A Noise Resistant Model Inference System

Eric McCreath and Mark Reid

Artificial Intelligence Group, School of Computer Science and Engineering
The University of New South Wales, Sydney NSW 2052, Australia
{ericm,mreid}@cse.unsw.edu.au

Abstract. Within the empirical ILP setting we propose a method of inducing definite programs from examples — even when those examples are incomplete and occasionally incorrect. This system, named NRMIS, is a top-down batch learner that can make use of intensional background knowledge and learn programs involving multiple target predicates. It consists of three components: a generalization of Shapiro's contradiction backtracing algorithm; a heuristic guided search of refinement graphs; and a LIME-like theory evaluator. Although similar in spirit to MIS, NRMIS avoids its dependence on an oracle while retaining the expressiveness of a hypothesis language that allows recursive clauses and function symbols. NRMIS is tested on domains involving noisy and sparse data. The results illustrate NRMIS's ability to induce accurate theories in all of these situations.

1 Introduction

An inductive logic programming (ILP) system can be loosely characterized as concept discovery tool that uses logic programs to describe its hypotheses. The use of logic programs as a hypothesis language offers several advantages over other choices of hypothesis representation such as clusters or decision trees. These advantages include the ability to describe a very rich class of concepts and a convenient way of providing systems with background knowledge about a domain. Also, hypotheses output as logic programs are generally easier for humans to understand and analyze.

There has been a recent trend in ILP to develop systems that can perform well in a variety of domains that are grounded in practical settings. Difficulties such as missing or inconsistent data are commonplace in these domains and have, to an extent, prevented the successful application of ILP techniques to real world problems.

The issue of inconsistent or *noisy* data has been addressed by systems such as FOIL [16], mFOIL [3,4], PROGOL [13] and LIME [11] to name but a few. A common characteristic of these systems, besides their robustness, is that their performance on training sets with missing or *sparse* data is fairly poor. On the other hand, systems such as MIS [19], LOPSTER [8], CRUSTACEAN [1], SKILIT [7], and FOIL-I [6] can learn from sparse data but do not handle noisy data at all. Systems that are robust to noisy training data tend to be systems that use

extensional (example-based) cover when determining the fitness of a hypothesis. Conversely, systems that use proof-based or *intensional* cover generally perform well on sparse data.

There has also been a growing interest in multiple predicate learning within ILP. A handful of systems (MIS[19], MPL[2], NMPL[5], and MULT_ICN[10]) can correctly induce programs with multiple target predicates but none can handle any amount of noise. All but MULT_ICN are intensional systems.

In this paper we propose an ILP system, NRMIS (pronounced "near-miss"), which modifies MIS so it can learn from noisy examples and without an oracle. NRMIS inherits from MIS, amongst other things, the ability to learn multiple target predicates simultaneously while improving on the efficiency of the search method of its predecessor. This can be seen as a step towards integrating noise handling techniques with the benefits of an intensional system.

Central to the NRMIS algorithm presented in Section 3 are the refinement graphs and markings described in Section 2. In Section 4 NRMIS is tested on several domains, covering the range of difficulties discussed earlier. A discussion of these results and future work is given in Section 5.

We use the following description of an ILP problem to set the scene and introduce some notation. Any concepts not thoroughly defined here can be found in [14].

Let L be a first-order logic derived from an alphabet which contains only finitely many predicate and function symbols. A unknown model $\mathcal{M}$ assigns a truth value to each ground fact in L. Ground facts of L are called *examples* and those which are true (resp. false) in $\mathcal{M}$ are called *positive* (resp. *negative*) examples. Given background knowledge $\mathcal{B}$ (a finite set of clauses true in $\mathcal{M}$), a set E^+ of positive examples, and a set E^- of negative examples we wish to find a hypothesis Σ (a finite set of clauses) that entails every positive example in E^+ and none of the negative examples in E^-. Such a hypothesis is called *correct*.

We will say a set of examples is *noisy* with respect to a *target* hypothesis Σ if Σ is not correct with respect to the examples. In this situation the aim is to find a hypothesis that has the same extension as the target hypothesis. The noise model we will use in this paper is outlined in Section 4.2.

The system described in this paper addresses the ILP problem in the *empirical setting* [2] which requires that: the entire example set be given in advance, no queries are made of the underlying model $\mathcal{M}$, and the initial hypothesis contains no definitions for the predicates being learned. Also, we will only consider the ILP problem for definite programs. This allows us to use SLD-refutation, denoted by $\vdash$, as the proof procedure to determine the intensional cover of a theory. In order to guarantee the termination of this procedure we have implemented it with a depth-bound which, in our experiments, is set large enough to avoid any problems.

2 Refinements and markings

The problem of finding a correct hypothesis can be seen as a search of the space of all definite clauses in L. This search space, we will call it L_{def}, can be structured through the use of a *refinement operator* [19].

Given a definite clause C and a refinement operator ρ the set $\rho(C)$ contains all of the most general specializations of C in L_{def}. The specialization order placed on the clauses in L_{def} is that of Plotkin's θ-subsumption [15]. The refinement operator and specialization order induce a *refinement graph* over the clauses in L_{def}. This graph has a directed edge from clause C to clause D if and only if $D \in \rho(C)$, i.e., when D is a *refinement* of C.
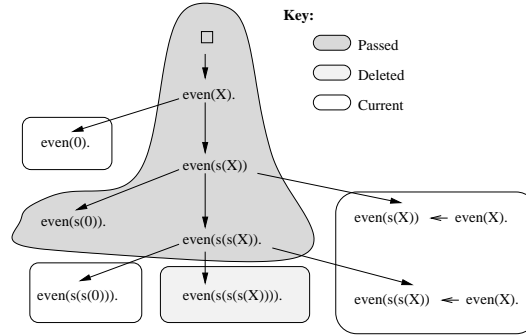

Fig. 1. A marking for L_{def}'s refinement graph

A *marking* [19] is a structure that is used intensively by NRMIS when searching for a correct hypothesis. It consists of three finite subsets of L_{def}: the *current* hypothesis M_{cur}, a set of *deleted* clauses M_{del}, and a set of clauses, M_{pass}, marked *passed*. The marking consisting of these sets will be denoted M. A diagrammatic example of a refinement graph and a marking is given in Figure 1. The alphabet for L in this case consists of the predicate *even/1*, the function *s/1* and the constant 0. The refinement operator used here is named ρ_2 in [18] and is based on a context-free transformation.

When the language L contains many predicates and function symbols the sets $\rho(C)$ can be quite large which causes the refinement graph for L to grow extremely quickly. In order to keep the search of this graph managable NRMIS makes use of user-provided mode and type information as well as the search heuristics outlined in Section 3.3.

3 The NRMIS algorithm

As the name suggests NRMIS is a modification of Shapiro's MIS [19]. Both are top-down intensional algorithms that form hypotheses by searching a space of

definite clauses using a refinement graph. Unlike MIS our system is not an incremental learner, that is, all the training examples are given to our algorithm in advance. Also, NRMIS does not require an oracle and can tolerate noise in the training examples it is given. These advantages are due to a generalization of Shapiro's contradiction backtracing algorithm and a theory evaluation heuristic similar to LIME's [11], respectively. These are detailed in Section 3.2 and Section 3.4.

An overview of NRMIS is given in Figure 2. Input to this algorithm consists of background knowledge $\mathcal{B}$ in the form of a definite program, a set of positive examples E^+ and a set of negative examples E^-. The marking M is initialized and specialization/generalization loop is entered. The following subsections detail the function of the subroutines `Decision`, `Generalize`, `Specialize`, and `Compress`.

<div style="border:1px solid">

`NRMIS(`$\mathcal{B}, E^+, E^-$`)`

$Q :=$ Empty priority queue
$M_{\mathrm{cur}} := \{\ \Box\ \}$
$M_{\mathrm{del}} := \{\ \ \}$
$M_{\mathrm{pass}} := \{\ \ \}$
repeat
  $E^-_{\mathrm{bad}} := \{e^- \in E^- \,|\, \mathcal{B} \cup M_{\mathrm{cur}} \vdash e^-\}$
  $E^+_{\mathrm{bad}} := E^+ - \{e^+ \in E^+ \,|\, \mathcal{B} \cup M_{\mathrm{cur}} \vdash e^+\}$
  case `Decision(`$E^-_{\mathrm{bad}}, E^+_{\mathrm{bad}}$`) of**
    specialize: `Specialize(`M, E^-_{bad}`)`
    generalize: `Generalize(`M, E^+_{bad}`)`
  `insert(Compress(`M_{cur}`)`$, Q)$
until $E^+_{\mathrm{bad}} \cup E^-_{\mathrm{bad}} = \emptyset$
output head of Q

</div>

Fig. 2. The `NRMIS` Algorithm

<div style="border:1px solid">

`NRBackTrace(`e^-, b`)`

$R :=$ a proof of e^-
repeat
  $(G, A, C) :=$ `last(`R`)`
  case Query truth value of A **of**
    true: $\sigma := 0$, $\tau := 1$
    false: $\sigma := 1$, $\tau := 0$
    unknown: $\sigma := \frac{1}{2}$, $\tau := \frac{1}{2}$
  `blame(`C, b, σ`)`
  $b := \tau b$
  $R :=$ `init(`R`)`
until (R is empty) $\vee$ ($b = 0$)

</div>

Fig. 3. `NRBackTrace`

Ideally, the NRMIS algorithm terminates when a correct hypothesis is found. This is not always possible when noise is present in the training examples. To overcome this the present implementation of NRMIS places an upper bound on the number of times the body of the main loop can be executed. If this upper bound is reached the best hypothesis in the priority queue Q is taken to be the system's final hypothesis. A hypothesis' position in Q is determined by a ranking given to it by `Compress`.

3.1 Decision

In Shapiro's MIS a generalization and specialization step is performed in every iteration of its main loop. When noise is present in the data this approach is too

coarse and frequently overlooks good candidate hypotheses. By separating these two steps NRMIS implements a finer search of the refinement graph.

Each repetition of the main loop in Figure 2 calculates E_{bad}^-, the set of negative examples that are implied by M_{cur}, and E_{bad}^+, the set of positive examples not implied by M_{cur}. The routine `Decision` guides the search for a correct hypothesis at a high level. It is a heuristic that simply aims to minimize the proportion of bad positive or bad negative examples. If the proportion $|E_{\text{bad}}^-|/|E^-|$ is larger than $|E_{\text{bad}}^+|/|E^+|$ a specialization is performed this loop, otherwise a generalization takes place[1].

3.2 `Specialize`

The procedure `Specialize` is used to reduce the number of negative examples covered by M_{cur}. It modifies the marking M by moving clauses from M_{cur} to M_{del}. The clauses to be moved depend on what examples appear in E_{bad}^-.

`Specialize` relies heavily on a modified version of Shapiro's *contradiction backtracing algorithm* [19]. We will refer to the original as `BackTrace` and the modification `NRBackTrace`. The main difference between them is the former requires access to an oracle – in the form of an enumeration of the underlying model or a user answering queries – whereas `NRBackTrace` does not.

If a negative example e^- can be covered by M_{cur} then there must be a SLD-refutation of e^- using clauses from the background knowledge and M_{cur}. We will represent an SLD-refutation of e^- as a sequence of resolution steps, $R_i = (G_i, A_i, C_i)$ for $i = 1 \ldots n$. Each G_i is a goal (a definite clause with no head literal), A_i is an atom of G_i and C_i is a clause from $M_{\text{cur}} \cup \mathcal{B}$ such that its head unifies with A_i. For $i = 1 \ldots n-1$, G_{i+1} is the resolvent of G_i and C_i on A_i. G_1 is the goal $\leftarrow e^-$ and G_n resolves with C_n to form the empty clause $\square$.

If $R = R_1 \ldots R_n$ is a proof then we denote by $\texttt{last}(R)$ the final resolution step R_n and the initial part of the proof, $R_1 \ldots R_{n-1}$, by $\texttt{init}(R)$. We illustrate this somewhat cumbersome definition with an example.

Example 1. Consider the following definite program Σ for the ternary predicates representing addition, *add/3*, and multiplication, *mult/3*, in terms of the successor function *s/1*:

μ_1: $mult(A,B,B).$ $\qquad\qquad\qquad\qquad$ α_1: $add(A,0,A).$

μ_2: $mult(A,s(B),C)\leftarrow mult(A,B,Z),add(A,Z,C).$ α_2: $add(A,s(B),s(C))\leftarrow add(A,B,C).$

μ_3: $mult(s(A),B,C)\leftarrow mult(A,B,Z),add(B,Z,C).$

This program is overgeneral due to the clause μ_1, hence we have an SLD-refutation of $mult(1,2,3)$ as seen in Figure 4. Each resolution in the sequence is represented as $\boxed{G_i} \overset{C_i}{\Longrightarrow}$ and each A_i is underlined in G_i.

We now detail our version of the backtracing algorithm, `NRBackTrace`, as shown in Figure 3. It is used to allocate a given amount of "blame", b, amongst

---

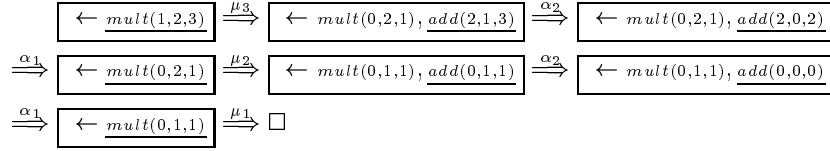[1] $|\cdot|$ is used to denote the cardinality of a set.

$$\boxed{\leftarrow mult(1,2,3)} \overset{\mu_3}{\Longrightarrow} \boxed{\leftarrow mult(0,2,1), \underline{add(2,1,3)}} \overset{\alpha_2}{\Longrightarrow} \boxed{\leftarrow mult(0,2,1), \underline{add(2,0,2)}}$$

$$\overset{\alpha_1}{\Longrightarrow} \boxed{\leftarrow \underline{mult(0,2,1)}} \overset{\mu_2}{\Longrightarrow} \boxed{\leftarrow mult(0,1,1), \underline{add(0,1,1)}} \overset{\alpha_2}{\Longrightarrow} \boxed{\leftarrow mult(0,1,1), \underline{add(0,0,0)}}$$

$$\overset{\alpha_1}{\Longrightarrow} \boxed{\leftarrow \underline{mult(0,1,1)}} \overset{\mu_1}{\Longrightarrow} \square$$

Fig. 4. An SLD-resolution of $mult(1,2,3)$ from Σ (see Example 1)

the clauses used in the proof of a negative example e^-. A proof, $R = R_1 \ldots R_n$, of e^- is found and the **repeat** loop is entered. Here the last resolution step, (G, A, C), is examined. If A can be shown to be false in the underlying model (via background knowledge or examples) then C must be false in the model as its head is false while its body is true (the rest of the steps in the proof show that each of C's body literals is true). C is therefore given all of the blame currently available by a call to $\texttt{blame}(C, b, \sigma)$ which adds $\sigma \times b$ units of blame to the clause C. Thus, in the case when A is false σ is set to 1.

If A is true the blame is passed back to the resolution step that led to G by setting τ to 1 and the process continues. Finally, if there is conflicting evidence for A's truth value or it is simply not known, half the blame is given to C and the rest is propagated back through the proof. An example of this process can be found in Example 2.

Note that if the truth values of every atom in the proof are known then $\texttt{NRBackTrace}$ will allocate all the blame to exactly one clause. In this sense $\texttt{NRBackTrace}$ can be seen to generalize Shapiro's $\texttt{BackTrace}$ algorithm to an oracle-free setting.

Example 2. Suppose we have the following sets of (noisy) positive and negative examples,

$$E^+ = \{ \ mult(0,2,1), add(0,0,0), add(2,0,2) \ \},$$
$$E^- = \{ \ mult(0,2,1), mult(1,2,3)) \ \},$$

and our hypothesis is the theory Σ given in Example 1. The result of calling the procedure $\texttt{NRBackTrace}$ with inputs $b = 1$ and $e^- = mult(1,2,3)$ is given in the table below.

Clause	μ_1	μ_2	μ_3	α_1	α_2
Blame	$\frac{1}{2}$	$\frac{1}{8}$	$\frac{1}{16}$	0	$\frac{5}{16}$

A description of the procedure $\texttt{Specialize}$ can now be given. For each $e^- \in E^-_{\text{bad}}$ a call to $\texttt{NRBackTrace}$ is made and blame is accumulated between calls (i.e., the blame is summed over every proof of a negative example a clause is involved in). At the end of this the most "guilty" clauses are removed from M_{cur} and placed in M_{del}. The most guilty clauses will be in some way responsible for

the hypothesis covering negative examples, therefore their removal from M_{cur} will result in a specialization.

A brief discussion of the time complexity of these algorithms is instructive at this point. The most time consuming part of the `Specialize` procedure is determining if negative examples are covered by its current hypothesis. As NR-MIS uses intensional cover, if a negative example is not covered by a hypothesis H, every possible depth-bounded SLD-refutation must be tried before returning a negative result (this is a version of negation by finite failure [14]). When H contains many interrelated and complicated clauses this search can be very inefficient. Care is taken when generalizing hypotheses to minimize this problem through the use of mode and type information as well as a bias for the addition of simple clauses over more complicated ones.

Once a proof of a negative example is found the `NRBackTrace` procedure's time complexity is roughly linear in the number of resolution steps in the proof. For each resolution step an additional expense is incurred when determining whether or not the atom resolved upon is true or false. This expense is dependent on the complexity of the background knowledge and can be reduced somewhat through standard dynamic programming techniques.

3.3 `Generalize`

The `Generalize` procedure used in the `NRMIS` algorithm (Figure 2) is much like that found in MIS. When there are positive examples not covered by M_{cur} clauses need to be added to the hypothesis. Clauses from M_{del} are moved to M_{pass} and the refinement operator ρ is applied to them. New clauses are chosen from these refinement sets and added to M_{cur}.

As the refinement graph can grow exponentially in the worst case it is imperative to keep the overall search efficient. A heuristic is therefore used to decide in which order the clauses in M_{del} should be refined. Given a way of measuring the "size" and "utility" of a clause preference is given to smaller and more useful clauses. We use the following definition for the size of a clause: The size of a clause is equal to the number of symbols, including punctuation, that appear in a clause minus the number of distinct variables (cf. [17]). A useful property of this size measure is that there are only finitely many clauses of any given size.

Like the notion of guilt used by `Specialize`, the utility of a clause is based on its involvement in the proof of examples. Each deleted clause C has a set $\text{covers}_H^+(C)$ of positive examples it helped cover when it was part of the hypothesis H. The utility of C is then defined to be the number of examples the sets $\text{covers}_H^+(C)$ and E_{bad}^+ have in common. Thus, a clause is considered useful if, in a past hypothesis H, it was used to cover several positive examples that are not covered by the current hypothesis M_{cur}.

Once a clause $C \in M_{\text{del}}$ is chosen for refinement all the clauses in $\rho(C) - M_{\text{pass}}$ are added to M_{cur}. Clauses in M_{pass} are not considered for addition to the hypothesis as they have already been considered and deleted. Clause C is moved from M_{del} to M_{pass} before the main loop of the NRMIS algorithm is executed again.

In MIS smaller clauses are also refined before larger ones but there is no method of choosing one clause over another should a tie occur. Instead, MIS refines all the smallest clauses and adds the resulting clauses to the overspecific hypothesis. By only adding the refinements of a single clause each time `Generalize` is called our system implements a much finer and more directed search for new hypotheses. As these hypotheses tend to be smaller, deciding what examples a hypothesis covers is usually less complicated in NRMIS.

3.4 `Compress`

Finally, the procedure `Compress` plays an important role in making NRMIS noise resistant and reducing the redundancy of the programs it outputs.

When a percentage of the examples given to a learning system are misclassified there is a tendency for the system to output hypotheses that explain these noisy examples. A common feature of these overzealous hypotheses is their large size. Striking a balance between accuracy and size is therefore a reasonable way of assessing the quality, $Q(H)$, of a hypothesis H. This philosophy is embodied in the *Q-heuristic* used in LIME [11]. `Compress` uses this heuristic to find an accurate and concise hypothesis $M_{\text{best}} \subset M_{\text{cur}}$. This is done using the following greedy strategy: An initial pruning of M_{cur} takes place in which only clauses that are involved in proofs of positive examples are kept. This results in a set $H_0 = \{\ C_1, \dots, C_n\ \} \subseteq M_{\text{cur}}$. For each $i = 1 \dots n$ we let $H_i = H_0 - \{\ C_i\ \}$. If $Q(H_0) \geq Q(H_i)$ for all i then `Compress` returns H_0 as the best hypothesis. Otherwise, the H_i with the highest quality is pruned and made the new H_0 and the procedure repeats. As this is a greedy search only $O(n^2)$ subsets of the original n clause H_0 are considered before the procedure terminates.

It is important to note that `Compress` does not actually modify the marking M in any way, rather, it outputs a compressed version of M_{cur} along with a number indicating its quality. It is this quality estimate which determines the hypothesis' position in the priority queue used in Figure 2.

4 Experimental Results

The focus of our experimental results is to demonstrate NRMIS's ability to correctly identify relations from training sets that are either incomplete or noisy. For these experiments we used an implementation of NRMIS written in the functional language Haskell which was generally an order of magnitude slower than the other systems.

4.1 Sparse data

In [6] a series of experiments are conducted that compare the performance of FOIL, PROGOL and FOIL-I on sparse or incomplete training sets. Table 4.1 shows how NRMIS and SKILIT [7] compare to these systems when tested on the same domains.

The domains used were *member/2*, *length/2*, *last/2* and *nth/3*. For each domain a complete initial portion of examples is generated. For example, in the *member/2* domain all possible ground facts involving the constants a,b and c and lists of length at most three were used. A training set drawn from this initial portion is said to have density 80% if the number of positive examples in the training set makes up 80% of the positive examples of the initial portion and the number of negative examples in the training set makes up 80% of the negative examples in the initial portion. For each density fifty random training sets are generated and each system's output on these training sets is checked for its correctness. The number of training sets the system correctly induced a hypothesis on is given in Table 4.1. [2]

Table 1. Comparing the performance of Nrmis, Foil-i, Foil, Skilit and Progol when learning from sparse data in various domains. The results for all the systems apart from Nrmis and Skilit are taken from [6]

Density	Correct theories (out of 50)					Density	Correct theories (out of 50)			
	Nrmis	Skilit	Foil-i	Foil	Progol		Nrmis	Foil-i	Foil	Progol
member						**length**				
80%	50	50	50	41	26	80%	41	38	38	39
50%	50	50	50	36	20	50%	29	18	18	13
30%	50	46	50	16	5	30%	22	6	4	0
20%	50	44	49	8	2	20%	17	4	2	0
10%	41	38	38	3	0	10%	0	0	0	0
7%	31	23	22	0	0					
last						**nth**				
80%	50	50	50	45	21	80%	50	50	0	43
50%	50	44	44	24	6	50%	47	49	6	43
30%	50	33	33	25	0	30%	28	46	5	19
20%	49	29	23	13	0	20%	33	27	0	0
10%	28	7	2	2	0	10%	13	1	0	0

We were unable to obtain sensible results from Skilit on the *length/2* and *nth/3* domains hence these results have been omitted from the table. This is most probably due to problems with our configuration of Skilit rather than a shortcoming of the system itself.

Unlike Foil-i and Skilit, Nrmis was not specifically developed to induce logic programs from sparse examples. Nevertheless, its performance on these training sets compares favourably against other two systems. This is a testament to the generality of the top-down refinement graph search used in Nrmis.

---

[2] For more information these experiments and some example training sets the reader is referred to [6] and http://www-itolab.ics.nitech.ac.jp/research/ilp/foili.html

4.2 Noisy data

NRMIS's noise handling ability is demonstrated when it is required to learn *add/3*, the addition relation. Its target concept can be found in Example 1.

Noise-free positive examples are generated by randomly choosing an instance of *add/3* and checking to see if it is true with respect to the target concept. If so, it is taken as a positive example. If not, another random instance is generated and this process continues until a positive example is found. An analagous process is used to generate noise-free negative examples. A noise rate of $\nu \in [0, 1]$ means, with probability ν, an instance of *add/3* will be drawn randomly and classified as positive (negative) without consulting the target concept.
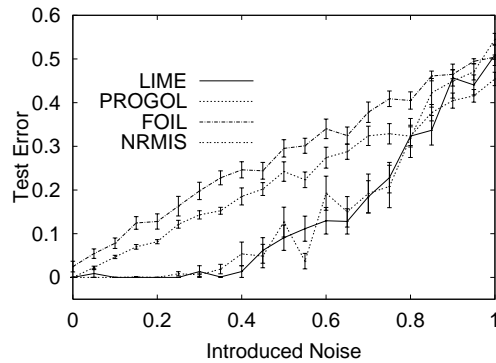

Fig. 5. Predictive Error vs Noise on the addition domain

Figure 5 compares the predictive error vs. noise rate curves for NRMIS, LIME[11, 12], PROGOL[13], and FOIL[16] [3]. For each noise rate, each of the systems are given a training set of 200 positive and 200 negative randomly generated, noisy examples. The predictive error for each hypothesis is approximated by averaging the proportion of misclassified positive and negative examples. These examples are drawn from a noise-free set of all possible instantiations of *add/3* with entries no greater than six (28 positive and 315 negative examples). Twenty of these trials are performed at each noise rate and the predictive error shown in the graph is averaged over these trials. As can be seen NRMIS performed similarly to LIME which also uses a theory evaluator based on the Q-heuristic.

5 Discussion and Future Work

We have presented in this paper an ILP system called NRMIS which is both noise-resistant and based on an intensional notion of cover. ILP systems in the

---

[3] We are using CProgol4.4 with the intensional cover flag set and Foil6.4

past have only ever met one of these two criteria and so have trouble with one or more of: multiple predicate learning, learning from sparse data, and learning from noisy data. The experiments outlined in this paper show Nrmis to be capable of performing well on the last two of these. In addition, we have been able to get Nrmis to correctly induce programs for the mutually recursive predicates *male-ancestor/2* and *female-ancestor/2* (as described in [2]) as well as *odd/1* and *even/1*.

The success of Nrmis is due to judicious use of the Q-heuristic and a modification of Shapiro's contradiction backtracing algorithm which removes the need for an oracle. Combined with the elegance of Mis's method of searching refinement graphs we have produced a system that uses a rich language to represent its hypotheses – recursive logic programs involving function symbols. This expressiveness comes with one major drawback, however, and that is the inefficiency of the search which is particularly apparent when the language consists of a large number of predicates and function symbols. Some progress has been made towards taming this problem both here and in other systems. We have managed to obtain large improvements in speed over a simple Mis-like refinement graph search while keeping the search complete. To get further improvement it may be necessary to consider greedy, incomplete strategies.

We have done some preliminary tests of Nrmis on larger domains such as document understanding [9] with some promising, though inconclusive, results. The biggest hurdle here is that the domain consists of over fifty predicates which means the refinement graph for this domain gets large very quickly. Further compounding the problem is the large number of examples that must be considered. We believe that, in principle, the Nrmis algorithm is capable of inducing an accurate hypothesis in this domain but our current implementation needs some optimizing before this will happen. This is our main focus for the near future.

Other ongoing work includes looking at formalisms that can provide theoretical basis for our modified backtrace algorithm, especially within a multiple predicate learning framework.

6   Acknowledgements

References

1. D. Aha, S. Lapointe, C. Ling, and S. Matwin. Learning singly recursive relation from small datasets. In F. Bergadano, L. DeRaedt, S. Matwin, and S. Muggleton, editors, *Proc. of the IJCAI-93 Workshop on Inductive Logic Programming*, pages 47–58, Chambery, France, 1993.

2. L. De Raedt, N. Lavrač, and S. Džeroski. Multiple predicate learning. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*. Morgan Kaufmann, San Mateo, CA, 1993.

3. S Džeroski. Handling noise in inductive logic programming. Master's thesis, Faculty of Electrical Engineering and Computer Science, University of Ljubljana, Slovenia, 1991.

4. S. Džeroski and I. Bratko. Handling noise in inductive logic programming. In *Proceedings of the Second International Workshop on Inductive Logic Programming*, 1992. Tokyo, Japan. ICOT TM-1182.

5. L. Fogel and G. Zaverucha. Normal programs and multiple predicate learning. In David Page, editor, *Proceedings of the 8th International Conference on Inductive Logic Programming*, Wisconsin,USA, July 1998.

6. N. Inuzuka et al. Top-down induction of logic programs from incomplete samples. In *Proceedings of the Sixth International Inductive Logic Programming Workshop*. Springer, 1996.

7. A. Jorge and P. Brazdil. Architecture for iterative learning of recursive definitions. In L. De Raedt, editor, *Advances in Inductive Logic Programming*. IOS Press, 1996.

8. S. Lapointe and S. Matwin. Sub-unification : A tool for efficient induction of recursive programs. In *Proceedings of the Ninth International Conference on Machine Learning*, pages 273–281, Aberdeen, Scotland, 1992. Morgan Kaufmann.

9. D. Malerba. Document understanding: A machine learning approach. Technical report, Esprit Project 5203 INTREPID, 1993.

10. Lionel Martin and Christel Vrain. MULTI_ICN: an empirical multiple predicate learner. In Luc De Raedt, editor, *Proceedings of the Fifth International Workshop on Inductive Logic Programming*, pages 129–144, 1995.

11. E. McCreath and A. Sharma. ILP with noise and fixed example size: A Bayesian approach. In *Fifteenth International Joint Conference on Artificial Intelligence*, volume 2, pages 1310–1315, 1997.

12. E. McCreath and A. Sharma. Lime: A system for learning relations. In *The 9th International Workshop on Algorithmic Learning Theory*. Springer-Verlag, October 1998.

13. Stephen Muggleton. Inverse entailment and progol. *New Generation Computing Journal*, 13, May 1995.

14. S.H. Nienhuys-Cheng and R. de Wolf. *Foundations of Inductive Logic Programming*. LNAI Tutorial 1228. Springer-Verlag, 1997.

15. G. Plotkin. A note on inductive generalization. *Machine Intelligence*, 5:153–163, 1970.

16. J.R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5(3):239–266, 1990.

17. John C. Reynolds. Transformational systems and the algebraic structure of atomic formulas. *Machine Intelligence*, 5:135–151, 1970.

18. E. Shapiro. Inductive inference of theories from facts. Technical Report 192, Computer Science Department, Yale University, 1981.

19. E. Shapiro. *Algorithmic Program Debugging*. MIT Press, 1983.