in this domain takes on exactly one value for each predicate. The concepts in this environment specify the properties a book must have for it to be enjoyed by a particular person. For example, the first two rules in the Figure 4.2 state that Person A will like reading book X if one of two conditions are met. Namely, that the book's genre is science-fiction and its author resides in the UK or that the author's nationality is Australian and the book was published in the 1990s.

Intuitively, these concepts fall into two groups of mutually related theories. The first group contains concepts A, D and E. These all express preferences in terms of genre, nationality and year. All of the concepts in this group mention

Person A	`like(X) :- genre(X,scifi), nation(X,uk).` `like(X) :- nation(X,aus), year(X,90s).`
Person B	`like(X) :- size(X,small), genre(X,romance).` `like(X) :- size(X,medium), genre(X,romance).`
Person C	`like(X) :- size(X,large), genre(X,romance).` `like(X) :- size(X,medium), genre(X,horror).`
Person D	`like(X) :- genre(X,scifi), nation(X,aus).` `like(X) :- nation(X,uk) ,year(X,90s).`
Person E	`like(X) :- genre(X,scifi), nation(X,usa).` `like(X) :- nation(X,aus), year(X,00s).`

FIGURE 4.2. The five example concepts for the Reading Preferences environment.

science-fiction as a genre and the UK or Australia as the author's nationality. The second group contains concepts B and C. These concepts in this group state reading preferences solely in terms of a book's size and genre and both use the constants 'medium' and 'romance' in one of their two rules. Notably, the concepts B and C, while similar, do not share any positive examples since there is no book which is simultaneously a small or medium romance novel (concept B) and a large romance novel or a medium horror novel (concept C).[4]

4.5.2. Descriptions and Consolidation. As described at the beginning of this chapter, the use of DEFT is broken into a consolidation phase and a transfer phase. During the consolidation phase DEFT repeatedly samples clauses and evaluates them on the examples from a support task. In this example run of DEFT the support tasks for concepts A, B, C, D and E each had 50 positive and 50 negative random examples generated using the procedure described in Section 5.2 of the next chapter. A DFT was constructed for each of these support tasks using the BUILDDFT procedure described above. The descriptor templates $\mathcal{T}$ passed to the procedure was the set containing the has_pred/1 and has_arg/3 templates. The sampling parameters S_{exs} and S_{cls} were both set to 20, meaning that a total of 400 clauses were sampled and evaluated during DFT construction. These clauses were restricted to having a maximum of 4 literals and a maximum variable depth of 2. As an example of DEFT's use, the following list of commands were those used to construct a DFT using the parameters just described for the support task E.

[4]The assumption that each book takes on one value per property ensures that there are no novels that are simultaneously romance and horror.

```
?- set(i,2).
?- set(clauselength,4).
?- set(deft_cls_sampled,20).
?- set(deft_exs_sampled,20).
?- set(deft_templates,[has_pred/1,has_arg/3]).
?- set(deft_table,'e.dft').
?- read_all('support_e').
?- deft:sample_clauses.
?- deft:save_table.
```

The settings `i` and `clauselength` are the standard ALEPH parameters which control the maximum variable depth and maximum clause length respectively. The settings `deft_cls_sampled` and `deft_exs_sampled` correspond respectively to the S_{cls} and S_{exs} parameters in the BUILDDFT procedure. The `deft_table` setting specifies the filename used to store the constructed DFT, in this case `'e.dft'`. The final three lines of the above example are commands.

The first command, `read_all('e')`, is a standard ALEPH command used to read in data from the positive, negative and background files for the given file-stem. In this case, the positive examples are stored in the file `'support_e.f'`, the negative examples in the file `'support_e.n'` and the background predicates and mode and type information in the file 'support_e.b'. The mode and type information used in this example is given in Section B.1 of Appendix B.

The command `deft:sample_clauses` is a call to the implementation of the BUILDDFT procedure within DEFT using the parameters specified through the `set/2` predicate. Once completed, the call to `deft:save_table` writes the collected descriptor frequencies out to the file `'e.dft'`. Section B.2 in the appendices shows the contents of this file. The term `total/1` stores the table of counts $\mathbf{t}$ described in Algorithm 7 while the `counts/3` terms store the sample counts $\mathbf{s}[d_k, v_k]$ for each of the non-default descriptor-value pairs $[d_k, v_k]$.

To compute clause descriptions the DESCRIBE procedure applies both descriptor templates to the clause. This results in descriptor-value pairs for each non-default value returned. For example, calling DESCRIBE with the clause `like(B) :- size(B, large), genre(B, scifi)` as input will first apply the has_pred template. This will return an instantiation of the has_pred/2 template for each of the predicates within the rule, namely `size/2` and `genre/2`. For these predicates the descriptor will take on the value "true". The has_arg/3 template is also applied to the rule resulting in one instantiation of the has_arg/3 descriptor for each of the `size/2` and `genre/2` predicates. The final sparse description is represented using the following Prolog list of lists:

```
[[has_pred(size/2),true],
 [has_pred(genre/2),true],
 [has_arg(size,2,large),true],
 [has_arg(genre,2,scifi),true]]
```

Each of the four lists above hold two entries. The first is a descriptor d_k and the second is its non-default value v_k. This representation is much smaller than what would be required if all 16 descriptor-value pairs for the rule were computed in full.

4.5.3. Priors and Descriptor Power. Table 4.1 shows the default table $\mathbf{z}$ and the update tables $\mathbf{u}[d_k, v_k]$ for the has_pred/1 descriptors that are derived from the DFT constructed for all five concepts in the example environment. The values in these tables are used to compute classification priors using the CalcPrior algorithm described above. Using the example description above, the prior for the clause `like(B) :- size(B, large), genre(B, scifi)` derived from the support task E can be computed as the product of the base table $\mathbf{z}_E$ and the update tables for the four descriptors in the aforementioned description. For example, if only the has_pred/1 descriptors were being used then the product would be

$$\begin{aligned} & \mathbf{z}_E \otimes \mathbf{u}_E[size/2, true] \otimes \mathbf{u}_E[genre/2, true] \\ = & \begin{bmatrix} .16 & .73 \\ .04 & .07 \end{bmatrix} \otimes \begin{bmatrix} .12 & .06 \\ .45 & .37 \end{bmatrix} \otimes \begin{bmatrix} .14 & .07 \\ .43 & .36 \end{bmatrix} \\ = & \begin{bmatrix} .0027 & .0031 \\ .0077 & .0093 \end{bmatrix} \end{aligned}$$

where the symbol $\otimes$ denotes a point-wise multiplication of the tables. The prior classification probabilities

$$\mathbf{q} = \begin{bmatrix} .12 & .13 \\ .34 & .41 \end{bmatrix}$$

for this rule using concept E as support are obtained by normalising the result of this product.

The values in the base and update tables can also be used to compute descriptor powers, as described in Section 4.4.5 above. Table 4.2 shows these powers for each of the four has_pred/1 descriptors based on the DFTs constructed for each of the five example concepts. The apparent similarity between concepts B and C and between concepts A, D, and E mentioned at the beginning of this section is reflected in the table. The has_pred/1 descriptors for the predicates `size/2` and `genre/2` have large descriptor power on concepts

TABLE 4.1. The default tables and $has\_pred(P/N)$ update tables derived from DFTs for concepts A, B, C, D and E. The update tables have been normalised for easier comparison. The notation $\mathbf{u}[p, v]$ in this table is shorthand for $\mathbf{u}[has\_pred(p), v]$.

	A		**B**		**C**		**D**		**E**	
z	.17	.71	.05	.91	.08	.87	.15	.79	.16	.73
	.05	.07	.02	.02	.02	.03	.03	.04	.04	.07
$\mathbf{u}[size/2, true]$	.11	.09	.22	.02	.22	.02	.09	.05	.12	.06
	.44	.37	.37	.39	.35	.41	.47	.38	.45	.37
$\mathbf{u}[genre/2, true]$	.16	.07	.25	.02	.24	.04	.16	.06	.14	.07
	.40	.37	.35	.38	.35	.37	.40	.38	.43	.36
$\mathbf{u}[nation/2, true]$	.20	.07	.13	.07	.16	.05	.20	.04	.21	.07
	.38	.36	.43	.37	.40	.39	.38	.38	.36	.35
$\mathbf{u}[year/2, true]$	.25	.09	.14	.17	.14	.23	.27	.12	.34	.10
	.33	.33	.37	.32	.36	.27	.30	.31	.27	.29

TABLE 4.2. The descriptor power of the four has_pred/1 descriptors for each of the five example reading preference concepts.

	Concept				
Power	*A*	*B*	*C*	*D*	*E*
$\beta[size/2, true]$	0.28	3.46	3.46	0.84	1.00
$\beta[genre/2, true]$	1.42	3.64	2.58	1.41	1.00
$\beta[nation/2, true]$	1.51	0.89	1.67	2.32	1.58
$\beta[year/2, true]$	1.47	-0.28	-0.72	1.17	1.77

B and C relative to those obtained for concepts A, D and E. Conversely, the has_pred/1 descriptor for the predicate `year/2` has a larger value on A, D and E than on B and C.

4.5.4. Transfer. During the transfer phase of Deft, priors for each clause are combined with classification probabilities derived from the training set of the target task. In this example, the target task will consist of 2 positive and 2 negative examples from concept A. DFTs for each of the four other concepts (B, C, D and E) will be separately used to construct priors during four separate attempts at learning the target concept from the 4 training examples. As an example, the settings and commands used to run Deft on the task for A using the DFT constructed for concept E are given below.

```
?- set(i,2).
?- set(clauselength,4).
?- set(evalfn,deft_m).
?- set(deft_m, 4).
?- set(deft_evalfn, coverage).
?- read_all('target_a').
```

TABLE 4.3. Comparison of scores given to acceptable rules for task A. The **bold** entries show the highest score in each column.

		Coverage		DEFT Coverage			
ID	**Rule Body**	**Train**	**Test**	**B**	**C**	**D**	**E**
a	`size(A,medium)`	1.0	0.0	1.3	1.4	1.0	0.8
b	`nation(A,uk)`	1.0	22.0	0.6	0.8	**1.4**	1.0
c	`size(A,medium),genre(A,scifi)`	1.0	6.0	**1.4**	**1.5**	1.1	1.0
d	`size(A,medium),nation(A,uk)`	1.0	2.0	1.2	1.2	1.2	1.0
e	`genre(A,scifi),nation(A,uk)`	1.0	**47.0**	1.1	1.2	1.3	**1.2**
f	`size(A,medium),genre(A,scifi),` `nation(A,uk)`	1.0	11.0	1.2	1.2	1.1	1.0

```
?- deft:read_table('e.dft').
?- induce.
```

The first two settings here were the same as those used to construct the DFT in the example above. They are standard ALEPH settings to limit the set of legal clauses during its search. The `evalfn` setting is also a standard ALEPH setting which specifies the evaluation function used to assess the quality of clauses. In this case, it is set to '`deft_m`' meaning that DEFT should be used to modify the evaluation function specified in the `deft_evalfn` setting using $M = 4$ as given by the `deft_m` setting. The evaluation function that is modified by DEFT is the "coverage" function that is used by ALEPH by default. The assessment of the quality of a clause r using this function is simply $n_{++}(r) - n_{+-}(r)$, that is, the number of positive examples covered less the number of negative examples covered.

Table 4.3 lists acceptable rules for concept A, in the order in which they were tested during a search using ALEPH. The test set used for this concept consisted 50 positive and 50 negative examples drawn at random. The columns in the table titled "Train" and "Test" show the coverage score of rules on the corresponding set. Unsurprisingly, rule e has the highest score of 47.0 on the test data as it is one of the clauses used to define concept A. On the training set however, all of the rules have a coverage score of 1.0. This means each rule covered either two positive training examples and one negative or one positive and no negative examples. The search policy employed by ALEPH only replaces its current best rule if another one has a strictly greater score. This results in the first rule, rule a, being returned, a sub-optimal choice since its test set score is 0.0, indicating that it covers an equal number of positive and negative test examples.

The last four columns in the table show the quality score given to the clauses by coverage function modified using priors derived from DFTs for tasks

B, C, D, and E. When B and C are used as secondary tasks, the rule returned by the search is *c*. In terms of test set score this is slightly better than rule *a*. When D is used as a support task, rule *b*, with a test set score of 22.0 is returned. The rule with the highest test set score, rule *e*, is returned when E is used as a support task.

Once again, this result is in line with the intuition given at the beginning of this example. The use of concept D and concept E as support tasks resulted in a ranking of clauses by quality scores that was more in line with the "true" quality, as estimated on the test set. In the next chapter further experiments are carried out with these artificial concepts in order to better understand how DEFT's various settings affect the quality of the clauses it induces.

In theory, there is no difference between theory and practice.
But, in practice, there is.

- Jan L. A. Snepscheut (1953-1994)

CHAPTER 5

Empirical Evaluation

This chapter aims to identify and reconcile the several differences between the theory of description-based transfer described in Chapter 3 and its implementation as DEFT in Chapter 4. Although the theory provides conditions for when description-based transfer will reduce evaluation error it is not clear that, due to the simplifying assumption of independent descriptors, the same will be true for DEFT. Also, the theoretical effect of transfer on evaluation was studied independently of the other aspects of a learning system including its search and evaluation biases. The evaluation function, pruning heuristics, representation and constraints used by a learner all determine whether the final theory induced is a good generalisation of the examples. How much of an improvement to accuracy a reduced evaluation error can make with all the other parts in play is a question best answered empirically. Further questions which are glossed over in theory but crucial in practice involve how to best select tasks and settings when using DEFT in practice. Answers to all of these depend, to some extent, on the environment of tasks to which DEFT is applied.

To investigate these questions DEFT was applied to four different sets of learning tasks. The first is based on the five artificial concepts regarding reading preferences that were introduced in the previous chapter. The results of these experiments, reported in Section 5.2 below, show how DEFT compares with a standard, single-task learner under a range of training set sizes and learning parameters. Section 5.3 reports DEFT's performance on an environment of chess tasks that has been used previously to test other inductive transfer approaches to rule learning. The experiments on this domain compare DEFT to the "Repeat Learning" system described in [Khan et al., 1998] as well as a combination of DEFT and Repeat Learning. In Section 5.4, results of experiments with DEFT on a medical domain are reported. In this domain learners are required to predict whether a patient has heart disease based on a handful of their symptoms and features. The tasks in this domain have also been used previously by Silver [2000] to investigate inductive transfer in artificial neural networks. Finally, experiments reported in Section 5.5 describe attempts to transfer inductive bias between the ILP benchmark problems mutagenesis [Srinivasan et al., 1994] and carcinogenesis [Srinivasan et al.,

1997].

5.1. Overview

The goal of description-based transfer is to be a general and practical method for a domain expert to modify the evaluation bias of a learner through the selection of a support task. When done appropriately, the modified evaluation bias should allow the learner to induce theories with good generalisation performance from a limited number of training examples. For this to be a practical way to improve performance it is important that it be inexpensive, both computationally and in terms of the decision-making required of the expert using it. Running times during the consolidation and transfer phases should not be prohibitive, choosing good support tasks should be intuitive and successful transfer should be robust with respect to small changes to DEFT's settings. It would also be advantageous for DEFT to be able to be used in conjunction with other bias modifications and inductive transfer techniques.

5.1.1. Questions. The main aims of the experimental evaluation of DEFT are driven by the following sets of questions:

Does it work? Given a target task with only a small number of training examples does transfer from a related support task actually improve generalisation performance? Is this a result of a reduction in CPM error? Does this occur between tasks which are intuitively similar in some respect?

Is it practical? Are the overheads incurred by DEFT (such as DFT construction time and computing descriptions) acceptable? How do they compare to the analysis of the previous chapter? Is positive transfer robust with respect to small changes in sampling settings or the examples used in a support task?

How does it compare to other methods? Does DEFT perform any better or worse than other types of inductive transfer for rule learning, including direct transfer? Can DEFT be used in conjunction with other systems? What are the relative strengths and weaknesses of DEFT compared to other modes of transfer?

The remainder of this chapter seeks to answer these questions empirically by running DEFT on tasks drawn from the four environments described above. All the experiments were all run on a Mac PowerBook (1.33GHz G4, 768Mb RAM running OS 10.3.9). The implementation of ALEPH used as a baseline learner and as the base-level learner for DEFT was a modified version of ALEPH 4.0 [Srinivasan, 2001]. It was run on top of Yap Prolog version 4.5.7 [Costa et al., 2005]. Much of the statistical analysis reported in this chapter was done

using the statistical software *R* [R Development Core Team, 2005] with the extension package *Hmisc* [Harrell, 2005].

5.2. The Reading Preferences Environment

The artificial reading preferences environment introduced in Section 4.5.1 of the previous chapter is a simple set of concepts which can be used to test various aspects of the DEFT algorithms. Each concept is described using two rules and each of these rules uses two of the four available predicates: `genre/2`, `nation/2`, `size/2` and `year/2`. These predicates can be instantiated with one of three different constants. These five concepts (summarised in Table 5.1) were chosen so as to have two subsets of intuitively similar tasks: concepts A, D, and E are all defined in terms of a book's genre, nation and year whereas the rules for concepts B and C test only the genre and size of a book.

The main hypothesis tested in this section is whether transfer between these intuitively similar tasks has a positive effect on learning when examples are scarce. This is established in Experiments RP-1 and RP-2. The first measures the performance of the baseline learner ALEPH on tasks of various sizes while the second applies DEFT to the same set of tasks when using a variety of support tasks. The remainder of the experiments investigate how stable the transfer effects are when the parameters used by DEFT and the intrinsic features of the support tasks are varied. In particular, Experiment RP-3 examines how transfer is affected by the size of a support task, Experiment RP-4 varies the sampling parameters used in DFT construction, Experiment RP-5 looks at the importance of the admissibility condition, Experiment RP-6 tries different combinations of descriptor templates and Experiment RP-7 investigates the influence of the M parameter on transfer.

5.2.1. Materials. The experiments in this section measure the effects of inductive transfer using DEFT on learning performance and efficiency. To do this, target and support datasets are required for each of the five concepts shown in Table 5.1. For each concept $T \in \{A, B, C, D, E\}$, the following method was used to generate the datasets:

(1) First, 10,000 book instances were created by assigning a random value to each of the four attributes: size (small, medium or large), genre (scifi, romance, horror), nation (aus, uk, usa) and year (00s, 90s, 80s). The values were drawn uniformly and independently for each attribute.
(2) The 10,000 instances were then classified according to the concept T and split into positive examples T^+ and negative examples T^-.

TABLE 5.1. A tabular representation of the rules describing the five reading preference concepts. The columns show the four predicates used in the domain while each row shows the constants used in those predicates for each rule. A blank entry in a row indicates the predicate was not used in the corresponding rule.

	genre	nation	size	year
A	scifi	uk		
		aus		90s
B	romance		small	
	romance		medium	
C	romance		large	
	horror		medium	
D	scifi	aus		
		uk		90s
E	scifi	usa		
		aus		00s

TABLE 5.2. Distribution of class labels over the randomly generated instances for each of the five Reading Preference concepts

	#. of Pos.	# of Neg.	Total
A	2240	7760	10000
B	2201	7799	10000
C	2196	7804	10000
D	2334	7666	10000
E	2248	7752	10000

Table 5.2 shows the distribution of positive and negative instances for each of the concepts.

(3) The first 1500 positive and first 1500 negative examples were taken from T^+ and T^- respectively and split into 30 pairs of sets T_t^+ and T_t^- for $t = 1, \ldots, 30$. These pairs were used to create 30 tasks $T_t = T_t^+ \cup T_t^-$ each with 100 examples.

(4) The next 100 positive examples from T^+ and the next 100 negative examples from T^- were designated as test examples and placed into the sets T_{test}^+ and T_{test}^-. Each pair was combined to create a single, 200 example test set $T_{\text{test}} = T_{\text{test}}^+ \cup T_{\text{test}}^-$ for each concept.

(5) Each of the 30 sets T_t were then used to create 240 tasks $T_{t,N}$ with sizes $N \in \{4, 6, 10, 20, 30, 40, 60, 100\}$. Each $T_{t,N}$ consists of $\frac{N}{2}$ examples which were drawn randomly and without replacement from T_t^+ and another $\frac{N}{2}$ were drawn in the same manner from T_t^-.

The end result of this process is the construction of $5 \times 8 \times 30 = 1200$ learning tasks. Each task is uniquely identified by its target concept, the number of instances it has and the index used to select them. As an example, the 12th task for concept B with 40 training examples is denoted $B_{12,40}$. These tasks and nomenclature are used throughout the experiments in this section.

It is important to note that the test sets for each concept were constructed to have 100 positive and 100 negative examples even though a randomly created instance is three to four times more likely to be classed negative than positive. This is to ensure the accuracy reported for a theory on one of these test sets is equivalent to its AUC, that is, the average of its true positive and true negative rates. As discussed in Section 2.1.3 above, this measurement of classification performance is invariant under changes in class distributions and so is better suited for comparison across learning tasks. The terms "accuracy" and "AUC" are therefore used interchangeably for the rest of this section.

5.2.2. Algorithms and Settings. Several learning parameters for ALEPH remain constant throughout all of the experiments in this section. These are shown in Table 5.3 and are used when ALEPH is used by itself and as a base-level learner for DEFT. The settings and their values are discussed in Section 2.3.2.

Since the aim of these experiments is to look at the relative performance of ALEPH and DEFT these settings were chosen somewhat arbitrarily but used consistently across both algorithms. They were, however, checked to ensure that each of the target concepts could be induced by ALEPH with high confidence from a training set of 30 positive and 30 negative examples.

TABLE 5.3. Settings used by ALEPH and DEFT for experiments within the Reading Preferences domain

Learner	**Setting**	**Value**
ALEPH	*clauselength*	4
	i	2
	minacc	1.0
	nodes	200
BUILDDFT	*admissibility*	true
	exs_sampled	20
	cls_sampled	20
	templates	{has_pred, has_arg}
DBSCORE	M	$\sqrt{N}$

The settings used in DEFT's consolidation and transfer algorithms are also factors which can influence the results of a learning trial. Several of the experiments in this section are constructed to empirically determine the sensitivity of learning performance and complexity to these factors. The design of these sensitivity experiments is to vary one of these settings while keeping all the others fixed at their default values. The default values for the DEFT algorithms are also shown in Table 5.3 and are explained in the previous chapter. The choices for these default values are now briefly discussed.

The default *templates* setting used in this domain was set to has_pred/1 and has_arg/3 as these generate simple, binary descriptors which appear to capture relevant features of the rules describing the concepts. As descriptors generated from the has_pred/1 template are unbiased the theory from Chapter 3 suggests a default value for the *admissibility* setting be "true" meaning only positive examples are sampled when constructing a DFT. Default values of 20 were chosen for the other two sampling settings *exs_sampled* and *cls_sampled*. The default value of 20 for *exs_sampled* was chosen so that DEFT samples about half of the 50 positive examples in each support task. The default setting of 20 for *cls_sampled* was chosen to make the total number of samples $t > 16590$ as per the analysis in Section 4.3 of the previous chapter. As there are 50 positive and 50 negative examples, choosing *exs_sampled* and *cls_sampled* both equal to 20 gives $t = 20000$.

5.2.3. Experiment RP-1: Baseline Learning Performance. The purpose of inductive transfer algorithms is to modify a learner's bias so as to improve learning performance when data is limited. Any empirical study of these algorithms must therefore compare the learner with a modified bias against the same learner with its original bias. The data gathered in this experiment creates a baseline for against which any changes due to DEFT can be compared.

5.2.3.1. *Method.* ALEPH was run using the above settings shown in Table 5.3 on the target tasks generated with the procedure described in Section 5.2.1 above. For each target concept $T \in \{A, B, C, D, E\}$, training size $N \in \{4, 6, 10, 20, 30\}$ and task index $t = 1, \ldots, 30$ ALEPH was run on the task $T_{t,N}$. The theory induced by ALEPH on task $T_{t,N}$ was then tested on the target concept's test set T_{test} to obtain an estimate of its accuracy.

As well as accuracy, the training time, number of clauses evaluated, theory size and other measurements were recorded for each of the runs. All of these values can be viewed as random variables which are dependent on the parameters T, N and t. By averaging over task index t in each case, statistics such

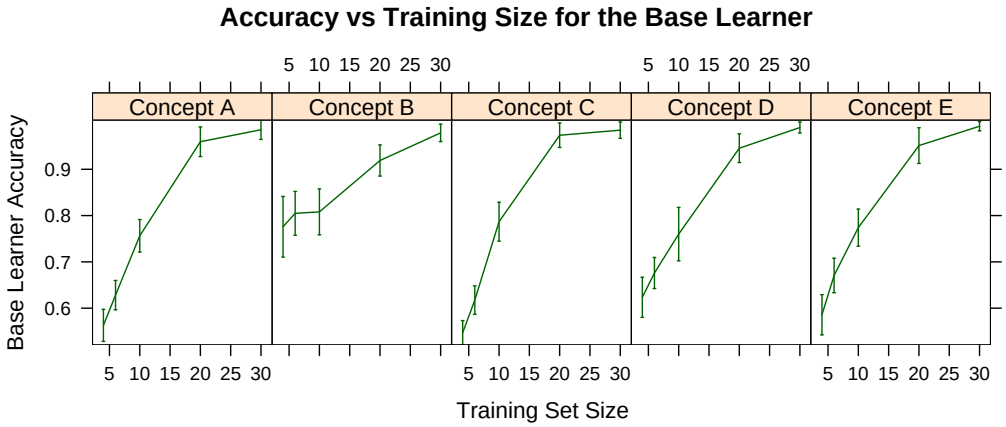


FIGURE 5.1. Base learner test accuracy on each of the five reading preferences concepts. Error bars show 95% confidence intervals for the mean test accuracies.

as means and confidence intervals were computed for these variables for each target concept and training size.

5.2.3.2. *Results.* Figure 5.1 shows the mean test accuracies and their 95% confidence intervals for hypotheses induced by ALEPH for each of the five target concepts at a range of training set sizes. Unsurprisingly, the graph for each concept shows roughly the same trend: as the number of examples increase from 4 to 30 the accuracy increases from around 0.5 (random guessing) to 1 (perfect classification). The only exception is Concept B where the accuracy at a training size of 4 is around 0.75 indicating it is much easier to learn from small datasets than the other four concepts. This is to be expected since it is the only one of the five concepts whose positive instances can all be covered with a single rule, namely `like(B) :- genre(B, romance)`.

The results show that generalisation accuracy is generally poor when tasks for these concepts have ten or fewer examples. These are therefore the "limited data" tasks on which DEFT is expected to help improve learning performance when appropriate support tasks are chosen. The next experiment tests this theory by using DEFT with each of the 25 possible support/target task pairs from this environment.

5.2.4. Experiment RP-2: Inductive Transfer with DEFT. The aim of this experiment is to determine whether the use of support tasks with DEFT has any appreciable influence on learning when compared to the standard ALEPH algorithm. The five concepts in this environment differ substantially, so it was expected that the use of DEFT would improve the accuracy when used to transfer bias between some pairs of tasks and have a negative effect

on accuracy when used for transfer between other pairs. As discussed in the beginning of this section concepts A, D and E appear to be intuitively similar, in that they can be described using similar predicates and constants however they appear quite different to the concepts B and C. It is therefore expected that DEFT will improve generalisation accuracy when transferring evaluation bias from a task for D to a task for A which has limited data. It is also likely that transfer from a task for B to a task for A will harm generalisation performance.

5.2.4.1. *Method.* The following method was used to test the effects of transfer using DEFT by applying it to the 25 possible pairs of support and target concepts for a range of training set sizes for the target tasks. The basic idea is to first create DFTs from tasks for each of the concepts A through E. Five tasks, each with 100 examples, were chosen for each concept. The large number of examples ensures conditions are favourable for producing reliable DFTs and five DFTs are created per concept to estimate the variability in the transfer results due to the randomised aspects of their construction. Once the DFTs are created DEFT is applied to each of the variously sized tasks for each concept while using each of the DFTs. The inferred theories in each case are then tested on the test dataset for each concept and their accuracy recorded. The exact details of the entire process are as follows.

(1) For each support concept $S \in \{A, B, C, D, E\}$, five DFTs $\Phi[S, s]$ were created by applying DEFT's BUILDDFT algorithm to each support dataset $S_{s,100}$ for $s = 1, \ldots, 5$. The settings used during this phase are given in Table 5.3 above.
(2) For each of the target concepts $T \in \{A, B, C, D, E\}$, training sizes $N \in \{4, 6, 10, 20, 30\}$ and task indices $t = 1, \ldots, 6$ DEFT was run on $T_{t,N}$ using each of the 25 DFTs created in the previous step. DEFT's M parameter was set equal to the square root of the target task size in each case, that is $M = \sqrt{N}$.
(3) Each theory induced by DEFT on task $T_{t,N}$ was then evaluated on the concept's test set T_{test} to estimate its accuracy.

As well as test set accuracy, measurements such as the number of nodes explored, search time, and theory size were recorded for each run. Each of these measurements was treated as a random variable dependent on the parameters N, T, S, t and s. Their sample means and standard deviations were computed for each support concept S, target concept T and training size N by averaging over the 30 pairs of target and support task indices $t = 1, \ldots, 6$ and $s = 1, \ldots, 5$.

5.2.4.2. *Results.* There are four main sets of results to report for this experiment. The first of these concerns the running time and output of the BUILDDFT algorithm. The second set compares the generalisation performance using DEFT to the baseline learner ALEPH. The final two sets of results report the search and theory complexity.

DFT Construction: As per the method described above, the BUILDDFT algorithm was applied to 25 support tasks resulting in five DFTs for each concept. On average, each DFT took 1.4 seconds to construct with a sample standard deviation of 0.06 seconds across the 25 runs. This lack of variability is to be expected since on each support task BUILDDFT is simply sampling and testing 400 clauses against 100 examples.

Of more interest is how the descriptor frequencies vary across the five different support concepts. Rather than tabulate the descriptor frequencies for all 25 tables, more insight can be gained by examining the average descriptor probability and descriptor "power" across each of the five concepts. The former is the frequency with which a descriptor takes on its non-default value during the sampling process while the latter, introduced in Section 4.4.5 above, is a measure of the influence a descriptor has on the final CPM for a rule. An increase by one of the power of a descriptor-value pair corresponds to a doubling in the odds factor of a positive prediction being correct when compared to the odds factor when the descriptor takes on its default value.

Table 5.4 shows the average values of both these statistics for all five concepts. Notably, some descriptors are not recorded for all concepts. For example, the `has_arg(genre,2,horror)` and `has_arg(genre,2,scifi)` descriptors do not appear in the DFTs constructed for concept B. The reason for this is that the admissibility constraint used when constructing these DFTs means only positive examples of concept B are sampled and all of these have romance as their genre. When DFTs for concept B are used as biases for learning, these missing descriptors will have no effect on the CPM priors generated for any rule and can be thought of as having power $\beta = 0$.

Comparing the relative powers and probabilities of descriptors across the concepts reveals that the DFTs are capturing some of the intuitively similar features of the concepts. For example, for concepts B and C a rule that tests whether a book's genre is romance is an indication of a correct positive class prediction (β = 4.4 and 2.4 respectively) whereas this descriptor is not useful for concepts A, D and E (β = -0.6, -0.7, and -0.5 respectively). The DFTs for A, D, and E all assign high descriptor power (β = 2.5, 2.0 and 2.1 respectively) to the descriptor which tests whether a book's genre is science fiction while the DFTs for B and C do not record entries for that descriptor at all. Similar

TABLE 5.4. Average DFT statistics for the five reading preference tasks. The first row in each group (genre, nation, size, and year) gives the power β and probability π for the corresponding `has_pred` descriptor. The remaining rows in each group show the same statistics for the `has_arg` descriptor that tests the value of the predicate's second argument. All values are averages over the five DFTs constructed for each concept.

		Task									
		A		*B*		*C*		*D*		*E*	
has_pred	**has_arg**	β	π	β	π	β	π	β	π	β	π
genre	-	2.1	0.74	4.3	0.74	3.3	0.72	1.5	0.73	1.8	0.73
	horror	-0.8	0.09	-	0	2.5	0.44	0.2	0.14	0	0.07
	romance	-0.6	0.08	4.4	0.74	2.4	0.28	-0.7	0.11	-0.5	0.12
	scifi	2.5	0.59	-	0	-	0	2.0	0.48	2.1	0.54
nation	-	2.2	0.65	1.1	0.66	1.2	0.65	2.1	0.66	2.2	0.66
	aus	0.3	0.25	0.8	0.22	0.4	0.23	2.2	0.37	0.9	0.33
	uk	2.7	0.40	0.2	0.21	1.1	0.22	1.0	0.29	-	0
	usa	-	0	1.6	0.23	0.8	0.20	-	0	2.4	0.32
size	-	1.1	0.80	2.8	0.81	3.3	0.80	0.9	0.80	1.1	0.78
	small	0.6	0.31	1.9	0.36	-	0	0.7	0.27	0.4	0.25
	medium	0.4	0.25	2.0	0.45	2.7	0.48	0.3	0.32	0.9	0.30
	large	0.7	0.24	-	0	1.8	0.32	0.5	0.21	0.4	0.22
year	-	0.0	0.42	-0.1	0.42	-0.2	0.43	0.4	0.41	0.7	0.43
	80s	-0.7	0.10	-0.5	0.12	0	0.16	-0.8	0.08	-0.9	0.11
	90s	0.8	0.20	0	0.18	-0.5	0.14	1.0	0.32	-1.1	0.04
	00s	-0.6	0.15	-0.3	0.12	-0.5	0.13	-0.3	0.21	1.2	0.28

differences occur between these two groups for descriptors which test for the presence of a nation/2 predicate ($\beta \approx 2$ for A, D and E, $\beta \approx 1$ for B and C) and the size/2 predicate ($\beta \approx 3$ for B and C, $\beta \approx 1$ for A, D and E).

The probability π of a descriptor in a DFT indicates how frequently a descriptor took on its non-default value. That is, the proportion of times that descriptor was applied to a rule that was sampled during the DFT construction and the result of that application was its non-default value. A powerful descriptor will not influence the results of a search if the rules it pertains to are very rare. This observation suggests that the product of the descriptor power and its probability, that is its "weighted power" $\beta\pi$, is a reasonable measure of both a descriptor's influence and relevance. As an example, the descriptor that tests whether a book's year is in the 1990s has a moderate negative power ($\beta = -1.1$) for concept E but occurs very infrequently ($\pi = 0.04$) giving it a weighted power of -0.04, implying this test is of limited use when deciding whether a rule will correctly predict a positive example of this concept.

Figures 5.2 and 5.3 are a visual comparison of the weighted power for each

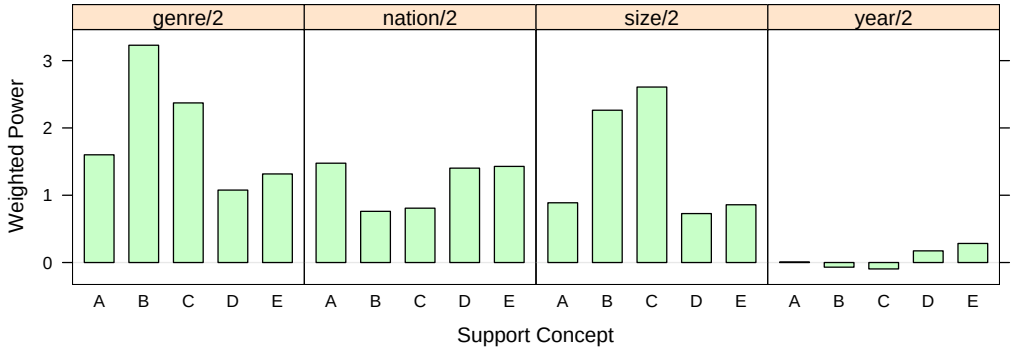


FIGURE 5.2. Average weighted power of `has_pred` descriptors for the five target concepts. The average is taken over the five tasks constructed for each of the five concepts.

of the descriptors across all of the concepts in the reading preference domain. The first shows the weighted power for the has_pred descriptors while the second is for the has_arg descriptors. The title above each subgraph denotes the descriptor in question while each bar in a graph shows a concept's average of $\beta\pi$ taken over the five DFTs generated for it.

Several of these bar graphs make clear the similarities between the concept groups {A, D, E} and {B, C}. In particular the graph labelled "genre = scifi", corresponding to the descriptor has_arg(genre, 2, scifi), has a weighted power of $\beta \approx 1$ for all concepts in the first group and a weighted power of zero for concepts B and C. The other graphs that characterise these differences are "size = medium" and "genre = romance". Within the {A, D, E} concept group the descriptors testing a specific value of nation can be seen differentiating between the concepts: A has a high "nation = uk" power, D a high "nation = aus" power, and E a high "nation = usa" power. This is unsurprising given the rules each of the concepts were generated from and suggests that the BUILDDFT algorithm is correctly discovering salient features of good quality rules for each concept. While this is encouraging, the real aim of this experiment was to determine whether using the DFTs constructed for these concepts as input to DEFT can improve learning performance.

Generalisation Performance: As described in the method above, DEFT was used to learn theories on each of the five reading preference concepts at a variety of training set sizes and with each of the 25 DFTs just analysed. A total of $5 \times 5 \times 5 = 125$ support, target, and training size configurations were run, each repeated 30 times (six different example sets per size and five DFTs per support concept) and the resulting generalisation accuracy was recorded.

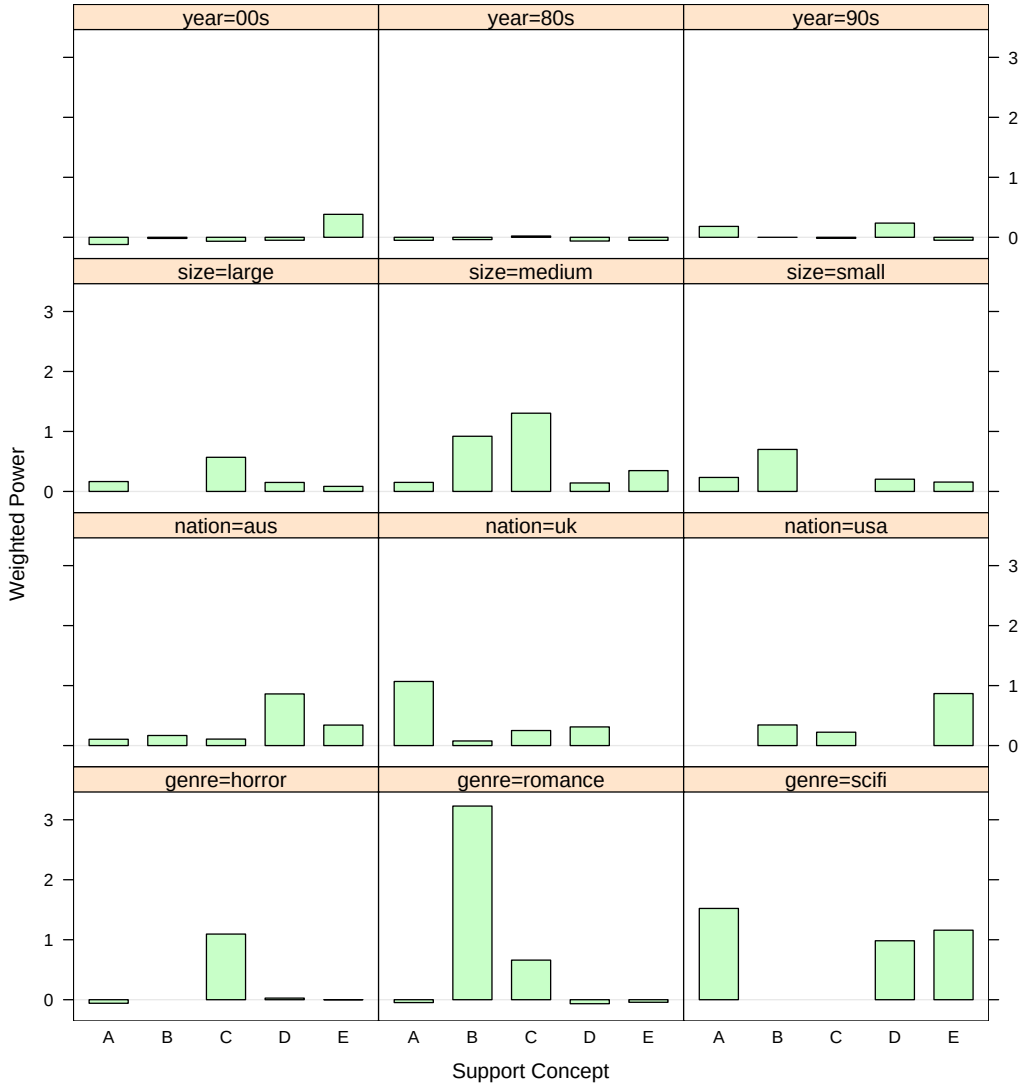


FIGURE 5.3. Average weighted power of `has_arg` descriptors for the five target concepts. Each graph entitled "pred=value" is for the descriptor `has_arg`(pred,2,value). The average is taken over the five tasks constructed for each of the five concepts.

The 125 accuracy measurements are difficult to present in a meaningful way in a tabular form so the 25 graphs shown in Figure 5.4 are given as an overview. Each of the 25 labelled subgraphs show the accuracies for a particular support/target concept pair at each of the eight training set sizes.[1] The column specifies which support task is used while the row specifies the target concept. Two plots - one solid, the other dashed - are shown for each of the 25 support/target pairs. The solid lines plot accuracy against training set size for DEFT when transferring bias between the support and target tasks while the dashed lines are for reference and are identical to the ALEPH accuracy results from Figure 5.1. As ALEPH is not affected by the choice of support task these reference plots are the same within each row. The error bars shown at each point on the plots are the 95% confidence intervals for the mean accuracy.

Some qualitative statements above the relative performance of DEFT and ALEPH can be drawn from Figure 5.4. Firstly and broadly, it is clear that on the smaller example sets the use of DEFT has a significant effect on accuracy when compared to single-task learning and the direction and magnitude of the effect are dependent on which concepts are used as support. The most dramatic improvements occur for transfer from concept D to concept A when the target task has 6 examples and from concept B to a 10 example task for concept C. In the former case accuracy increased from 0.63 to 0.90 (significant at $p < 0.001$ using a two-sided t-test), while in the latter case the increase is from around 0.77 to 1.0 ($p < 0.001$) meaning a perfect theory is being induced from only 10 examples - a third of what is normally required to induce high quality theories for concept C. The largest decreases in accuracy due to the use of DEFT can be seen when DFTs from concepts B and C are used to bias learning on tasks of size 20 for concept A. In these cases the decrease in accuracy is from 0.96 to 0.86 ($p < 0.001$) using B as support and to 0.84 ($p < 0.001$) using C as support. A further analysis of these results and a comparison to the theoretical guarantees regarding CPM error are reported in Section 5.2.10 below.

In general, it can be said that the effects on generalisation performance due to DEFT are compatible with the intuitive grouping of concepts into the sets of similar concepts {A, D, E} and {B, C}. Transfer between two concepts within one of these sets typically results in an increase in accuracy or at worst no change. Transfer between concepts in different sets never resulted in an increase in accuracy and when the support task was B or C transfer was slightly

[1]The graphs along the main diagonal are for when the support and target task are the same concept. While this is not a realistic case in practice it does provide a best case scenario for inductive transfer.

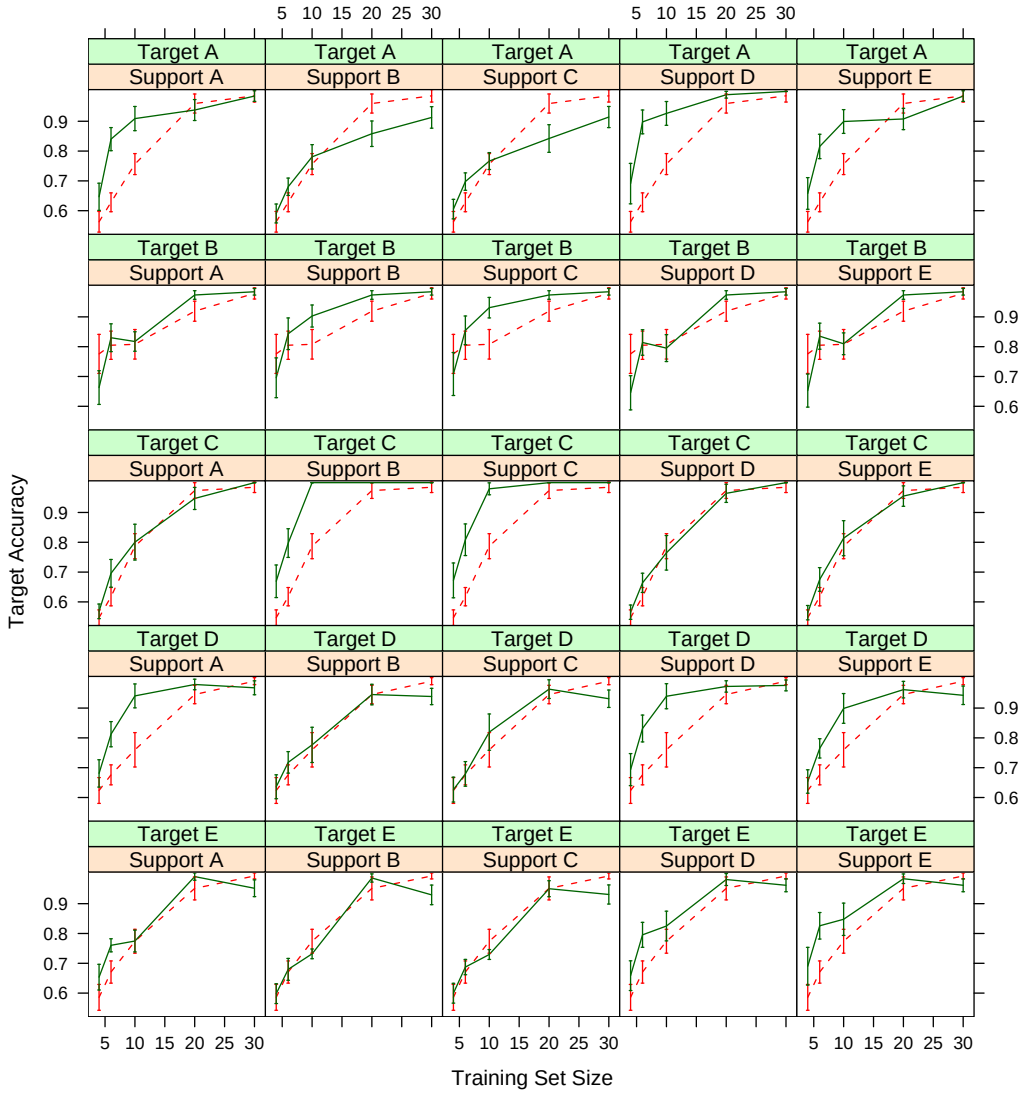


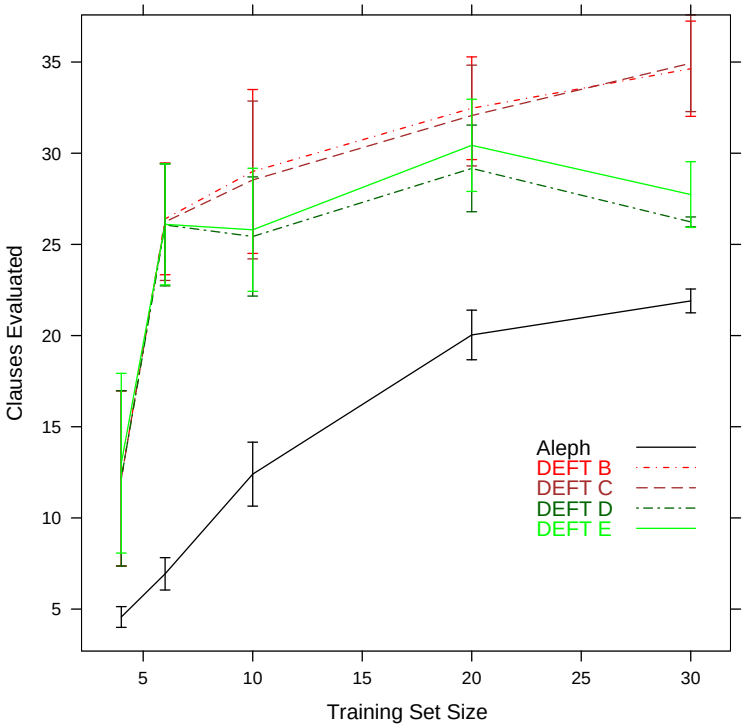
FIGURE 5.4. Target task accuracies for ALEPH (dashed line) and DEFT (solid line) as a function of training set size for all pairs of target and support tasks in the reading preferences environment. Error bars show 95% confidence intervals for the means.

detrimental to learning. Transfer from A, D or E to B or C had no significant effect in either direction.

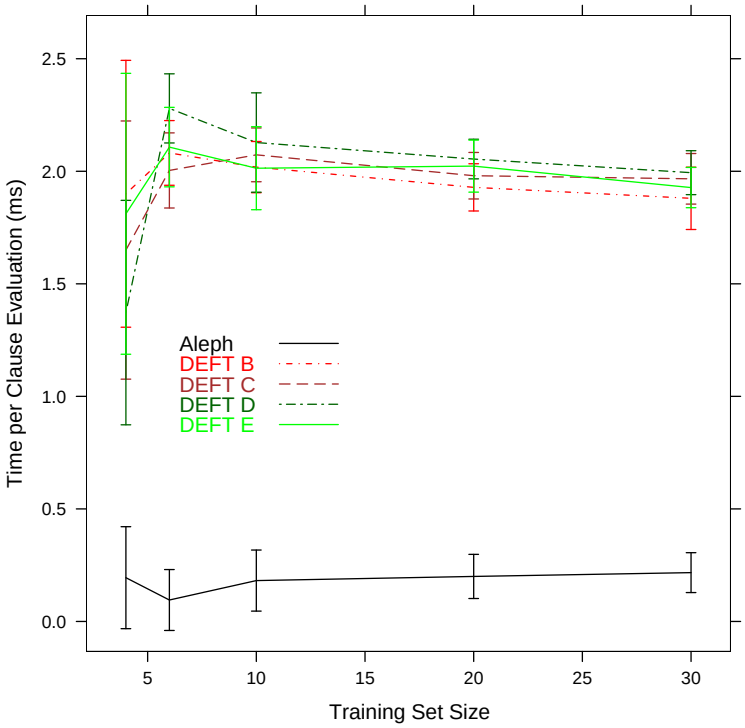
Search Complexity: The number of clauses evaluated and the total time taken to induce each hypothesis were recorded on each run of DEFT and ALEPH. The following set of reported results are only concerned with target tasks drawn from concept A with support tasks drawn from concepts B, C, D, and E. These were chosen as they are representative of the results obtained on the other runs. The graphs in Figure 5.5 provide a summary of the theory and search complexity for target tasks drawn from concept A. The left graph plots the average number of clauses that were evaluated during searches by ALEPH and DEFT using support tasks B, C, D, and E at a range of training set sizes. The graph on the right is organised in the same manner and shows the average CPU time spent evaluating each clause. This evaluation time per clause for each run was calculated by dividing the total running time required to induce a theory by the total number of clauses evaluated during the induction. This time per clause value was then treated like the other trial statistics and averaged over all the tasks attempted for a given support concept, target concept and training size.

It is clear from Figure 5.5 (a) that using DEFT incurs a penalty in the form of larger search complexity. On training sets with six examples the base learner ALEPH found clauses to add to the induced theory after only examining, on average, around six candidate clauses. In contrast, DEFT explored over 25 candidate clauses regardless of the support task being used. This difference in search complexity is entirely due to the modifications of the SCORE and BOUND procedures described in Section 4.1.3 of the previous chapter. Both of these procedures are used to guide the branch-and-bound search used by ALEPH and the modifications weaken the bound by assuming that refinements of a clause during the search could potentially have a perfect prior CPM. This makes it more difficult for the search to prune portions of the candidate space which results in more candidates being evaluated. One possible way to strengthen this bound and reduce the search complexity is discussed in the next chapter.

The effectively constant overhead in evaluation time per clause due to DEFT, as shown in Figure 5.5 (b), agrees with the analysis of the DESCRIBE procedure in Section 4.2.4. For this domain the overhead is of the order of a couple of milliseconds suggesting that the computation of a clause's description is quite efficient though still more time consuming than evaluating a clause against a training set. The reason clause evaluation is so efficient is that ALEPH caches evaluation results for a clause and reuses them when



(a) Clauses evaluated during search



(b) Evaluation time per clause

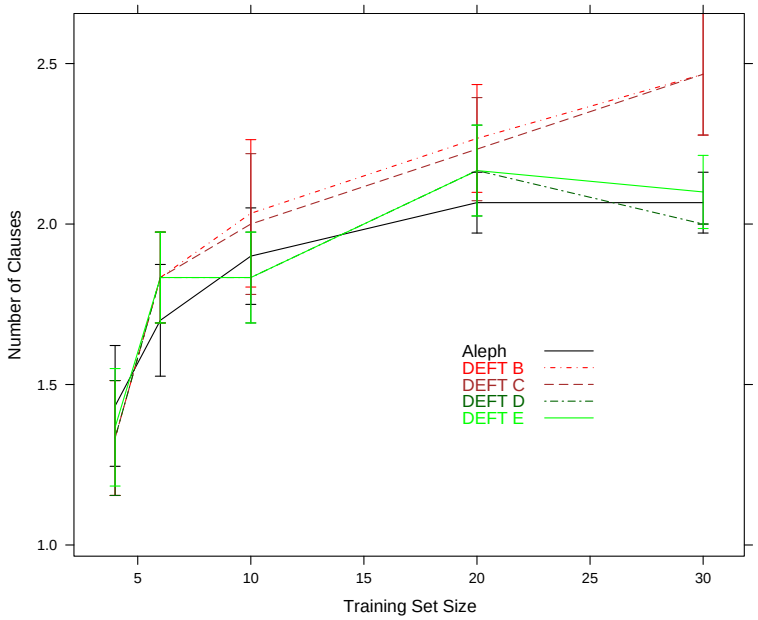
FIGURE 5.5. Search and time complexity on tasks at a range of training set sizes for Concept A. Each graph shows values for ALEPH and DEFT using support tasks B, C, D and E. Error bars show 95% confidence intervals

computing evaluations for the clause's refinements. Given some fairly weak requirements regarding descriptors, a similar caching strategy could be employed to improve the efficiency of computing clause description during a search. This improvement is also outlined in the next chapter.

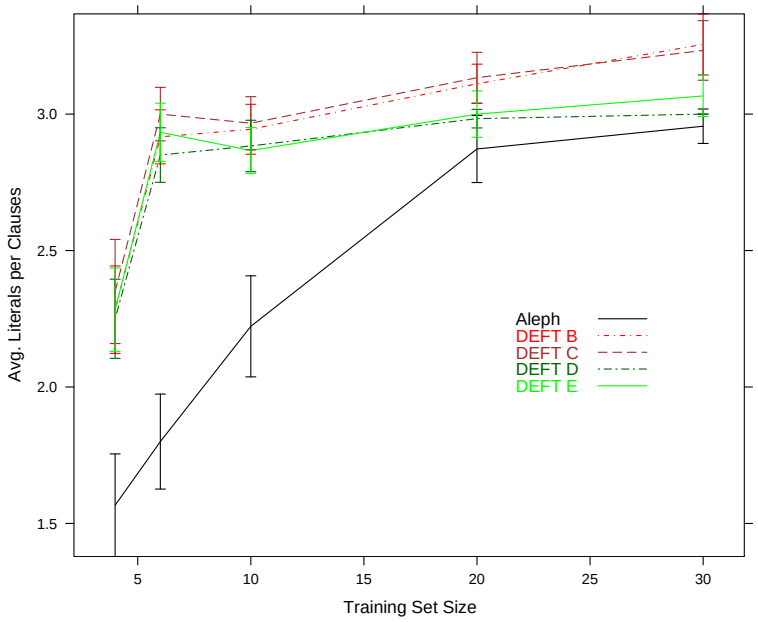
Theory Complexity: Counts of the total number of clauses and literals in each induced theory were recorded at the end of each run of DEFT or ALEPH on a target task. These values were then averaged over all the tasks for a given support concept, target concept and training size. Once again, the key points can be made with reference to results for target tasks from concept A. These statistics are summarised in the graphs of Figure 5.6. The five plots on the left (one for ALEPH and four for DEFT using support tasks B, C, D, and E) show the number of clauses induced from training sets at various sizes. The corresponding plots on the right show the average number of literals per clause. These figures were obtained by dividing the total number of literals (including head literals) by the total number of clauses in each theory.

There is a noticeable difference between the average number of literals induced by DEFT compared to the number induced by ALEPH on the smaller target tasks regardless of the support task that was used. This is to be expected since all the support task DFTs have at least one has_pred descriptor with a power of two or more. Evaluation functions that use CPM priors constructed from these DFTs will therefore assign a higher score to clauses which include a gainful predicate compared to those which do not. On target training sets of size six, the clauses include an average of around two body literals compared to the 0.6 body literals on average when no inductive transfer is used. This ability of DEFT to return larger clauses even though the data does not support it shows that its modification the evaluation function implements a preference bias for longer clauses. It is this bias that allows DEFT to achieve higher generalisation accuracies from smaller training sets when bias is correct (when concepts D and E are assumed to be similar) and causes it to perform worse when the bias is incorrect (when B and C are used for support).

When the support task used by DEFT is for concepts B or C, the influence of modified evaluation function leads to over-specific clauses. As shown in Figure 5.6 (b), when the target task has 30 examples, using DEFT with these support tasks can be seen to induce, on average, clauses with slightly more literals than ALEPH. As these larger clauses are more specific they will cover fewer positive examples which means more clauses must be found to cover the remainders. This increase in the number of clauses for support tasks B and C can be seen in Figure 5.6 (a) with an average of around 2.5 clauses per theory. This same effect is not observed when concepts D or E are used as support.



(a) Clauses in Hypothesis



(b) Average Literals per Clause

FIGURE 5.6. Theory complexity results for a range of training set sizes for target tasks for concept A. Each graph shows values for ALEPH and DEFT using support tasks for concepts B, C, D and E. Error bars show 95% confidence intervals.

The following is an example of one of the three-clause theories learnt from a 30 example task for concept A by DEFT using a task from concept B as support:

```
like(A) :- size(A,small), genre(A,scifi), year(A,'90s').
like(A) :- genre(A,scifi), nation(A,uk).
like(A) :- genre(A,romance), nation(A,aus), year(A,'90s').
```

The influence of the support task is primarily due to the `genre(A,romance)` and `size(A,small)` literals. Their inclusion in the third clause have over-specialised it resulting in uncovered positive examples while the first clause covers some negative examples reducing the overall accuracy of the theory to 84.5%.

5.2.5. Experiment RP-3: Sensitivity to support task size. Results from the previous experiment show that the choice of support concept has a strong effect on the magnitude and direction transfer has on generalisation performance. The support tasks used in that experiment were all of a generous size so that the descriptor frequencies present in the DFTs were good estimates of the true descriptor probabilities for each concept. With fewer examples available for a support task it is possible that the descriptor frequencies will be poor estimates and using such a DFT will not be as effective in modifying the bias for the target task. The purpose of this experiment is to test whether or not this is the case.

As the previous experiment has shown, the concepts in the reading preferences domain can be separated by similarity into the groups {A,D,E} and {B,C}. The effect DEFT has on accuracy, whether positive or negative, depends on whether the support and target tasks were drawn from the same group or not. To simplify the remaining experiments in this section representatives from each group will be chosen rather than running trials on all 25 pairs of support and target combinations at all training sizes. When testing the effect of support task size on accuracy target tasks were drawn from concept A only and support tasks came from concepts C and D. The target tasks contained either 6 or 10 examples while the support tasks' sizes varied from 4 to 100 examples. DFTs were built for each of the support tasks and then DEFT was applied to the target tasks using each of these DFTs. The exact details of the method are given below.

5.2.5.1. *Method.* The BUILDDFT algorithm (using the settings given at the beginning of this section) was used to construct a DFT $\Phi[S, s, N]$ from the tasks $S_{s,N}$ for each support concept $S \in \{C, D\}$, training size $N \in$

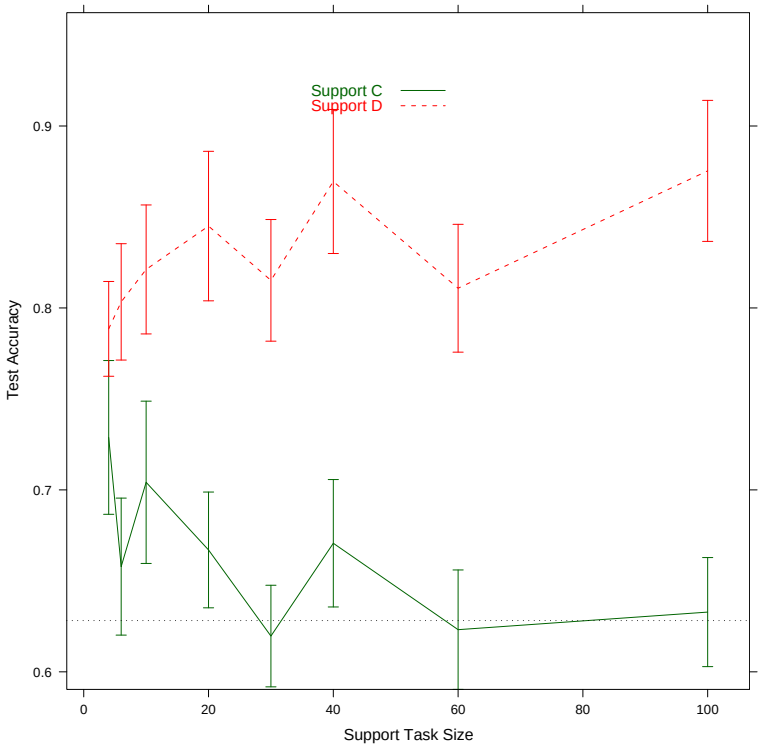
$\{4, 6, 10, 20, 30, 40, 60, 100\}$ and support task index $s = 1, \ldots, 5$. The 80 resulting DFTs were each used by DEFT to learn a theory from the target tasks $A_{t,6}$ and $A_{t,10}$ for each target task index $t = 1, \ldots, 6$. The theory induced on each task was tested on A_{test} to estimate its accuracy. As in the previous experiments, various measurements, including accuracy, were made at the completion of each run. These measurements were then averaged over the 30 pairs of s and t indices.

5.2.5.2. *Results and Analysis.* The time taken to construct a DFT did not vary significantly across the 80 different DFTs built for this experiment. The running time of BUILDDFT in each case was around 1.1 seconds regardless of the support concept used or the number of examples in the support task. This is slightly surprising as one would expect that when the support tasks have 100 examples the time taken to evaluate each rule during DFT construction will take around 25 times longer than when only 4 examples are available. The reason that this is not the case is that rule evaluation is much faster than the procedure used to sample rules. Regardless of whether there are 4 or 100 examples, 20 rules are sampled from each of 20 bottom clauses. The time taken to determine the sampling distribution over each bottom clause contributes to the majority of the 1.1 seconds of DFT construction time.

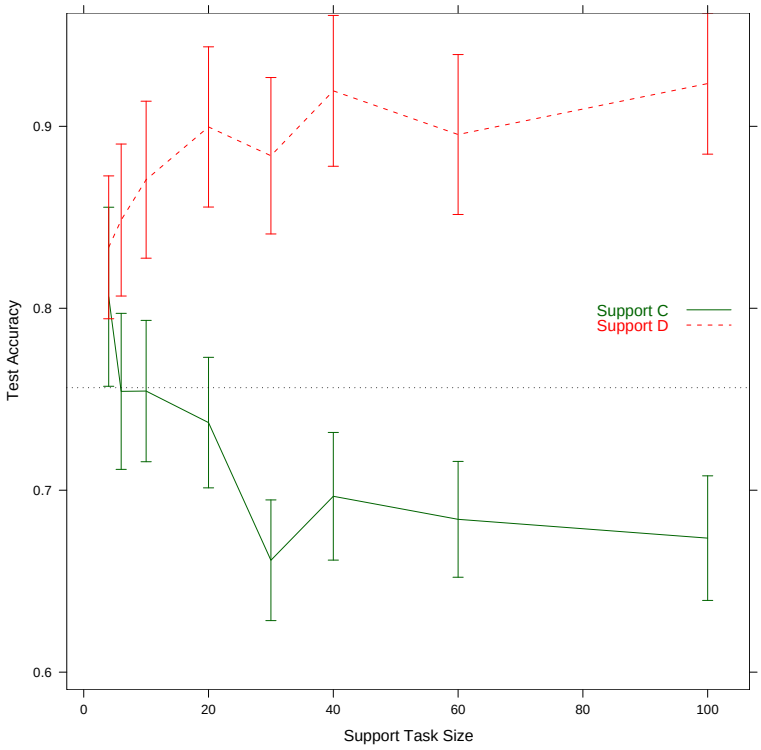
While DFT construction time does not vary with a changing support size, the effect on target task accuracy does depend on how many examples are available in the support task. The two graphs in Figure 5.7 summarise how the mean accuracies vary with support task size for the two target tasks for concept A described in the method above. The first graph in the figure shows the results for the size 6 task and the second graph shows the results for the size 10 task.

Noticeably, the largest difference between the accuracies when not using a support task (shown by the horizontal, dotted line in each graph) and when using one for support concept D occurs when the support tasks have 100 examples. The largest negative effects when using concept C as support also occur when support tasks for that concept have 100 examples.

According to the analysis regarding sampling parameters made in Section 4.3.4, to ensure that descriptor frequencies are estimated confidently and accurately, approximately 16000 rule/example pairs need to be evaluated. Given that the BUILDDFT was used with sampling parameters which meant 400 rules are created there needs to be at least 40 examples in a support task to ensure good descriptor frequency estimates. As the graphs in Figure 5.7 show, this size of support task gives transfer effects comparable to when 100 examples are used. While it is not always possible to choose exactly how many



(a) Test accuracy on tasks $A_{t,6}$



(b) Test accuracy on tasks $A_{t,10}$

FIGURE 5.7. Test accuracies for Experiment RP-3. Plots show mean test accuracy as a function of support task size for DFTs built using Concept C (solid line) and Concept D (dashed line). Error bars show 95% confidence intervals for the means. The dotted line shows ALEPH's mean accuracy on the target tasks.

support task examples will be available to DEFT, the results here suggest that if the support concept can be learnt accurately from the support task then the DFTs constructed from that support task should be suitable for transfer. If a suitable number of support task examples are available the most pertinent issue becomes how best to choose the sampling parameters.

5.2.6. Experiment RP-4: Sensitivity to sampling parameters. In this experiment, the parameters *exs_sampled* and *cls_sampled* are varied over a range of values to determine what effect they have on transfer using DEFT. The method given below describes in detail how the sampling parameters were varied. These two parameters control the number and distribution of rules that are generated when the BUILDDFT algorithm is used to estimate descriptor frequencies for a support task. The total number of rules sampled is just the product of the parameters and it is expected that if this product is too small the estimates of descriptor frequency will be poor which, in turn, will lead to poor estimates for the prior CPMs used in inductive transfer by DEFT. The *exs_sampled* is believed to be particularly important in obtaining good quality priors. This parameter controls how many examples are used to construct the bottom clauses from which rules are sampled. If this is too small, rules may not be created which cover certain disjuncts within a concept. This may also have an adverse effect on the quality of descriptor estimates and therefore transfer.

As discussed in Experiment RP-3 above, there is little need to run transfer trials over all 25 pairs of support and target concepts when the effects of parameters are being investigated. The primary aim when using DEFT is to improve generalisation accuracy when learning on small training sets. As shown in the earlier experiments, one situation when this occurs when support tasks for concept D are used as support for learning on target tasks for concept A of size 10. The average accuracy for DEFT in these cases was 92% compared to 74% for ALEPH. These sets of support and target tasks therefore leave room for both positive and negative change in accuracy as the sampling parameters for DEFT are varied. The method used to measure these variations is described below.

5.2.6.1. *Method.* Ten tasks $A_{t,10}$ (for $t = 1, \ldots, 10$) from concept A were used as target tasks for DEFT. Each application of DEFT to each target task was performed using one of 360 DFTs computed from support tasks with 100 examples drawn from concept D. Each DFT is created by applying BUILDDFT to one of ten support tasks $D_{s,100}$ ($s = 1, \ldots, 10$) using a *exs_sampled* parameter set to a value from {2, 5, 10, 20, 40, 80} and a *cls_sampled* parameter set to a value from {2, 5, 10, 20, 40, 80}. The ten different support tasks were

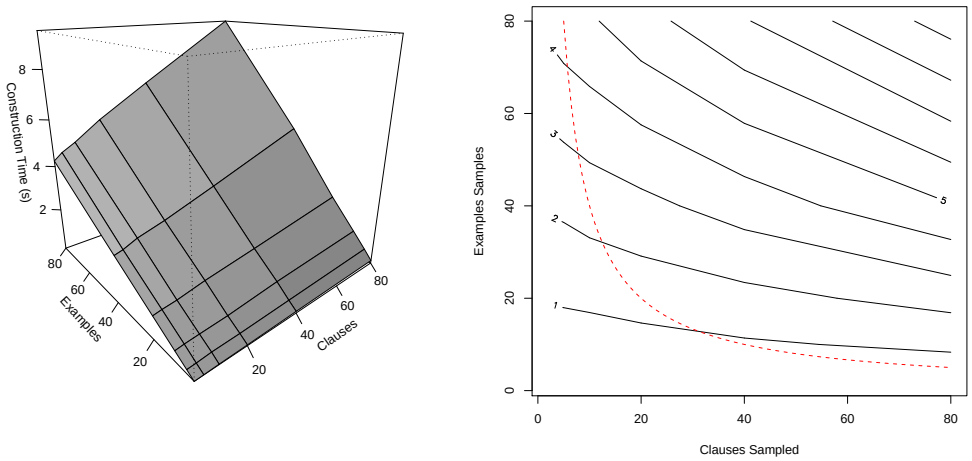
TABLE 5.5. Total number of clauses sampled by BUILDDFT when using the given values for the sampling parameters *exs_sampled* and *cls_sampled*.

	exs_sampled					
cls_sampled	**2**	**5**	**10**	**20**	**40**	**80**
2	4	10	20	40	80	160
5	10	25	50	100	200	400
10	20	50	100	200	400	800
20	40	100	200	400	800	1600
40	80	200	400	800	1600	3200
80	160	400	800	1600	3200	6400

used when constructing DFTs to reduce the variability of the results for each parameter setting due to the incidental properties of the support examples. Similarly, the ten different target tasks were used to reduce the random effects of the target sample used. Each of the 36 different pairs of parameter settings were therefore used for 100 separate support-to-target transfers. As in the earlier experiments, the generalisation accuracy of each theory learnt by DEFT is computed through its application to the test set A_{test}. The accuracy values reported for each of the 36 parameter settings is taken to be the average over the 100 trials for each setting. Construction times were also recorded for each DFT. The average construction time for each parameter setting is taken over the values recorded for each of the ten support tasks.

The values the sampling parameters vary over were chosen to maximise the number of pairs taken from these sets which have the same product. The analysis in Section 4.3.4 shows that the total number of clauses sampled (the product of the sampling parameters) when creating a DFT affects the quality of its descriptor frequency estimates. Having many pairs of parameters with the same products means the clause sample size can be held constant while the effect of varying the parameters is studied. To clarify this, Table 5.5 shows all 36 pairs of sampling parameters and their products. From this it can be seen, for example, that there are five pairs of sampling parameters which result in 400 clauses being sampled.

5.2.6.2. *Results.* There are two quantities of interest in this experiment: the average time it takes to construct DFTs and the average generalisation accuracy of theories induced for the target task when using these DFTs. Both of these quantities can be thought of as functions of the two parameters *exs_sampled* and *cls_sampled*. Figures 5.8 and 5.9 each show two representations of the construction time and accuracy functions respectively. The perspective plot in each case is a straight-forward rendering of the function as a



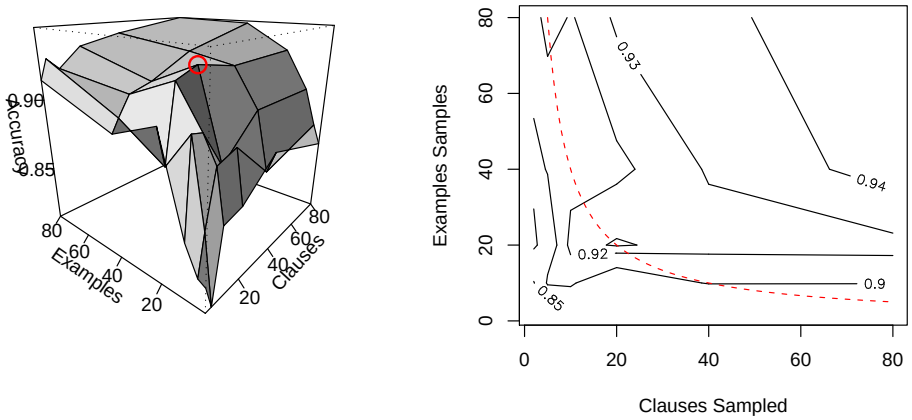
(a) Perspective View (b) Contour Plot

FIGURE 5.8. Perspective (left) and contour (right) plots of the average construction time in seconds of DFTs for concept D as a function of the sampling parameters *exs_sampled* and *cls_sampled*. Solid lines on the contour plot are iso-lines for construction time and the dashed line indicates where the product of the parameters is equal to 400.

surface in three dimensions. The surface is a linear interpolation of the values of the function at each parameter setting. The contour plot shows a top-down view of the function surface. Each iso-line in a contour plot shows a set of parameter values which all have the same associated value on the interpolated surface.

The plots in Figure 5.8 reveal a simple relationship between the number of clauses and examples sampled and the time taken to construct a DFT. Construction time increases monotonically with an increasing number of sampled examples or clauses. This increase occurs faster as the number of examples increase compared to the same increase in the number of clauses. An examination of the dashed line in the contour plot shows that for a fixed total number of samples (400) the time to construct a DFT is about a second when the *cls_sampled* is 40 and the *exs_sampled* parameter is set to 10. Increasing the *cls_sampled* parameter to 80 and dropping the *exs_sampled* parameter to 5 reduces the construction time to slightly under a second. However, sampling 80 examples and 5 clauses per example during DFT construction takes an average of over four seconds.

This strong influence of the *exs_sampled* parameter on construction time is due to two factors. First, a bottom clause must be constructed for each



(a) Perspective View (b) Contour Plot

FIGURE 5.9. Perspective (left) and contour (right) plots of the average accuracy of DEFT as a function of the sampling parameters *exs_sampled* and *cls_sampled*. Solid lines on the contour plot are iso-lines for accuracy and the dashed line indicates where the product of the parameters is equal to 400.

example from which the clauses used to build the DFT are drawn. Secondly, only legal clauses are to be sampled from each bottom clause. This requires an estimate of the distribution of legal clauses of varying lengths (see Section 4.3.2 of the previous chapter). To do this, 100 subsets of the bottom clause are taken for each length up to the value of the *clauselength* parameter. Each subset is then tested against the mode and type constraints to test whether it is legal. Since the *clauselength* parameter is set to 4 in the reading preferences experiment, each extra example that is used for DFT construction requires the sampling of 400 extra clauses.

All else being equal, the above results suggest choosing the *exs_sampled* parameter to be as low as possible if DFT construction time is a concern. However, this parameter also has an effect on accuracy, as shown in Figure 5.9. The perspective plot in this figure shows a fairly complex relationship between the sampling parameters and the resulting effect on accuracy. Broadly speaking, the results show that sampling more clauses during DFT construction results in higher accuracies for transfer using DFTs for concept D on tasks for concept A. These reached 94.6% (with a s.d. of 9.2%) when $exs\_sampled = cls\_sampled = 80$ and dropped to 82.7% (s.d. 11.5%) when $exs\_sampled = cls\_sampled = 2$. This last figure is not statistically different ($p = 0.22$) to the 74% accuracy reported by ALEPH on the same tasks.

TABLE 5.6. The effect of the *exs_sampled* parameter (columns) on generalisation accuracy (and s.d.) for a fixed number of total clauses sampled (rows). *Italic* values show entries corresponding to *cls_sampled* settings equal to 5 or 10. **Bold** values in a given row differ from the *exs_sampled* = 20 entry in the same row at the $p < 0.05$ level of significance using a paired, two-sided t-test.

	exs_sampled				
	5	**10**	**20**	**40**	**80**
100	0.89 (0.10)	*0.90 (0.10)*	*0.88 (0.10)*	-	-
200	**0.86 (0.11)**	**0.88 (0.10)**	*0.93 (0.10)*	*0.90 (0.10)*	-
400	**0.87 (0.10)**	**0.90 (0.10)**	0.93 (0.10)	*0.91 (0.10)*	*0.92 (0.9)*
800	-	0.90 (0.10)	0.93 (0.10)	0.92 (0.10)	*0.92 (0.9)*
1600	-	-	0.93 (0.10)	0.93 (0.10)	0.93 (0.9)

The effect of the sampling parameters on accuracy can be better understood with reference to Table 5.6. In this table each row shows the effect of changing the *exs_sampled* parameter while holding the total number of sampled clauses fixed. For example, the third row in the table corresponds to the dashed line in the contour plot of Figure 5.9 where the product of the parameters is 400. If *exs_sampled* is set to 5 or 10 examples, the difference in accuracy compared to a setting of 20 is significant when a total of 200 or 400 clauses are sampled. However, this effect on accuracy is not observed for low values of *cls_sampled*. All of the higher settings for *exs_sampled* correspond to lower values for *cls_sampled* for any given row in the table. None of the entries corresponding to *cls_sampled* settings lower than 20 (shown in italics in the table) are significantly lower than any of the other entries within the same row.

5.2.6.3. *Conclusions.* Both the construction time and accuracy results point to *exs_sampled* being the more important of the two settings used to construct DFTs. Higher values for this parameter lead to longer times for constructing DFTs but when these used in a situation that is favourable to positive transfer the effects of the DFT are stronger, leading to higher accuracies.

5.2.7. Experiment RP-5: The admissibility condition. In Chapter 3, the theoretical analysis of descriptor-based transfer found that to increase the usefulness of descriptors it was necessary to restrict the rule space used by the support and target concepts to the set of *admissible* rules for those concepts, that is rules that cover at least one positive example of the concept. As the search used by ALEPH considers only admissible rules for inclusion in a theory it is believed that evaluating rules during DFT construction using the same bias will increase their effectiveness when used during inductive transfer. All of the experiments reported above were carried out using this assumption

and it is the aim of this experiment to empirically test whether or not the admissibility condition is of any importance.

5.2.7.1. *Method.* The method used for this experiment is almost identical to that of Experiment RP-2 described in Section 5.2.4.1. The only difference is that during DFT construction the five DFTs $\Phi[S, s]$ were constructed with the BUILDDFT algorithm's admissibility condition set to false. The rest of the method is identical, using the DFTs built for this experiment in place of those constructed for Experiment RP-2.

The DFTs and transfer results were collected in such a way to allow comparison with the results from Experiment RP-2. Of particular interest is the difference in accuracy between DEFT when sampling only admissible rules and when sampling all rules. This was calculated for all target tasks T, support tasks S and target training set sizes N by taking the mean of the pairwise difference in accuracy between the Experiment RP-2 runs and those performed here. Letting $\text{acc}_1(T_{t,N}, S_s)$ denote the accuracy obtained by applying DEFT to task $T_{t,N}$ using the DFT constructed from admissible rules for the support task S_s and, similarly, letting $\text{acc}_0(T_{t,N}, S_s)$ be the accuracy using a DFT constructed using all rules, then

$$\Delta(T, S, N) = \frac{1}{30} \sum_{t=1}^{6} \sum_{s=1}^{5} \text{acc}_1(T_{t,N}, S_s) - \text{acc}_0(T_{t,N}, S_s)$$

is the mean difference in accuracy for target task T of size N using support task S. The measurement is used to compare the performance of DEFT in Experiment RP-2 with its performance in this experiment. The values of these mean differences as well as a comparison of the weighted powers for the DFTs constructed here and in Experiment RP-2 are presented below.

5.2.7.2. *Results.* By definition, non-admissible rules cover no positive examples of the support task being used during DFT construction. On average then, non-admissible rules must therefore cover more negative examples compared to admissible rules. When a DFT is constructed using BUILDDFT, each descriptor-value pair $[d, v]$ has associated with it a CPM that is computed by averaging the CPMs of sampled rules which have value v for descriptor d. In this experiment non-admissible rules were included in that average so the resulting CPM for each descriptor-value pair in the DFT must necessarily have a larger false positive rate.

This increase in false positive rate can be observed as a decrease in weighted power for each of the descriptors. The power for a descriptor d with value v

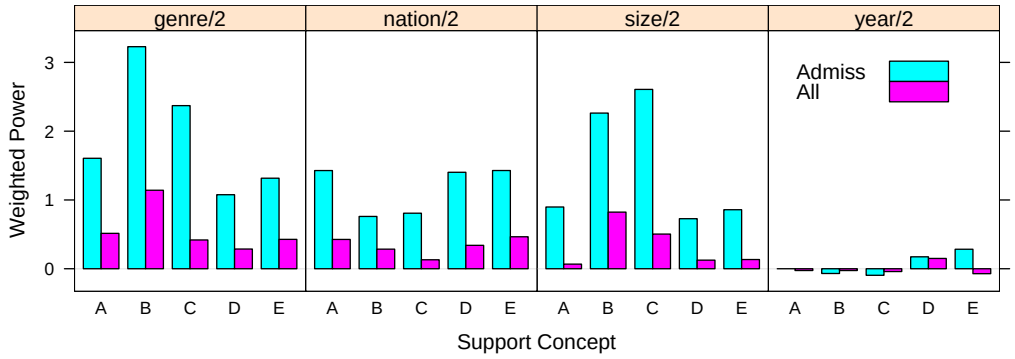


FIGURE 5.10. Weighted power for `has_pred` descriptors on the five support concepts. The left (right) bar for each concept shows the power when the admissibility condition is (not) used.

and default value v_0 is given by

$$\beta[d, v] = (q_{++}[d, v] - q_{+-}[d, v]) - (q_{++}[d, v_0] - q_{+-}[d, v_0])$$

where the q_{ij} are entries in the CPM averages for the descriptor-value pairs. If the increase in the false positive rate for the value v is greater than that for the default value v_0 then the power of the pair $[d, v]$ will decrease. This can be seen in the comparison of the powers with and without the admissibility condition shown in Figures 5.10 and 5.11. The layout of these figures is the same as for Figures 5.2 and 5.3 from Experiment RP-2 above except that two values are reported for each concept. The left (respectively, right) value for each concept is the average weighted power based on DFT computed using admissible (respectively, all) rules. The left-hand values are identical to those shown in the earlier figures from Experiment RP-2.

In every case except one, the value for the non-admissible DFT powers is less than that for those computed for the admissible DFTs. The slightly higher value in the case when "year = 90s" is not statistically significant and may be due to the stochastic nature of the DFT construction procedure. In the graphs of Figure 5.11 the difference between the admissible and non-admissible scores is most noticeable, with some descriptors (*e.g.*, "genre = scifi", "nation = usa" and "size = small") having large negative weighted power in the non-admissible case. The negative powers when all rules are sampled highlight a number of differences between the concepts that were not present when only admissible rules were sampled. For example, concepts B and C differ dramatically for the descriptors "size = large", "size = small" and "genre = horror". Similarly, concepts A and D have very different powers compared

TABLE 5.7. Number of common positive and negative examples between each pair of Reading Preference concepts. Each entry +P -N denotes that there are P positive and N negative examples shared by the concepts labelling the row and column of the entry. The values are not shown under the the main diagonal as they are identical to those on the opposite side of it.

	A	**B**	**C**	**D**	**E**
A	+18 -63	+2 -47	+2 -47	+6 -51	+0 -45
B	.	+18 -63	+0 -45	+2 -47	+2 -47
C	.	.	+18 -63	+2 -47	+2 -47
D	.	.	.	+18 -63	+3 -48
E	.	.	.	.	+18 -63

to concept E for the descriptors "nation = uk" and "nation = usa". These differences can be understood with reference to Table 5.1. Since both rules describing concept B test for "genre = romance", no positive examples can have "genre = horror" or "genre = scifi" meaning rules with those descriptors can only cover negative examples. Similarly, positive examples for concepts A and D must either have "nation = aus" or "nation = uk" meaning rules that test "nation = usa" will cover only negative examples of those concepts.

The differences in power brought about by sampling non-admissible rules has a measurable effect on accuracy when learning with DEFT. Figure 5.12 shows the difference $\Delta(T, S, N)$ described in the method above for each target task, support task and training set size. The layout of these values is similar to the accuracy plots shown in Figure 5.4 above except that the differences between accuracies, rather than absolute accuracies are shown. The solid, horizontal line in each of the 25 plots is for reference and shows a difference of zero. The only large and significant differences are for transfer from E to A, A to E, B to C, C to B and C to C. In all of these cases, restricting DFTs to admissible rules only leads to a higher accuracies when compared to using DFTs that sample admissible and non-admissible rules. The largest of these differences was 0.2 for transfers from concept B to tasks of size 10 for concept C. The reason for the difference in accuracy in the C to C case is that the descriptor powers for admissible DFTs have a larger positive power than the DFTs when all rules are sampled. This means the preference bias for rules with high true accuracy will be stronger when using the admissible DFTs.

Apart from the concept C to concept C transfer the other four cases in which admissible sampling gives improved accuracy can be explained by the descriptors with negative power. These cause the evaluation of rules that satisfy those descriptors to be lower than when admissible DFTs are used.

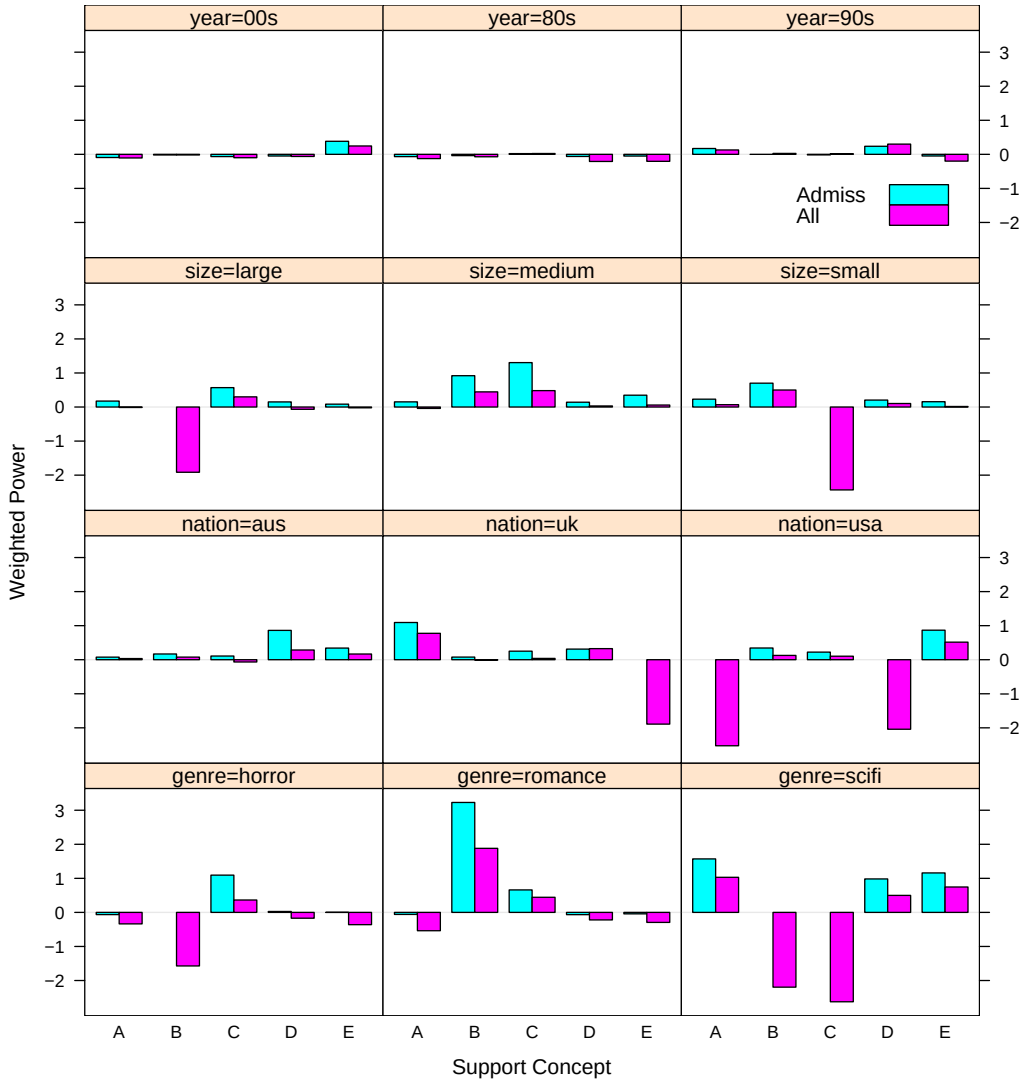


FIGURE 5.11. Weighted power for `has_arg` descriptors. The left (resp. right) bar shows the power when the admissibility condition is (resp. is not) used.

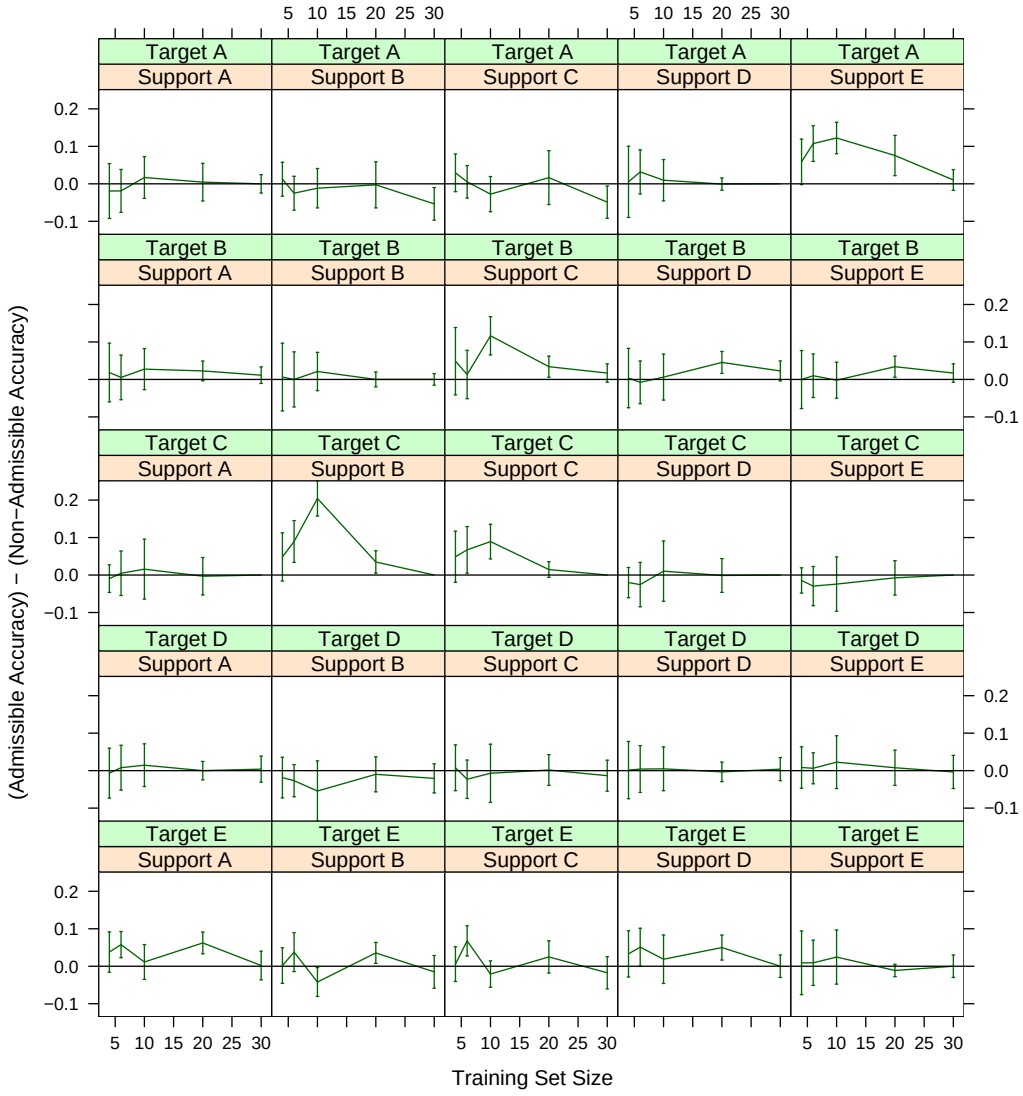


FIGURE 5.12. Difference in test accuracies between DEFT using DFTs constructed using only admissible rules and those constructed using all rules. Error bars show 95% confidence intervals for the true mean difference.

Significantly, the concept pair A and E share no positive examples. That is, none of the 81 possible instantiations of a book instance is covered by both concept A and concept E. The pair B and C also share this property whereas all of the other concept pairs have at least two positive examples in common, as shown in Table 5.7.

5.2.7.3. *Conclusions.* The above results show that sampling only admissible rules during DFT construction leads to more opportunities for DEFT to improve generalisation accuracy from small training sets. In the cases where the support and target tasks do not share any common positive examples, using the admissibility condition reduces the influence of descriptors with large, negative power and allows DEFT to exploit other, positive similarities between those concepts. Furthermore, there were no situations within the reading preference environment where sampling only admissible rules lead to significantly lower accuracies. These observations suggest that a reasonable default setting is to use the admissibility condition when constructing DFTs.

5.2.8. Experiment RP-6: Sensitivity to descriptor templates. In this experiment the use of the default *templates* setting {has_pred, has_arg} was investigated to see whether some other combination of descriptor templates can provide better transfer results. This was done by considering each of these templates in isolation and also in conjunction with the num_lits descriptor template described in the previous chapter. This template constructs exactly one descriptor which counts the number of body literals present in a clause. This descriptor template is also considered in isolation so as to determine whether it is useful in its own right.

As in previous experiments, the focus here is limited to considering transfer between a single pair of concepts, in this case from concept D to concept A. This allows a range of possible settings for the *templates* parameter to be explored and analysed.

5.2.8.1. *Method.* For each support task D_s for $s = 1, \ldots, 5$ and non-empty set of descriptor templates $\tau \subseteq$ {has_pred, has_arg, num_lits}, a DFT $\Phi[s, \tau]$ was constructed by BUILDDFT using settings identical to those in Table 5.3 but with the *templates* parameter set to τ. The code used to implement the three templates can be found in Appendix A. The 35 resulting DFTs were then used as input to DEFT when learning from each of the 30 tasks $A_{t,N}$ where $t = 1, \ldots, 6$ and $N \in \{4, 6, 10, 20, 30\}$. The accuracies for the resulting 1050 theories induced by DEFT for each transfer was computed by applying each theory to the test set A_{test}.

TABLE 5.8. Power and probabilities for the values of the num_lits/0 descriptor recorded for concept D. The entries show the mean value (s.d.) taken over the five tasks of size 100 used to construct the DFTs $\Phi[s, \{num\_lits\}]$.

	num_lits			
	0	**1**	**2**	**3**
power (β)	0.00 (0.00)	0.79 (0.31)	1.42 (0.14)	2.77 (0.38)
prob. (π)	0.02 (0.00)	0.05 (0.00)	0.21 (0.00)	0.72 (0.01)

The parameters of interest in this experiment are the set of templates τ and the training set size N. The mean accuracy $\text{acc}(\tau, N)$ for each of their settings was calculated by averaging over the 30 accuracies obtained by varying the support task index s and target task index t.

5.2.8.2. *Results.* All the experiments prior to this one have only used the template set {has_pred, has_arg} and so these have been the only descriptors for which powers have been reported. Table 5.8 provides the powers and probabilities recorded for each of the values the num_lits/0 descriptor takes in the DFTs for concept D. Unlike the other two descriptor types which only take on the values "true" and "false" (the default), the num_lits descriptor's values are integers between zero (the default) and three, each number describing the number of body literals in a clause. By definition, the default value of zero literals has power equal to zero while the powers for the other values are the increase in log-odds relative to the default. As the table shows, longer admissible rules have a stronger correlation with correctly predicting positive examples. This is not surprising since more specific admissible rules cover at least one positive example but fewer negative examples than the shorter rules.

Whether or not a general preference for longer rules improves predictive accuracy when examples are limited can be answered with reference to Figure 5.13. These plots provide a compact way of comparing the accuracies across a range of training set sizes for concept A and all the combinations of descriptor templates used to construct DFTs from concept D. Each of the eight subsets of the templates {has_pred, has_arg, num_lits} are identified by the presence or absence of each template. The solid (respectively, dashed) plots within each of the four graphs in the figure show accuracy results when the has_pred template was used (respectively, not used) when constructing DFTs. The two leftmost (respectively, two rightmost) graphs show accuracies when the template has_arg was not (respectively, was) used during DFT construction while the first and third (respectively, second and fourth) plots report results for template sets which did not include (respectively, included)

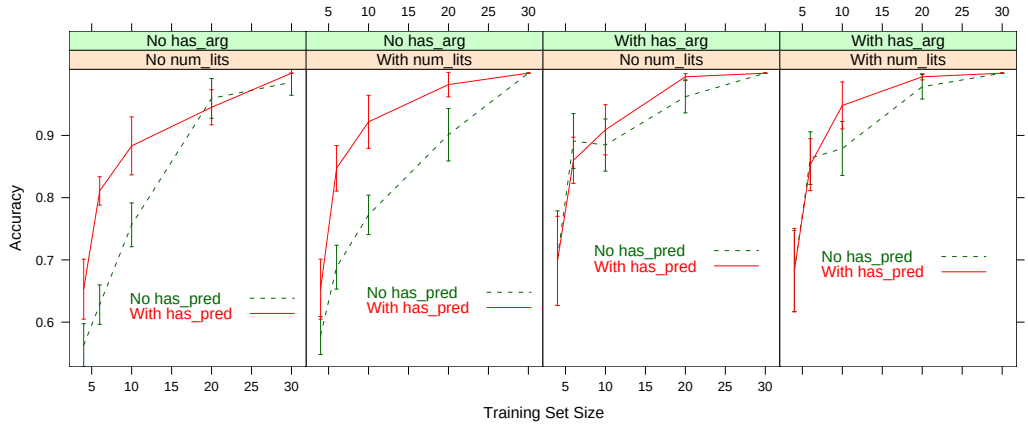


FIGURE 5.13. Accuracy vs. training size for tasks for transfer from concept D to concept A using eight different *templates* settings. Each of the eight graphs shows how accuracy varies with training set size when a particular subset $\tau \subseteq$ {has_pred,has_arg,num_lits} is used to construct a DFT for concept D. The error bars show 95% confidence intervals for the mean.

the num_lits template.

The dashed plot in the second graph from the left corresponds to the use of the template set {num_lits}. The dashed plot in the leftmost graph uses an empty template set and so reports the same results as baseline learning on concept A from Experiment RP-1. Comparing these two plots reveals that using the num_lits descriptor gives a accuracy higher than the baseline by 0.06 ($p < 0.02$ using a two-sided paired t-test) when there are six training examples but a lower accuracy than the baseline by 0.06 ($p < 0.02$) when there are 20 training examples. Both the increase and the decrease in accuracy relative to the baseline learner are due to the preference for longer clauses when the $\tau =$ {num_lits} DFTs are used. The search order the baseline learner uses means the shortest rule that covers the most positive examples while excluding all the negative examples will be selected. On small training set sizes a limited number of negative examples means this search will terminate early, selecting an over-general rule. When the {num_lits} DFTs are used, however, the evaluation function is modified so as to prefer longer rules thereby reducing the false positive rate. This same preference for longer rules is a liability when more training data is available, forcing the search to over-specialise, creating extra rules which ultimately increase the false positive rate. When used in conjunction with other descriptors, however, the addition of the num_lits descriptor does not significantly increase or decrease the accuracy of DEFT.

The interaction between the has_pred and has_arg descriptors is also fairly minimal. Both descriptor types when used alone give substantial increases in accuracy over the baseline learner, especially at small training set sizes. Comparing the dashed plots in the first and third graphs of Figure 5.13 shows an increase of around 0.24 ($p < 0.001$) on training sets of size six when the has_arg descriptor is used by itself. Similar gains can be seen when comparing the solid and dashed plots in the leftmost graph. The solid plot corresponds to the exclusive use of has_pred descriptors while the dashed plot shows the baseline learning results. The combination of the has_pred and has_arg descriptors (as shown by the solid plot in the third graph from the left) is not significantly different to the has_pred descriptors used alone or the has_arg descriptors used alone. The only exception to this is on training sets of size 20. In this case the use of the combination returns theories that are 0.05 higher than using only has_pred ($p < 0.001$) and 0.03 higher than using only has_arg ($p < 0.001$).

5.2.8.3. *Conclusions.* When other descriptor templates are used the presence or absence of the num_lits template has no discernible effect on accuracy. When used alone, however, the num_lits descriptor provides a small increase in accuracy on small datasets when compared to the baseline results. This suggests that this descriptor is not well suited for this environment as it is not able to separate rules into to meaningful similarity classes.

In contrast, the similarity classes created by the other two descriptors, has_pred and has_arg, allow for large increases in accuracy when used individually to transfer bias from concept D to concept A. When combined, a slight increase in accuracy was observed on larger training sets and no adverse effects were recorded. This suggests that of the eight possible subsets of {num_lits, has_pred, has_arg}, the use of {has_pred, has_arg} is a reasonable default setting.

A final conclusion that can be drawn from these results is that adding extra templates never decreased the accuracy of theories learnt by DEFT. Further experiments with a larger number of varied templates would be required to test whether this is true in general.

5.2.9. Experiment RP-7: Sensitivity to the M parameter. The final transfer parameter that the user can control is the M parameter. This determines the number of virtual examples that are used when DEFT's DBSCORE and DBBOUND procedures combine a rule's prior CPM with its actual CPM. Intuitively, this controls how much influence a support task has over the evaluation bias on a target task. In this experiment DEFT was used with

a variety of M settings to transfer bias from concepts B, C, D and E to target tasks from concept A. The details of this experiment are given in the method below.

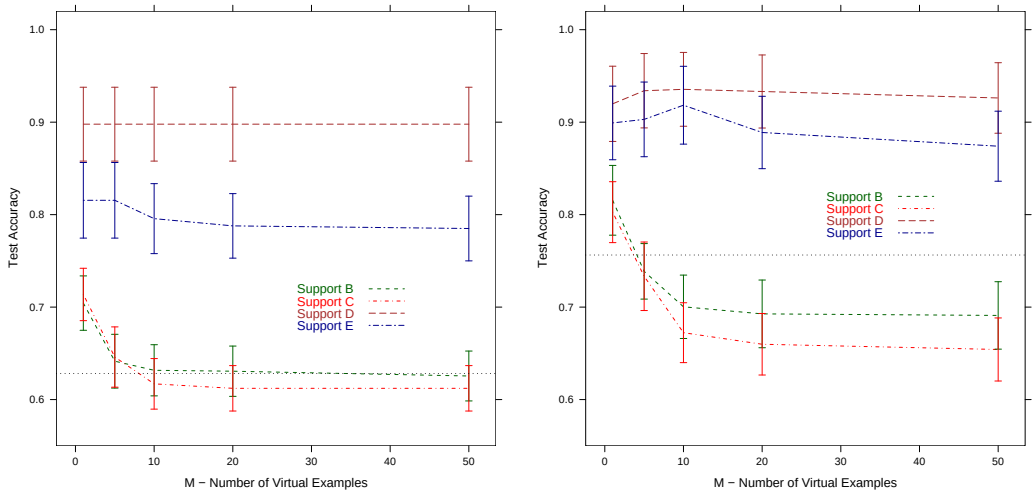
5.2.9.1. *Method.* The target concept used in this experiment was concept A and the support concepts S were drawn from {B, C, D, E}. The 20 DFTs $\Phi[S_s]$ constructed in Experiment RP-2 for each of the support tasks S_s, $s = 1, \ldots, 5$ were reused for this experiment. Using each of these DFTs in turn, DEFT was applied using each parameter setting $M \in \{1, 5, 10, 20, 50\}$ to the tasks $A_{t,N}$ with training set sizes $N \in \{6, 10, 20\}$ and task indices $t = 1, \ldots, 6$. Each theory induced by DEFT was applied to the test set A_{test} to obtain an accuracy measurement $\text{acc}(S_s, A_{t,N}, M)$. The resulting 1800 trials were organised by support task, target task size and M parameter setting and the mean accuracy $\text{acc}(S, N, M)$ was computed by averaging over the task indices s and t:

$$\text{acc}(S, N, M) = \frac{1}{30} \sum_{s=1}^{5} \sum_{t=1}^{6} \text{acc}(S_s, A_{t,N}, M).$$

5.2.9.2. *Results.* Figure 5.14 contains three graphs: the left, centre and right showing the accuracies $\text{acc}(S, N, M)$ on target tasks for concept A with sizes $N = 6$, 10 and 20 respectively. In each graph, ALEPH's accuracy for each target task size is shown as a horizontal, dotted line. The value reported by this line comes from the results of Experiment RP-1. The other four lines within each graph show how the accuracies for DEFT change as a function of the parameter M for each of the four support concepts B, C, D and E.

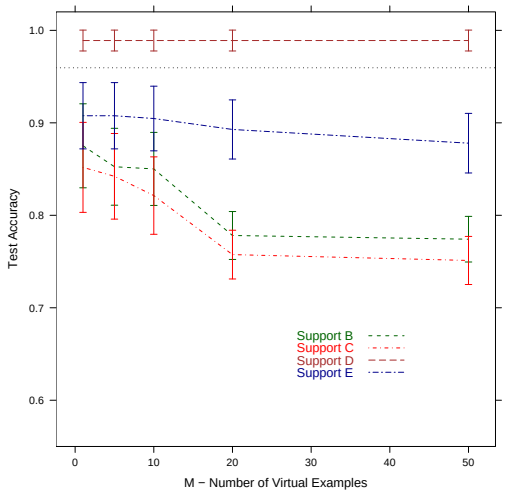
A striking feature of the graphs in Figure 5.14 is how little impact the M parameter has on the accuracies reported when DEFT was used with DFTs for concepts D and E. Regardless of the target task size the accuracy for these support concepts is virtually identical as M ranges between 1 and 50. While there is slightly more variance when concept E is used as support, the accuracies in both these cases are never higher or lower than about 3% from the $M = 1$ value. This can be explained by considering the interaction between the CPM classes for the target concept A and the descriptor classes for the support concepts D and E.

As described in Chapter 3, the set of rules for a learning task can be partitioned into classes based on the CPMs assigned to the rules by the training data. When examples are limited these CPM classes can contain many rules and, as more examples become available, the number of possible CPMs increases and so the number of rules per CPM class falls. The evaluation function used by the learner assigns scores to each of these classes based on the entries in each class's CPM. Poor test accuracies occur when the high scoring CPM



(a) Training Size = 6

(b) Training Size = 10



(c) Training Size = 20

FIGURE 5.14. Mean accuracy as a function of the M parameter for DEFT on target tasks for concept A for various training set sizes. The dotted horizontal line in each graph shows the accuracy of the baseline learner. Error bars show 95% confidence intervals.

classes based on a limited number of training examples have a large number of rules which are not in the high scoring CPM classes for the test examples. Description classes - sets of rules that share a common description - are used by DEFT to define a partition independent of the CPM partitions. Each descriptor class is assigned a prior CPM based on the descriptor frequencies in the DFT used as support. The evaluation function assigns a score to intersections of CPM and description classes based on the combination $\mathbf{p}^* = \frac{N}{M+N}\mathbf{p} + \frac{M}{M+N}\mathbf{q}$ of the training CPM $\mathbf{p}$ and prior CPM $\mathbf{q}$.

In these experiments the evaluation function f is the *coverage* function, returning the true positive count minus the false positive count of a rule. The independence of the test accuracy and the M parameter in the cases described above can be attributed to the linearity of this evaluation function. That is,

$$f(\mathbf{p}^*) = f\left(\frac{N}{M+N}\mathbf{p} + \frac{M}{M+N}\mathbf{q}\right) = \frac{N}{M+N}f(\mathbf{p}) + \frac{M}{M+N}f(\mathbf{q}).$$

When the rules which maximise $f(\mathbf{q})$ are a subset of the rules which maximise $f(\mathbf{p})$ these same rules must also maximise $f(\mathbf{p}^*)$ for all $M > 0$. This is what is happening when the priors $\mathbf{q}$ are constructed from support tasks for concepts D and E. When there are rules which do not maximise $f(\mathbf{p})$ and $f(\mathbf{q})$ simultaneously small values of M will make the learner return rules which maximise the former while large M will result in rules which maximise the latter. This is the case for when the priors are constructed using concepts B and C as support and the reason for the poor performance of DEFT for large M values.

The above observations can also be used to explain another interesting feature of the graphs in Figure 5.14, namely using the concepts B and C (which are dissimilar to concept A) as support can give higher accuracies at low M values than the baseline learner. Even a small M value lets the evaluation function differentiate between rules on the basis of their description. The rules which primarily maximise $f(\mathbf{p})$ but also take into account $f(\mathbf{q})$ perform better than the rules found by purely maximising $f(\mathbf{p})$.

5.2.9.3. *Conclusions.* When transferring bias from concepts D and E to concept A, the M parameter was found to have very little influence on the DEFT's generalisation accuracy, an effect is explained by the linearity of the evaluation function used in these experiments. Small M values gave as much an improvement in accuracy as large M values when the support concepts were similar, as was the case with concepts D and E. It was also found that, for small M values, DEFT performs better than the baseline learner even when using the concepts B and C, which are dissimilar to the target concept A. Taken together, these results suggest that M values should be set to be small relative to the

number of training examples. The use of the maximum likelihood estimate for $M = \sqrt{N}$ as a default setting for DEFT ensures that this is the case.

5.2.10. Experiment RP-8: Analysis of CPM Error. The results of the experiments described above show that DEFT is transferring a bias between the tasks in the reading preferences domain. Furthermore, when this transfer is between tasks from the intuitively similar groups A, D and E, or B and C, the effect of this transfer is an increase in generalisation performance. According to the theory developed in Chapter 3, one explanation for these improvements is due to the modifications made to the base learner's CPM estimates which take into account the performance of similar rules on the support task. If the tasks are related - that is, their dissimilarity is low enough - then the Transfer Theorem implies that the average CPM error using the support task should be less than the average CPM error without any support.

There are two properties of the reading preferences domain which make testing this theorem relatively easy in practice. The first is the small size of the example and rule space. Each of the four predicates `genre/2`, `nation/2`, `size/2` and `year/2` can only take on one of three possible values so there are only $3^4 = 81$ possible book instances and $4^4 = 256$ unique rules that can be constructed. This makes it possible to compute the exact CPM error $\text{err}_T(\mathbf{p}_E)$ for an estimated CPM function $\mathbf{p}_E$ based on the examples E for the task T. The second property of this domain is that the descriptor templates `has_pred` and `has_arg` create very fine-grained descriptions of rules. Indeed, each description corresponds to exactly one rule. Computing the similarity class for any given rule is therefore trivial as it is the set containing only the rule in question. Also, the prior CPM for that class is just the CPM for its single rule evaluated on the support task. In practice however, the implementation of DEFT assumes descriptors are independent and computes CPM priors using the product of the frequencies of the individual descriptors. The reading preferences environment provides a perfect opportunity to analyse the difference the practical assumptions make in light of the theory.

5.2.10.1. *Method.* Theorem 3.18 of the previous chapter gives an upper bound on the CPM error when using classification priors from a support task in terms of the normal CPM error for the target task and the dissimilarity and irregularity of the support and target tasks. To test whether or not the bound holds in practice these quantities need to be computed and compared to the empirically obtained CPM error for the DEFT evaluated rules.

This was done using the same collection of training folds and sizes for the target tasks for A and the DFTs for the support tasks for B, C, D, and

E that were used in Experiment RP-2 above. Using the method described below, baseline CPM errors $\text{err}_A(\bar{\mathbf{p}})$ were computed for each of the six training sets $A_{t,N}$ for the target concept A and training sizes $N = 4, 6, 10, 20, 40$. The DEFT CPM errors $\text{err}_A(\mathbf{p}^*_S)$ were also computed on the same six training sets. This was done using DFTs for each of the five example sets $S_{s,100}$ for the four support concepts $S \in \{B, C, D, E\}$.

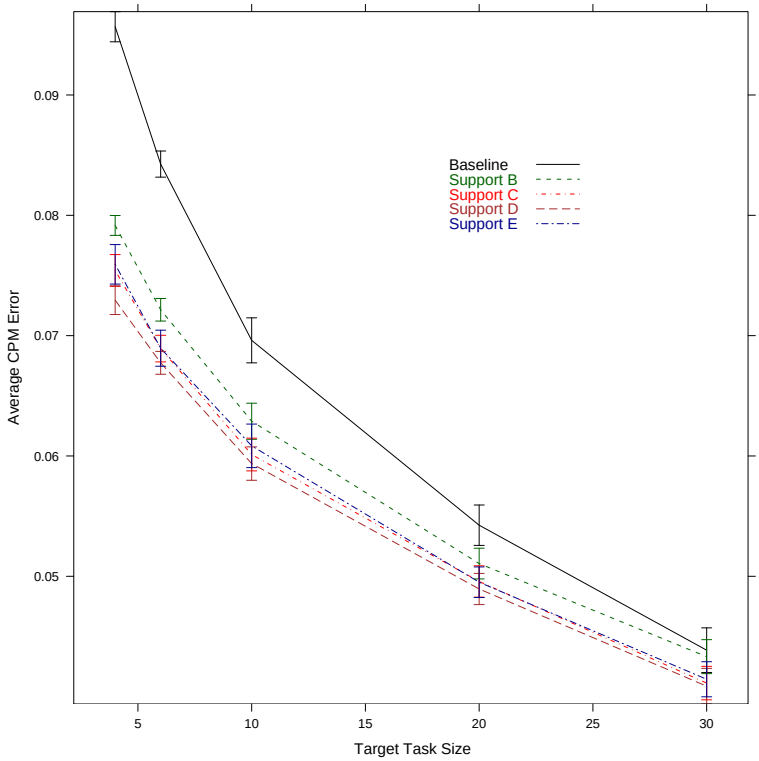
The CPM error was computed by applying the estimated CPM function in question $\mathbf{f}$ (either for the baseline $\bar{\mathbf{p}}$ or DEFT $\mathbf{p}^*_S$) to each admissible rule $r \in R_A$ for the concept A and taking the distance between the resulting CPM $\mathbf{f}(r)$ and the true CPM $\mathbf{p}_A(r)$. For each rule, the estimated CPMs were computed against the training set $A_{t,N}$ while the true CPM was computed by evaluating the given rule on every one of the 81 possible instances in the Reading Preferences domain and comparing the rule's classification to the true classification for concept A. The Euclidean norm (see Section 3.4.1 above) was used when computing the difference between CPMs. By definition, the final CPM error for the estimated and true CPM functions is the average of these CPM distances taken over all the admissible rules for the concept A.

For a given support task S and training size N, the empirical difference between the CPM errors for the baseline CPM function and the DEFT CPM function was computed for the six training tasks $A_{t,N}$ and each of the five DFTs created for the concept S. A rearrangement of equation 3.1 in the Transfer Theorem says that this difference $d(A, S) = \text{err}_A(\bar{\mathbf{p}}) - \text{err}_A(\mathbf{p}^*_S)$ should satisfy

$$d(A, S) \geq \frac{\sqrt{N}}{N + \sqrt{N}} \frac{\Pr(R')}{\Pr(R_A)} \left[\Delta_{R'}(\bar{\mathbf{p}}, \mathbf{p}) - (\delta_{R'}(A, S) + \gamma_{R'}(A))\right] \tag{5.6}$$

where R' is the set of rules in the domain $\text{dom}(A, S)$ of the similarity map between concepts A and S. This lower bound was tested by computing the set R', the distance $\Delta_{R'}(\bar{\mathbf{p}}, \mathbf{p})$ between the estimated and actual CPM function, the dissimilarity $\delta_{R'}(A, S)$ between A and the support S, and the irregularity $\gamma_{R'}(A)$.

Several properties of the Reading domain and the descriptors used for transfer make it relatively easy to compute many of these terms. As mentioned above, descriptions uniquely determine a rule and so each description-based similarity class contains only that described rule. The domain $\text{dom}(A, S)$ of the similarity map between concepts A and S is simply the set of rules that are admissible for both concepts. That is, $R' = R_A \cap R_S$ which was computed by iterating through all 81 rules and keeping those that covered at least one positive example of both concepts. The irregularity $\gamma_{R'}(A)$ was zero for all sets R' since the average CPM for each similarity class was just the CPM for



(a) CPM Error over all rules

FIGURE 5.15. Mean CPM Error of ALEPH and DEFT CPM estimates on tasks for concept A. Error bars show 95% confidence intervals.

the single rule in that class. Computing $\Delta_{R'}(\bar{\mathbf{p}}, \mathbf{p})$ was done by evaluating both CPM functions on each rule in R' and taking that average Euclidean distance between the CPMs they returned. The computation of the dissimilarity $\delta_{R'}(A, S)$ was made easier since for each rule $r \in R'$ the prior CPM $\mathbf{q}_T(r)$ is just the CPM $\mathbf{p}_T(r)$ as the similarity class for r only contains r. As the dissimilarity is the average CPM distance over all similarity classes for rules in R' the calculation reduced to computing the average CPM distance over all rules in R'. Finally, the probability measure Pr was taken to be uniform over the rules and so the ratio $\frac{\Pr(R')}{\Pr(R_A)}$ is just the number of rules in R' divided by the number of admissible rules for concept A which is 112.

5.2.10.2. *Results and Analysis.* Figure 5.15 shows the average CPM errors for the estimated CPM functions on the tasks $A_{t,N}$ for the various example set sizes N. The values shown at each size are the mean and 95% confidence interval taken over all the tasks $t = 1, \ldots, 6$ and the five DFTs generated from the support tasks $S_{s,100}$, $s = 1, \ldots, 5$. As can be seen, the CPM error for the DEFT modified CPM estimates are consistently lower on average than the

TABLE 5.9. Dissimilarity between the reading preference concepts. Each cell shows the dissimilarity between the row and column concepts. Entries below the main diagonal are not shown due to the symmetry of the dissimilarity measure on these tasks.

	A	**B**	**C**	**D**	**E**
A	0	0.055	0.047	0.015	0.021
B	.	0	0.019	0.055	0.055
C	.	.	0	0.047	0.047
D	.	.	.	0	0.032
E	.	.	.	.	0

baseline CPM estimate on the smaller example set sizes. The CPM estimates when using concept B or C as support are lower than the raw CPM estimate. Furthermore, the CPM error when using concept C is indistinguishable from those made when using the more similar concepts D and E. While surprising, this is still consistent with the results from the experiments above since CPM error only measures the *average* difference between the estimated and true CPM functions. Generalisation performance, as measured by accuracy or AUC, depends on the other biases of the learner and, ultimately, on which rules are selected for the induced theory. Although the use of concepts C and E as support result in similar CPM error on average, the bias provided by concept E leads to better rules being selected for the induced theories.

To see how these empirical CPM error results compare with those predicted by the theory the values for the terms in Equation 5.6 are required. The number of rules admissible for concept A that were also admissible for concepts B, C, D and E were 42, 46, 80 and 56, respectively making the proportions $\frac{\Pr(R')}{\Pr(R_A)}$ approximately equal to 0.375, 0.411, 0.714, and 0.500. The dissimilarities $\delta_{R'}(A, S)$ are shown in the top row of Table 5.9 and agree with the intuition and earlier results that suggest concepts D and E are more similar to A than B or C.[2]

These values were used to compile Table 5.10 by computing the empirical difference between the raw and posterior CPM error and theoretical lower bound on the tasks of size 10 for concept A. As there were six example sets of size 10 and five DFTs for each support task, a total of thirty differences and bounds were computed for each support task. The values in the second and third columns of the table show the mean values and sample standard deviations. The final column is a tally of the number of times the lower bound was valid out of the 30 trials.

[2]Although it is not the case in general, this table is symmetric because the similarity classes for this domain only contain a single rule.

TABLE 5.10. The empirical difference, theoretical lower bound and number of times the lower bound was respected for the raw and posterior CPM errors for tasks A with 10 examples using the support tasks B, C, D and E.

S	$d(A, S)$	*bound*	$d(A, S) \geq bound$
B	0.0067 (0.0003)	0.0022 (0.0002)	30/30
C	0.0095 (0.0004)	0.0038 (0.0003)	30/30
D	0.0102 (0.0005)	0.0149 (0.0004)	0/30
E	0.0088 (0.0004)	0.0097 (0.0002)	12/30

The theoretical lower bound held for all 30 of the posterior CPM functions built using the support tasks B and C but did not hold at all for support tasks for concept D and less than half the time for concept E. These results show that the Transfer Theorem over-estimated the amount of improvement in CPM error that was expected when using the tasks D and E as support for the similar task A. This discrepancy between theory and practice is due to the approximations made in the implementation of similarity-based transfer as DEFT, the main one being the decomposition of description classes into descriptor frequencies. This simplification, along with the use of sampling techniques to estimate the frequencies, make the DEFT-constructed posterior CPM function differ from the theoretical one. This experiment has shown that these simplifications weaken the guarantees made by the theoretical analysis of similarity-based transfer in the Chapter 3. Adapting the theory to take into account these simplifications is discussed further as future work in Chapter 6.

5.2.11. Summary of Results and Conclusions. The experiments and analysis reported in this section aimed to understand under what circumstances the use of the inductive transfer system DEFT can improve learning performance. The simplicity of the five reading preference concepts meant this question could be thoroughly explored by carefully varying and controlling the many parameters and factors that were suspected to have an influence on inductive transfer. Furthermore, the small number of rules and instances in this domain meant that CPM error could be explicitly computed and compared with the implications of the Transfer Theorem.

The first two experiments, Experiments RP-1 and RP-2, were used to compare the baseline learner ALEPH with DEFT over a range of training set sizes and support and target pairs using a default set of parameter settings. The results of these experiments confirmed that, with an appropriate choice of support task, DEFT can achieve large improvements in accuracy over the baseline learner ALEPH when training examples are limited. These gains were shown

to be due to a preference, imparted by the support task priors, for rules which were more specific than would normally be suggested by the training data alone. DFT construction time was shown to be significant relative to learning times, however the use of DFTs once they were constructed cost very little in terms of overhead.

The results of Experiment RP-3 examined the effect the size of support tasks had on the results from the previous experiments. It was found that DEFT performed best provided there were enough examples to induce an accurate theory from the support task. Increasing the number of examples in support tasks beyond this did not significantly effect the transfer results.

Experiments RP-4 and RP-5 were designed to investigate the effects of varying the DFT construction parameters. In Experiment RP-4 the sampling parameters *exs_samples* and *cls_sampled* were varied and the resulting changes in DFT construction times and transfer accuracies recorded. It was found that of these two parameters varying *exs_sampled* caused the greatest changes in the recorded quantities. DFT construction times grew fastest when increasing *exs_sampled* and too small a value for this parameter had an adverse effect on transfer accuracy. Some heuristics for choosing values for the sampling parameters include making *exs_sampled* large enough to ensure every disjunct in the target concept is sampled from and setting the product of the two parameters to be larger than the value derived from the analysis of BuildDFT in Section 4.3.4. The results of Experiment RP-5 also suggest that the best set of descriptor templates to use is {*has_pred*, *has_arg*}. Transfer using these performed slightly better than using either one alone while the addition of the template num_lits did not have any measurable effect on accuracy.

The admissibility condition used by DEFT when constructing DFTs was investigated in Experiment RP-6. When admissible and non-admissible rules were sampled during DFT construction the overall power of each descriptor-value pair in the resulting DFTs dropped. These differences gave no improvement in transfer results compared to when only admissible rules were sampled. Furthermore, support and target pairs that showed positive transfer effects when only admissible rules were used, such as concepts B and C or A and E, no longer yielded improvements in accuracy when used by DEFT. It was observed that these pairs shared no positive examples and so it was concluded that only admissible rules be sampled during DFT construction, especially if transfer between disjoint concepts is desired.

The amount of influence a DFT has on rule evaluation is controlled by DEFT's M parameter. By default this is set to its maximum likelihood es-

timate, that is, the square root of the number of available training instances in the target task. The results of Experiment RP-7 showed that, in the cases where support concepts had a positive influence on accuracy, changing the value of the M parameter had no significant effect on this influence. Analysis showed that this was due to the linearity of the evaluation function used in these experiments. Increasing the M parameter did, however, magnify the decrease in accuracy when a bad choice of support task was made. The conclusion was that M should be chosen to be small relative to the number of examples N. This is something that he default setting of $M = \sqrt{N}$ ensures.

The culmination of the above results suggest that the choice of parameter settings for DEFT do not have a great effect on the transfer results in the Reading Preferences domain. By and large, the most important factor in determining whether or not the use of DEFT will result in an increase in generalisation accuracy is the choice of support task. This sits well with the intended use of DEFT as a tool which allows a domain expert to select an inductive bias for a limited data learning task without necessarily requiring a deep understanding of the details of the learning and transfer algorithms.

Finally, the experiment and analysis reported in Experiment RP-8 suggests more work is needed in order to improve the theory of Chapter 3 to take into account the simplifications made in order to implement DEFT. In order to make the computation of prior CPM functions tractable it was necessary to assume the descriptors used in a transfer were independent. This is clearly not the case in the experiments performed above and this difference is reflected by the theoretical CPM error improvement being larger than those observed empirically.

5.3. The Chess Environment

Chess has long been studied by artificial intelligence researchers as it ties together many areas of the discipline: planning, knowledge representation, learning, search, and pattern recognition. From an ILP researcher's point of view, the arrangement and movement of pieces is best represented relationally, using predicates to state which piece threatens other pieces, the relative position of pieces, and movement on the board.

Learning symbolic descriptions of rules and strategies for chess has had a long history. The problem of learning descriptions of legal moves for various pieces from examples was introduced by Vere [1977] and later studied by De Raedt and Bruynhooghe [1992], Datta and Kibler [1993], and Khan et al. [1998] as an environment of related learning tasks. Both Khan et al. [1998] and Datta and Kibler [1993] used the chess movement environment to investigate

inductive transfer using predicate invention. They only considered moves on empty chessboards. In Datta and Kibler's work, moves by the Rook, Bishop, Queen and Knight were considered as a sequence with concepts shared between each task. In the Repeat Learning (RL) work by Khan and others,[3] only King and Knight movements were considered, with invented predicates being transferred from King to Knight, King to King, Knight to King and Knight to Knight. The same datasets and methodology used by Khan et al. [1998] are used here to evaluate DEFT.

Of the five inductive transfer approaches reviewed in Section 2.5, Repeat Learning was the most appropriate inductive transfer approach with which to compare DEFT. The foremost reason for this is that RL is the simplest of all of the approaches and was originally implemented to use the same base learning algorithm as DEFT. This not only meant the experiment reported here was easier to set up and run but also that differences in the performance of the two systems would be due to transfer effects and not properties of the base learner. Furthermore, comparisons with the other inductive transfer approaches to rule learning seemed to have dubious benefit. The predicate invention and rule model approaches of MOBAL-MAT and CLINT-CIA were designed for learning apprentice systems and require interaction with a human expert. The relational cliché techniques used by CLUSE and LRC extend FOCL's greedy, hill-climbing search by adding extra search actions to it, however the search performed by ALEPH is complete so the addition of extra search actions via clichés would, at best, only reorder the search. Finally, the concept sharing approach of MFOCL is a special case of the generalised repeat learning used in these experiments.

The experiment reported here had three main aims. The first was to test DEFT on another environment of tasks. As the chess tasks have been used to test other approaches to inductive transfer they make a good, independent testbed for DEFT. The second was to compare the performance of two different approaches to inductive transfer, namely DEFT and Repeat Learning. To this end, the RL algorithm was re-implemented for these experiments so as to use ALEPH as the base level learner. This not only allowed for a fairer comparison of the two algorithms but also functioned as a replication of the original RL experiments. The third aim was to see if the two inductive transfer methods could be combined and whether such a combination is beneficial. The effect DEFT and RL have on a learner's bias are more or less independent: the former modifies its evaluation bias while the latter modifies its language bias through

[3]See Section 2.5 for an overview of RL.

TABLE 5.11. Description of the predicates used in the Chess domain. The `king/2` and `knight/2` predicates are the target predicates for the King and Knight concepts respectively. The type `f` and `r` specify the types "file" and "rank" respectively. The '?' mode is shorthand for input (+), output (-) and constant (#).

Predicate Mode	Description
`rdiff(+r, +r, ?int)`	Third argument is the number of ranks from first to second.
`fdiff(+f, +f, ?int)`	Third argument is the number of files from first to second.
`king(pos(+f,+r), pos(+f,+r))`	Moving from first argument's file and rank to second is a legal King move.
`knight(pos(+f,+r), pos(+f,+r))`	Moving from first argument's file and rank to second is a legal Knight move.

the invention of predicates on a support task which are then used when learning on the target task. A search using RL invented predicates can be guided by an evaluation function modified by DEFT.

5.3.1. Materials. The chess movement tasks involve learning rules to describe the legal moves of King and Knight pieces on an empty board.[4] The predicates to be learnt are `king/2` and `knight/2` where the first argument of each predicate is the start position, represented by the function `pos(File,Rank)`. The second argument holds the end position, also represented using the function `pos/2`. The background knowledge consists of the predicates `rdiff/3` and `fdiff/3`. The former describing the difference between two ranks, `rdiff(R1,R2,Diff)`, and the latter the difference between two files, `fdiff(F1,F2,Diff)`. These predicates and their modes are summarised in Table 5.11.

5.3.1.1. *Datasets.* This experiment used the background and example sets made available by Khan et al. [1998], who describe their construction as follows. First, all 64×64 possible pairs of chessboard positions were generated. These were then classified according to whether they are legal moves for the Knight or King resulting in two "gold standard" datasets Ki_{test} for the King movement concept and Kn_{test} for the Knight movement concept. Both the

[4]Some preliminary experiments were tried using Queen, Rook and Bishop movement but compared to the Knight and King movement domains they are very easy to learn. Both the Rook and Bishop movements can be described in two short clauses while Queen movement is just the disjunction of the Rook and Bishop theories. ALEPH was able to learn accurate theories from tiny datasets obviating the need for inductive transfer.

King and Knight datasets contained roughly 8 times more negative examples than positive examples.

Training sets were then derived from the complete datasets by random sampling with replacement. In total, 140 training sets were created for each of the two chess pieces. Twenty tasks, indexed by $t = 1, \ldots, 20$, were created for seven different training set sizes $N \in \{10, 20, 30, 40, 60, 80, 160\}$. For convenience, these will be denoted $Ki_{t,N}$ for the King tasks and $Kn_{t,N}$ for the Knight tasks. Each task was constructed to contain an equal number of positive and negative examples.

5.3.1.2. *Algorithms.* As well as DEFT, three other types of learning algorithms were applied to the chess tasks. These were ALEPH, RL and RL+DEFT. The RL algorithm is a re-implementation and slight extension of the Repeat Learning system of Khan et al. [1998] and RL+DEFT denotes its simultaneous use with DEFT as described below. As in the reading preferences environment, ALEPH was used by itself as a baseline for learning performance and complexity.

The RL Algorithm - In its original incarnation, the implementation of repeat learning used PROGOL [Muggleton, 1995] as a base level learner and used its built-in constraint solving procedures to invent predicates from support tasks. Predicate invention is performed as part of a standard induction by the base learner. As described in Section 2.5 of the literature review, the predicates invented from support tasks using this technique are then added to the background knowledge when learning on a target task.

RL, the implementation of repeat learning used in this experiment, differs from the original in two ways. Firstly, ALEPH rather than PROGOL is the system used as the engine that performs the search during the invention and reuse stages of repeat learning. As both systems use inverse entailment for their search the effects of this difference are negligible.

The second modification RL introduces is to automate the reuse of invented predicates. In its original implementation, Repeat Learning writes to file the models of the invented predicates. The experimenter then chooses which predicates to reuse on a target task by adding their models to the background knowledge and setting the mode and type information that is given to the base learning system. As the newly invented predicates are only defined in terms of ground models the applicability of the approach is limited. The main difference between that original form and its implementation here is that *all* the predicates invented on the support task are reused on the target. No selection of appropriate predicates by the experimenter takes place.

For the chess tasks, RL was configured to invent predicates over the constants appearing in the third argument of the `fdiff/3` and `rdiff/3` predicates. This is consistent with the way in which repeat learning was used in the experiments by Khan et al. [1998]. Predicates invented in this way should exploit symmetries in the Knight and King movement definitions by allowing clauses to express disjuncts of file and rank differences.

The RL+DEFT Algorithm - The effect DEFT and RL have on a learner's bias are more or less independent: the former modifies its evaluation bias while the latter weakens its language bias. A search using RL invented predicates can be guided by an evaluation function modified by DEFT.

Combining the RL and DEFT algorithms for inductive transfer is straightforward. First, the RL predicate invention method is applied to a support task. DEFT's consolidation procedure BUILDDFT is then applied separately to the same support task. This results in a set of invented predicates and a DFT for the support task. To apply RL+DEFT to a target task the invented predicates are added to the target background knowledge as just described for RL. The DFT built on the support task is then used by DEFT on the target task in the normal manner.

It is worth noting that the DFT build on the support task will not mention any of the predicates invented by RL as the invention and consolidation processes are run separately. This was the simplest way of combining the two algorithms. Many of the clauses generated on the target task will, however, contain the invented predicates. These will have no effect on the construction of CPM priors by DEFT using the DFT.

5.3.1.3. *Settings.* All three inductive transfer methods, DEFT, RL and RL+DEFT, use ALEPH as their base level learner. The settings used for ALEPH were kept constant across all the methods and are summarised in Table 5.12. The settings for *clauselength*, *i* and *nodes* are slightly higher than those used in the Reading Preferences environment as the target concepts for chess movement are slightly more complicated. It is important to note that the ALEPH settings described here are the same ones used by RL when inventing as well as using predicates.

TABLE 5.12. Settings used by ALEPH and DEFT for experiments within the Chess domain

Learner	Setting	Value
ALEPH	*clauselength*	6
	i	3
	minacc	1.0
	nodes	1000
BUILDDFT	*admissibility*	true
	exs_sampled	40
	cls_sampled	10
	templates	{has_pred, has_arg}
DBSCORE	M	$\sqrt{N}$

Table 5.12 also summarises the default settings for DEFT in the chess experiment. These are predominantly the same as those used in the reading preferences environment. Default values of 40 and 10 were chosen for the two sampling settings *exs_sampled* and *cls_sampled* respectively. These were chosen for reasons similar to those for the reading preferences environment in Section 5.2 above. That is, 40 examples are sampled as this is half of the total number of positive examples and 10 clauses per example were sampled to ensure the total number of clause evaluations was larger than the lower bound derived in the previous chapter.

5.3.2. Experiment Chess-1: Comparison of ALEPH, DEFT and RL. In this experiment the generalisation performance of DEFT, RL and the combination of DEFT and RL were all compared to the baseline performance of ALEPH on the King and Knight movement tasks. As discussed in the introduction to this section, the aims of this experiment are to try DEFT on a new environment, compare DEFT with repeat learning, and determine whether there is any value in combining the two approaches to inductive transfer. All three aims are carried out using the King and Knight as a support and target pair. When King tasks were used as a target, the inductive transfer algorithms used a large Knight task as support and *vice versa*. In each case, DEFT used the support task to construct a DFT while the RL algorithm used the support task to invent predicates. When RL+DEFT were used in conjunction on a target task both the priors from the DFT and the invented predicates were used. The details of this procedure are described in the method below.

5.3.2.1. *Method.* The method used for this experiment can be broken up into three main phases: 1) establishing the baseline learning results for ALEPH, 2) DFT construction using DEFT and predicate invention using RL, and 3) applying DEFT, RL and RL+DEFT to the target tasks. The first phase consisted

of the following steps. For each target concept $T \in \{Ki, Kn\}$ ALEPH was run on the 140 tasks $T_{t,N}$ for training set size $N \in \{10, 20, 30, 40, 60, 80, 160\}$ and task index $t = 1, \ldots, 20$. The resulting theories were tested on the corresponding test set T_{test} for the target concept.

For the second phase, DFT construction and predicate invention was carried out using the following two steps:

(1) Descriptor frequency tables Φ_{king} and Φ_{knight} were created for King and Knight movement concepts respectively. The was done by applying DEFT to the tasks $Ki_{1,160}$ and $Kn_{1,160}$ using the sampling settings and descriptor templates described above.
(2) Predicates were invented using the RL algorithm for the support tasks $Ki_{1,160}$ and $Kn_{1,160}$. RL was restricted to invented predicates over the constants appearing in the third arguments of the `fdiff/3` and `rdiff/3` predicates. The sets of invented predicates are denoted Q_{king} and Q_{knight} respectively.

Finally, DEFT, RL and RL+DEFT were applied to the target tasks using the results of the above consolidation steps. The method used for this third phase was as follows:

(1) First, DEFT was applied to all 140 of the King and Knight tasks using the Φ_{king} as the support DFT when applied to the Knight tasks. The Φ_{knight} DFT was used when DEFT was applied to the 140 King tasks.
(2) RL was applied to the 140 tasks for the King concept using the invented predicates Q_{knight} as extra background knowledge. Similarly, RL was also applied to the 140 Knight tasks with the Q_{king} predicates added to the background knowledge.
(3) Finally, RL+DEFT was applied to each of the 140 King tasks using the Q_{knight} predicates as extra background knowledge and prior CPMs were created using the DFT Φ_{knight}. Similarly, RL+DEFT was applied to the 140 Knight tasks using Q_{king} as extra background and the DFT Φ_{king} was used for the creation of priors.
(4) All the theories induced from the 280 tasks $T_{t,N}$ in the previous three steps were tested on their corresponding test set T_{test}. The true positive and false positive rates computed from the test set were then averaged to determine the AUC score for each theory. The number of clauses and literals used in each theory was also recorded.

The AUC score used here is the same performance measure as used by Khan et al. [1998], though they only refer to it as the average of the true positive and true negative rates. These two measures are equivalent [Fawcett, 2004] and are

TABLE 5.13. Default priors and descriptor power for the DFTs for the King and Knight tasks.

		Task			
		King		*Knight*	
has_pred	**has_arg**	**Power (β)**	**Prob. (π)**	**Power (β)**	**Prob. (π)**
`fdiff`	`-`	1.04	0.99	0.37	0.99
	`-2`	-	0	2.66	0.09
	`-1`	1.62	0.25	0.79	0.24
	`0`	-0.34	0.61	-0.32	0.56
	`1`	1.82	0.27	0.66	0.17
	`2`	-	0	2.76	0.11
`rdiff`	`-`	-0.81	0.99	0.47	0.99
	`-2`	-	0	1.40	0.21
	`-1`	1.52	0.26	1.49	0.09
	`0`	-0.65	0.62	-0.25	0.64
	`1`	1.48	0.28	1.51	0.12
	`2`	-	0	1.32	0.21

used instead of test accuracy due to the large class imbalance of around eight negative examples for every positive example in the test sets. As in the reading preference experiments, mean values and standard deviations for the theories induced by each learning algorithm were calculated for each target task T and training set size N by averaging over all twenty task indices $t = 1, \ldots, 20$.

5.3.2.2. *Results and Analysis.* The results and analysis of this experiment are presented in three parts. The first part examines the auxiliary information - the DFTs and invented predicates - created by DEFT and RL. The comparison of these support task summaries for the two target concepts gives some insight into how the respective algorithms modify the bias used by the base learner. The second part reports the AUC scores achieved by the baseline learner, DEFT, RL and the combination RL+DEFT on the King and Knight tasks. Statistical comparisons of AUC scores are performed between all four learning algorithms and reveal improvements by all three inductive transfer approaches. The final part of this section attempts to understand the gains in AUC by examining the theories induced by the baseline learner and each of the transfer methods.

A summary of the DFTs constructed for this experiment is provided in Table 5.13. The presentation used is the same as for the DFTs in the reading preference experiments, showing the power for each of the non-default descriptor-value pairs. The powers for the descriptor-value pairs in the DFTs built from the King and Knight tasks show a strong correspondence indicating that the well performing clauses for the two concepts are similar. In particu-

TABLE 5.14. Predicates invented by RL on the King and Knight tasks.

Invented predicates Q_{king}	Invented predicates Q_{knight}
`q0(-1). q0(0). q0(1).`	`q3(-2). q3(-1). q3(1). q3(2).`
`q1(-1). q1(1).`	`q4(-1). q4(1).`
`q2(-1). q2(0).`	`q5(-2). q5(-1). q5(1).`
	`q6(-2). q6(2).`
	`q7(-1). q7(1). q7(2).`
	`q8(-2). q8(-1).`
	`q9(1). q9(2).`

lar, the use of the `fdiff(F1,F2,-1)` and `fdiff(F1,F2,1)` literals correlates well with correct positive predictions on both tasks. The descriptors present in the Knight DFT but missing in the King DFT are due to the difference in admissible clauses for each concept. Examples of Knight movements always involve a rank or file difference of two whereas a King changes its rank or file by at most one. When learning using these DFTs as support, a preference should be shown towards clauses which use these constants over those which do not.

Predicates invented by RL on the King and Knight tasks are shown in Table 5.14. Each predicate is defined by its ground instances which range over integer values. On the King task RL invented three distinct predicates, `q0` through `q2`, each defining a disjunction of values between -1 and 1. On the Knight task, RL invented seven predicates `q3` through `q9`, each defining a disjunction of values between -2 and 2. The predicates `q1` and `q4` express the same disjunction of values, namely -1, 1. This disjunction is particularly useful for learning both the King and Knight concepts as it can be used to express differences of one square in either direction along a rank or file. It is expected that these two predicates will feature prominently in the theories induced by RL and RL+DEFT when these are examined below.

The AUC scores for all three transfer methods and the baseline learner are summarised for the two chess concepts in Tables 5.15 and 5.16. The first table shows the scores on the King tasks at the seven different training sizes while the second reports the same information for the Knight tasks. For both chess concepts the AUC scores for theories learnt by ALEPH increase as more training examples become available. Theories learnt from 160 examples consistently have almost perfect prediction regardless of the learning algorithm used. An example of a theory learnt by Aleph on a large King task is shown in Figure 5.16. This shows how each of the eight possible King moves are each covered by a single clause.[5]

[5]The other clause `king(pos(A,B),pos(A,B))` covers the case when the King stays in its starting position. This was added as a "legal" King move in the experiments reported by Khan et al. [1998] and so it is done so here.

```
[ +8,-0] king(pos(A,B),pos(A,B)).
[+11,-0] king(pos(A,B),pos(A,C)) :- rdiff(C,B,-1).
[+10,-0] king(pos(A,B),pos(A,C)) :- rdiff(C,B,1).
[ +7,-0] king(pos(A,B),pos(C,B)) :- fdiff(C,A,-1).
[ +6,-0] king(pos(A,B),pos(C,B)) :- fdiff(C,A,1).
[ +9,-0] king(pos(A,B),pos(C,D)) :- rdiff(D,B,-1), fdiff(C,A,-1).
[+12,-0] king(pos(A,B),pos(C,D)) :- rdiff(D,B,1), fdiff(C,A,1).
[+13,-0] king(pos(A,B),pos(C,D)) :- rdiff(D,B,-1), fdiff(C,A,1).
[ +4,-0] king(pos(A,B),pos(C,D)) :- rdiff(D,B,1), fdiff(C,A,-1).
```

FIGURE 5.16. A theory induced by ALEPH from a training set with 160 examples of the King concept. The numbers in square brackets before each clause indicate the number of true positives and false positives that clause covers.

On tasks with fewer than 60 examples DEFT achieves higher AUC scores than the baseline learner ALEPH on both the King and Knight tasks. The differences between these two algorithms becomes larger as fewer examples are available to the learners. The largest average difference of 10% occurs on King training sets with 10 examples. These results are consistent with the behaviour of DEFT observed in the reading preferences domain and suggest that the bias modification made by using support task DFTs improve evaluation estimates when training data is very limited.

The RL algorithm also shows significant gains in AUC score over the baseline learner on both chess concepts. Unlike DEFT, the larger gains due to RL occur on medium-sized King tasks with the largest average improvement of 8.2% recorded on King tasks of size 40 respectively. On the King and Knight tasks with only 10 examples the improvements due to RL are much smaller than those due to DEFT while on the largest training sets both transfer approaches perform as well as the baseline learner. These results suggest that when sufficiently many positive examples become available the predicates invented by RL are used in single clauses to cover sets of examples that would otherwise require several separate clauses.

In the discussion at the beginning of this section it was argued that the hybrid transfer approach should capitalise on the advantages of its component systems as the bias modified by each is independent of the other. DEFT changes the base learner's evaluation by adding CPM priors derived from DFTs while RL changes the representation bias used by the base learner by adding predicates invented from the support task. If this was the case, the improvements made by DEFT on the very small training sets should also been made by RL+DEFT as well as the improvements due to the addition of invented predicates on medium-sized tasks. This behaviour was observed on the King tasks and, to a lesser extent, on the Knight tasks, and in all cases the RL+DEFT

TABLE 5.15. Mean AUC values of all four algorithms as percentages on the King tasks. The figures in brackets show the sample standard deviation. **Bold** entries indicate a score greater than ALEPH while "R" and "D" indicate a score greater than RL or DEFT respectively. All tests are single-sided, paired t-tests at the $p < 0.01$ level of significance.

	ALEPH	DEFT	RL	RL+DEFT
10	61.3 (1.1)	**71.3 (1.4)** [R]	66.3 (1.9)	**73.2 (1.5)** [R]
20	72.4 (1.4)	**79.1 (1.6)**	**78.2 (2.2)**	**82.4 (1.8)** [D,R]
30	77.5 (1.0)	**85.7 (0.7)**	**84.5 (1.1)**	**87.5 (0.9)** [R]
40	84.9 (1.1)	**90.2 (1.2)**	**93.1 (1.3)** [D]	**93.1 (1.1)** [D]
60	91.2 (1.3)	**94.8 (1.0)**	**95.7 (1.3)**	**96.8 (1.0)** [D]
80	95.6 (1.0)	97.1 (0.7)	**98.9 (0.5)** [D]	**98.9 (0.5)** [D]
160	99.6 (0.3)	99.6 (0.3)	99.8 (0.2)	99.8 (0.2)

TABLE 5.16. Mean AUC values of all four algorithms as percentages on the Knight tasks. The figures in brackets show the sample standard deviation. **Bold** entries indicate a score greater than ALEPH while "R" and "D" indicate a score greater than RL or DEFT respectively. All tests are single-sided, paired t-tests at the $p < 0.01$ level of significance.

	ALEPH	DEFT	RL	RL+DEFT
10	62.4 (1.0)	**71.1 (1.1)** [R]	**67.3 (1.5)**	**71.9 (1.1)** [R]
20	71.8 (1.0)	**79.7 (1.2)** [R]	**77.2 (1.7)**	**80.6 (1.5)** [R]
30	79.6 (1.3)	**88.5 (1.0)**	**87.8 (1.3)**	**89.1 (1.2)**
40	86.3 (1.5)	**92.4 (1.3)**	**92.4 (1.5)**	**93.1 (1.3)**
60	93.3 (1.3)	**97.3 (0.8)**	**97.8 (0.7)**	**97.8 (0.7)**
80	96.4 (0.9)	97.9 (0.7)	98.2 (0.6)	98.2 (0.6)
160	99.8 (0.2)	99.8 (0.2)	99.8 (0.2)	99.8 (0.2)

combination performs at least as well as the RL or DEFT algorithms alone.

On the small King tasks (those with 10, 20 and 30 examples) and the small Knight tasks (with 10 and 20 examples) the RL+DEFT combination significantly outperforms RL used by itself. The AUC scores achieved in these cases are at least as good as those reported when using DEFT on its own and better by about 3% than *both* the DEFT and RL algorithms on the King tasks of size 20. When the amount of training data increases to 40, 60 and 80, the hybrid system outperforms DEFT alone on the King tasks with AUC scores comparable to those recorded for RL alone. No significant improvement over DEFT was recorded for these sizes on the Knight tasks as all three approaches had similar AUC scores.

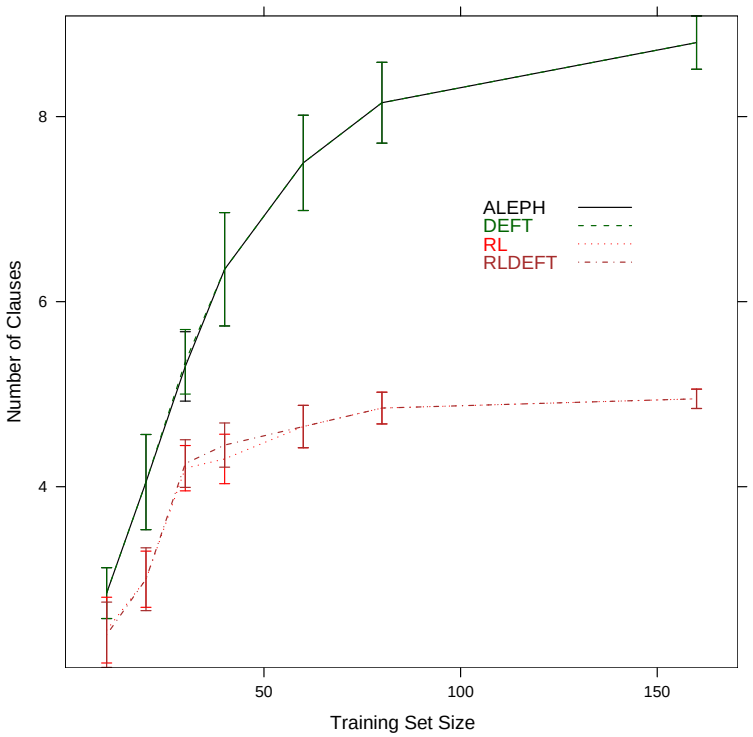
These improvements made by DEFT, RL and RL+DEFT can be better understood by examining the theories induced by each approach. In the reading

preference experiments, counting the number of clauses and literals in each theory proved to be a quick way to form a picture of how DEFT was modifying the preferences of the base learner. This technique is employed again for this experiment and the theory and clause sizes are displayed as functions of training size for the King tasks in Figure 5.17 and the Knight tasks in Figure 5.19.

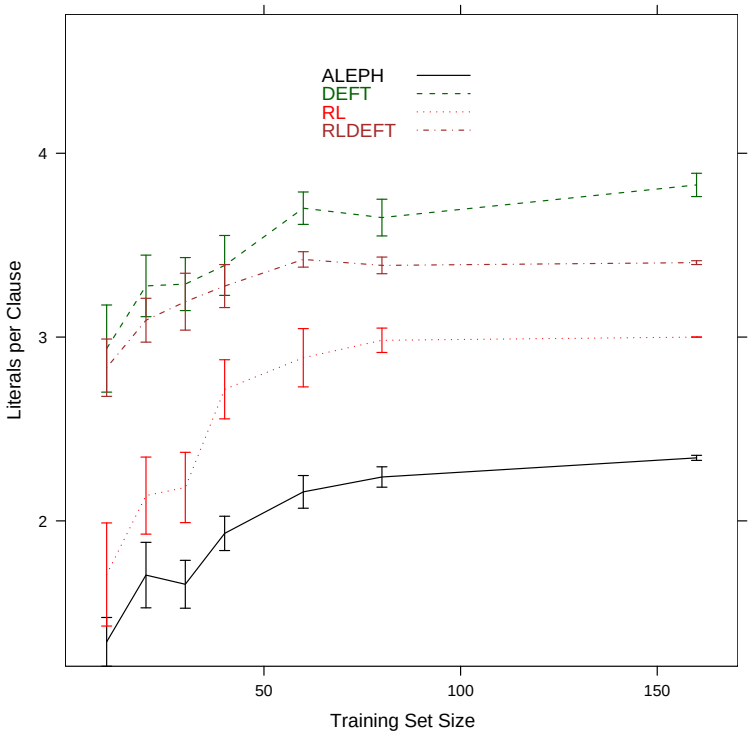
The first graphs in both figures are somewhat difficult to read as the plots for the number of clauses in theories induced by ALEPH and DEFT coincide, as do the plots for RL and RL+DEFT. There is, however, a large difference in the number clauses induced by the RL and non-RL approaches on larger task sizes. The invented predicates used by both RL and RL+DEFT enable accurate theories for both the King and Knight tasks to be represented using half as many clauses as required to express accurate theories without them. The use of these invented predicates can be observed in the plots of average clause length in the second graphs in both figures. On the larger Knight and King tasks RL induces clauses that are, on average, about one literal longer than those found by ALEPH. An example of a theory induced by RL on a large King task is given in Figure 5.18. Each pair of clauses in the theory induced by ALEPH (shown in Figure 5.16) representing moves within a file or rank has been replaced by single one which uses an extra `q4` literal.

Table 5.17 shows how frequently each invented predicate was used in theories induced by RL and RL+DEFT from tasks at each training size for the King and Knight concepts. The predicates `q1` and `q4` are used very often by both RL and RL+DEFT when learning theories for the Knight and King tasks, respectively. This is because the disjunction of values expressed by these predicates allows a greater number of positive examples to be covered with fewer clauses. RL+DEFT also infrequently includes the less useful predicates `q2` and `q9` on the smaller training set sizes.

Longer clauses are a distinct feature of both DEFT approaches on all tasks to which they are applied. This suggests that the preference bias due to the DFTs is causing the base learner to specialise clauses further than the data alone would require. This specialisation is advantageous on the very small datasets as witnessed in the performance results discussed above. More specific clauses have a lower false positive rate and hence theories which use them have a higher AUC score. What is slightly puzzling is why these over-specialised clauses do not reduce AUC scores on the larger training set sizes. An examination of the theories induced by DEFT on the 160 example King tasks showed that many of theories looked like the one shown in Figure 5.20. Here can be seen clauses with extraneous literals and literal combinations which do not affect which instances are covered. For example, the literals like `fdiff(A,A,0)` in



(a) Number of Clauses



(b) Literals per Clause

FIGURE 5.17. Theory complexity on King tasks as a function of training set size for the standard ALEPH algorithm and the three transfer approaches using the Knight concept as support. Error bars show 95% confidence intervals for the mean.

```
[+21,-0] king(pos(A,B),pos(A,C)) :- rdiff(B,C,D), q4(D).
[+22,-0] king(pos(A,B),pos(C,D)) :- rdiff(D,B,-1), fdiff(A,C,E), q4(E).
[+16,-0] king(pos(A,B),pos(C,D)) :- rdiff(D,B,1), fdiff(A,C,E), q4(E).
[+13,-0] king(pos(A,B),pos(C,B)) :- fdiff(A,C,D), q4(D).
[ +8,-0] king(pos(A,B),pos(A,B)).
```

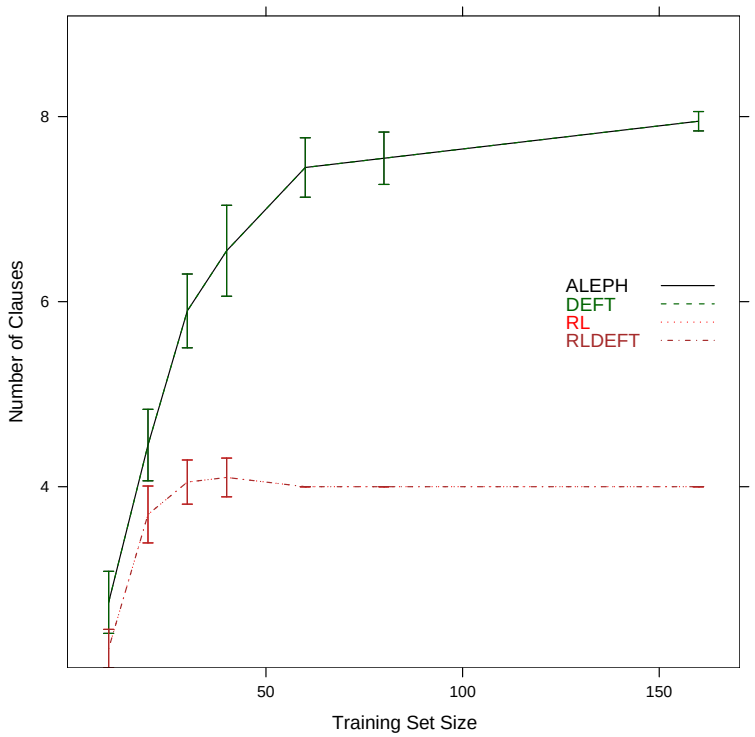
FIGURE 5.18. A theory induced by RL from a training set with 160 examples of the King concept. The numbers in square brackets before each clause indicate the number of true positives and false positives that clause covers.

TABLE 5.17. The distribution of invented predicates in theories induced by RL and RL+DEFT. Columns specify the training task size and the rows describe the algorithm, target task and invented predicate of interest. Each entry shows how many of the twenty tasks at each training size contained the specified predicate.

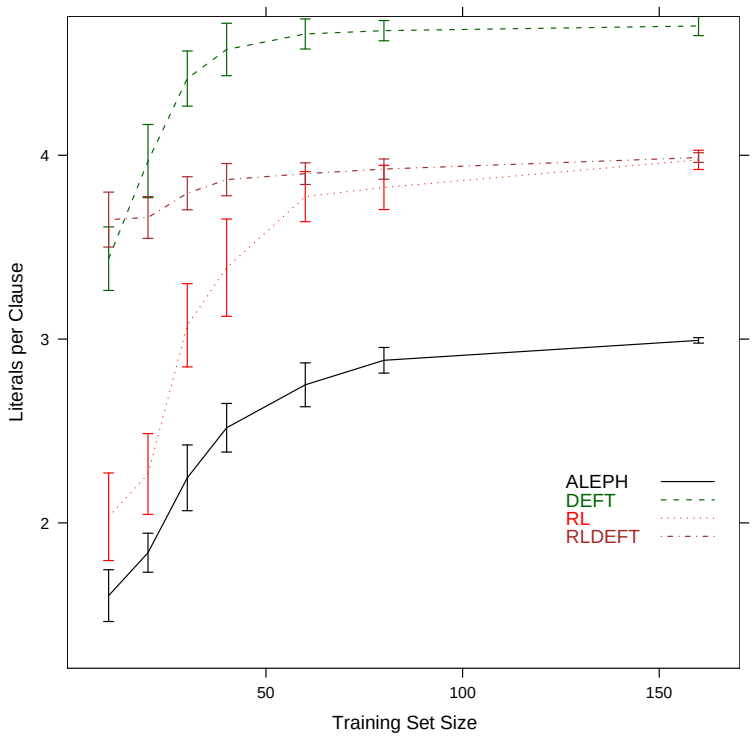
			Training Size						
Alg.	**Task**	**Pred.**	10	20	30	40	60	80	160
RL	King	`q4`	8	15	16	19	20	20	20
	Knight	`q1`	9	13	19	20	20	20	20
RL+DEFT	King	`q4`	11	16	17	18	20	20	20
		`q9`	4	1	6	2	1	0	0
	Knight	`q1`	11	13	19	20	20	20	20
		`q2`	1	1	0	0	0	0	0

the first through fifth clauses are tautologies. In the sixth through ninth clauses combinations like `rdiff(D,B,-1), rdiff(B,D,E), fdiff(A,C,E)` are just verbose ways of writing the simpler conjunction `rdiff(D,B,-1), fdiff(A,C,1)`. Adding these extra literals to a clause will not change its CPM estimate but it will change the CPM priors calculated from the Knight DFT. Since the Knight DFT assigns a slightly larger positive power to `has_pred(fdiff/3)` than the negative power assigned to `has_arg(fdiff,3,0)` the net change in the CPM prior will be to increase the true positive rate more than the false positive rate and so the evaluation of such a clause with the additional literal will be positive. These changes in evaluation affect both DEFT and RL+DEFT equally as the addition of predicates invented by the RL component of RL+DEFT does not change the CPM priors for a clause.

5.3.3. Conclusions. The results for the above experiment provide a number of insights into inductive transfer using both description-based transfer and repeat learning. Firstly, DEFT was shown to work on a second environment of related tasks with results that were competitive with a representation-based inductive transfer technique. The chess movement tasks are ideally suited to



(a) Number of Clauses



(b) Literals per Clause

FIGURE 5.19. Theory complexity on Knight tasks as a function of training set size for the standard ALEPH algorithm and the three transfer approaches using the King concept as support. Error bars show 95% confidence intervals for the mean.

```
[ +8,-0] king(pos(A,B),pos(A,B)) :- fdiff(A,A,0).
[+11,-0] king(pos(A,B),pos(A,C)) :- rdiff(C,B,-1), fdiff(A,A,0).
[+10,-0] king(pos(A,B),pos(A,C)) :- rdiff(C,B,1), fdiff(A,A,0).
[ +7,-0] king(pos(A,B),pos(C,B)) :- fdiff(A,A,0), fdiff(C,A,-1).
[ +6,-0] king(pos(A,B),pos(C,B)) :- fdiff(C,A,1), fdiff(C,C,0).
[ +9,-0] king(pos(A,B),pos(C,D)) :- rdiff(D,B,-1), rdiff(B,D,E), fdiff(A,C,E).
[+12,-0] king(pos(A,B),pos(C,D)) :- rdiff(B,D,E), fdiff(C,A,1), fdiff(A,C,E).
[+13,-0] king(pos(A,B),pos(C,D)) :- rdiff(D,B,-1), rdiff(B,D,E), fdiff(C,A,E).
[ +4,-0] king(pos(A,B),pos(C,D)) :- rdiff(B,D,E), fdiff(C,A,-1), fdiff(C,A,E).
```

FIGURE 5.20. A theory induced by Deft applied to a training set with 160 examples of the King concept. The numbers in square brackets before each clause indicate the number of true positives and false positives that clause covers.

the predicate invention and transfer employed by repeat learning. Describing the King and Knight movements using the rank and file difference predicates requires many small disjuncts, each handling different directions of what are symmetrical movements. As the results of this experiment showed, the predicates invented by RL are able to capture some of the symmetries making it easier to define the pieces' movements using fewer clauses. Somewhat surprisingly then, DEFT was also able to improve generalisation performance over the baseline learner to a degree similar to that of RL. On very small training set sizes, DEFT outperformed RL suggesting that, under these circumstances, the use of priors to improve evaluation estimates is a more appropriate approach to bias learning than a representational change such as predicate invention.

The second observation was that the combination of these techniques was shown to be better than either used alone. This showed that the strengths of each type of bias transfer - evaluation-based and representational - was combined without any detrimental effect. DEFT is able to specialise clauses without an abundance of negative examples. In a complementary fashion, the invented predicates used by RL are helpful when there are sufficient negative but too few positive examples. Each rule using the invented predicates can cover what would require several small disjuncts without them. The combination of these pressures, as exhibited by RL+DEFT results in a hybrid inductive transfer system that has a larger range of task sizes for which it will perform well.

The final observation that came out of the above experiment and analysis was that DEFT can sometimes add too many literals to a clause without affecting the instances it covers. As discussed above, this extra-evidential specialisation is what makes DEFT useful when training data is limited. However, more work needs to be done on ways to avoid the addition of unnecessary literals as the resulting theories are more difficult to understand than their

counterparts induced without DEFT.

5.4. Heart Disease Environment

The previous two experiments provide evidence for the utility of DEFT as a inductive transfer technique on two different artificial domains. The main purpose of this experimental section is to examine whether these positive results carry over to data taken from a real world source.

The Heart Disease domain consists of three tasks, each of which require a learner to predict whether or a patient has heart disease based on several of their readily available features such as the patient's age, gender, blood pressure, ECG readings and the type of chest pain the patient is reporting. Each task consists of patient data from one of three hospitals, one in the U.S.A., one in Canada and the other in Hungary. These tasks are believed to be related as they all require a learner to construct a model for predicting heart disease, however each task may be slightly different due to differences in the patient populations in each country.

These tasks were first used in a inductive transfer context by Silver [2000] in his Ph.D. thesis to empirically test his Task Rehearsal Method (TRM) approach to inductive transfer for artificial neural networks. An overview of the TRM is given in Section 3.6.3 above. The purpose of the experiments in this section was to apply DEFT to the same set of tasks and compare its performance with Silver's TRM approach for neural networks.

The experimental set up and method to test transfer using DEFT on versions of the tasks with artificially limited data are given in the Sections 5.4.1 and 5.4.2. The results presented in Section 5.4.3 show that using DEFT on these tasks never harms generalisation performance relative to learning without transfer and occasionally outperforms it. These improvements using DEFT are then compared in Section 5.4.4 to "theory transfer": simply applying a theory learnt on a support task to the target task. DEFT is found to outperform theory transfer in a number of cases suggesting that DEFT's use of support and limited target task information can work better than the exclusive use of support data. Many of these results are at least qualitatively similar to those reported by Silver for his TRM approach and are discussed in Section 5.4.5 before concluding with Section 5.4.6.

5.4.1. Materials. The tasks used in these experiments are derived from the Heart Disease dataset found at the UCI Repository [Newman et al., 1998].[6]

[6]In accordance with the dataset's terms of use, the following institutions and researchers are acknowledged for providing the original data from which these tasks were derived: Hungarian Institute of Cardiology, Budapest: Andras Janosi, M.D ; University Hospital, Zurich,

TABLE 5.18. Description of the predicates used in the Heart Disease domain. Each patient has a unique identifier of type `id`. Values in the description shown in `typewriter` font are values for the constant mode argument (`#arg`) in the predicate. The `disease/1` predicate is the learning target.

Predicate Mode	**Description**
`age(+id, -number)`	The age of the patient in years.
`sex(+id, #sex_type)`	The gender of the patient (`male`, `female`).
`cp(+id, #cp_type)`	Type of chest pain: typical angina (`typical`), atypical angina (`atypical`), non-anginal pain (`non`), or asymptomatic (`asympt`).
`trestbps(+id, -number)`	Resting blood pressure upon admission in mm Hg.
`restecg(+id, #restecg_type)`	Resting electrocardiograph results: normal (`normal`), having ST-T wave abnormality (`abnormal`), or showing left ventricular hypertrophy (`hyper`).
`lteq(+number, #number)`	The "less than or equal to" relation.
`gteq(+number, #number)`	The "greater than or equal to" relation.
`disease(+id)`	Patient has a greater than 50% narrowing of any major coronary artery .

Each task consists of data taken from patients at the Hungarian Institute of Cardiology in Budapest (*hung*), the V.A. Medical Center in Long Beach (*vamc*), and the Cleveland Clinic Foundation (*cleve*). The target concept for each task is to predict whether or not a patient at the hospital is suffering from heart disease based on their physical attributes, the results of various tests and the symptoms they present. In order to compare the work here to the results obtained by Silver's study of the same domain, the features used in this experiment are the same as those described in [Silver, 2000, §6.3] and are summarised in Table 5.18 along with their mode and type constraints.

The disease/1 predicate is taken to be the learning target for tasks in this experiment. Table 5.19 shows the number of examples in each of the three datasets as well as their class distribution.[7] Notably, the class distribution of the *vamc* data has a ratio of roughly 3:1 positive to negative examples while the *cleve* and *hung* datasets have ratios close to 1:1 and 1:2, respectively.

Switzerland: William Steinbrunn, M.D. ; University Hospital, Basel, Switzerland: Matthias Pfisterer, M.D. ; V.A. Medical Center, Long Beach and Cleveland Clinic Foundation: Robert Detrano, M.D., Ph.D.

[7]There is a small discrepancy between the size and skew of the datasets used here and those reported in [Silver, 2000]. His example counts for *hung* and *vamc* also differ from those in the original UCI dataset.

TABLE 5.19. Summary of the Heart Disease tasks showing the total number of examples available from each hospital as well as their class distributions.

Dataset	Hospital	# of Pos.	# of Neg.	Total
cleve	Cleveland Clinic	125	157	282
hung	Hungarian Institute of Cardiology	106	188	294
vamc	V. A. Medical Center	149	51	200

The parameter settings used by ALEPH and DEFT in this experiment are given in Table 5.20. The value 0.8 was chosen for the *minacc* parameter after some preliminary runs at a variety of *minacc* settings. Using this value, ALEPH was able to induce theories from all three tasks with relatively high AUC scores. The large value for the *exs_sampled* setting was chosen as a result of the analysis of the experiment reported in Section 5.2.6 above suggest that larger values of this setting result in better DFTs. The `has_argsym` descriptor template was used instead of the `has_arg` template from the previous experiments. Unlike the `has_arg` template used in the experiments of the previous sections, `has_argsym` only constructs `has_arg` descriptors for non-numeric arguments. This was done to avoid a large number of descriptors like `has_arg(gteq,2,57)` that DEFT constructed using the original `has_arg` template on preliminary runs on the datasets. The experiments in this section were tried with both the `has_arg` and `has_argsym` templates and no significant difference in performance was detected so the `has_argsym` template was used to keep the DFTs smaller and easier to analyse.

TABLE 5.20. Settings used by ALEPH and DEFT for experiments within the Heart Disease domain

Learner	Setting	Value
ALEPH	*clauselength*	6
	i	2
	minacc	0.8
	nodes	5000
BUILDDFT	*admissibility*	true
	exs_sampled	100
	cls_sampled	50
	templates	{has_pred, has_argsym}
DBSCORE	M	$\sqrt{N}$

5.4.2. Method. The efficacy of inductive transfer using DEFT compared to standard, single task learning using ALEPH was measured by evaluating theories induced by both algorithms using four performance metrics: accuracy,

true positive rate, true negative rate and AUC. Due to its invariance to class distribution, AUC will be used as the primary measure of learning performance while the other measures are used for analysis (TPR, FPR) and comparison to Silver's results (accuracy).

Repeated measurements on each task were turned into summary statistics using ten-fold cross-validation. That is, letting $E = F_1 \cup \ldots \cup F_{10}$ denote a dataset and its ten folds, the i^{th} test set is F_i and the corresponding training set $E_i = E - F_i$. When using the complete datasets, learners are trained on the examples in each E_i and the resulting theories tested on F_i. The collection of ten pairs for each hospital will be denoted by the dataset names *cleve*, *hung* and *vamc*. Performance results are determined by averaging the values recorded on each of the test folds.

Limited data training sets, denoted $\hat{E}_i$, were constructed by randomly drawing five positive and five negative examples without replacement from each training fold E_i. Learners were applied to each $\hat{E}_i$ and the resulting theories tested on the corresponding test examples F_i. The collection of ten limited training and test pairs for each hospital are denoted $cleve_{10}$, $hung_{10}$ and $vamc_{10}$. Constructing the limited data training sets in this way ensures that the test sets are independent. Furthermore, the test sets are the same as those used for the full training data trials and so results for the limited and full data trials can be compared using paired t-tests.

To establish the baseline learning results, ALEPH was used to induce theories for the full data and limited data collections for each of the three hospitals using the settings described in the section above. DEFT was also applied to the three limited data collections $cleve_{10}$, $hung_{10}$ and $vamc_{10}$ using DFTs constructed from each of the complete datasets *cleve*, *hung* and *vamc*. Transfer between support and target tasks for the same hospital were not attempted as the examples for these pairs are not independent. Excluding these pairs left a total of six transfer pairs: *cleve* to $hung_{10}$ and $vamc_{10}$, *hung* to $cleve_{10}$ and $vamc_{10}$, and *vamc* to $cleve_{10}$ and $hung_{10}$. A single DFT for each of the datasets *cleve*, *hung* and *vamc* was built using the settings shown in Table 5.20 and used for all transfers involving the respective support task. Preliminary experiments showed that there was little to no change in the transfer results when other DFTs were constructed and used.

5.4.3. Results and Analysis. The results of using DEFT to transfer bias between the hospital tasks are summarised in Table 5.21. Each row gives the results for a specific support and target pair and the entries in a row report the mean and sample standard deviation for each of the four performance

TABLE 5.21. Accuracy, true positive/negative rates and AUC scores on the Heart Disease tasks with 10 examples. Entries in **bold** are significantly larger than the baseline figures at the $p < 0.05$ level using a single-sided paired t-test.

Target	Support	Accuracy	TPR	TNR	AUC
$cleve_{10}$	*none*	65.4 (3.0)	65.2 (8.9)	65.5 (4.7)	65.4 (3.4)
	hung	72.1 (2.9)	58.6 (7.8)	**82.8 (2.8)**	70.7 (3.2)
	vamc	72.4 (2.9)	64.4 (8.3)	**79.0 (3.3)**	71.1 (3.3)
$hung_{10}$	*none*	60.8 (5.0)	74.0 (5.3)	53.5 (8.4)	63.8 (4.0)
	cleve	67.0 (5.2)	71.0 (4.5)	64.6 (8.0)	67.8 (4.4)
	vamc	64.5 (4.5)	68.0 (4.3)	63.0 (7.7)	65.5 (3.6)
$vamc_{10}$	*none*	36.5 (3.5)	29.6 (5.7)	57.0 (8.7)	43.3 (3.5)
	cleve	**50.5 (3.0)**	**48.3 (3.2)**	57.0 (9.2)	**52.6 (4.6)**
	hung	**48.6 (4.1)**	**45.7 (5.0)**	56.3 (9.4)	51.0 (4.9)

measures. The rows are split into three groups according to which task was used as the target. The first row in each group shows the performance results for the baseline learner ALEPH while the other two show the results for DEFT using DFTs constructed from the task shown in the second column.

In all cases the AUC for the DEFT results are consistently higher than the corresponding measurements for the baseline learner indicating that the evaluation bias transfered by DEFT has a positive effect on learning performance. The AUC results for the *cleve* to $vamc_{10}$ transfer is statistically significantly higher than the baseline results at the $p < 0.04$ level using a single-sided paired t-test. The AUC improvement when transferring bias from *hung* to $vamc_{10}$ is at the $p < 0.09$ level of significance. From these results it can be concluded that DEFT was able to successfully transfer bias between all the tasks in this domain but, apart from transfer to $vamc_{10}$, the resulting improvements are not as significant as those observed in the Reading or Chess environments.

One reason the improvements due to transfer were not significant on the $cleve_{10}$ and $hung_{10}$ tasks is because the theories induced from the limited training sets by the baseline learner were almost as good as those learnt from the full training sets. The entries in Table 5.22 show the mean performance measures and sample standard deviations for the baseline learner on the full training sets *cleve*, *hung* and *vamc*. A comparison of the limited dataset and full dataset results reveals that *vamc* is the only learning task that is made dramatically more difficult by limiting the training examples. Whereas the mean paired differences between the full and limited AUC results for the *cleve* and *hung* tasks are 1.8% and 1.1% respectively, the mean difference for the *vamc* task is 13.2% ($p < 0.002$).

The performance improvements that were observed can be attributed to

TABLE 5.22. Mean performance measures (as percentages) for ALEPH on the three complete datasets for Heart Disease tasks. The bracketed numbers show sample standard deviation.

	Accuracy	**TPR**	**TNR**	**AUC**
cleve	68.2 (3.2)	58.4 (5.6)	76.0 (3.9)	67.2 (3.3)
hung	69.8 (2.0)	47.2 (4.9)	82.5 (3.9)	64.8 (2.0)
vamc	68.5 (2.8)	81.3 (4.2)	31.7 (6.9)	56.5 (3.2)

the similarity between the three tasks, as evidenced by the comparable powers and probabilities in the DFTs summarised in Table 5.23. For example, the `has_pred(cp)` has positive power on all three tasks and frequently occurs. The strength of this descriptor is larger for *cleve* and *hung* than *vamc*. Similarly, asymptotic chest pain, tested by the `has_argsym(cp,2,asympt)` descriptor, is frequently correlated with positive prediction on all three tasks, though once again the powers on *cleve* and *hung* are larger than for *vamc*.

The DFT for the *vamc* task also suggests why it may have been a more difficult task to learn. Unlike the *cleve* and *hung* DFTs, there is no descriptor which is both frequently occurring and has high power on the *vamc* task. This means that any rules that discriminate well between positive and negative examples do not have many descriptors (and therefore literals) in common whereas clauses containing the literal `cp(A,asympt)` will generally perform well on the *cleve* and *hung* tasks. This, in turn, means that the *vamc* concept is difficult to accurately describe using large disjuncts. The high mean true positive rate (TPR) of 81.3% and low mean true negative rate (TNR) of 31.7% on the full *vamc* task show that the clauses induced by ALEPH were over-general, incorrectly covering too many negative examples. However, the 3:1 ratio of positive to negative examples in the full *vamc* training sets meant these were preferable to more specific clauses.

On the $vamc_{10}$ tasks the proportion of positive and negative examples is equal. Under these circumstances, many of the theories induced by ALEPH included clauses with a single, ground head literal which was only true on a specific positive training example. These theories therefore covered few positive examples in the test sets leading to a poor TPR. The TNR of 57% implies that the examples that were covered in the test sets were roughly have positive and half negative.

DEFT was able to significantly improve the TPR on the $vamc_{10}$ tasks by modifying ALEPH's evaluation function so as to prefer clauses that generalised away from the training examples even if they only covered a single positive example and therefore scored as well as a single, ground literal. An examina-

TABLE 5.23. DFT summaries for the *cleve*, *hung* and *vamc* tasks. The first row in each group (age, cp, restecg, sex and trestbpm) gives the power β and probability π for the corresponding `has_pred` descriptor. The remaining rows in each group show the same statistics for the `has_argsym` descriptor that tests the value of the predicate's second argument.

		Task					
		cleve		*hung*		*vamc*	
has_pred	**has_argsym**$(\cdot, 2, \cdot)$	β	π	β	π	β	π
age	-	-0.16	.77	-0.16	.77	0.04	.82
cp	-	1.3	.83	1.6	.83	0.40	.85
	asympt	1.8	.72	2.2	.65	0.73	.63
	atypical	-2.4	.02	-4.1	.04	-2.0	.05
	non	-2.7	.05	-2.5	.09	-1.0	.14
	typical	-2.2	.03	-0.01	.05	0.76	.03
restecg	-	0.25	.78	0.12	.76	0.06	.80
	abnormal	4.0	.01	-0.13	.17	-0.54	.37
	hyper	1.0	.50	0.81	.01	0.41	.08
	normal	-1.1	.26	0.13	.58	0.60	.35
sex	-	0.33	.60	0.31	.61	-0.05	.65
	female	-1.3	.12	-2.2	.08	1.7	.01
	male	0.6	.48	0.50	.53	-0.06	.64
trestbps	-	-0.11	.82	-0.37	.83	0.12	.64
gteq	-	0.13	.51	-0.07	.53	0.43	.53
lteq	-	-0.41	.36	-0.46	.36	-0.33	.33

tion of the complexity of the theories produced by ALEPH and DEFT confirms this, as shown in Table 5.25. The number of clauses in the theories for $vamc_{10}$ induced by ALEPH and DEFT were identical on every fold with a mean of 2.8 clauses per theory taken over the folds. This was the case regardless of the support task used. The average number of literals per clause, however, is only 1.8 when ALEPH is used compared to 5.5 and 5.4 when DEFT is used with *cleve* and *hung* respectively. The same increase in literals was observed in the theories for the $cleve_{10}$ and $hung_{10}$ tasks when DEFT was used. On the $cleve_{10}$ tasks, the more specific clauses significantly improved the TNR without adversely affecting the TPR, thereby raising the AUC overall while on the $hung_{10}$ tasks this effect was also present but less pronounced.

These results show that the modifications made by DEFT evaluation function used by ALEPH result in a preference for more specific clauses. The observed improvements to AUC are due to this preference reducing the number of incorrectly covered negative examples (for $cleve_{10}$ and $hung_{10}$) or forcing ALEPH to generalise away from the training examples (for $vamc_{10}$). The fact

TABLE 5.24. The complexity of theories induced for the limited data tasks in the Heart Disease domain. Complexity is measured by the mean number of clauses per theory and mean number of literals per clause (including the head literal).

Target	Support	Clauses	Lits. per Clause
$cleve_{10}$	*none*	2.0 (0.0)	2.2 (0.2)
	hung	2.0 (0.0)	4.9 (0.1)
	vamc	2.0 (0.0)	5.2 (0.1)
$hung_{10}$	*none*	1.4 (0.2)	2.4 (0.3)
	cleve	1.4 (0.2)	4.0 (0.5)
	vamc	1.4 (0.2)	4.0 (0.5)
$vamc_{10}$	*none*	2.8 (0.2)	1.8 (0.2)
	cleve	2.8 (0.2)	5.5 (0.1)
	hung	2.8 (0.2)	5.4 (0.2)

that the number of positive examples covered was not adversely affected by these specialisations shows that the preference bias that DEFT transfers is capturing some information about the concepts that is relevant to all three tasks.

5.4.4. Theory Transfer. An alternative to bias transfer which Silver [2000] explored in the Heart Disease environment was the transfer of entire theories. This is referred to as *direct transfer* in Section 2.4.4 while Silver calls it *theory transfer*. This involves constructing a "standard model" for heart disease prediction using all the training examples from one hospital (the support) and then testing it on examples for a second, target hospital. If the examples for all three hospitals were drawn from the same underlying distribution, then the models learnt on each hospital would be expected to be interchangeable, performing equally well across the tasks. This hypothesis was tested using ALEPH using the same settings as the previous section and the performance results compared to those obtained when DEFT was used for transfer.

The method for this experiment involved applying ALEPH to the entire example set for one of the hospitals and evaluating the resulting theory on the complete example sets for the other two hospitals. Once again, training and testing ALEPH on datasets for a single hospital was excluded leaving a total of six transfer pairs: *cleve* to *hung* and *vamc*, *hung* to *cleve* and *vamc*, and *vamc* to *cleve* and *hung*. The order in which examples are selected by ALEPH for bottom clause construction and search during the FINDRULE procedure can have a large effect on the quality of the induced theory. For this reason, the training and testing for each pair of tasks was repeated 30 times allowing ALEPH to select different start examples in each case. The resulting test

TABLE 5.25. Accuracy, true positive/negative rates and AUC scores (as percentages) on the complete Heart Disease datasets using theory transfer.

Test	Train	Accuracy	TPR	TNR	AUC
cleve	*hung*	61.9 (2.2)	33.7 (4.6)	84.4 (4.7)	59.0 (2.2)
	vamc	64.9 (6.5)	87.6 (3.3)	46.8 (12.5)	67.2 (5.8)
hung	*cleve*	70.6 (4.0)	43.6 (16.2)	85.9 (3.6)	64.7 (6.6)
	vamc	66.6 (10.8)	88.9 (3.8)	54.0 (18.2)	71.4 (8.0)
vamc	*cleve*	57.9 (7.5)	57.1 (11.3)	60.1 (5.4)	58.6 (4.2)
	hung	54.4 (5.0)	48.8 (9.4)	70.8 (9.5)	59.8 (2.4)

accuracies, true positive and true negative rates and AUC were averaged over these 30 runs for each transfer pair to obtain mean performance values.

Table 5.25 shows performance results for the six pairs of theory transfer trials. The entries in the second column specify the training dataset that ALEPH used to induce a theory while the first column names the dataset that was used to test the resulting theory. The remaining four columns give the mean performance for each of the measures taken over the 30 trials along with the sample standard deviation.

These results show that ALEPH learns quite different theories for each of the three hospital datasets, as exhibited by the AUC and true positive/negative rate scores. The models induced from the *cleve* examples have a higher TPR and lower TNR when tested on the *hung* examples than when tested on the *vamc* dataset. The 6.1% difference in AUC on these test sets is significant at the $p < 0.001$ using a one-sided difference of means t-test. The AUC value of 58.6% for the *cleve* theories tested on the *vamc* examples is also significantly lower than the AUC of 67.2% reported in the first row of baseline results Table 5.22 ($p < 0.001$). The baseline AUC of 64.8% for the *hung* task was also significantly higher than the theory transfer results for *hung* theories to the *cleve* or *vamc* examples ($p < 0.001$). In contrast, the AUC score for the *vamc* theories are larger when tested on the *hung* examples than on the *cleve* examples ($p < 0.05$) and both are larger than the baseline AUC value of 56.5% for the *vamc* task ($p < 0.001$). These results confirm that the difference in example sets between the three hospitals results in different theories being induced for each. The main reason for this is the different class distributions across the three tasks. No information about the target task is used when performing theory transfer.

When DEFT is used to transfer bias between the hospital tasks it makes use of classification priors constructed from the support task and combines them with rule evaluations on the limited number of examples from the target

task. A comparison of the theory transfer results with the performances of ALEPH and DEFT on the limited data tasks (Table 5.21) shows that neither theory transfer nor DEFT is a superior method of transfer for all pairs of tasks. The AUC score for the *hung* to *cleve* theory transfer is lower by 11.7% than reported when using DEFT to perform transfer to the $cleve_{10}$ task ($p < 0.001$). Also, the AUC for the *vamc* to *cleve* theory transfer was lower by 3.9% when compared to the DEFT transfer for the same tasks ($p < 0.01$). In contrast, theory transfer had a 5.9% higher AUC score ($p < 0.02$) for *vamc* to *hung* transfer while the rest of the DEFT and theory transfer comparisons were statistically indistinguishable at the $p < 0.05$ level.

5.4.5. Discussion. In Experiment 14 of his doctoral thesis, Silver [2000] used the Heart Disease datasets to evaluate his neural network based, multi-task learning system ηMTL and the Task Rehearsal Method (TRM). Three types of inductive transfer were attempted on the Heart Disease tasks in his experiment. In the first type of attempt a previously trained network for *cleve*, *hung* or one of four artificially created tasks were used exclusively to provide a bias when learning from a *vamc* task with only five positive and five negative examples. Secondly, theory transfer was tried by applying the networks trained on *cleve* and *hung* tasks to the *vamc* task. The third type of transfer used all six tasks simultaneously as support tasks for *vamc*. Since DEFT does not perform transfer from multiple support tasks this last type of transfer will not be considered further here. Section 6.2.3 in the next chapter discusses how DEFT may be modified in the future to handle multiple support tasks.

In broad terms, conclusions drawn from Silver's transfer experiments using the single support tasks and theory transfer are consistent with results observed when DEFT was applied to the same pairs of support and target tasks. That is, the use of bias learnt from a support task can improve learning performance when training data is limited on a target task and in some cases inductive transfer can achieve better results than theory transfer. The replication of these two results using two different transfer methods strengthens the claim that biases exist that are useful to all three Heart Disease tasks even though the best target theories for each are different. However, there are some differences in methodology and results between the experiments reported here and Silver's that are worth remarking upon.

Although both DEFT and the TRM were able to achieve significant improvements in AUC and accuracy when applied to $vamc_{10}$ as a target task, the performance of the baseline learner and DEFT were much worse than reported by Silver. His baseline neural network was able to achieve an accuracy

of 66.65% compared to the 36.5% accuracy here. Transfer using the TRM from the *cleve* and *hung* task resulted in accuracies of 76.1% and 74.7% respectively. The increase in accuracy is similar using DEFT (*i.e.*, around 10 percentage points) but the final accuracy of the induced theories is around 50%. Furthermore, his attempt at theory transfer from the *cleve* task to the limited *vamc* task performed significantly worse than transfer using ηMTL. This was not the case when DEFT and ALEPH were used with the same pairs of tasks.

These discrepancies can be attributed to a couple of differences between the experimental setup and algorithms used here and in Silver's work. Firstly, the base learners used here and in Silver's work are very different and secondly, Silver repeatedly used the same train, test and validation set for the limited *vamc* task whereas the methodology here used ten independent training sets of size 10. In future work, a better comparison of DEFT and ηMTL could be made by trying ηMTL on different limited *vamc* tasks as well as on the other transfer pairs.

5.4.6. Conclusions. The three tasks that constitute the Heart Disease environment are intuitively similar as all three tasks require a learner to construct a predictive model of heart disease using the same patient features. The successful transfer of inductive bias between these tasks by DEFT and ηMTL [Silver, 2000] confirm this intuition.

One weakness of rule learning compared to whole theory learning such as neural networks is that the available examples strongly influence the overall structure of the induced set of rules. It is possible that transfer with ηMTL provided a bias for part of the model that was not supported by the available examples when using DEFT as rule learning can only construct rules for disjuncts that are supported by the examples. The gains made by DEFT on these tasks are once again due to the DFTs creating a bias that prefers specialised clauses over simpler ones.

It would also be instructive to apply Silver's ηMTL to the other pairs of tasks to see whether the similar or better gains than DEFT can be made on limited data versions of the *cleve* and *hung* datasets. The DFTs produced by DEFT suggest that these two tasks that have the more in common than they do with *vamc*.

5.5. Molecular Biology Environment

This section introduces a learning environment consisting of two well-known tasks from the ILP literature: mutagenesis [Srinivasan et al., 1994]

and carcinogenesis [Srinivasan et al., 1997]. The target theories for these tasks are used to predict cancer-related behaviour of nitroaromatic molecules from their two- and three-dimensional atomic structure and chemical properties. Mutagenic substances are known to cause changes to cellular DNA which often results in carcinogenic behaviour. For this reason it seems reasonable to assume that this pair of tasks would be amenable to inductive transfer.

The aim of the experiment reported in this section was to determine whether DEFT was able to improve learning on artificially limited training sets for the mutagenesis and carcinogenesis tasks. The results show this was not the case, and an examination of the DFTs constructed from the full datasets of each task reveal that the tasks are not a similar as first thought.

5.5.1. Materials. In the original datasets for the mutagenesis (*mut*) and carcinogenesis (*carc*) tasks, the background knowledge used by the problems to describe molecules contained relations for their atoms and bonds, chemical features, the presence of particular molecular groups such as benzene, and their three-dimensional structure. A total of 125 positive and 63 negative examples are available for the *mut* task and 182 positive and 148 negative examples for the *carc* task. The only relations immediately common to both tasks were those concerning the graphical structure of the molecules, that is, the atoms they contained and the bonds between them. Srinivasan et al. [2003] called these the M0 and C0 predicate groups for the *mut* and *carc* tasks respectively. To avoid complicating the experiment and its analysis theories and DFTs for the *mut* and *carc* tasks will be constructed using only these common predicates. Table 5.26 gives a short description of each of these predicates along with the mode and type information that was used to constrain the rule space.

Once again, ALEPH was used as the baseline learner and DEFT as the transfer method. Table 5.27 lists the settings used for both of these algorithms in the trials reported in this section. The settings for ALEPH were chosen in line with what have been reported as reasonable default settings for these problems in the past [Srinivasan et al., 2003]. As in the Heart domain, the *exs_sampled* and *cls_sampled* settings for DEFT were chosen so that 5000 clauses were sampled to construct the DFTs. As examples are prevalent on both tasks the *exs_sampled* setting was taken to be higher than the *cls_sampled* setting in line with the results from Experiment RP-4 in Section 5.2.6 above.

Preliminary DFT construction using these settings and the has_pred and has_arg descriptor templates resulted in hundreds of descriptors of the form `has_arg(`*p*`, 2, `*float*`)` where *float* is a floating point number and *p* was one of `lteq/2`, `gteq/2` or `=/2`. These all had very low power and frequency and

TABLE 5.26. Description of the predicates, modes and types used in the mutagenesis and carcinogenesis tasks. The `active/1` predicate is the target predicate for the tasks.

Predicate Mode	Description
`atm(+mol,-id #elem,#int,-chg)`	The molecule `mol` contains an `elem` atom identified by `id` of a particular type (`int`) with the given partial charge `chg`.
`bond(+mol,+id,±id,#int)`	The molecule `mol` has a bond between the atoms identified in the second and third arguments (`id`) with a bond type given by the fourth argument (`int`).
`lteq(+charge, #number)`	The "less than or equal to" relation for charges.
`gteq(+charge, #number)`	The "greater than or equal to" relation for charges.
`=(+charge, #number)`	The "equal to" relation for charges.
`active(+mol)`	The molecule is in the positive class for the task (*i.e.*, mutagenic or carcinogenic).

were therefore removed through the introduction of the `has_argnf` descriptor template. Like the `has_argsym` template used in the Heart Disease experiments, this template constructs exactly the same descriptors as the original `has_arg` template except for those that contain a floating point number in the third argument. The results of the experiment below were unchanged by the use of `has_argnf` in place of `has_arg` and meant the constructed DFTs were smaller and easier to analyse.

TABLE 5.27. Settings used by ALEPH and DEFT for experiments within the Molecular Biology domain

Learner	Setting	Value
ALEPH	*clauselength*	4
	i	2
	minacc	0.8
	nodes	5000
BUILDDFT	*admissibility*	true
	exs_sampled	100
	cls_sampled	50
	templates	{has_pred, has_argnf}
DBSCORE	M	$\sqrt{N}$

5.5.2. Method. To simulate limited datasets for the *mut* and *carc* tasks the original datasets, containing 188 and 330 examples respectively, were split into ten folds and sampled from using the same method as for the Heart Disease tasks. Letting $E = F_1 \cup \ldots \cup F_{10}$ represent one of the original datasets. Each

of the test folds F_i contained approximately 10% of the original examples in as close to E's class distribution as possible. The ten full-sized training and test pairs are then (E_i, F_i) where $E_i = E - F_i$. The training example folds $\hat{E}_i$ for the limited data version of the dataset were sampled without replacement from the corresponding full data training set E_i. The limited data tasks for this experiment were constructed to have roughly 20% of the original dataset with approximately the same class distribution as the full dataset. For the *mut* task, this meant there were 24 positive examples and 12 negative examples in each $\hat{E}_i$ and for the *carc* task there were 36 positive and 30 negative examples in each $\hat{E}_i$. The collection of limited data tasks for mutagenesis is denoted $mut_{20\%}$ and, similarly, the set of limited data carcinogenesis tasks is denoted $carc_{20\%}$.

To establish a baseline level of performance on the $mut_{20\%}$ and $carc_{20\%}$ tasks ALEPH was applied to each of the training folds using the settings shown in Table 5.27. The resulting theory in each case was evaluated on the corresponding test fold. DEFT was applied in the same manner to each of the limited data training folds for both tasks using the settings shown in the above table. When the training fold was for a $mut_{20\%}$ task a DFT built from the *carc* task was used and *vice versa*. The DFTs were built by applying DEFT once to the entire datasets for the *mut* and *carc* tasks. The parameters used for BUILDDFT procedure were those shown in Table 5.27. The evaluations of the theories returned by ALEPH and DEFT were averaged over the ten folds to obtain mean values and sample standard deviations for the AUC and true positive/negative rates.

5.5.3. Results and Analysis. The main result to report for the experiment described above is a negative one. No improvement in generalisation performance was observed when using DEFT on either of the two limited target tasks. As shown in Table 5.28 there is no significant difference between the AUC scores for the baseline learner and DEFT on either of the artificially limited $mut_{20\%}$ or $carc_{20\%}$ tasks. Differences were tested for using two-sided, paired t-tests at the 0.05 level of significance.

There are at least two reasons as to why no difference in performance was observed. One is that DEFT was unsuited to this domain and was unable to exploit any similarities between the two problems. This could be due to a poor choice of descriptor templates or bad example and clause sampling settings. Alternatively, there may be no similarity to exploit and the assumption that the *mut* and *carc* tasks are related is incorrect. The remainder of this analysis provides evidence for the latter possibility over the former.

TABLE 5.28. Mean performance measures and their sample standard deviation for DEFT and the baseline learner on the limited example mutagenesis and carcinogenesis tasks.

	AUC	**TPR**	**TNR**
Baseline $mut_{20\%}$	68.5 (2.8)	74.2 (4.3)	62.9 (6.9)
DEFT *carc* to $mut_{20\%}$	67.5 (3.4)	75.0 (3.4)	60.0 (8.3)
Baseline $carc_{20\%}$	53.3 (1.9)	41.7 (2.9)	65.0 (4.3)
DEFT *mut* to $carc_{20\%}$	50.2 (2.8)	39.6 (4.0)	60.7 (4.0)

Evidence against the DEFT being unsuitable for transfer in this domain was gathered by applying DEFT to the limited data tasks using the DFT for the same task, reusing the same training and test folds as the original experiment. That is, the $mut_{20\%}$ tasks were learnt from using DEFT with the *mut* DFT and the $carc_{20\%}$ tasks were tried with the *carc* DFT. The expectation was that no positive transfer would be observed in these trials if DEFT or its configuration were unsuitable for these tasks. This was not the case. When using the *mut* DFT as support for the $mut_{20\%}$ task the mean AUC was 74.9% (with a sample standard deviation of 3.3%). This represents an improvement of 6.4% points over the baseline significant at the $p < 0.01$ level. Most of this increase was due to a higher TPR of 82.2 (3.4) and a slightly higher TNR of 67.6 (7.7). The TPR increase is significant at the $p < 0.01$ level while the increase is TNR is significant at the $p < 0.05$ level. No significant difference in AUC, TPR or TNR was observed for the DEFT *carc* to $carc_{20\%}$ trials.

These results suggest two things. Firstly, the DFT for the *mut* task is capturing a useful bias that enables positive transfer to the limited data *mut* tasks. Secondly, no such useful bias is being captured in the *carc* DFT. The lack of useful bias for the *carc* task explains the lack of transfer in the *carc* to $mut_{20\%}$ trials. If the *carc* DFT cannot impart a useful bias to a task that is known to be similar (*i.e.*, the *carc* task itself) there is little chance of it doing so for a possibly different task. The positive transfer from the *mut* DFT to the limited data $mut_{20\%}$ tasks and the lack of transfer to $carc_{20\%}$ tasks suggest that the mutagenesis and carcinogenesis problems are not similar enough for successful transfer to take place.

Further evidence for the tasks being dissimilar was found in an examination of the DFTs constructed for each task. A total of 43 descriptors were created from the *mut* dataset, of which 5 were for the `has_pred` template and the other 38 for the `has_argnf` template. The DFT from the *carc* dataset also contained 5 `has_pred` descriptors as well as 60 `has_argnf` descriptors. The `has_pred` descriptors were common to the DFTs for both tasks and had almost identical

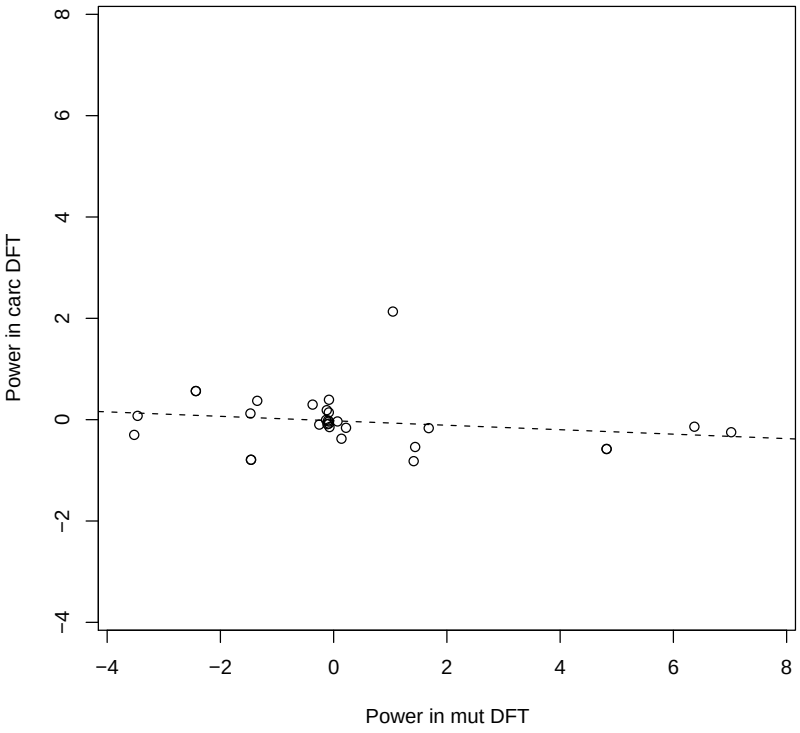


FIGURE 5.21. The powers of descriptors in the *mut* and *carc* DFTs. The horizontal and vertical position of each point corresponds to the power for a descriptor in the *mut* and *carc* DFT respectively. The dashed line is the least-squares line of best fit.

probabilities and power in each, except for the `has_pred('='/2)` descriptor which had a power of 1.86 in the *mut* DFT and a power of -0.02 in the *carc* DFT.

There was a much greater difference in the `has_argnf` descriptors in the for DFTs. A total of 68 unique descriptors were created from that template type across the two DFTs of which only 30 of those were common to both. The powers for descriptors in the *mut* DFT ranged from -3.52 for `has_argnf(atm,4,35)` which had a probability of 0.004, to 12.93 for `has_argnf(atm,4,28)` which had a probability of 0.038. The range of powers in the *carc* DFT went from -5.90 for `has_argnf(atm,4,60)` (probability 0.001) to 7.35 for `has_argnf(atm,4,41)` (probability 0.004). The high powers for the descriptors at these extremes are most likely due to their rarity.

Figure 5.21 shows a plot of the powers of the 30 `has_argnf` descriptors that are common to the *mut* and *carc* DFTs. The horizontal position represents a descriptor's power in the *mut* DFT while its vertical position gives its

power in the *carc* DFT. Two observations can be made from this plot. First, of those descriptors common to both DFTs, the range of powers in the *carc* DFT is much smaller than the range in the *mut* DFT. The highest power is 2.13 for the `has_argnf(atm,4,29)` descriptor while the lowest power is -0.81 for the `has_argnf(atm,4,8)` descriptor. The lack of highly discriminating descriptors for the *carc* task is probably due to its difficulty to learn using only the C0 background predicates. The AUC recorded for ALEPH applied to the full *carc* dataset was 51.3% with a standard deviation of 3.3% using ten-fold cross validation. In contrast, the *mut* DFT has powers for the common descriptors ranging from -3.52 for the `has_argnf(atm,4,35)` descriptor to 7.02 for the `has_argnf(atm,4,51)` descriptor. The mean AUC on the full *mut* dataset was 73.9% (3.1%), significantly higher than the AUC for the full *carc* task. The second observation that can be made from the figure is that there is no correlation ($r = -0.040$) between the powers in the two DFTs as shown by the dashed line in the figure. The relatively few common descriptors and the lack of correlation between their powers suggests that the tasks are not as similar as first thought.

5.5.4. Conclusions. The conclusion from the experiments and analysis in this section is that the mutagenesis and carcinogenesis tasks, while both superficially about cancer-causing properties of molecules, are not similar in a manner that can be exploited by the DEFT approach to inductive transfer. If any similarity exists it is of a subtler form than can be detected with the descriptors used here. This finding is consistent with observations made by Srinivasan [2002] who tried some preliminary transfer experiments with these tasks assuming they were similar and found "that the data were about such different sets of chemicals that this was not really the case". The selection of a support task is a form of expert provided bias and in the case of the mutagenesis and carcinogenesis tasks neither provided appropriate biases tasks for the other. The examination of the DFTs for the tasks provided some insight into why these tasks were not appropriate for description-based transfer.

5.6. Summary and Conclusions

The experiments reported in this chapter provide evidence that the approach to inductive transfer implemented in DEFT is an effective way to improve the performance of clausal theories induced from small training sets. The four environments used in the experiments showed that DEFT can be used under a variety of different circumstances with varying degrees of success. Analysis of these successes and failures led to several discoveries regarding the

practical application of DEFT, how it differs to the theory, its strengths and limitations, the configuration of its settings and the selection of support tasks. The purpose of this final section is to summarise these discoveries and use them to answer the three questions that were asked in the opening section of this chapter: Does DEFT work? Is it practical? And how does it compare to other inductive transfer techniques?

5.6.1. Does it work? Of the four sets of tasks used to evaluate DEFT two sets - the Heart Disease and Molecular Biology environments - were derived from naturally occurring data while the other two - the Reading Preferences and Chess environments - were constructed artificially. In three of these four task environments, DEFT was able to successfully transfer bias between at least one pair of tasks, leading to improved generalisation performance from limited data compared to learning on the same task without inductive transfer.

The largest improvements were observed for transfer in the two artificial environments. In the Reading Preferences environment of Section 5.2 DEFT was shown to induce near-perfect theories from a third as much data as would normally be required without inductive transfer. Reductions in generalisation performance were also observed in this environment when transfer took place between the tasks that were intuitively dissimilar. Improvements over the base-level learner were also observed on the Chess tasks of Section 5.3 with DEFT adding up to 10 percentage points to the baseline accuracy on datasets containing less than a tenth of the examples required to induce a perfect theory for King and Knight movement. The success of DEFT in these environments was not too surprising: the Chess tasks were known to be amenable to transfer due to earlier research by Khan et al. [1998] and Datta and Kibler [1993] while the Reading Preference tasks were designed to exhibit similarities that were easily captured by the predicate and argument descriptor templates. Furthermore, the tasks were all "ideal" in these sense that they were noise-free and generated from model theories.

Unlike the Reading Preferences and Chess tasks, there was no known ideal theory for the learning tasks in the Heart Disease and Molecular Biology environments of Sections 5.4 and 5.5. The larger search spaces, lack of model theories, and the presence of noise hampered both the learning of theories and the construction of DFTs, as evidenced by the low generalisation performance and low descriptor frequencies. Regardless of these problems, improvements over single-task learning were observed when DEFT was to transfer bias to limited data tasks from the Heart Disease environment, though these were not as significant as for the artificial environments. These results were consistent

with the successful transfer reported by Silver [2000] using his multi-task learning system for neural networks. No improvements were observed when using DEFT to transfer bias between the mutagenesis and carcinogenesis tasks from the Molecular Biology environment. An analysis of the descriptor powers and DFTs for these two tasks revealed that they were less similar than initially thought and an argument was made that these tasks are not related in the sense required for DEFT to successfully transfer bias between them. That is, there is little in common between descriptions for the rules that perform well on carcinogenesis and those that perform well on mutagenesis.

When improvements in generalisation performance due to DEFT were observed they were all for the same reason. The classification priors used by DEFT when evaluating rules created a preferential bias for rules with additional predicates that were not justified by the limited number of examples available in the target tasks. This extra-evidential specialisation led to reductions in false positive and, surprisingly, false negative rates of theories induced by DEFT. This second effect was due to modified evaluation functions scoring rules that generalised away from ground facts higher than simply asserting the ground facts as part of the theory - a common occurrence when training data is limited. Experiments on the Chess tasks showed that this preference for specialised rules can lead to the superfluous addition of literals to rules. This suggests that some kind of post-pruning might be useful to incorporate within DEFT.

Experiment RP-8 in the Reading Preferences environment confirmed that the extra-evidential bias provided by DEFT led to a reduction in CPM error which meant that, on average, the true classification probabilities of rules in that domain were being estimated better when using classification priors. The actual reduction in CPM error was not, however, as large as that predicted by the theory in Chapter 3 which suggests that the theory needs to be amended to take into account the independence assumption made in order to decompose descriptions and efficiently implement description-based transfer.

In summary, the experiments and analysis in this chapter show that DEFT does work. That is, with an appropriate choice of support task, the application of DEFT to a target task with limited training data results in theories with higher generalisation performance than standard, single-task learning.

5.6.2. Is it practical? The practicality of DEFT as a useful technique for bias transfer depends on whether or not the cost of using it outweighs its benefits outlined above. Compared to single-task learning using ALEPH, using DEFT to perform inductive transfer requires more of the machine it is run on.

It also requires more decision-making on behalf of the practitioner using it in terms of the selection of support tasks and new parameters introduced by DEFT. Both of these aspects are considered below.

5.6.2.1. *Parameter and Support Task Selection.* The parameters which control the behaviour of DEFT fall into two categories: those with an influence over DFT construction (the sampling parameters, admissibility condition, and choice of descriptor templates), and those which affect how the base-level learner's evaluation bias is modified (the number of virtual examples M and the choice of DFT to use). While all of these parameters are important, the Reading Preferences experiments of Section 5.2 showed that reasonable defaults can be used for most of them. In particular, good default settings were found for the descriptor templates, admissibility condition and M parameter. Experiment RP-7 and the analysis in Section 3.6.5 both suggest that choosing M to be the square root of the number of training examples is a reasonable default. Experiment RP-5 showed that setting the admissibility condition to *true* during DFT construction does not reduce the magnitude of any positive transfer effects and, in the cases where the support and target tasks do not share any common positive examples, can improve generalisation performance. The ability to specify sets of descriptors through the templates mechanism makes the configuration of this aspect of DEFT quite easy. The user does not need to know in advance which descriptors will be required for a given support task as the templates construct them as required. Experiment RP-6 showed that the `has_pred` and `has_arg` templates together contribute most to the positive transfer effects in the Reading Preferences environment and so make for good default choices. This is confirmed by the positive transfer results using these templates on the other environments.

Choosing sampling parameters and selecting the support task for a given problem requires more involvement on the part of an expert. As shown in Experiment RP-4, the *exs_sampled* parameter controls a trade-off between the running time of the BUILDDFT algorithm and the positive influence of the resulting DFT. Large values of this parameter lead to longer running times but better descriptor frequency estimates. Finding an appropriate value for this and the *cls_sampled* parameter will depend on the support task but some guidance, in the form of a lower bound, is given by the analysis in Section 4.3.4. As the experiments did not show any degradation in transfer when large numbers of rules were sampled, a practitioner would be advised to err on the side of sampling a large number of rules if the computational resources are available.

The selection of an appropriate support task was shown to be the most important decision in determining whether any positive benefit will be had

through using DEFT. As discussed in the previous chapter, leaving this decision to a domain expert was part of the original design criteria for DEFT. Given some knowledge of what the descriptor templates will be extracting as rule features, the choice of a support task allows the expert to express a bias for the target task at a high level. Arguably, the instances of successful transfer in the Reading Preferences, Chess and Heart Disease environments were all between tasks that an expert would deem similar given the `has_pred` and `has_arg` templates. The use of the Chess and Heart Disease tasks for the study of inductive transfer by other researchers suggests that they are thought to be similar in some regard. The cases in the Reading Preferences environment where the support and target tasks were intuitively dissimilar led to DEFT reporting worse performance than the base-level learner alone. The only anomaly in this correspondence between expert opinion and results was for the transfer between the mutagenesis and carcinogenesis tasks. In this case, the belief that the mutagenesis and carcinogenesis tasks were related was based on the similarity of the M0 and C0 sets of predicates and the opinion of the author who is not a chemist or molecular biologist. This can be seen as an argument for expert guidance in the selection of appropriate support tasks.

5.6.2.2. *Computational Complexity.* As analysed in Section 4.2.4 of the previous chapter, the computational cost of using DEFT with the `has_pred` and `has_arg` descriptor templates should only add some constant overhead to the standard rule evaluation carried out by ALEPH. This was found to be the case in experiment RP-2 for the Reading Preferences environment. It was also the case for the other environments, as shown in Table 5.29. This shows for both ALEPH and DEFT the total time taken to induce a theory from typical target tasks from each of the four environments as well as the total number of rules evaluated during the search. The final row of the table shows the ratio of the times and counts for DEFT and ALEPH. Typically, DEFT must evaluate up to 2.6 times as many rules and spend roughly ten times as long evaluating each rule. The increase in the number of rules evaluated is due to the weaker upper bound for DEFT evaluation when pruning the search performed by ALEPH. The roughly constant increase in evaluation time per rule is due to the extra work that must be done to compute rule descriptions and prior probabilities. Together, these increases can mean DEFT takes between 12 and 39 times longer to find a theory than ALEPH. While this is quite large, it is not prohibitive for the environments considered in this chapter.

The other additional computation overhead is the construction of DFTs from support tasks. Table 5.30 compares the time and number of rules evaluated when constructing a DFT and performing a standard induction with

TABLE 5.29. Average amount of time taken and number of rules evaluated during a rule search using ALEPH and DEFT on each of the four domains. The time values are given in seconds.

	Reading		**Chess**		**Heart**		**Molecular**	
	Time	**Rules**	**Time**	**Rules**	**Time**	**Rules**	**Time**	**Rules**
DEFT	0.75	34	3.0	1500	1.2	510	120	66000
ALEPH	0.052	20	0.25	1200	0.031	140	5.3	25000
DEFT : ALEPH	14:1	1.7:1	12:1	1.2:1	39:1	3.6:1	23:1	2.6:1

ALEPH on typical support tasks from each of the four environments examined in this chapter. In all cases, the time taken to build a DFT was always larger than the time taken to induce a theory from the same example set even when fewer rules were evaluated, as in the Chess and Molecular Biology environments.

Strikingly, the average time taken per rule evaluation for the Chess environment was 260 times larger than the rule evaluation time during theory construction. The main reason for this large difference is the complexity of the legal rules for the Chess tasks. The maximum variable depth and maximum clause length for legal rules were, respectively, set to 3 and 6. While ALEPH searches the space of these clauses from least specific to most specific, DEFT samples rules uniformly from this space. This means the rule evaluations DEFT must perform involve longer and more complex clauses than ALEPH, most likely around the computationally expensive "phase transition" area [Giordana and Saitta, 2000]. In order to keep DFT construction times in check, future implementations of DEFT may resort to non-uniform sampling techniques that focus on the areas in which the base-level learner will actually be evaluating rules.

Even with these inefficiencies the time for DFT construction was, once again, not prohibitive for the domains considered. Furthermore, DFT construction time is a once-off cost and can be amortized over several repeated uses of DEFT with the same support task.

Overall, the extra overhead of DFT construction and the calculation of classification priors during rule evaluation does not make the use of DEFT impractical with regards to computational resources. Also, the broad applicability of default values for half of DEFT's parameters leaves only the selection of an appropriate support task and sampling parameters to the user. Compared to the technical knowledge required for other methods of bias selection, it can be argued that these decisions are easier to make for a domain expert unfamiliar with machine learning.

TABLE 5.30. A comparison of DFT construction and theory construction complexity on a support task from each of the four experimental environments. Values in the "Time" columns are in seconds and values in the "Rules" columns give the number of rules evaluated during the runs.

	Reading		**Chess**		**Heart**		**Molecular**	
	Time	**Rules**	**Time**	**Rules**	**Time**	**Rules**	**Time**	**Rules**
DFT	1.4	400	26	400	63	5000	179	5000
Theory	0.008	23	0.37	1440	1.4	3844	115	81850
DFT:Theory	175:1	17:1	73:1	0.28:1	45:1	1.3:1	1.6:1	0.06:1

5.6.3. How does DEFT compare to other approaches? Other approaches to using extra examples from support tasks were considered as alternatives to DEFT in this chapter. One of the simplest of these is to use the examples for the support task as though they were extra examples for the target task. The analysis in Experiment RP-5 showed that that this approach would not lead to improved performance since some pairs of tasks (*e.g.*, tasks B and C from the Reading Preferences environments and the King and Knight movement tasks) had no positive examples in common even though their concepts were similar. Combining examples in this case would be equivalent to adding noisy data to the training set. DEFT, however, was able to exploit the concept similarity between tasks such as these and improve generalisation performance.

Another simple approach to transfer, is called "theory reuse" or "direct transfer". In the case of the Reading Preference tasks which share no positive examples, simply inducing a theory on a support task and using it for classification on the target performs very poorly. Theory transfer was also explored in the Heart Disease environment. For these tasks it was shown that this approach has some merit. However, there were pairs of tasks for which transfer using DEFT performed better than theory transfer. The Heart Disease environment was also used to compare DEFT with the results of experiments with transfer using sub-symbolic methods. The tasks that DEFT successfully transfered bias between were the same as those Silver [2000] reported successful transfer for using his ηMTL system.

One advantage of DEFT over non-symbolic approaches to transfer is that the DFTs produced from a support task are inspectable by the user. The powers and frequencies of these descriptors can give some insight into the task and the rules that will perform well on it. Improvements could be made to improve this method of analysis by reducing the number of superfluous descriptors that appear in DFTs. This could be achieved by pruning those with low power

and/or frequency. As exhibited in the Heart Disease and Molecular Biology experiments, special care is required when predicates can contain numerical arguments. Future work could also investigate descriptors that know more about the semantics of the predicates such as those representing greater-than and less-than and combine counts for their numeric arguments into ranges.

In the Chess environment, DEFT was compared to an existing inductive transfer technique, Repeat Learning. The combination of DEFT's evaluation-based transfer and representation-based transfer of RL was shown to be a promising one as the two approaches complemented each other. DEFT was able to compensate for a lack of negative training data on very small example sets, forcing the base-level learner prefer specialised rules. On larger datasets, the extra predicates added to the language by RL meant rules could express internal disjuncts, generalising beyond the available positive examples. This hybrid transfer system RL+DEFT was shown to outperform either method alone on the Chess movement tasks.

The experiments in this chapter have shown that DEFT has advantages over other techniques for exploiting information present in support tasks. In particular, DEFT is able to improve generalisation performance where simple techniques like example and theory transfer fail. DEFT was also successfully combined with the Repeat Learning approach to inductive transfer.

5.6.4. Conclusions. DEFT was intended to be a tool that allows a domain expert to modify the evaluation bias of the learning algorithm ALEPH in order to improve generalisation performance when faced with a target task with limited training examples. This is achieved by allowing the expert to choose a support task that has high quality rules that are believed to be similar to high quality rules on the target task.

The experiments of this chapter have shown that DEFT is a suitable and practical tool for this purpose. Its relationship with other transfer techniques was also examined and the transfer of evaluation bias by DEFT was found to complement the transfer of language bias. The empirical work also revealed several short-comings of the system and theory. Ways in which these problems may be overcome are the topic of the next chapter.

Hofstadter's Law: It always takes longer than you expect, even when you take into account Hofstadter's Law

- Hofstadter [1999, pg. 152]

CHAPTER 6

Future Work and Conclusions

As the world around us becomes more complex we require inductive tools to help us make sense of it by finding, summarising and communicating patterns we would otherwise not detect. Like any tool, the quality of the results we obtain from our learning algorithms depends on how appropriate they are for the task at hand and how deftly they are wielded. If the cost, in terms of expert time and effort, of adjusting an inductive algorithm for each task outweighs the benefits of its results then it is no longer a useful tool.

This dissertation has presented theoretical and empirical evidence for the efficacy of a novel approach for configuring concept learning tools. This approach, DEFT, enables domain experts to tune the evaluation bias of a rule learning system through the selection of support tasks they believe to be similar to the task to be solved. This final chapter reviews the preceding work, points out some of its weaknesses and suggests some directions for future research.

6.1. Review and Contributions

This thesis focused on the types of problems for which an interpretable model was required to be induced from a small number of training examples and where a domain expert was able to provide a bias in the form of related learning tasks. Background to this problem was provided in Chapter 2 along with a survey of systems applicable to this type of problem. Each of the existing systems were analysed according to the type of bias it modified. This revealed that existing systems capable of using inductive transfer in a rule learning setting have only been able to transfer language or search biases.

To fill this gap a theoretical basis for the similarity-based transfer of evaluation bias for rule learning was proposed in Chapter 3. The main contributions of this theory are a novel definition of task similarity based on rule similarity and a theorem that provides sufficient conditions for successful transfer. A specific type of rule similarity, called description similarity, was also introduced which allowed the prior classification probabilities to be decomposed and computed efficiently. Using this decomposition, an implementation of description-based transfer called DEFT was presented in Chapter 4. The two main algorithms in this implementation allow for a sequential approach

to transfer. During the consolidation phase this involves the construction of a data structure called a Descriptor Frequency Table (DFT) which, during the transfer phase, can be used to efficiently create classification priors when evaluating rules. Central to these algorithms is the computation of rule descriptions. An efficient method for the computation of rule descriptions was presented which made use of a sparse description representation and descriptor templates. The modifications required to incorporate these extra algorithms into a rule learning system such as Aleph were minimal and, as their analysis showed, reasonably efficient.

The implementation of Deft was tested on four different environments and the results reported in Chapter 5. These results suggest that Deft is an intuitive and effective way to transfer bias between tasks. The results were also shown to be consistent with the theory and analysis of the preceding chapters as well as results obtained by other researchers who had used the same environments with other transfer systems. To the author's knowledge, this is the first comparative study of inductive transfer techniques for symbolic learning. Deft was successfully combined with Repeat Learning, an existing inductive transfer technique, and shown to outperform either approach when used alone. The experiments also revealed that bias imparted through the use of priors for classification probabilities meant rules could be specialised further than the data alone would normally allow. This extra-evidential specialisation was shown to be what led to the observed improvements in generalisation performance.

6.2. Limitations and Future Work

The main intention of the research presented here was to propose and examine a new form of inductive transfer. Many decisions regarding its direction were made to keep it focused and manageable. This included restricting its application to rule learning algorithms. Many avenues of theoretical and experimental investigation were therefore left unexplored and several improvements to Deft were left unimplemented. The purpose of this section is to briefly highlight some of the limitations of this work and examine some of the ways in which it could be expanded upon in the future.

6.2.1. Implementation Improvements. Apart from the description-based decomposition of classification priors and the use of descriptor templates to construct sparse rule descriptions there was little attempt made to optimise the implementation of the BuildDFT and CalcPrior algorithms within Deft. As observed in Section 5.6.2 of the previous chapter, the weak

upper bound implemented by DBBOUND means ALEPH's rule search explores more candidate rules than necessary. This bound could be easily improved and would make the search slightly more efficient. Other efficiency gains could be made by caching rule descriptions as rules are searched and incrementally updating them as rules are refined.

As well as being made more efficient, the classification priors created by DEFT could be made more robust to chance occurrences during rule sampling. Descriptor Frequency Tables constructed in the Heart Disease and Molecular Biology environments exhibited descriptors with very low frequency and high power. Whether due to noise or distributional properties of the domain these descriptors have the potential to strongly and incorrectly bias a learner on a target task. Future revisions of DEFT may use some kind of pruning strategy to eliminate extremely low probability descriptors to avoid this problem. Pruning may also be used to eliminate descriptors with zero power from the DFT. Doing so would have no impact on bias transfer and may lead to improved efficiency in environments with large numbers of descriptors.

If DEFT is truly meant to be a tool for people with little experience with machine learning then further work also needs to be done in making it easier to use. While good defaults exist for the descriptor templates and M parameters, it is not always clear what value should be chosen for the sampling parameters. One approach to this problem is to allow DEFT to sample and evaluate rules until the classification probability priors for each descriptor stabilise according to some kind of statistical test. Further experimental work would be required to determine the impact of this and the previously proposed improvements.

6.2.2. Beyond Predictive Rule Learning. One of the largest restrictions placed on the scope of this research was the decision to limit the investigation to inductive transfer for rule learning algorithms that use a covering strategy to build theories. However, the core idea of description-based transfer - improving classification estimates using priors based on syntactic features of models - is potentially applicable to a much wider class of symbolic concept learning algorithms.

The simplest extension to this work would be to see whether DEFT has any use in conjunction with associative, rather than predictive, rule learning. Classification tables are used extensively in systems such as TERTIUS [Flach and Lachiche, 2001] for relational association rule mining. DEFT could be used there to improve estimates for those tables based on other rule mining tasks. This would make for an interesting alternative to the algorithms described by Zhang and Zhang [2002] for association rule mining from small databases.

Classification tables also play an important role in an alternative to ILP called analogical prediction [Muggleton and Bain, 1999] so the DEFT approach to transfer may also be applicable to systems built upon that mode of inference.

Another straightforward extension to DEFT would be to multi-class rule learning [Fürnkranz, 1999]. Learning rules for k-class prediction would mean using contingency tables of size $k \times k$. The equation for computing the posterior classification probabilities remains the same, namely $\mathbf{p}^* = \mathbf{p} + M\mathbf{q}$. All of the algorithms and data structures used by DEFT can be trivially extended to handle these larger matrices.

As well as rule sets, decision trees are another popular symbolic representation for classifiers. Algorithms for learning decision trees such as C4.5 [Quinlan, 1993] use a gain heuristic to search the candidate space of decision trees. Bensusan [1999] gives a number of decision tree features that could be used as descriptors for a DEFT-like approach to augmenting decision-split evaluation in a manner similar to that of Caruana's multitask approach to decision tree learning [Caruana, 1997].

6.2.3. Multiple Support Tasks. Another limitation of description-based transfer as it currently stands is its restriction to single support tasks. This restriction was made for the sake of simplicity and it would not be difficult to extend DEFT so as to take into account more than a single support task. In essence, if multiple support tasks are available virtual contingency tables can be constructed for each and combined to create a classification prior. The simplest way to combine these priors is through a linear combination. If λ_i are k positive numbers such that $\sum_{i=1}^{k} \lambda_i = 1$ then a posterior CPM can be defined for k support tasks by

$$\mathbf{p}^* = \mathbf{p} + M \sum_{i=1}^{k} \lambda_i \mathbf{q}_i$$

where $\mathbf{q}_i$ is the prior CPM for the i^{th} task. The terms λ_i determine how much of an influence the corresponding support task will have on the learner's evaluation bias.

This construction and the use of the weighting terms λ_i is analogous to the transfer rates R_i of Silver's ηMTL system Silver [2000] or the task weights in the TC system of Thrun and O'Sullivan [1996]. Both systems can automatically determine good transfer weights by dynamically adjusting them during learning. Determining good λ_i values for a given set of tasks using DEFT would require more research into practically computable estimates of task similarity. Some directions for this research are discussed further below.

6.2.4. Similarity: Theory and Practice. The theory of task similarity proposed in Chapter 3 of this dissertation was motivated by a desire to understand how the use of priors might improve estimates of classification probabilities. As the construction and use of these priors were stated quite generally - using an abstract similarity relation over candidate models - the main transfer theorem was also framed in general terms. While this had the advantage of focusing the theory on the key mechanisms of evaluation transfer it also meant important details of actual learning systems were ignored. In particular, the transfer theorem provides a bound on CPM error that is independent of any evaluation function a real learning system may be using. This makes it difficult to use the theory to draw any strong conclusions regarding the empirical behaviour of DEFT reported in Chapter 5. Future research may focus on specific performance criteria such as generalisation accuracy and examining how the use of classification priors affects the performance of learners that specifically aim to maximise this criteria.

The other disadvantage of the very general definition of task similarity proposed in this work is that it is not practically computable since it requires the evaluation of every candidate model against complete training sets for the support and target tasks. This problem may be able to be overcome by careful examination of how the CPM error bound is weakened when estimates of classification probabilities are made using samples of the candidate and example spaces. Ultimately, it would be desirable to have a theory that uses estimates of task similarity derived from values found in descriptor frequency tables rather than the current, unrealistic definition of similarity.

A quicker way to a practical measure of task similarity might be to start with the approach used in the previous chapter. There, comparisons of the powers and frequencies of descriptors derived from a task's DFT were used to make an intuitive appraisal of the differences between two tasks. It may be possible to quantify this appraisal by defining some weighted sum of the differences between descriptor powers appearing in the DFTs for two tasks. Using a large number of tasks, an experiment could be run to see whether a correlation exists between the change in performance when transferring bias between two tasks and their estimated similarity. If found to be reliable, a practical measure of task similarity could be used to set the task weights λ_i (as described above) in an extension of DEFT to multiple support tasks.

Although descriptions in this thesis were defined purely as syntactic features, future work may investigate the feasibility of semantic descriptions of rules. That is, descriptions and similarity based upon extensional definition of a rule with respect to some background theory. This would require careful

thought since determining whether or not two rules have the same extension can be a computationally demanding task in general.

Finally, it would be instructive to find stronger ties between the theory of similarity presented here and those put forward by Ben-David and Schuller [2003] and more recently Juba [2006] as discussed in Chapter 3. Links with Ben-David and Schuller's work are especially tantalising since they also define task similarity using partitions of the hypothesis space.

6.3. Conclusions

A good tool kit not only contains tools for the job at hand but also equipment for the correct adjustment and maintenance of those tools. This thesis has presented description-based inductive transfer as a new method for sharpening rule learning tools. Through the selection of tasks similar to the primary one, domain experts are able to express preferences for models to a learning system at a high level, away from many of the system's technical details. This allows domain knowledge that may otherwise be difficult to express to be incorporated into a learning system enabling it to make better inferences from fewer training instances.

While more work needs to be done, the research here has demonstrated the potential of using priors for classification probabilities as a practical and general way to incorporate preference biases into symbolic forms of learning such as rule learning. For this reason, it is hoped the work presented here will find a place in the inductive tool kits of the future.

Any problem in computer science can be solved with another layer of indirection.

- David Wheeler

Bibliography

W. R. Ashby. *An Introduction to Cybernetics*. Chapman and Hall, London, UK, second edition, 1956. URL `http://pespmc1.vub.ac.be/books/IntroCyb.pdf`.

C. Babbage. *On the Economy of Machinery and Manufactures*. Project Gutenburg, 1832. URL `http://www.gutenberg.org/etext/4238`.

X. Bao, J. L. Herlocker, and T. G. Dietterich. Fewer clicks and less frustration: Reducing the cost of reaching the right folder. In *Proceedings of the International Conference on Intelligent User Interfaces*, Sydney, Australia, January 2006. URL `http://doi.acm.org/10.1145/1111449.1111490`.

J. Baxter. A model of inductive bias learning. *Journal of Artificial Intelligence Research*, 12:149–198, 2000. URL `citeseer.nj.nec.com/article/baxter00model.html`.

J. Baxter. Learning internal representations. In *Proceedings of the 8th International Workshop on Computational Learning Theory*. Morgan Kaufmann Publishers, 1995.

S. Ben-David and R. Schuller. Exploiting task relatedness for multiple task learning. In B. Scholkopf and M. K. Warmuth, editors, *Proceedings of the 16th Annual Conference on Computational Learning Theory*, volume 2777 of *Lecture Notes in Computer Science*. Springer, 2003.

H. N. Bensusan. *Automatic bias learning: an inquiry into the inductive basis of induction*. PhD thesis, University of Sussex, February 1999.

Y. M. M. Bishop, S. E. Fienberg, and P. W. Holland. *Discrete Multivariate Analysis: Theory and Practice*. The MIT Press, Cambridge, MA, 1975.

M. Bongard. *Pattern Recognition*. Spartan Books, New York, 1970.

I. Bratko. *Prolog: Programming for Artificial Intelligence*. Addison-Wesley Publishers, Wokingham, England, 2nd edition, 1990.

I. Bratko. Refining complete hypotheses in ILP. In Džeroski and Flach [1999], pages 44–55.

B. Carlin and T. Louis. *Bayes and Empirical Bayes Methods for Data Analysis*. CRC Press, 2000.

R. Caruana. Multitask learning. *Machine Learning*, 28(1):41–75, 1997. URL `citeseer.nj.nec.com/caruana97multitask.html`.

R. Caruana, S. D. L, J. Baxter, T. M. Mitchell, L. Y. Pratt, and S. Thrun, editors. *Learning to Learn: Knowledge consolidation and transfer in inductive systems*, 1995. Workshop, held at NIPS-95.

B. Cestnik. Estimating probabilities: A crucial task in machine learning. In *Proceedings of the Ninth European Conference on Artificial Intelligence*, pages 147–149, London, 1990. Pitman.

P. Clark and T. Niblett. The cn2 induction algorithm. *Machine Learning*, 3: 261–283, 1989.

W. Cohen. Rapid prototyping of ILP systems using explicit bias. In F. Bergadano, L. De Raedt, S. Matwin, and S. Muggleton, editors, *Proceedings of the IJCAI-93 Workshop on Inductive Logic Programming*, pages 24–35. Morgan Kaufmann, 1993. URL `citeseer.nj.nec.com/cohen93rapid.html`.

W. W. Cohen. Learning trees and rules with set-valued features. In *AAAI/IAAI*, volume 1, pages 709–716, 1996.

W. W. Cohen. A web-based information system that reasons with structured collections of text. In K. P. Sycara and M. Wooldridge, editors, *Proceedings of the 2nd International Conference on Autonomous Agents (Agents'98)*, pages 400–407, New York, 1998. ACM Press. URL `citeseer.nj.nec.com/32521.html`.

W. W. Cohen. Grammatically biased learning: learning logic programs using an explicit antecedent description language. *Artificial Intelligence*, 68:303–366, 1994.

W. W. Cohen. Fast effective rule induction. In *Proceedings of the 12th International Conference on Machine Learning*, pages 115–123, 1995a.

W. W. Cohen. PAC-learning non-recursive prolog clauses. *Artificial Intelligence*, 79(1):1–38, 1995b.

W. W. Cohen and D. Kudenko. Transferring and retraining learned information filters. In *AAAI/IAAI*, pages 583–590, 1997. URL `citeseer.nj.nec.com/cohen97transferring.html`.

V. S. Costa, L. Damas, R. Reis, and R. Azevedo. *Yap User's Manual*. LIACC, Universidade do Porto, 4.5.7 edition, 2005.

E. Crawford, J. Kay, and E. McCreath. An intelligent interface for sorting electronic mail. In *IUI '02: Proceedings of the 7th international conference on Intelligent user interfaces*, pages 182–183, New York, NY, USA, 2002. ACM Press. URL `http://doi.acm.org/10.1145/502716.502747`.

J. Cussens. Using prior probabilities and density estimation for relational classification. In Page [1998], pages 106–115. URL `citeseer.nj.nec.com/cussens98using.html`.

P. Datta and D. F. Kibler. Concept sharing: A means to improve multi-concept learning. In *Proceedings of the 10th International Conference on Machine Learning*, pages 89–96, 1993. URL `citeseer.nj.nec.com/datta93concept.html`.

L. De Raedt. Logical settings for concept learning. *Artificial Intelligence*, 95: 187–201, 1997.

L. De Raedt and M. Bruynhooghe. Interactive concept-learning and constructive induction by analogy. *Machine Learning*, 8:107–150, 1992.

L. De Raedt and M. Bruynooghe. Constructive induction by analogy. In *Proceedings of the Sixth International Workshop on Machine Learning*, pages 476–477. Morgan Kaufmann, 1989.

L. De Raedt and L. Dehaspe. Clausal discovery. *Machine Learning*, 26(2/3): 99–146, 1997.

L. Dehaspe and L. De Raedt. DLAB: A declarative language bias formalism. In *International Syposium on Methodologies for Intelligent Systems*, pages 613–622, 1996. URL `citeseer.nj.nec.com/dehaspe96dlab.html`.

L. DeRaedt and M. Bruynooghe. On interactive concept-learning and assimilation. In D. Sleeman, editor, *Proceedings of the Third European Working Session on Learning*, pages 167–176. Pitman, 1988.

B. Dolsak and S. Muggleton. *Inductive Logic Programming*, chapter The Application of Inductive Logic Programming to Finite Element Mesh Design, pages 453–472. Academic Press, 1992.

P. Domingos. A process-oriented heuristic for model selection. In *Proceedings of the Fifteenth International Conference on Machine Learning*, Madison, WI, 1998. Morgan Kaufmann. URL `citeseer.nj.nec.com/article/domingos98processoriented.html`.

P. Domingos and M. Pazzani. On the optimality of the simple bayesian classifier under zero-one loss. *Machine Learning*, 29:103–130, 1997.

S. Džeroski and P. Flach, editors. *Proceedings of the 9th International Workshop onInductive Logic Programming*, volume 1634 of *Lecture Notes in Artificial Intelligence*, Bled, Slovenia, June 1999. Springer.

S. Džeroski, S. Muggleton, and S. Russell. PAC-learnability of determinate logic programs. In *Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory (COLT-92)*. ACM Press, 1992.

T. S. Eliot. Choruses from 'The Rock'. In *Selected Poems*. Faber and Faber, London, 1961.

C. Elkan. The Foundations of Cost-Sensitive Learning. *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, 340, 2001.

D. C. Englebart. Augmenting human intellect: A conceptual framework. Summary Report AFOSR-3233, Stanford Research Institute, Menlo Park, CA, 1962. URL http://www.bootstrap.org/augdocs/friedewald030402/augmentinghumanintellect/ahi62index.html.

T. Fawcett. ROC graphs: Notes and practical considerations for researchers. *Submitted to Machine Learning*, 2004. URL http://citeseer.ist.psu.edu/fawcett04roc.html.

U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. From data-mining to knowledge discovery in databases. *AI Magazine*, 17:37–54, 1996.

P. A. Flach and N. Lachiche. 1BC: A first-order bayesian classifier. In Džeroski and Flach [1999], pages 92–103.

P. A. Flach and N. Lachiche. Confirmation-guided discovery of first-order rules with tertius. *Machine Learning*, 42(1/2):61–95, 2001.

P. Flener and S. Yilmaz. Inductive synthesis of recursive logic programs: Achievements and prospects. *Journal of Logic Programming*, 41(2-3):141–195, 1999.

Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *European Conference on Computational Learning Theory*, pages 23–37, 1995. URL citeseer.ist.psu.edu/freund95decisiontheoretic.html.

J. Fürnkranz. Separate-and-conquer rule learning. *Artificial Intelligence Review*, 13(1):3–54, February 1999. URL citeseer.ist.psu.edu/26490.html.

J. Fürnkranz and P. A. Flach. An analysis of rule evaluation metrics. In *Proceedings of the 20th International Conference on Machine Learning (ICML'03)*, pages 202–209. AAAI Press, January 2003. ISBN 1-57735-189-4.

A. Giordana and L. Saitta. Phase transitions in relational learning. *Machine Learning*, 41:217–251, 2000.

C. Giraud-Carrier, R. Vilalta, and P. Brazdil. Special issue: Meta-learning. *Machine Learning*, 54(3), March 2004.

I. J. Good. *The Estimation of Probabilities: An Essay on Modern Bayesian Methods*. MIT Press, Cambridge, MA, 1965.

N. Goodman. Seven strictures on similarity. In N. Goodman, editor, *Problems and projects*, pages 437–447. Bobbs-Merrill, New York, 1972.

N. Goodman. *Fact, Fiction and Forecast*. Harvard University Press, fourth edition, 1983.

P. Graham. A plan for spam, 2002. URL http://www.paulgraham.com/spam.html.

J. A. Hanley and B. J. McNeil. The meaning and use of the area under a receiver operating characteristic (ROC) curve. *Radiology*, 143(1):29–36, April 1982.

F. E. Harrell. *Hmisc: A Package of Miscellaneous S Functions*. Vanderbilt University, Nashville, Tennessee, USA, 2005. URL `http://biostat.mc.vanderbilt.edu/s/Hmisc`.

T. Heskes. Empirical bayes for learning to learn. In *Proceedings of the 17th International Conference on Machine Learning*, pages 367–374. Morgan Kaufmann, San Francisco, CA, 2000. URL `citeseer.nj.nec.com/heskes00empirical.html`.

V. Ho, W. Wobcke, and P. Compton. EMMA: An e-mail management assistant. In J. Liu, B. Faltings, N. Zhong, R. Lu, and T. Nishida, editors, *Proceedings of the IEEE/WIC International Conference on Intelligent Agent Technology*, pages 67–74, Los Alamitos, CA, 2003. IEEE Computer Society. URL `http://doi.ieeecomputersociety.org/10.1109/IAT.2003.1241050`.

Y. Ho, R. Sreenivas, and P. Vakili. Ordinal optimization in DEDS. In *Discrete Event Dynamic Systems: Theory and Applications*, volume 2, pages 61–68, Boston, MA, 1992. Kluwer Academic Publishers.

D. R. Hofstadter. *Gödel, Escher, Bach: An Eternal Golden Braid*. Basic Books, Inc., New York, 20th anniversary edition, April 1999.

R. C. Holte, L. E. Acker, and B. W. Porter. Concept learning and the problem of small disjuncts. In *In Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 813–818, 1989.

E. T. Jaynes. *Probability Theory: The Logic of Science*. Cambridge University Press, Cambridge, UK, 2003. ISBN 0 521 59271 2.

R. Jeffrey. *Subjective Probability: The Real Thing*. Cambridge University Press, 2004.

D. Jensen and J. Neville. Linkage and autocorrelation cause feature selection bias in relational learning. In C. Sammut and A. G. Hoffman, editors, *Proceedings of the 19thInternational Conference on Machine Learning*, pages 259–266, Sydney, Australia, July 2002. University of New South Wales, Morgan Kaufmann Publishers. ISBN 1-55860-873-7.

B. Juba. Estimating relatedness via data compression. In *Proceedings of the 23rd International Conference on Machine Learning*, 2006.

M. J. Kearns and U. V. Vazirani. *An introduction to computational learning theory*. MIT Press, Cambridge, MA, 1994.

M. J. Kearns, R. E. Schapire, and L. Sellie. Toward efficient agnostic learning. *Machine Learning*, 17:341–352, 1994. URL `citeseer.ist.psu.edu/`

kearns94toward.html.

K. Khan, S. Muggleton, and R. Parson. Repeat learning using predicate invention. In Page [1998], pages 165–174.

J.-U. Kietz and S. Wrobel. Controlling the complexity of learning in logic syntactic and task oriented models. In S. Muggleton, editor, *Inductive Logic Programming*, pages 335–359. Academic Press Ltd., London, 1992.

R. King, K. Whelan, F. Jones, P. Reiser, C. Bryant, S. Muggleton, D. Kell, and S. Oliver. Functional genomic hypothesis generation and experimentation by a robot scientist. *Nature*, 427:247–252, January 2004.

S. Kramer. Predicate invention: A comprehensive view. Technical Report OFAI-TR-95-32, Austrian Research Institute for Artificial Intelligence, Schottengasse 3, A-1010 Vienna, Austria, 1995. URL citeseer.ist.psu.edu/97560.html.

M.-A. Krogel, S. Rawles, F. Železný, P. A. Flach, N. Lavrăc, and S. Wrobel. Comparative evaluation of approaches to propositionalization. In T. Horváth and A. Yamamoto, editors, *Proceedings of the 13th International Conference on Inductive Logic Programming*, number 2835 in Lecture Notes in Artificial Intelligence, pages 197–214. Springer-Verlag, 2003.

P. Langley and H. A. Simon. Applications of machine learning and rule induction. *Communications of the ACM*, 38(11):54–64, 1995. URL citeseer.ist.psu.edu/langley95applications.html.

P. Latinne, M. Saerens, and C. Decaestecker. Adjusting the outputs of a classifier to new a priori probabilities may significantly improve classification accuracy: Evidence from a multi-class problem in remote sensing. In *Proc. 18th International Conf. on Machine Learning*, pages 298–305. Morgan Kaufmann, San Francisco, CA, 2001. URL citeseer.ist.psu.edu/article/latinne01adjusting.html.

N. Lavrač and P. A. Flach. An extended transformation approach to inductive logic programming. *ACM Transactions on Computational Logic (TOCL)*, 2 (4):458–494, 2001. ISSN 1529-3785. URL http://doi.acm.org/10.1145/383779.383781.

M. Li and P. Vitanyi. *An Introduction to Kolmogorov Complexity and its Applications*. Springer-Verlag, New York, 1997.

Z. Marx, M. T. Rosenstein, T. G. Dietterich, and L. P. Kaelbling. Two algorithms for transfer learning. To appear in "Inductive Transfer: 10 years later", 2007.

E. McCreath. *Induction in First Order Logic from Noisy Training Examples and Fixed Example Set Sizes*. PhD thesis, School of Computer Science and Engineering, University of New South Wales, 1999.

E. McCreath and M. Reid. A noise resistant model inference system. In S. Arikawa and K. Furukawa, editors, *Proceedings of the 2nd International Discovery Science Conference*, volume 1721 of *Lecture Notes in Artificial Intelligence*, pages 252–263, Tokyo, Japan, Dec. 1999. Springer.

E. McCreath and A. Sharma. LIME: A system for learning relations. In *Algorithmic Learning Theory*, pages 336–374, 1998. URL `citeseer.nj.nec.com/mccreath98lime.html`.

D. L. Medin, R. L. Goldstone, and D. Gentner. Respects for similarity. *Psychological Review*, 100(2):254–278, 1993.

R. S. Michalski. On the quasi-minimal solution of the covering problem. In *Proceedings of the 5th International Conference on Information Processing*, volume A3 (Switching Circuits), pages 125–128, Bled, Yugoslavia, 1969.

T. M. Mitchell. Version spaces: A candidate elimination approach to rule learning. In *Proceedings of the Fifth International Joint Conference on AI*, pages 305–310, Cambridge, MA, 1977. MIT Press.

T. M. Mitchell. The need for biases in learning generalizations. Technical Report CBM-TR-117, Rutgers University, New Brunswick, New Jersey, 1980. URL `citeseer.nj.nec.com/mitchell80need.html`.

T. M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.

K. Morik. Sloppy modeling. In K. Morik, editor, *Knowledge Representation and Organization in Machine Learning*, volume 347 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, Berlin and New York, 1989.

K. Morik. Balanced cooperative modelling. *Machine Learning*, 11:217–235, 1993.

J. Morin. *Learning Relational Clichés with Contextual Generalization*. PhD thesis, School of Information Technology and Engineering, University of Ottawa, Canada, 1999.

J. Morin and S. Matwin. Relational learning with transfer of knowledge between domains. In H. J. Hamilton, editor, *Advances in Artificial Intelligence: 13th Biennial Conference of the Canadian Society for Computational Studies of Intelligence*, volume 1822 / 2000 of *Lecture Notes in Computer Science*, page 379, Montréal, Quebec, Canada, May 2000. Springer-Verlag. URL `http://www.springerlink.com/link.asp?id=yljd7y9djd8gq1by`.

S. Muggleton. Learning from positive data. In S. Muggleton, editor, *Proceedings of the 6th International Workshop on ILP*, number 1314 in Lecture Notes in Artificial Intelligence, pages 358–376. Springer, 1996.

S. Muggleton. Exceeding human limits. *Nature*, 440:409–410, March 2006.

S. Muggleton and M. Bain. Analogical prediction. In Džeroski and Flach [1999].

S. Muggleton and C. Feng. Efficient induction of logic programs. In *Proceedings of the 1st Conference on Algorithmic Learning Theory*, pages 1–14, Tokyo, Japan, 1990.

S. Muggleton, A. Srinivasan, and M. Bain. Compression, significance and accuracy. In D. Sleeman and P. Edwards, editors, *Proceedings of the 9th International Workshop on Machine Learning*, pages 338–347. Morgan Kaufmann, 1992. URL `citeseer.ist.psu.edu/muggleton92compression.html`.

S. H. Muggleton. Inverse entailment and progol. *New Generation Computing*, 13(3,4):245–286, 1995.

C. Nédellec, C. Rouveirol, H. Adé, F. Bergadano, and B. Tausend. Declarative bias in ILP. In L. De Raedt, editor, *Advances in Inductive Logic Programming*, volume 32 of *Frontiers in Artificial Intelligence and Applications*, pages 82–103. IOS Press, Amsterdam, 1996.

D. Newman, S. Hettich, C. Blake, and C. Merz. UCI repository of machine learning databases, 1998. URL `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

S.-H. Nienhuys-Cheng and R. de Wolf. *Foundations of Inductive Logic Programming*. Number 1228 in Lecture Notes in Artificial Intelligence. Springer-Verlag, Berlin Heidelberg, 1997.

D. A. Oblinger, M. Reid, M. Brodie, and R. d. S. Braz. Cross training and its application to skill mining. *IBM Systems Journal*, 41(3):449–460, 2002.

D. Page, editor. *Proceedings of the 8th International Conference onInductive Logic Programming*, volume 1446 of *Lecture Notes in Artificial Intelligence*, Madison, Wisconsin, USA, July 1998. Springer.

M. Pazzani and D. Kibler. The utility of knowledge in inductive learning. *Machine Learning*, 9(1):57–94, 1992. URL `http://dx.doi.org/10.1023/A:1022628829777`.

B. Pfahringer. *Practical Uses of the Minimum Description Length Principle in Inductive Learning*. PhD thesis, Technische Universität Wien, 1995.

B. Pfahringer, H. Bensusan, and C. Giraud-Carrier. Meta-learning by landmarking various learning algorithms. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 743–750, San Francisco, California, 2000. Morgan Kaufmann. ISBN 1-55860-707-2. URL `citeseer.ist.psu.edu/pfahringer00metalearning.html`.

G. Plotkin. A further note on inductive generalization. *Machine Intelligence*, 6:101–124, 1971.

J. R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 9:239–266, 1990.

J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, California, 1993.

R Development Core Team. *R: A language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria, 2005. URL `http://www.R-project.org`.

R. Raina, A. Y. Ng, and D. Koller. Constructing informative priors using transfer learning. In *Proceedings of the 23rd International Conference on Machine Learning*, 2006.

R. Rao, D. Gordon, and W. Spears. For every generalization action, is there really an equal and opposite reaction? analysis of the conservation law for generalization performance. In *Proceedings of the 12th International Conference on Machine Learning*, pages 115–121, Tahoe City, CA, 1995. Morgan Kaufmann.

M. Reid and M. R. K. Ryan. Using ILP to improve planning in hierarchical reinforcement learning. In J. Cussens and A. Frisch, editors, *Proceedings of the 10th International Conference onInductive Logic Programming*, volume 1866 of *Lecture Notes in Artificial Intelligence*, London, UK, July 2000. Springer.

M. D. Reid. Using multitask learning to improve rule evaluation. In *Proceedings of the 14th International Conference on ILP*, pages 252–269. Springer-Verlag, 2004.

J. Rissanen. Modeling by shortest data description. *Automatica*, 14:465–471, 1978.

H. Robbins. An empirical Bayes approach to statistics. *Proc. 3rd Berkeley Sympos. Math. Statist. Probability*, 1:157–163, 1956.

M. R. K. Ryan and M. Reid. Learning to fly: An application of heirarchical reinforcement learning. In *Proceedings of the 17thInternational Conference on Machine Learning*, 2000.

Saki. The square egg. In *The Best of Saki (H. H. Munro)*. Ure Smith, Sydney, Australia, 1968.

C. Sammut. *Learning Concepts by Performing Experiments*. PhD thesis, University of New South Wales, 1981. URL `http://www.cse.unsw.edu.au/~claude/research/papers/thesis/index.html`.

C. Sammut and R. B. Banerji. Learning concepts by asking questions. In R. Michalski, J. Carbonell, and T. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach Volume 2*, pages 167–191. Morgan Kaufmann, 1986.

C. Schaffer. A conservation law for generalization performance. In *Proceedings of the 11th International Conference on Machine Learning*, pages 259–265,

1994.

J. Schmidhuber. Reinforcement learning with self-modifying policies. In Thrun and Pratt [1997a], pages 293–309.

J. Schmidhuber. Optimal ordered problem solver. *Machine Learning*, 54:211–254, 2004.

M. Sebag and C. Rouveirol. Resource-bounded relational reasoning: Induction and deduction through stochastic matching. *Machine Learning*, 38:41–62, 2000.

E. Y. Shapiro. *Algorithmic Program Debugging*. MIT Press, Cambridge, MA, 1983.

D. Silver. *Selective Transfer of Neural network Task Knowledge*. PhD thesis, Graduate Program in Computer Science, University of Western Ontario, London, Ontario, Canada, June 2000.

D. Silver and R. Mercer. Toward a model of consolidation: The retention and transfer of neural net task knowledge. In *Proceedings of the INNS World Congress on Neural Networks*, volume III, pages 164–169, Washington, DC, 1995. URL `citeseer.nj.nec.com/silver95toward.html`.

D. L. Silver and R. E. Mercer. *Learning to Learn*, chapter The Parallel Transfer of Task Knowledge Using Dynamic Learning Rates Based on a Measure of Relatedness, pages 213–233. In , Thrun and Pratt [1997a], 1998.

G. Silverstein and M. J. Pazzani. Relational clichés: Constraining constructive induction during relational learning. In *Proceedings of the 8th International Conference on Machine Learning*, pages 203–207. Morgan Kaufmann, 1991.

G. Silverstein and M. J. Pazzani. Learning relational clichés. In *Proceedings of the IJCAI-93 Workshop onInductive Logic Programming*, pages 71–82, Chambery, France, 1993.

H. A. Simon. *The Sciences of the Artificial*. The MIT Press, Cambridge, MA, third edition, 1999.

A. Srinivasan. A study of two sampling methods for analysing large datasets with ILP. *Data Mining and Knowledge Discovery*, 3(1):95–123, 1999. URL `citeseer.nj.nec.com/srinivasan99study.html`.

A. Srinivasan. ALEPH: A learning engine for proposing hypotheses, September 2001. URL `http://www.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/misc/aleph.tex`.

A. Srinivasan. Personal communication. Email regarding mutagenesis and carcinogenesis datasets, September 2002.

A. Srinivasan and R. Camacho. Numerical reasoning with an ILP program capable of lazy evaluation and customised search. *Journal of Logic Programming*, 40(2,3):185–214, 1999. URL `ftp://ftp.comlab.ox.ac.uk/pub/`

Packages/ILP/Papers/AS/jlp99.ps.gz.

A. Srinivasan and R. C. Camacho. Experiments in numerical reasoning with ILP. Technical Report PRG-TR-22-96, Oxford University Computing Laboratory, Oxford, 1996.

A. Srinivasan and R. D. King. Feature construction with inductive logic programming: a study of quantative predictions of biological activity aided by structural attributes. *Data Mining and Knowledge Discovery*, 3(1):37–57, 1999.

A. Srinivasan, S. Muggleton, R. D. King, and M. J. E. Sternberg. Mutagenesis: ILP experiments in a non-determinate biological domain. In S. Wrobel, editor, *Proceedings of the Fourth Inductive Logic Programming Workshop*, volume 237, pages 217–232. Gesellschaft für Mathematik und Datenverarbeitung MBH, 1994. URL http://citeseer.ist.psu.edu/srinivasan94mutagenesis.html.

A. Srinivasan, R. D. King, S. Muggleton, and M. J. E. Sternberg. Carcinogenesis predictions using ILP. In S. Džeroski and N. Lavrač, editors, *Proceedings of the 7th International Workshop on Inductive Logic Programming*, volume 1297, pages 273–287. Springer-Verlag, 1997. URL citeseer.nj.nec.com/srinivasan97carcinogenesis.html.

A. Srinivasan, R. D. King, and M. E. Bain. An empirical study of the use of relevance information in inductive logic programming. *Journal of Machine Learning Research*, (4):369–383, 2003.

S. Thrun and J. O'Sullivan. Discovering structure in multiple learning tasks: The TC algorithm. In L. Saitta, editor, *Proceedings of the 13thInternational Conference on Machine Learning*, pages 489–497. Morgan Kaufmann Publishers, 1996. URL citeseer.nj.nec.com/thrun96discovering.html.

S. Thrun and L. Pratt, editors. *Learning to Learn*. Kluwer Academic Publishers, Boston, MA, 1997a.

S. Thrun and L. Pratt. *Learning to Learn*, chapter Learning to Learn: Introduction and Overview, pages 3–13. In , Thrun and Pratt [1997a], 1997b.

P. Turney. Types of costs in inductive concept learning. In *Proceedings of the Cost-Sensitive Learning Workshop at the 17th ICML-2000 Conference*, Stanford, CA., July 2000. URL http://iit-iti.nrc-cnrc.gc.ca/publications/nrc-43671_e.html. NRC 43671.

L. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11): 1134–1142, 1984.

S. A. Vere. Induction of relational productions in the presence of background information. In *Proceedings of the Fifth International Joint Conference on AI*, pages 349–355, 1977.

R. Vilalta and Y. Drissi. A perspective view and survey of meta-learning. *Journal of Artificial Intelligence Review*, 18(2):77–95, 2002.

R. Vilalta and D. Oblinger. A quantification of distance-bias between evaluation metrics in classification. In *Proc. 17th International Conf. on Machine Learning*, pages 1087–1094. Morgan Kaufmann, San Francisco, CA, 2000. URL `citeseer.nj.nec.com/vilalta00quantification.html`.

C. S. Wallace and M. P. Georgeff. A general objective for inductive inference. Technical Report TR 32, Department of Computer Science, Monash University, Australia, 1983. URL `http://www.csse.monash.edu.au/~lloyd/tildeMML/Structured/TR32/`.

R. E. Walpole and R. H. Myers. *Probability and Statistics for Engineers and Scientists*. Macmillan, New York, second edition, 1978.

G. M. Weiss and F. Provost. The effect of class distribution on classifier learning. Technical Report ML-TR-44, Department of Compter Science, Rutgers University, New Brunswick, NJ, 2001.

D. H. Wolpert. The lack of A priori distinctions between learning algorithms. *Neural Computation*, 8(7):1341–1390, 1996. URL `citeseer.ist.psu.edu/wolpert96lack.html`.

S. Wrobel. *Concept Formation and Knowledge Revision*. Kluwer Academic Publishers, Dordrecht, 1994.

B. Zadrozny, J. Langford, and N. Abe. Cost-sensitive learning by cost-proportionate example weighting. *Data Mining, 2003. ICDM 2003. Third IEEE International Conference on*, pages 435–442, 2003.

C. Zhang and S. Zhang. *Association Rule Mining: Models and Algorithms*, volume 2307 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 2002. ISBN 3-540-43533-6.

APPENDIX A

Implementation of DESCRIBE

```
:- use_module(library(lists)).
% == Descriptor Template Predicates ==
%  describe(+Clause, -Desc) - Get a sparse description for Clause
%  default(?Desc, -DefValue) - Get the default value for Desc
%  value(?Desc, +Clause, -Value) - Get value of Desc on Clause

describe(Clause, Desc) :- all([D,V], value(D, Clause, V), Desc).

% num_lits -- Returns the number of literals in the body of a clause.
default(num_lits, 0).
value(num_lits, Clause, Length) :-
   body_lits(Clause, BodyLits), length(BodyLits, Length).

% has_pred(P/N) -- Tests whether a predicate P with arity N appears in a clause.
default(has_pred(_/_), false).
value(has_pred(PredName/Arity), Clause, true) :-
   body_lits(Clause, BodyLits), member(Lit, BodyLits),
   functor(Lit, PredName, Arity).

% has_arg(Pred,Index,Val) -- Test whether a predicate Pred is present in a clause
% and the argument indexed by Index in the predicate is equal to the value Val.
default(has_arg(_, _, _), false).
value(has_arg(PredName, ArgIndex, ArgValue), Clause, true) :-
   body_lits(Clause, BodyLits),
   member(Lit, BodyLits),
   functor(Lit, PredName, Arity),
   between(ArgIndex, 1, Arity),
   arg(ArgIndex, Lit, ArgValue),
   \+ var(ArgValue).

% -- between(?N, +Low, +High) -- True when Low <= N <= High
between(N,N,High).
between(N,Low,High) :- Low < High, Low1 is Low+1, between(N, Low1, High).

% -- body_lits(+Clause, -BodyLits) -- Get literals from the body of Clause.
body_lits((_:-(Lit,Lits)), [Lit|Rest]) :- body_lits((_:-Lits), Rest), !.
body_lits((_:-Lit), [Lit]).
```

APPENDIX B

Reading Preferences Example

B.1. Mode and Type Settings

```
% Background knowledge for the example reading preferences learning tasks.
:- mode(1,like(+book)).
:- mode(*,size(+book, #size_val)).
:- mode(*,genre(+book, #genre_val)).
:- mode(*,nation(+book, #nation_val)).
:- mode(*,year(+book, #year_val)).

:- determination(like/1,size/2).
:- determination(like/1,genre/2).
:- determination(like/1,nation/2).
:- determination(like/1,year/2).

% Values
size_val(small).
size_val(medium).
size_val(large).

genre_val(scifi).
genre_val(romance).
genre_val(horror).

nation_val(aus).
nation_val(uk).
nation_val(usa).

year_val('00s').
year_val('90s').
year_val('80s').
```

B.2. Example DFT for Concept E

```
% DFT for the example reading preferences concept E.
total([[3976,1126],[16025,18873]]).
counts(has_arg(genre,2,scifi),true,[[1469,245],[7531,8755]]).
counts(has_arg(size,2,small),true,[[1258,175],[5442,6525]]).
counts(has_arg(nation,2,aus),true,[[1516,282],[7234,8468]]).
counts(has_pred(genre/2),true,[[1758,355],[10392,11795]]).
counts(has_pred(size/2),true,[[1917,374],[11183,12726]]).
counts(has_pred(nation/2),true,[[1891,307],[9259,10843]]).
counts(has_arg(year,2,'00s'),true,[[1501,103],[4549,5947]]).
counts(has_pred(year/2),true,[[1621,205],[5679,7095]]).
counts(has_arg(size,2,large),true,[[318,89],[2432,2661]]).
counts(has_arg(genre,2,romance),true,[[181,91],[1969,2059]]).
counts(has_arg(year,2,'80s'),true,[[26,45],[424,405]]).
counts(has_arg(nation,2,usa),true,[[375,25],[2025,2375]]).
counts(has_arg(size,2,medium),true,[[341,110],[3309,3540]]).
counts(has_arg(genre,2,horror),true,[[108,19],[892,981]]).
counts(has_arg(year,2,'90s'),true,[[94,57],[706,743]]).
```