Improving Rule Evaluation Using Multitask Learning

Mark D. Reid

School of Computer Science and Engineering
University of New South Wales
Sydney, NSW 2052, Australia
mreid@cse.unsw.edu.au

Abstract. This paper introduces DEFT, a new multitask learning approach for rule learning algorithms. Like other multitask learning systems, the one proposed here is able to improve learning performance on a primary task through the use of a bias learnt from similar secondary tasks. What distinguishes DEFT from other approaches is its use of rule descriptions as a basis for task similarity. By translating a rule into a feature vector or "description", the performance of similarly described rules on the secondary tasks can be used to modify the evaluation of the rule for the primary task. This explicitly addresses difficulties with accurately evaluating, and therefore finding, good rules from small datasets. DEFT is implemented on top of an existing ILP system and the approach is tested on a variety of relational learning tasks. Given appropriate secondary tasks, the results show that DEFT is able to compensate for insufficient training examples.

1 Introduction

Obtaining correctly classified examples for a supervised learning problem is a crucial but sometimes costly task, especially if experts are required to classify the examples. This can result in too few training examples for the problem at hand. In some chemical domains for example, small datasets are "not unusual" since "data are often sparse, and bioassays expensive" [1]. The goal of the research presented in this paper is to improve the quality of learning when data is limited by making use of classified examples gathered for other tasks in the same domain. This places the research in the field of multitask learning [2].

The main problem when learning from small amounts of data is that many candidate hypotheses can fit the data equally well. In this situation, a learner must rely heavily on its inductive bias to decide between them [3]. In rule learning, these decisions involve the suitability of individual rules for a hypothesis and the bias of a rule learning algorithm is influenced by several factors. These include choices that determine which rules are admissible, how they are searched and what heuristic is used to evaluate and rank them [4,5]. Making the best choices for a particular learning problem requires expertise in the problem domain as well as a good knowledge of rule learning algorithms and can be more costly than obtaining more classified examples.

R. Camacho, R. King, A. Srinivasan (Eds.): ILP 2004, LNAI 3194, pp. 252–269, 2004.
© Springer-Verlag Berlin Heidelberg 2004

Multitask learning eases the burden of choosing a bias for a task with limited data by assuming that related, secondary tasks are available. The learner uses these to learn a bias that can improve its performance on the primary task. Existing multitask approaches for rule learning [6,7,8] have focused on learning language or search biases. In contrast, the DEFT system[1] introduced in this paper is concerned exclusively with improving a learner's evaluation of rules on limited data.

The DEFT approach is outlined in Section 2 where *rule descriptions* are introduced to define classes of similar rules. An assessment of a rule on limited primary task data can be augmented using the performance of similar rules on secondary tasks. When the secondary and primary tasks are related the expanded assessment can improve the reliability of an evaluation metric. Section 3 describes an implementation of this approach for ILP that uses ALEPH [9] as the underlying concept learner. This implementation is evaluated on a range of datasets and the results presented in Section 4. Related work from ILP and bias learning is presented in Section 5 and the paper is concluded with a discussion of future directions in Section 6.

2 Description-Based Evaluation

The relative quality of rules according to an evaluation metric can be poorly estimated when only a small number of training examples are available. This section describes a method, called DEFT, that improves these estimates by transforming any evaluation metric into a representation-based one. The transformed metric bases its evaluation of a rule on the training data for the primary task as well as the performance of similar rules on other, secondary, tasks. Rule similarity is defined in Section 2.2 in terms of functions, called *descriptors*, that are used to transform a rule into an attribute vector called a *description*. Section 2.3 shows how classification probabilities can be estimated from a secondary task based on a rule's description. These probabilities can be used to create a *virtual contingency table* for a rule on a secondary task. Section 2.4 explains how these can be combined with a rule's real contingency table for the primary task. The result can then be fed into an evaluation metric to provide an assessment that takes into account a rule's description and its similarity to rules that have been evaluated on the secondary task.

2.1 Preliminaries

For the purposes of this paper, a supervised concept learning task consists of an instance space X, a set of class labels $Y = \{+, -\}$, and training examples $E \subset X \times Y$ for some unknown *target concept* $t : X \to Y$. A *rule* $r = h \leftarrow b$ consists of a label $h \in Y$, called the *head*, and a condition b, called the *body*, that can be tested against instances. A rule is said to *match* an instance $x \in X$ if its

[1] Description-based Evaluation Function Transfer

body is true for that instance. In this case, the rule classifies the instance with the label h.

A rule learning algorithm solves a learning task by finding a set of rules that (ideally) classifies every instance in accordance with the target concept. Most existing rule learning algorithms adopt a strategy whereby a set of rules is created by repeatedly finding a single rule that explains some of the training examples and then removing them from the training set. For a comprehensive survey of this "covering" approach to rule learning we refer the reader to [5].

Our primary concern is with the assessment of individual rules on the training examples. In particular, we are interested in "purity-based" evaluation metrics: those that assess a rule based on the type and number of misclassifications it makes on the training examples. These are summarised using a *contingency table*. The contingency table for a rule r and examples E is written in matrix form like so

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

Each entry n_{ij} is a count of the number of times the rule r assigned the label i to an example $(x, j) \in E$. A matrix containing the relative frequencies of each type of classification can be derived from a contingency table. We call this a *classification probability matrix* (CPM) and define it to be $\mathbf{p}_E(r) = \frac{1}{N}\mathbf{n}_E(r)$ where $N = \sum_{ij} n_{ij}$ is the number of examples in E. The values $p_{ij}(r)$ in a CPM can be viewed as estimates of the true classification probability of a rule, $\Pr_{x \in X}(r(x) = i, t(x) = j)$.

2.2 Rule Descriptions

The key to the DEFT multitask learning approach is to combine the assessment of rule on a primary task with the assessment of the same rule on one or more secondary tasks. The hope is that if the target concepts are similar[2] the combined evaluation will, on the primary task, prefer rules that are similar to those that perform well on the secondary tasks. A straight-forward way to combine task assessments would be to treat examples from a secondary task as extra training examples for the primary task. Caruana's MTL for decision tree learning [2] does exactly this when choosing between splits while building a decision tree. The problem with this approach for evaluating rules is that entire rules are assessed and not the steps in building them. Consider the rules in concepts A and D of Fig. 3. While they are quite similar, the first rule of A will not cover any example covered by the first rule of D. This means the classification probabilities for the two rules will be quite different even though they "look" the same. The problem becomes worse when the same representation language is only partially shared between the tasks. What is required is a looser notion of similarity than "covering the same examples".

[2] Concept similarity or relatedness is an important but difficult issue in multitask learning. Space prevents it being discussed here so the reader is referred to [2,10].

We would like to say two rules are similar if they share some salient features such their length, particular conditions or constants. We call these rule features *descriptors* and use them to transform a rule into an attribute-value vector. More formally, a descriptor $d : R \to V_d$ is a function from the set of rules R to the descriptor values V_d. A collection of descriptors $D = \{d_k\}_{k=1}^{n}$is called a *descriptor set* and induces a function $\mathbf{d} : R \to V_{d_1} \times \ldots \times V_{d_n}$, called a *description*, that maps a rule to a attribute-value vector $\mathbf{d}(r) = (d_1(r), \ldots, d_n(r))$. In general, rules may share the same description or part description. As a shorthand, we will write $r' \in d(r)$ when $d(r') = d(r)$, and write $r' \in \mathbf{d}(r)$ when $r' \in d_k(r)$ for all $k = 1 \ldots n$. Treating descriptions as equivalence classes for rules allows to generalise over them and therefore learn functions of rules based on their description.

2.3 Learning Classification Probabilities

Given a task with target concept t, classification probabilities for this task can be seen as functions $p_{ij}(r)$ that map rules to values in $[0, 1]$. As already discussed, these functions can return very different values for two rules that have a similar description. One way to fix this is to smooth the functions by averaging them over rules that share the same description:

$$q_{ij}(r) = \Pr_{r',x}(r'(x) = i, t(x) = j | r' \in \mathbf{d}(r)).$$

The matrix $\mathbf{q}(r) = (q_{ij}(r))$ is analogous to the CPM $\mathbf{p}(r)$ except that the classification probabilities are based on rule descriptions. It is therefore called a *description-based CPM* (DCPM). Using $\Pr(i, j|\mathbf{d}(r))$ as a shorthand, the Bayes' identity and a naïve Bayes' assumption about the independence of the descriptors d_i lets us express

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

If values for $\Pr(d_k(r)|i, j)$ and $\Pr(i, j)$ can be determined, $q_{ij}(r)$ can be computed since the $\Pr(\mathbf{d}(r))$ term just normalises the $q_{ij}(r)$ so that $\sum_{i,j} q_{ij}(r) = 1$. The former terms can be derived from $\Pr(i, j, d_k(r))$ so this is what we now focus on estimating.

The probability we wish to estimate measures the chances of drawing an instance $x \in X$ and rule $r' \in R$ such that r' classifies x with label i, the target concept classifies x with label j and that $d_k(r')$ and $d_k(r)$ have the same value $v \in V_{d_k}$. Given a sample of rules $\overline{R} \subseteq R$ and a set of examples E, the number of times this event occurs in these samples can be counted:

$$\mathbf{s}(d_k, v) = \sum_{\substack{r \in \overline{R} \\ d_k(r) = v}} \mathbf{n}_E(r) \tag{1}$$

where $\mathbf{n}_E(r)$ is the contingency table for r on the examples E. The collection of matrices $\mathbf{s}_D = \{\mathbf{s}(d_k, v) : d_k \in D, v \in V_k\}$ is called a *descriptor frequency table*

(DFT) for D. By letting $s_{ij} = \sum_{d,v} s_{ij}(d,v)$ and $s = \sum_{i,j} s_{ij}$, a DFT can be used to estimate the probabilities $\Pr(i,j) \approx \frac{s_{ij}}{s}$ and $\Pr(d_k(r)|i,j) \approx \frac{s_{ij}(d_k,d_k(r))}{s_{ij}}$. The functions $q_{ij}(r)$ can therefore be approximated by

$$q_{ij}(r) \approx \frac{s_{ij}}{s} \prod_{k=1}^{n} \frac{s_{ij}(d_k, d_k(r))}{s_{ij}}. \tag{2}$$

In practice, the q_{ij} functions are approximated using rules and examples taken from a secondary task. This requires some way of sampling rules for that task and is discussed in Section 3.3. The quality of the approximation will depend on the size and bias of the rule sample and the examples. It will also depend on the descriptor set used and the validity of the naive Bayes' assumption for those descriptors. While all these factors are important, it must be remembered that the DCPM is only designed to crudely estimate a rule's CPM on a secondary task based on its description. We now introduce a method for combining values in a DCPM with examples from the primary learning task.

2.4 Virtual Contingency Tables

The values $q_{ij}(r)$ in a DCPM $\mathbf{q}(r)$ express the chances that an example with label j will be given label i by r based on its description. If the rule was to classify M examples, the expected number with each type of classification can be summarised in a *virtual contingency table* $\mathbf{m}(r) = M\mathbf{q}(r)$. This can be used to increase the number of examples used to asses a rule as follows. If $\mathbf{n}_E(r)$ is the contingency table for r on some small set of examples E, we can define its *augmented contingency table* $\mathbf{n}^*(r) = \mathbf{n}_E(r) + \mathbf{m}(r)$. The relative size of the number of virtual examples, M, to the number of real examples, N, determines how much emphasis is given to the assessment of the rule on the primary task compared to the secondary task that generated $\mathbf{q}$.

An augmented CPM, $\mathbf{p}^*(r) = \frac{1}{N+M}\mathbf{n}^*(r)$, can be derived from an augmented contingency table. The entries in the resulting matrix are

$$p^*_{ij}(r) = \frac{n_{ij}(r) + m_{ij}(r)}{N + M}$$

and can be seen as "linearly squashing" the values $p_{ij}(r)$ towards the priors $q_{ij}(r)$, or equivalently, assuming that the $n^*_{ij}(r)$ values have a Dirichlet distribution with parameters $m_{ij}(r)$ [11, §4.1].

Any evaluation metric h can be transformed into a description-based metric h^* by using classification probabilities from a rule's augmented CPM. As a special case, this transformation can be used to turn the *precision* metric $prec_E(r) = \frac{pos}{pos+neg}$ into the generalised m-estimate [12] $gm_E(r) = \frac{pos+a}{(pos+a)+(neg+b)}$ where a and b are fixed and $pos = n_{++}(r)$, $neg = n_{+-}(r)$. The transformed metric $prec^*_E(r)$ is equal to $gm_E(r)$ with $a = m_{++}(r)$ and $b = m_{+-}(r)$. Using a description-based precision metric can therefore be seen as using a generalised m-estimate where the costs have been learnt from a secondary task.

3 Implementation

This section briefly outlines an implementation[3] of DEFT for inductive logic programming using ALEPH [9] as the base rule learner. Details of the ALEPH system that are relevant to this paper are provided in Section 3.1 along with the modification that allows it to use description-based evaluations. A discussion of the three main procedures required to implement DEFT make up the remainder of this section. These are procedures to: compute rule descriptions (Section 3.2), build descriptor frequency tables from secondary tasks (Section 3.3), and determine classification priors (Section 3.4). These procedures are applied to an example domain in Section 3.5.

3.1 The Base and Deft Learners

The ILP system ALEPH is designed to replicate the behaviour of many different kinds of rule learning systems. As the base learner for DEFT, however, we are only interested in its implementation of the inverse entailment algorithm PROGOL. Details glossed over in the following summary can be found in [13].

Given a collection of positive and negative ground facts, ALEPH induces a theory as a set of Horn clauses using a covering strategy. Each clause (or rule) in the theory is found by searching a space of *legal* rules. Predicates in the body of a legal rule must come from the *background knowledge* provided to the learner. Exactly how they can be combined in a rule's body is determined by *mode and type constraints*. These restrict the ways variables in the rule can be shared between its conditions. Legal rules are also constrained by an upper limit on their *variable depth*. In ALEPH, this is controlled by the `i` setting. For any positive example there is a most specific legal clause, called the *bottom clause*, that entails the example given the background knowledge. Any other legal clause that is required to cover the same positive example must also subsume the bottom clause. This fact is used to limit the search to a complete, general-to-specific, breadth-first search of subsets of the bottom clause that meet the legality requirements. The search is further restricted by requiring the subsets' sizes be no larger than the value specified in the `clauselength` setting. For efficiency reasons, a limit on the total number of rules examined in a single search can be controlled by the `nodes` setting.

By default, ALEPH uses the *coverage* metric[4] $cov_E(r) = n_{++}(r) - n_{+-}(r)$ to evaluate rules. The rule returned by the search is one that is *acceptable* and maximises this metric on the training data. The acceptability of a rule is controlled by the `noise` and `minacc` settings. The `noise` setting is an upper bound on the number of negative examples covered by a rule while `minacc` places a lower bound on the accuracy of a rule.

To use the DEFT approach described in the last section, the coverage metric of the base learner is transformed to $cov^*_E(r) = n^*_{++}(r) - n^*_{+-}(r)$, where the

[3] The source for the implementation is available from the author upon request.

[4] In [12, Theorem 4.1] this metric is shown to be equivalent to the accuracy metric $acc_E(r) = p_{++} + p_{--}$ and can therefore be defined in terms of a rule's CPM.

$n^*_{ij}(r)$ are the values taken from the rule's augmented contingency table. When the term "DEFT learner" is used in this paper, it refers to the base learner with its coverage metric modified in this way. As well as all the base learner's settings, the DEFT learner is additionally parameterised by the DFT it uses, the number of virtual examples M and the functions used to create rule descriptions.

3.2 Clause Descriptions

This paper will only consider two types of descriptors for clauses. The first type, denoted $pred(P/N)$, tests for the presence of predicate P with arity N in the body of a clause. The second type, $arg(P/N, I, Const)$, tests whether a clause has a predicate P/N with its Ith argument equal to $Const$. Both types of descriptors are Boolean-valued, returning either "true" or "false" when applied to a clause. Defining descriptors in terms of types avoids having to explicitly construct a descriptor set for a task. This is useful when the legal clauses for a search, and hence their predicates and constants, are difficult to determine in advance.

Computing values for descriptors is implemented in Prolog by a `value/3` predicate that 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. Each descriptor is also associated with a default value, implemented by the `default/2` predicate. As an example, the implementation of $pred(P/N)$ is shown in Fig. 1.

```
default(pred(P/N), false).
value((Head :- Body), pred(P/N), true) :-
    lit_list(Body,Lits), member(L,Lits), functor(L, P, N).
```

Fig. 1. Prolog code for the $pred(P/N)$ descriptor. The predicate `lit_list/2` takes a goal and turns it into a list of literals.

Due to Prolog's backtracking behaviour, the `value/3` relation can be used to find all descriptors that would return non-default values on a clause. This allows for a sparse representation for clause description s by assuming any descriptor not satisfying `value/3` takes on its default value.

Using a sparse representation has several advantages. Most importantly, the time taken to compute a description for a clause can be made a function of the complexity of clause instead of the description set. This is significant since ALEPH's general-to-specific search will mainly require descriptions for short clauses. Computing descriptions for clauses is also crucial for the construction of descriptor frequency tables from secondary tasks which will be discussed in the next section.

3.3 Descriptor Frequency Tables

As described in Section 2.3, making a DFT for a secondary task requires rules to be sampled from that task and evaluated on its training examples, E. The `make_DFT`(E) procedure in Fig. 2 shows how this is implemented. The clause sampling procedure `scs` in line 1 is ALEPH's implementation of the stochastic clause selection algorithm, described in [14]. In a nutshell, `scs`($\perp$) returns a clause that entails the bottom clause $\perp$ by randomly choosing a subset of its literals. Efficient techniques are used to ensure the procedure only draws legal clauses with uniform probability.

Each drawn clause r has its contingency table n computed against the examples E and added to *total*. The loop at line 2 then finds non-default descriptor-value pairs (d, v) for r using the `value/3` predicate described earlier, and for each pair adds the matrix n is added to an accumulation of counts in $counts[d, v]$. The whole process is repeated `sample_cl` times per bottom clause, each generated from one of `sample_ex` positive examples drawn without replacement from E.

The values collected in the structures *total* and *counts* are sufficient for computing the matrices $\mathbf{s}(d, v)$ described in Section 2.3, and hence represent a DFT. Clearly, for any descriptor d in the DFT and one of its non-default values v, $\mathbf{s}(d, v) = counts[d, v]$. When v_0 is a default value for d, $\mathbf{s}(d, v_0) = total - \sum_{v \neq v_0} counts[d, v]$. This can be seen by observing that the sets of rules $R_{d,v} = \{r \in R \,|\, d(r) = v\}$ partition R for each d. Summing both sides of equation 1 over $v \in V_d$ shows that the matrix $total = \sum_{v \in V_d} \mathbf{s}(d, v)$ for all d.

3.4 Calculating Classification Priors

A DFT constructed using the `make_DFT` procedure can be used estimate values for the $q_{ij}(r)$ functions of equation 2. The quantities s_{ij} and s in that equation are fixed and can be precomputed once for a DFT since $s_{ij} = total_{ij}$ and $s = \sum_{i,j} s_{ij}$. In the worst case, computing the product term would require iterating through each descriptor d in the DFT, applying it to the rule r and multiplying together the values $\frac{s_{ij}(d,d(r))}{s_{ij}}$. Since most descriptors will take on their default value for any given rule, we can precompute a *base table* $\mathbf{b}$ that contains q_{ij} values for the default description vector (where all descriptors take on their default values) and update it as follows. Letting D denote the descriptors in the DFT, the values in its base table are

$$b_{ij} = \frac{s_{ij}}{s} \prod_{d \in D} \frac{s_{ij}(d, v_{d,0})}{s_{ij}}$$

where $v_{d,0}$ denotes the default value for d. Given a rule r, $q_{ij}(r)$ can be estimated by multiplying b_{ij} by $u_{ij}(d, v) = \frac{s_{ij}(d,v)}{s_{ij}(d,v_{d,0})}$ for each d and v satisfying `value(`r`,`d`,`v`)`. The matrices $\mathbf{u}(d, v)$ are called the *update tables* and can also be precomputed for a DFT. This means the time taken to calculate a prior for a rule depends only on the time it takes to compute its description and not the size of the DFT.

```
procedure make_DFT(E)
    repeat sample_ex times
        Select new example e ∈ E and saturate to create bottom clause ⊥
        repeat sample_cl times
1           Draw clause r using scs(⊥) and compute matrix n = n_E(r)
            total ← total + n
2           foreach d, v satisfying value(r, d, v) do
                counts[d, v] ← counts[d, v] + n
    return total, counts
```

Fig. 2. Given a set of examples E, `make_DFT`(E) returns a matrix *total* and a hashtable *counts* with descriptor-value pairs as keys and matrices as values.

3.5 Example Tasks

To clarify the procedures described in the previous sections we introduce four example learning tasks, A, B, C, and D, with target concepts shown in Figure 3. They are all simple attribute-value concepts represented using Horn clauses and instances for the tasks take on the values 0,1 or 2 for each of the four attributes a_0, a_1, a_2 and a_3.

A
```
a(X) :- a1(X,0), a2(X,1).
a(X) :- a2(X,0), a3(X,1).
```

B
```
b(X) :- a0(X,0), a1(X,1).
b(X) :- a0(X,1), a1(X,1).
```

C
```
c(X) :- a1(X,0), a2(X,0).
c(X) :- a1(X,1), a2(X,1).
```

D
```
d(X) :- a1(X,0), a2(X,0).
d(X) :- a2(X,1), a3(X,1).
```

Fig. 3. Four example concepts.

Intuitively, rules for concepts A and D are most similar due to their shared predicates and constants while rules from A and B are quite dissimilar. The descriptors $pred(P/N)$ and $arg(P/N, I, Const)$ are able to express these similarities and are used to construct DFTs from training sets containing 50 positive and 50 negative examples of each concept. Figure 4 shows the base tables and some update tables for those DFTs. These can be used to compare the priors given to a description across the four concepts. The ratio $\beta = u_{++} : u_{+-}$ in an update table indicates how the virtual true positive and false positive rates for a concept will shift when a description includes the descriptor-value pair for that table. For $pred(a_0/2) - true$, the values for β on concepts A, B, C and D are 1.2, 11, 1.83 and 1.8 respectively. A rule containing the predicate a_0 will therefore have a much higher true positive count on a virtual contingency table for concept B than for A, C or D. This means an evaluation metric modified by the DFT for B will prefer such a rule more than the same metric modified by the other DFTs. We now look at how this affects the bias of a rule search.

	A	B	C	D
b	.17 .71 .05 .07	.05 .91 .02 .02	.08 .87 .02 .03	.15 .79 .03 .04
$\mathbf{u}(pred(a_0/2), true)$	.11 .09 .44 .37	.22 .02 .37 .39	.11 .06 .45 .38	.09 .05 .47 .38
$\mathbf{u}(pred(a_1/2), true)$	.16 .07 .40 .37	.25 .02 .35 .38	.24 .04 .35 .37	.16 .06 .40 .38
$\mathbf{u}(pred(a_2/2), true)$	.20 .07 .38 .36	.13 .07 .43 .37	.25 .04 .34 .37	.20 .04 .38 .38
$\mathbf{u}(pred(a_3/2), true)$	.25 .09 .33 .33	.14 .17 .37 .32	.12 .13 .40 .35	.20 .16 .33 .32

Fig. 4. The base tables and $pred(P/N)$ update tables for DFTs for concepts A, B, C and D. The update tables have been normalised for easier comparison.

Table 1 lists acceptable rules for task A, in the order in which they were tested, during a search using Aleph. The training (resp. test) set for the task consisted of 10 (resp. 100) examples with equal number labelled positive and negative. The columns titled "Train" and "Test" show the cov_E score of rules on the respective set. Unsurprisingly, rule *f* has the highest score (32.0) on the test data as it is one of the target rules for task A. On the training set however, rules *a*, *b*, *d*, *e*, *f* and *h* all share the highest score (3.0). The search policy employed by Aleph only replaces its current best rule if another one has a strictly greater score. This results in rule *a* being returned, a sub-optimal choice since its test set score is -8.0.

The last three columns in the table show the score given to the rules by coverage metrics modified using DFTs for tasks B,C and D, all using an M parameter set to 10 (equal to the training set size). When B and C are used as secondary tasks, the rule returned by the search is *d*. The correct rule, *f*, is returned when D is the secondary task but is assigned the lowest ranking by B's DFT. Since the ranking of rules in column D are closer to that in the Test column, we would expect a learner to perform better when using an evaluation metric modified by D and worse when modified by B.

4 Empirical Results

To test whether Deft can improve learning performance on small datasets it needs to be compared to a baseline learner in an *environment* of two or more related learning tasks. This section assesses Deft on a number tasks of varying size drawn from three different environments.

The first environment, used in Section 4.1, consists of the four concepts A, B, C and D introduced earlier. The results demonstrate that learning performance on tasks for A can be improved when using DFTs from concepts C and D. However, using a DFT from concept B is shown to harm generalisation accuracy.

Section 4.2 tests Deft on a chess movement environment that has been used by other researchers to test relational multitask learning systems [15,6,7]. The experiments on this domain compare Deft to the "Repeat Learning" system described in [7]. The results show that Deft can improve generalisation accuracy

Table 1. Comparison of scores given to acceptable rules for task A.

		Coverage		DEFT Coverage		
ID	Rule	Train	Test	B	C	D
a	`t(A) :- a0(A,2)`	3.0	-8.0	2.71	1.22	2.05
b	`t(A) :- a2(A,0)`	3.0	14.0	0.72	3.72	4.69
c	`t(A) :- a0(A,2), a1(A,2)`	1.0	-4.0	3.21	1.30	0.81
d	`t(A) :- a0(A,2), a2(A,0)`	3.0	2.0	3.34	3.74	3.99
e	`t(A) :- a0(A,2), a3(A,1)`	3.0	11.0	1.48	1.86	2.86
f	`t(A) :- a2(A,0), a3(A,1)`	3.0	32.0	-0.02	3.37	5.12
g	`t(A) :- a1(A,2), a2(A,0)`	1.0	9.0	1.47	3.29	2.54
h	`t(A) :- a0(A,2), a2(A,0), a3(A,1)`	3.0	10.0	2.77	3.14	3.60
i	`t(A) :- a0(A,2), a1(A,2), a2(A,0)`	1.0	2.0	1.88	1.78	1.32
j	`t(A) :- a0(A,2), a1(A,2), a3(A,1)`	1.0	4.0	1.67	1.05	1.09
k	`t(A) :- a1(A,2), a2(A,0), a3(A,1)`	1.0	15.0	0.78	1.72	2.07

across these tasks. Furthermore, a combination of DEFT and Repeat Learning is tested and shown to outperform both.

The final environment (Section 4.3) consists of the mutagenesis [16] and carcinogenesis [17] problems from molecular biology. While the results show no significant advantage in using DEFT across these tasks, they do suggest the usefulness of DEFT in this environment.

For reference, a summary of the systems' settings used in each environment is provided in Appendix A.

4.1 Example Environment

The experiment in this section tests whether DEFT can exploit the apparent similarities between concept A and concepts C and D to improve the base learner's generalisation accuracy on small datasets. Concept A was used to generate primary tasks with training sets of size $N = 4, 6, 8, 10, 14, 20, 30$. Twenty tasks were created for each N, each with an equal number of positive and negative examples. Four more tasks with 100 examples and balanced class labels, one for each of concept A, B, C and D, were also created. The task for concept A was used as a test set while the `make_DFT` procedure was applied to the others to create DFTs using the $pred(P/N)$ and $arg(P/N, I, Const)$ descriptors.

The learner was run on each primary training set E with four different evaluation metrics: the standard coverage metric, cov_E, and cov_E^S for $S \in \{\text{B,C,D}\}$. Each cov_E^S is a coverage metric modified by DEFT using the DFT for the secondary concept S and DEFT's M parameter set to the size of E. The performance for each evaluation metric on datasets of size N was quantified by averaging the test set accuracies using that metric on all the training sets of size N. The results are summarised in Fig. 5.

The results show that the improvement obtained when using DEFT concurs with our expectations in Section 3.5 regarding the relative similarity of concepts B, C and D to concept A. Using the most similar concept (D) as a secondary

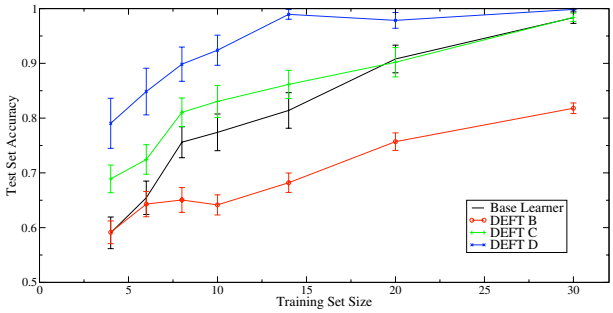


Fig. 5. Average generalisation accuracy on tasks for Concept A using the standard coverage metric and the coverage metric modified by DFTs created by DEFT from tasks for concepts B, C and D. Error bars show the sample standard deviation.

task results in large accuracy gains over the base learner while using the least similar (B) harms its learning performance.

4.2 Chess Environment

The chess movement tasks involve learning rules to describe the legal moves of King and Knight pieces on an empty board. These can be seen as similar concepts since both types of movement are short-range and have an eight-fold symmetry. Each example in a King or Knight task specifies the rank and file for the start and end positions of a move and whether or not it is legal for that piece. Background predicates `rdiff/3` and `fdiff/3` allow rules to determine differences between two ranks or two files.

The background and example sets used in this section are the same as those in [7] and consist of 140 training sets - 20 each of size 10, 20, 30, 40, 60, 80 and 160 - and each with balanced class labels. The supplied test sets, one for each piece, have the 64^2 possible chessboard movements classified according whether they are legal for the piece in question. These datasets were originally used to demonstrate the effectiveness of Repeat Learning (RL) as a bias learning method. The basic idea of RL (details are in [7]) is to invent predicates when learning on a secondary task and then use those predicates as background when learning on the primary task. On the King and Knight problems, the predicates invented by RL allow several file or rank differences (e.g., -1, 0 and 1) to be expressed in a single rule, thus the learner can express the target concept with fewer rules. This change of representation overcomes the small disjunct problem [18] in which rules with small extension do not have any representatives in a small example set.

The effect DEFT and RL have on a learner's bias are more or less independent: the former modifies its search bias while the latter weakens its language bias. A search using RL invented predicates can be guided by an evaluation function modified by DEFT. This combination was tested along with each method individually on primary tasks for the King movement problem. A Knight task

with 160 examples was used as a secondary task by RL to invent predicates. The same task was used by DEFT to construct a DFT using the $pred(P/N)$ and $arg(P/N, I, Const)$ descriptors. The base learner ALEPH was run four times on each primary task, once without any modifications, once using the invented predicates as background, once using an evaluation metric modified by the Knight DFT and once using both the invented predicates and modified metric. The performance of each approach was assessed using the same *balanced test accuracy* (BTA) measure as [7]. This is the mean of the true positive and true negative rate of a theory on the test set[5]. For each training size N, the BTA of each theory induced was averaged over the 20 datasets of size N. The results are presented in Table 2.

Table 2. Comparison of balanced test accuracy for the Base learner, DEFT, Repeat Learning (RL) and both DEFT and Repeat Learning (DEFT+RL) on the King movement problem. **Bold** entries are different from the Base entries at the 0.05 level of significance using a paired t-test. Entries for DEFT+RL with a R (resp. D) are significantly better than RL (resp. DEFT) alone.

	Training Size						
	10	20	30	40	60	80	160
Base	61.3 (1.1)	72.4 (1.4)	77.5 (1.0)	84.9 (1.1)	91.2 (1.3)	95.6 (1.0)	99.6 (0.2)
DEFT	**71.1 (1.4)**	**79.8 (1.6)**	**87.3 (0.9)**	**89.1 (1.1)**	**93.8 (1.0)**	96.3 (0.8)	99.3 (0.5)
RL	**66.3 (2.0)**	**77.8 (2.2)**	**83.7 (1.0)**	**92.0 (1.3)**	**95.5 (1.3)**	**98.9 (0.5)**	99.8 (0.2)
DEFT+RL	**72.3 (1.5)$^{\mathbf{R}}$**	**80.8 (1.7)**	**88.6 (0.9)$^{\mathbf{R}}$**	**90.5 (1.1)**	**96.5 (0.8)$^{\mathbf{D}}$**	**98.5 (0.5)$^{\mathbf{D}}$**	99.6 (0.2)

The results show that DEFT improves on the base learner to a greater degree than RL on very small datasets. To a lesser degree, this situation reverses when datasets become larger. The DFT used in these experiments has high β values (between 2 and 6) for descriptors $arg(P/3, 3, C)$ where P is `rdiff` or `fdiff` and C is $\pm 1, \pm 2$. Even when few negative examples are available, the modified evaluation metric prefers rules with those predicates whenever it does not harm their real true positive rate too much. This reduces the real false positive rate since the induced theories have several specific clauses rather then only a few over-general ones plus ground facts for the exceptions. On the other hand, the invented predicates used by RL are helpful when there are sufficient negative but too few positive examples as each rule using the extra predicates can cover what would require several small disjuncts without them. The two approaches therefore complement one another. RL compensates for missing positive examples when there are sufficient negative ones available, while DEFT compensates for missing negative examples. The accuracies for the combined approach reflect this.

[5] A preferable measure to standard accuracy since the positive to negative example ratio on the test set is 1:8.

4.3 Molecular Environment

This section reports the use of DEFT on a pair of benchmark ILP problems - mutagenesis [16] and carcinogenesis [17]. Both require the learner to predict cancer-related properties of organic molecules so it is reasonable to believe that one may act a useful bias for the other.

The background knowledge used by the problems to describe molecules can be partitioned into several groups concerning their atoms and bonds, chemical features, and three-dimensional structure. Only the atoms and bonds group is used here as its predicates (`atm/5`, `bond/4`, `gteq/2`, `lteq/2`, `=/2`) are common to both domains[6]. There are a total of 125 positive and 63 negative examples for the mutagenesis task and 182 positive and 148 negative for carcinogenesis. The carcinogenesis concept proved too difficult to learn using only this group as background (theories returned by the base learner and DEFT were no better than the majority class classifier) so we therefore focused on mutagenesis as the primary task. The complete mutagenesis and carcinogenesis datasets were used to construct two DFTs: Mut and $Carc$. Both types of descriptors, $pred(P/N)$ and $arg(P/N, I, Const)$ were used in the construction.[7]

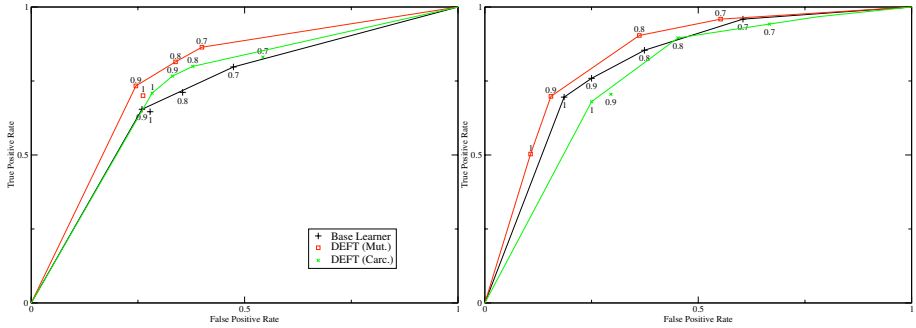
Ten-fold cross-validation was used to determine generalisation accuracy for this task. Each fold holds out 10% of the data while the other 90% is used for training. In order to test DEFT's performance on small datasets in this domain, the 90% sets were randomly sub-sampled once on each fold to create example sets with 24 positive and 12 negative examples - roughly 20% of the original dataset. On both the 90% and 20% datasets, the base learner's `minacc` parameter was varied over the values 0.7, 0.8, 0.9 and 1.0 to assess the performance of the learner over a range of classification costs. The value for M was set to the training set size, $M = 36$ on the 20% tasks and $M = 162$ on the 90% tasks. The resulting ROC curves [19] on the two dataset sizes are shown in Fig. 6 for an unmodified base learner as well as a Mut-modified and $Carc$-modified learner. The test accuracies for each point in the figure are given in Table 3.

The use of the Mut DFT on the mutagenesis tasks is, in a sense, "cheating" as it indirectly gives the learner access to extra mutagenesis examples it otherwise would not have. The results in these best-case scenarios are not intended as proof of DEFT's effectiveness, but rather to remove possible explanations of the poor performance of DEFT when using the $Carc$ DFT. If no improvement was seen when using the Mut DFT for mutagenesis (where the primary and secondary tasks are decidedly similar) then fault would lie with the DEFT algorithms or the choice of descriptors. However, the improvement was significant when using the Mut DFT while none was seen when using the $Carc$ DFT, warranting a closer look at which rules are preferred in each case.

The Mut DFT has high β values (2200, 7100, 47, 55) for rules that mention an an atom of certain types (195, 28, 49, 51). The main improvement in the generalisation accuracy on the 20% task is due to an increase of the true positive

[6] These are called the M0 and C0 predicate groups in [19].

[7] One detail about the arg descriptors is pertinent here: the $Const$ parameter cannot take on floating point values. Descriptors with specific floats match very few rules.



(a) 20% of Training Set (b) 90% of Training Set

Fig. 6. ROC curves showing the performance of the base learner and DEFT on the mutagenesis domain for two training set sizes. Two DEFT curves are shown per graph, one using the Mut DFT and the other the $Carc$ DFT .

rate. Like the chess tasks, this is due to the DFT-modified metric preferring many specific rules over a few general ones and several ground facts.

While there is also an improvement in the true positive rate when using the $Carc$-DFT, it comes as a trade-off for a higher false positive rate. This is because the $Carc$-modified evaluations also prefer rules over ground facts but the preferred rules generalise badly. The explanation is that the descriptors with high β values for the Mut-DFT either do not appear in the $Carc$-DFT (atom type = 195, 28) or are given indifferent scores (type = 49, 51 both have β around 1). Furthermore, some high β descriptors for $Carc$ (type = 94 : β= 6) have low scores (β= 0.5) on Mut. The few descriptors both DFTs prefer (type = 29, 52) are responsible for the slight gains on the 20% task but cannot overcome their differences which have a detrimental effect when 162 virtual carcinogenesis examples are used with the 90% task.

The conclusion to be drawn from these results is that, as suggested by the Mut DFT results, DEFT could improve learning from small datasets in this domain. However, carcinogenesis appears to have a very different set of high performing clauses and so was not useful as a secondary task for mutagenesis.[8]

5 Related Work

The role of evaluation metrics in rule learning and the relationships between them has been a topic of interest recently [21,22,12]. Adding values to entries in a contingency table is a general way of parameterising evaluation metrics of which the Laplace and m-estimates [23] are special cases. The DEFT method of

[8] This is consistent with another researcher's attempt to exploit the apparent similarity between the domains. [20] notes "that the data were about such different sets of chemicals that this was not really the case".

Table 3. Test accuracies on the mutagenesis tasks for the base learner and DEFT using the *Mut* and *Carc* DFTs. **Bold** entries differ from the Base figures at a significance level of 0.05 using a paired t-test.

train size	20%				90%			
minacc	0.7	0.8	0.9	1.0	0.7	0.8	0.9	1.0
Base	70.6 (3.0)	68.5 (3.3)	68.0 (4.0)	66.8 (4.1)	77.1 (2.4)	77.6 (2.1)	75.5 (2.8)	73.4 (3.1)
Mut	75.2 (2.3)	**77.2 (2.3)**	**74.9 (2.7)**	**73.8 (2.3)**	78.8 (2.4)	**81.4 (2.5)**	74.6 (2.9)	63.3 (3.0)
Carc	68.5 (3.4)	72.9 (2.5)	71.1 (3.5)	70.0 (3.7)	73.9 (1.6)	77.7 (2.1)	70.4 (3.1)	70.3 (2.7)

learning those parameters can be seen as a special case of Bayesian approaches to learning to learn [24,25]. DEFT's use of priors for rules is similar to those in positive-only learning [13,26] and LIME [27]. Those systems randomly generate unlabelled examples (as opposed to using labelled examples from secondary tasks) to estimate the size of a rule's extension. This helps rule evaluation when negative examples are scarce or unavailable.

Unlike DEFT, other attempts at multitask learning in ILP have not considered learning evaluation bias. The MFOCL system [6] is similar to the Repeat Learning system [7] described in Section 4.2. Its "concept sharing" approach reuses rules and parts of rules from concepts learned on secondary tasks in the same manner as the invented predicates in RL. Since MFOCL's base learner performs a greedy search, the new predicates allow it to avoid local minima. This is also the motivation for the CLUSE/XFOIL system [8] in which CLUSE uses a "contextual least-general generalisation" procedure on secondary tasks to learn relational clichés. These combinations of literals expand the actions available to the greedy search used by its FOIL variant.

Descriptors are similar to first-order features used by systems to transform relational problems into propositional ones (e.g., [28]). The main difference is that the latter is a transformation on examples, allowing a search of propositional rules, whereas descriptors act on first-order rules and are used to improve their evaluation. However, it would be possible to apply DEFT to the propositional rule learners used in such systems to help their rule evaluation on limited data.

6 Conclusions and Future Work

Evaluating rules when there is only a small amount of data is an intrinsically difficult problem. It is also an important one, especially when classified data is expensive or difficult to obtain. DEFT, the multitask approach presented here, demonstrates how the evaluation of rules can be modified so as to take into account the performance of similar rules on secondary tasks. Rule descriptions are introduced as a general way to define rule similarity. This forms the basis of a simple Bayesian technique to calculate classification priors for rules. These are used to improve estimates of classification probabilities from small datasets. Evaluation metrics that are functions of the improved estimates provide more reliable assessments of rule quality.

DEFT was tested empirically on three environments. The first, a toy environment, confirmed that the approach can improve learning performance when the primary and secondary tasks are "intuitively" similar. This was the case again on the chess movement environment. Furthermore, DEFT was successfully combined with a different, predicate invention-based approach that had previously been used on the same tasks. Results on the third environment were less conclusive but strongly suggested that description-based evaluation could with help learning in biological domains.

Future work on the theoretical front will include a better characterisation of similarity in terms of descriptions and descriptor tables and investigate links with existing work on representation-based metrics, especially those using MML/MDL. Improvements to the current implementation of DEFT will include adding more sophisticated descriptors and statistical pruning techniques to manage larger DFTs more efficiently. The impact of these new descriptors will need to be thoroughly empirically tested, as will DEFT's sensitivity to its M parameter and the secondary tasks used to create DFTs.

Acknowledgements. The author thanks Ashwin Srinivasan for his help with ALEPH and discussions about the molecular datasets. Thanks also to Claude Sammut for his feedback on this paper. Suggestions from the anonymous reviewers were also valuable.

References

1. Srinivasan, A., King, R.D.: Feature construction with inductive logic programming: a study of quantative predictions of biological activity aided by structural attributes. Data Mining and Knowledge Discovery **3** (1999) 37–57
2. Caruana, R.: Multitask learning. Machine Learning **28** (1997) 41–75
3. Mitchell, T.M.: The need for biases in learning generalizations. Technical Report CBM-TR-117, Rutgers University, New Brunswick, New Jersey (1980)
4. Nédellec, C., Rouveirol, C., Adé, H., Bergadano, F., Tausend, B.: Declarative bias in ILP. In: Advances in ILP. Volume 32 of Frontiers in AI and Applications. IOS Press (1996) 82–103
5. Fürnkranz, J.: Separate-and-conquer rule learning. Artificial Intelligence Review **13** (1999) 3–54
6. Datta, P., Kibler, D.F.: Concept sharing: A means to improve multi-concept learning. In: Proc. of the 10th ICML. (1993) 89–96
7. Khan, K., Muggleton, S., Parson, R.: Repeat learning using predicate invention. In: Proc. of the 8th ILP, Springer (1998) 165–174
8. Morin, J.: Learning Relational Clichés with Contextual Generalization. PhD thesis, School of Information Technology and Engineering, University of Ottawa, Canada (1999)
9. Srinivasan, A.: ALEPH: A learning engine for proposing hypotheses. Prolog code (2001) http:// www.comlab.ox.ac.uk/ oucl/ research/ areas/ machlearn/ Aleph/.
10. Silver, D.: Selective Transfer of Neural network Task Knowledge. PhD thesis, Graduate Program in Computer Science, University of Western Ontario, London, Ontario, Canada (2000)

11. Good, I.J.: The Estimation of Probabilities: An Essay on Modern Bayesian Methods. MIT Press (1965)
12. Fürnkranz, J., Flach, P.A.: An analysis of rule evaluation metrics. In: Proc. of the 19th ICML, AAAI Press (2003) 202–209
13. Muggleton, S.H.: Inverse entailment and progol. New Generation Computing **13** (1995) 245–286
14. Srinivasan, A.: A study of two sampling methods for analysing large datasets with ILP. Data Mining and Knowledge Discovery **3** (1999) 95–123
15. De Raedt, L., Bruynhooghe, M.: Interactive concept-learning and constructive induction by analogy. Machine Learning **8** (1992) 107–150
16. Srinivasan, A., Muggleton, S., King, R.D., Sternberg, M.J.E.: Mutagenesis: ILP experiments in a non-determinate biological domain. In: Proc. of the 4th ILP. (1994)
17. Srinivasan, A., King, R.D., Muggleton, S., Sternberg, M.J.E.: Carcinogenesis predictions using ILP. In: Proc. of the 7th ILP. (1997) 273–287
18. Holte, R.C., Acker, L.E., Porter, B.W.: Concept learning and the problem of small disjuncts. In: Proc. of the 11th IJCAI. (1989) 813–818
19. Srinivasan, A.: Extracting context-sensitive models in inductive logic programming. Machine Learning **44** (2001) 301–324
20. Srinivasan, A.: Personal communication. Email regarding chemical data (2002)
21. Lavrač, N., Flach, P., Zupan, B.: Rule evaluation measures: A unifying view. In: Proc. of the 9th ILP, Springer (1999) 174–185
22. Vilalta, R., Oblinger, D.: A quantification of distance-bias between evaluation metrics in classification. In: Proc. of the 17th ICML. (2000) 1087–1094
23. Cestnik, B.: Estimating probabilities: A crucial task in machine learning. In: Proc. of the 9th European Conference on AI, Pitman (1990) 147–149
24. Baxter, J.: A model of inductive bias learning. Journal of Artificial Intelligence Research **12** (2000) 149–198
25. Heskes, T.: Empirical bayes for learning to learn. In: Proc. of the 17th ICML, Morgan Kaufmann (2000) 367–374
26. Cussens, J.: Using prior probabilities and density estimation for relational classification. In: Proc. of the 8th ILP, Springer (1998) 106–115
27. McCreath, E., Sharma, A.: LIME: A system for learning relations. In: Proc. of the ALT-98. (1998) 336–374
28. Lavrač, N., Flach, P.A.: An extended transformation approach to inductive logic programming. ACM Trans. on Computational Logic (TOCL) **2** (2001) 458–494

A Experimental Settings

	clauselength	i	nodes	noise	minacc	sample_ex	sample_cl
Toy	4	2	200	0	-	40	100
Chess	6	3	200	0	-	40	100
Molecular	4	2	10000	-	0.67-1.0	100	50