Programming Research Group

NOTE ON THE USE OF STATISTICAL PROCEDURES AS BACKGROUND PREDICATES IN ILP

Ashwin Srinivasan

PRG-RR-03-09

Oxford University Computing Laboratory
Wolfson Building, Parks Road, Oxford OX1 3QD

Abstract

Procedures, broadly under the umbrella of statistical modelling, construct robust and powerful models from sample data. Examples include parametric techniques for regression, and non-parametric ones like classification and regression trees. The techniques are essentially propositional, and a first-order model constructor embodied by an Inductive Logic Programming (ILP) system should, in theory, be able to utilise them by simply providing appropriate (background) predicates. By "utilise", we mean here constructing an appropriate statistical model (estimating relevant parameters) and using it as part of a first-order hypothesis (for prediction). An example is the incorporation of a regression equation as a literal in a hypothesised clause. Nevertheless, the representation adopted by most ILP systems (logic programs) results in some special difficulties in both parameter estimation and prediction. This report presents the principal difficulties and some solutions.

1 Introduction

Given domain-specific background knowledge and data in the form of examples, an Inductive Logic Programming (ILP) system extracts models in the data-analytic sense—that is, descriptions of real or potential data. A long and honourable tradition of model-construction from sample data exists within statistics; broadly speaking, some of the techniques widely used within the field of machine learning (like decision trees, "naive" Bayesian classifiers etc.) can be thought of as being subsumed under the umbrella of statistical modelling. We will simply use the phrase "statistical modelling" to mean constructing a model from sample data are in the form of a table of values for specific variables (sometimes called attributes or features). This tabular representation of data is essentially propositional, and is thus a special case of the first-order logic representation used by an ILP system. In theory, therefore, any form of statistical modelling could be achieved from first principles by an ILP program. In practice, this has proved to be extremely difficult (due to time complexity), and upon reflection, is an unnecessary requirement. Since well-established methods to do this already exist, an ILP program should be able to benefit by simply incorporating them as background knowledge. This should allow construction of rules of the following type for a drug-design problem of predicting binding affinity of a small molecule ("ligand") to a larger one ("target") [4]:

```
The affinity of ligand M is A if:
      M has a hydrogen donor at position P1 and
      M has a hydrogen acceptor at position P2 and
      M has a hydrogen acceptor at position P3 and
      the distance between P1 and P2 is D1 and
      the distance between P1 and P3 is D2 and
      the distance between P2 and P3 is D3 and
      A = 0.8 + 2.3 D1 - 5.1 D2 + 3.8 D3.
```

Here, the equation that calculates affinity using Euclidean distances is obtained from a standard linear regression procedure. The parameter estimates (0.8, 2.3, -5.1, 3.8) are to be obtained during hypothesis construction, and the rule is intended for predicting the affinity of potential ligands. The following questions then arise:

Q1. Ligand M may have several hydrogen donors and acceptors, at different positions, resulting in different sets of values for D1, D2 and D3 (the logic programmer will recognise this as different substitution-sets arising from a non-determinate definition for the donor and acceptor predicates). However, the domain dictates that only one of these sets is relevant for parameter estimation. Which one?

Q2. The same problem arises during the use of the rule for prediction. Each set of values for D1, D2 and D3 results in a prediction for A. Which of these values should be returned[1]?

These difficulties are unlike those confronted by a statistician during normal discourse. There it is common, due to random variation, to obtain *different affinity values* for repeated measurements of the *same* values for D1, D2 and D3. Machine learning researchers will recognise the questions here as consequences of a multiple-instance representation [2].

This report presents procedures to address each of these questions. For illustrative purposes, we will use the case of using linear regression as a background predicate, although the procedures proposed are not restricted to that particular modelling technique. In particular we will consider learning the following rule using an ILP system (no chemical significance should be attached to this rule):

```
The affinity of molecule M is Y if:
      M has a hydrogen donor at position X and
      Y = m X + c.
```

Here m and c are parameters to be estimated. Suppose data available are as follows: (1) affinity values of 5 different molecules; and (2) records of hydrogen donor locations on each molecule (a molecule can have more than one donor). The data are tabulated in Fig. 1. Logically speaking, the rule above states that the affinity of the molecule is a linear function of one of the hydrogen donor positions (without specifying which one: the logician will recognise this as a consequence of the variable X being existentially quantified within the rule body).

Since for each N, it is not apparent which of the X values are to be paired with the corresponding Y, a combinatorial problem arises[2]. For this simple problem, the combinatorics are manageable: there are only $3 \times 1 \times 2 \times 1 \times 2 = 12$ different tables containing exactly 1 entry for each of the 5 values of X. Two such tables are shown in Fig 2.

Parameter estimates can now be obtained using each such table. The estimates returned are those that result in the best fitting model. For the data in Fig 2, the best

[1] We could consider using some representative set of values for distances (for example, the average) for both parameter estimation and prediction. However, what this representative should be may not be apparent and in any case, will require a different rule to the one used in this example.

[2] Some theoretical results known for multiple-instance learning can be found in [3].

Mol (M)	Aff. (Y)	Donor (X)
		25
1	50	33
		29
2	100	50
3	150	73
		75
4	200	90
5	250	120
		110

Figure 1: "Training" data for the problem of predicting affinity Y using hydrogen donor location X. For a given value of N, it is known that the value of Y depends on one of X values—but it is not known which one.

Mol (N)	Aff. (Y)	Donor (X)
1	50	25
2	100	50
3	150	73
4	200	90
5	250	120

Mol (N)	Aff. (Y)	Donor (X)
1	50	33
2	100	50
3	150	75
4	200	90
5	250	110

Figure 2: Example tables obtained by combinatorial enumeration of the data in Fig. 1. The table on the left yields the model Y = 2.161 X - 4.739, with sample correlation coefficient 0.997. The one on the right yields Y = 2.565 X - 33.677, with sample correlation coefficient 0.998.

fitting model (Y = 2.572 X - 33.155) is obtained with the table in Fig 3. The resulting rule is therefore:

```
The affinity of molecule M is Y if:
      M has a hydrogen donor at position X and
      Y = 2.572 X - 33.155.
```

Section 3 describes procedures that equips an ILP system to reach such a model (or at least, an approximation to it) given the data in Fig 1.

Now consider using this rule on the data shown in Fig. 4. It is evident that simply executing the rule above will yield two predictions: Y = 188.037 (using X = 86) and Y = 203.469 (using X = 92). Which of these predictions should be returned is the problem posed by (Q2). Strictly speaking, there is no mechanism of enforcing such a selection within the first-order framework used by most ILP systems (although it is presumably

Mol (N)	Aff. (Y)	Donor (X)
1	50	33
2	100	50
3	150	73
4	200	90
5	250	110

Figure 3: Table that results in the best fitting linear model relating Y and X values. The best fitting model is Y = 2.572 X - 33.155, with sample correlation coefficient 0.999.

straightforward within higher-order systems proposed by John Lloyd and colleagues). In Section 4, we examine some options which may be unsettling for the ILP-purist.

Mol (N)	Aff. (Y)	Donor (X)
6	?	86
		92

Figure 4: "Test" data. The task is to predict the affinity of molecule 6 given its hydrogen donor locations.

2 Terminology

It is convenient at this point to introduce some terminology from the machine learning literature on multiple instance learning. The following statement of the multi-instance learning task is largely from [8]. Data consists of a set of n bags ($1 \leq n < \infty$). The i^{th} bag consists of m_i instances ($1 \leq m_i < \infty$) and a label y_i (which may be nominal or real-valued). Instance j of bag i is described by a d-dimensional vector of values ($1 \leq d < \infty$). The task is to construct a model predicting the y values. For each bag i, it can be assumed that one of the m_i instances is sufficient for predicting y_i.

It should be evident that the standard statistical task of predicting a dependent variable, given values for independent variables is a special case of the multi-instance task as presented here (with one instance per bag). The formulation here is slightly different to the original description of multi-instance modelling in [2], which only deals with binary classification problems (that is, the y_i take one of two values: "positive" or "negative"). There, bags are treated asymetrically. If i is a "positive" bag, it is assumed that one of the m_i instances is sufficient for predicting y_i. If i is a "negative" bag, it is assumed that *all* the m_i instances are *necessary* for predicting y_i. We have found this distinction to be unnecessary here as the same effect is achieved by using m_i different "negative" bags, each with a single instance.

In Fig. 1 therefore, there are 5 bags (one for each molecule). Bag 1, with y value 50, has 3 instances; Bag 2, with y value 100, has 1 instance and so on.

3 Parameter Estimation

Parameter estimation requires the following:

1. All values of relevant variables required by the modelling procedure are to be collected (in the example introduced earlier, this corresponds to all the values of N, Y and X in Fig. 1).

2. A search for the best parameters is then to be conducted using subsets of values collected in Step 1 (each such subset results in a table of the type in Fig. 2).

An ILP system capable of "lazy evaluation" (as described in [10]) is sufficiently powerful to allow this form of parameter estimation. For a predicate specified as being lazily evaluated, certain arguments are annotated as being functionally dependent on others. We can loosely term the former "output" and the latter "input" arguments. The actual values of the output arguments to be used during normal execution of the literal are computed as follows:

(a) The ILP system collects all possible values of the input arguments.

(b) The ILP system assumes that the background knowledge contains a definition specifying the computation of the output arguments as a function of the values collected above. This definition is then used to compute the values of output arguments.

Without entering into implementation details, it suffices to state here that (a) is sufficient to achieve the requirement of Step 1 above; and the definition in (b) can be an implementation of the search procedure required in Step 2.

This still leaves open the actual search procedure to be used. An exhaustive strategy—as was used in the simple example in Section 1—although optimal, may not be tractable. An efficient, but approximate, procedure modelled on the Expectation-Maximization (EM) algorithm was proposed by Ray and Page [8] specifically for use in multi-instance linear regression. The same procedure can, in fact, be adapted for use with any modelling procedure that returns some estimate of the error in fitting a model to (a table of) data. The procedure is shown in Fig. 5.

Provided model construction in Step 14 terminates, it is easy to show that the entire procedure terminates (the procedure contains two loops, bound by finite constants R and T). The run-time complexity of the procedure primarily depends on the complexity of the model construction in Step 14 and the computation of "closest" instances in Step 11 (the error calculation in Step 12 can in fact be done in conjunction with Step 11). Statistical model construction procedures are usually $O(n)$ algorithms (and rarely worse than quadratic). For many reasonable error measures, Steps 11–18 are usually $O(d \cdot m \cdot n)$ where $m = max_i(m_i)$. There is no guarantee that the model returned will be optimal.

In summary, we propose that an ILP system capable of lazy evaluation, using the model-identification procedure in Fig. 5 for lazily evaluating output values, can address the two basic requirements of parameter estimation presented at the outset of this section.

Input: integers R and T $(1 \leq R, T < \infty)$, and n bags $(1 \leq n < \infty)$ where bag i contains a set of d-dimensional vectors $(1 \leq d < \infty)$.
Output: A model M, the instances I used to construct the model, and its error E

01. $m_{best} \leftarrow$ a randomly constructed model
02. $i_{best} \leftarrow \emptyset$
03. $e_{best} \leftarrow \infty$
04. **for** $r = 1 \ldots R$
05. $m_0 \leftarrow$ a randomly constructed model
06. $i_0 \leftarrow \emptyset$
07. $e_0 \leftarrow \infty$
08. $t \leftarrow 1$
09. $done \leftarrow false$
10.   **while** (not done **and** $t \leq T$)
11. $i_t \leftarrow$ the "closest" instances in the n bags, given m_{t-1}
12. $e_t \leftarrow$ the error of m_{t-1} on i_t
13.   **if** $(e_t < e_{t-1})$
14. $m_t \leftarrow$ the model constructed with instances i_t
15. $t \leftarrow t + 1$
16.   **else**
17. $done \leftarrow true$
18.   **endif**
19.   **endwhile**
20.   **if** $(e_{t-1} < e_{best})$
21. $m_{best} \leftarrow m_{t-1}$
22. $i_{best} \leftarrow i_{t-1}$
23. $e_{best} \leftarrow$ the error of m_{best} on i_{best}
24.   **endif**
25. **endfor**
26. **return** $m_{best}, i_{best}, e_{best}$

Figure 5: An EM-like procedure for model construction with multi-instance data. The actual model construction procedure (Step 14) is deliberately left unspecified, as is the meaning of "closest" in Step 11. It is normally acceptable for numerical prediction to adopt some quadratic loss measure (for example, squared difference between predicted and actual value) to measure error. The same measure is then used to determine closeness. The reason for returning the data instances used to construct the model will become apparent in Section 4.

4 Prediction

In Section 1, it was illustrated how multiple data instances can result in several predictions. The reason for this is that the example rules shown there all act as "instance-level" predictors. However, what is usually needed is a "bag-level" predictor (we want a single prediction for the molecule's affinity, for example). Two solutions suggest themselves:

1. Select a single instance and return the prediction based on that instance; or
2. Compute the bag-level prediction as some function of the set of instance-level predictions (this could be, for example, some representative value like the median).

Of these, the second solution is more complex, requiring at least second-order logic constructs as well as domain-specific knowledge about the actual function to be used. We will concentrate here on domain-independent options available for implementing the first solution. In effect, this requires one additional constraint be included in the rules constructed by the ILP program. For example:

```
The affinity of molecule M is Y if:
      M has a hydrogen donor at position X and
      X is the best instance and
      Y = 2.572 X - 33.155.
```

where the condition "X is the best instance" is true for just one of the different possible values of X. But how is this selection to be achieved? First-order regression implemented in [5] and [10] both pick the first value of X (which is tantamount to random selection). This solution, which is easy to implement, can lead to counter-intuitive results. Consider, for example, using the rule above to predict Y-values for the training data in Fig. 1. Recall that the sample correlation of the model was 0.999. The bag-level predictions using the "choose first instance" option are shown in Fig. 6. The correlation coefficient now is 0.997, and we are in the unusual position of being unable to reproduce our performance on data that was used to construct the model in the first place.

Mol (N)	Actual Affinity	Predicted Affinity	Instance Used (X)
1	50	31.145	25
2	100	95.445	50
3	150	154.601	73
4	200	198.325	90
5	250	249.765	110

Figure 6: Actual and predicted values of affinity using the first instance in each bag in Fig. 1. Predictions are done using the equation $Y = 2.572\ X - 33.155$.

The difference arises, clearly, because we cannot guarantee that the "choose first instance" rule will select the same instances as were used to estimate the parameters. A more informed choice is possible if we retain the actual instances I used for parameter estimation. The "best" instance could then be selected as the first one that is within some pre-specified distance (tolerance) to an instance in I. For example, the rule above would now look like:

```
The affinity of molecule M is Y if:
      M has a hydrogen donor at position X and
      Y = 2.572 X - 33.155 using instances I and
      X is the best instance given I.
```

This requires the model construction step to estimate parameters *and* return the instances on which these estimates were based—which is fulfilled by the procedure in Fig. 5. With a sufficiently small tolerance (0, for example) we can ensure performance on the training data at least will be reproducible. With a sufficiently large tolerance (∞, for example), the constraint's behaviour degenerates to that of choosing the first instance. Superficially, using such a constraint appears related to the approach in [7] where the probability of a class label for new data point depends on the proximity of that data point to training instances. For new data, instances selected will be those that within some tolerance of the instances used for parameter estimation. It remains an open question as to what should be done if none, or several, of the new instances satisfy this constraint.

5 Concluding Remarks

It is desirable to have some mechanism by which an ILP system can utilise modelling tools developed in other areas of machine learning and in statistics. We have presented some specific issues that arise during parameter estimation and prediction when using such tools in conjunction with non-determinate (that is, relational) predicates within a first-order logic setting. To address these, we propose an ILP system that: (a) estimates parameters using the procedure in Section 3; and (b) augments the resulting hypotheses with a constraint (as described in Section 4) to ensure deterministic prediction.

The literature on attempting to incorporate statistical models within ILP hypotheses is relatively sparse. Both [5] and [10] are concerned with the use of linear regression as a background predicate for an ILP program. In the former, the multiple instance problem is avoided by restricting background predicates that introduce the independent variables to be strictly deterministic (that is, functional). The latter simply ignores the multi-instance problem and treats all instances as independent data points. Adopting the same approach, Slattery and Craven [1] examine the use of a "naive" Bayesian classifier. An entirely different approach—that of incorporating ILP hypotheses into statistical models (see for example [6, 11])—is outside the scope of this report.

The obvious limitation of the procedures proposed here is the lack of provable properties, in particular, about the optimality of models constructed. In their absence, the next step would be to examine the practical utility of such engineering workarounds.

References

[1] M. Craven and S. Slattery. Relational Learning with Statistical Predicate Invention: Better Models for Hypertext. *Machine Learning*, 43(1-2):97–119, 2001.

[2] T. Dietterich, R. Lathrop, and T. Lorano-Perez. Solving the multiple-instance problem with axis-parallel rectanges. *Artificial Intelligence*, 89:31–71, 1997.

[3] D.R. Dooly, S.A. Goldman, and S.S. Kwek. Real-Valued Multiple-Instance Learning with Queries. In N. Abe, R. Khardon, and T. Zeugmann, editors, *Proceedings of the Twelfth Conference on Algorithmic Learning Theory (ALT 2001)*, volume 2225 of *LNAI*, pages 167–180, Berlin, 2001. Springer-Verlag.

[4] P. Finn, S. Muggleton, D. Page, and A. Srinivasan. Pharmacophore Discovery using the Inductive Logic Programming system Progol. *Machine Learning*, 30:241–270, 1998.

[5] A. Karalic and I. Bratko. First-Order Regression. *Machine Learning Journal*, 26:147–176, 1997. Special Issue on ILP.

[6] R. D. King, A. Srinivasan, and L. DeHaspe. WARMR: A Data Mining Tool for Chemical Data. *Computer Aided Molecular Design*, 15:173–181, 2001.

[7] O. Maron. *Learning from Ambiguity.* PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA, 1998.

[8] S. Ray and C.D. Page. Multiple Instance Regression. In *Proceedings of the Eighteenth International Conference on Machine Learning*, San Mateo, CA, 2001. Morgan Kaufmann.

[9] A. Srinivasan. The Aleph Manual. Available at http://www.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/, 1999.

[10] A. Srinivasan and R.C. Camacho. Numerical reasoning with an ILP program capable of lazy evaluation and customised search. *Journal of Logic Programming*, 40(2,3):185–214, 1999.

[11] A. Srinivasan and R.D. King. Feature construction with Inductive Logic Programming: a study of quantitative predictions of biological activity aided by structural attributes. *Data Mining and Knowledge Discovery*, 3(1):37–57, 1999.

A An Example

This section contains input files for the ILP system Aleph [9] for predicting Y values in Fig. 1.

A.1 Background knowledge

The background knowledge consisted of the following definitions (we have omitted the definitions of some obvious utility predicates):

```
:- modeh(1,affinity(+mol,+aff)).
:- modeb(*,hdonor(+mol,-x)).
:- modeb(1,lin_regress1(+mol,+aff,+x,#model,#instances,#error)).

:- determination(affinity/2,hdonor/2).
:- determination(affinity/2,lin_regress1/6).

:- lazy_evaluate(lin_regress1/6).

% locations of hydrogen donors as shown in Fig. 1

hdonor(m1,25).
hdonor(m1,33).
hdonor(m1,29).

hdonor(m2,50).

hdonor(m3,73).
hdonor(m3,75).

hdonor(m4,90).

hdonor(m5,120).
hdonor(m5,110).


% linear regression
% definition for use during lazy evaluation
% calls EM search in Fig. 5 with R=10 and T=5.
lin_regress1(B,Y,X,Model,Instances,Error):-
        nlist(Y), nlist(X), var(Model), !,
        em_model(10,5,B,Y,X,Model,Instances,Error).

% definition for prediction
% includes constraint to select the best instance
% as described in Section 4.
lin_regress1(_,Y,X,Model,Instances,_):-
        number(X), nonvar(Model),
        best_instance(X,Instances), !,
        test_model([Y,X],Model,_).

% otherwise: default catch-all
lin_regress1(_,Y,_,[0.0,Y],[],0.0):-
        (var(Y) ->
```

```
                default_value(Y);
                true
        ).

nlist([]).
nlist([_|_]).

default_value(0.0).

% check to see if instance is within some delta-neighbourhood
% of data used to construct the model
best_instance(X,[_,XVals]):-
        delta(Delta),
        in_nbd(XVals,X,Delta).

delta(0.5).

in_nbd([X1|_],X,Delta):-
        Dist is (X1-X),
        Dist >= -Delta,
        Dist =< Delta, !.
in_nbd([_|XVals],X,Delta):-
        in_nbd(XVals,X,Delta).

% not interested in rules that do not contain a regression literal
% forces rule to predict value of Y
false:-
        hypothesis(_,Body,_),
        not(has_literal(Body,lin_regress1/6)).

has_literal((Lit,_),Name/Arity):-
        functor(Lit,Name,Arity), !.
has_literal((_,Lits),Pred):-
        has_literal(Lits,Pred).
has_literal(Lit,Name/Arity):-
        functor(Lit,Name,Arity), !.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% EM-like procedure for estimating best fitting line through instances
%        finds best line after R random restarts, each of which iterates at most
%        T times.
% This is the procedure in Fig. 5.
em_model(R,T,[Bags|_],[YVals|_],[XVals|_],Model,Instances,Error):-
        Inf is inf,
        em_model(R,T,Bags,YVals,XVals,[],[],Inf,Model,Instances,Error).

em_model(0,_,_,_,_,Model,Instances,Error,Model,Instances,Error):- !.
em_model(R,T,Bags,YVals,XVals,_,_,E0,M,I,E):-
```

```
        write('**** Restarts Left: '), write(R), write(' ****'), nl,
        random_model(Mr),
        Inf is inf,
        em_model(T,Bags,YVals,XVals,Mr,[],Inf,M1,I1,E1),
        E1 < E0, !,
        R1 is R - 1,
        em_model(R1,T,Bags,YVals,XVals,M1,I1,E1,M,I,E).
em_model(R,T,Bags,YVals,XVals,M0,I0,E0,M,I,E):-
        R1 is R - 1,
        em_model(R1,T,Bags,YVals,XVals,M0,I0,E0,M,I,E).

em_model(0,_,_,_,M,I,E,M,I,E):- !.
em_model(T,Bags,YVals,XVals,M0,_,E0,M,I,E):-
        find_new_instances(Bags,YVals,XVals,M0,0.0,YVals1,XVals1,E1),
        E1 < E0, !,
        train_model([YVals1,XVals1],M1,_),
        T1 is T - 1,
        em_model(T1,Bags,YVals,XVals,M1,[YVals1,XVals1],E1,M,I,E).
em_model(_,_,_,_,M,I,E,M,I,E).

find_new_instances(Bags,YVals,XVals,M,E,YVals1,XVals1,E1):-
        get_closest_instances(Bags,YVals,XVals,M,E,[],E1,Instances),
        get_bag_instances(Instances,YVals1,XVals1).

get_closest_instances([],_,_,_,E,I,E,I).
get_closest_instances([Bag|Bags],[Y|YVals],[X|XVals],M0,E0,I0,E,I):-
        test_model([Y,X],M0,Error),
        (del(Bag-[OldE,_],I0,Left) ->
                (Error < OldE ->
                        I1 = [Bag-[Error,[Y,X]]|Left],
                        E1 is E0 + Error - OldE;
                        I1 = I0,
                        E1 = E0);
                I1 = [Bag-[Error,[Y,X]]|I0],
                E1 is E0 + Error),
        get_closest_instances(Bags,YVals,XVals,M0,E1,I1,E,I).

get_bag_instances([],[],[]).
get_bag_instances([_-[_,[Y,X]]|Instances],[Y|YVals],[X|XVals]):-
        get_bag_instances(Instances,YVals,XVals).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% model-specific code

% obtain least-squares line
train_model([YVals,XVals],[A,B],Error):-
        length(YVals,N),
        N > 0,
        sumprod(XVals,YVals,SumXY),
```

```
        sumprod(XVals,XVals,SumSqX),
        sum(XVals,SumX),
        sum(YVals,SumY),
        B is (N*SumXY - SumX*SumY)/(N*SumSqX - SumX^2),
        MeanY is SumY/N,
        MeanX is SumX/N,
        A is MeanY - B*MeanX,
        sum_error([YVals,XVals],[A,B],Error),
        print_model([YVals,XVals],[A,B],Error).

% predict y-value
test_model([Y,X],[C0,C1],Error):-
        Y1 is C0 + C1*X,
        (var(Y) ->
                Y = Y1,
                Error = 0.0;
                Error is (Y - Y1)^2
        ).

% obtain a random linear model
random_model([C0,C1]):-
        get_random(-100,100,C0),
        get_random(-100,100,C1),
        write('Random Model: '),
        write('Y = '), write(C1), write('X'),
        write(' + '), write(C0), nl.
```

A.2 Examples

The actual Y values in Fig. 1 are stored in a file of "positive" examples:

```
affinity(m1,50).
affinity(m2,100).
affinity(m3,150).
affinity(m4,200).
affinity(m5,250).
```

B Output

The following is a transcript of the output of Aleph using the information of the previous section (some irrelevant output has been removed). It is evident that the "best" model (see Fig. 3) is identified on the eighth restart ("`**** Restarts Left:  3 ****`").

```
   ?- [aleph4].
 [ consulting aleph4... ]

A L E P H
Version 4


yes
   ?- read_all(miline).
 [ reconsulting miline.b... ]
[consulting pos examples] [miline.f]
[consulting neg examples] [miline.n]

yes
   ?- sat(1).
[sat] [1]
[affinity(m1,50)]

[bottom clause]
affinity(A,B) :-
   hdonor(A,C),
   hdonor(A,D),
   hdonor(A,E),
   lin_regress1(A,B,E,[0.0,50],[],0.0),
   lin_regress1(A,B,D,[0.0,50],[],0.0),
   lin_regress1(A,B,C,[0.0,50],[],0.0).
[literals] [7]
[saturation time] [0.01]

yes
   ?- reduce.
[reduce]
[best label so far] [[1,0,2,1]/0]
affinity(A,B).
[constraint violated]
[5/0]
affinity(A,B) :-
   hdonor(A,C).
[constraint violated]
[5/0]
affinity(A,B) :-
   hdonor(A,C).
[constraint violated]
[5/0]
affinity(A,B) :-
   hdonor(A,C).
[constraint violated]
[5/0]
[lazy evaluation] [lin_regress1]
```

```
**** Restarts Left: 10 ****
Random Model: Y = 99X + 2
Instances:
Y     X
50    25
100    50
150    73
200    90
250    110

Model: Y = 2.36849X + -14.8471
Error: 130.831

Instances:
Y     X
50    29
100    50
150    73
200    90
250    110

Model: Y = 2.46992X + -23.8824
Error: 53.8003

Instances:
Y     X
50    29
100    50
150    73
200    90
250    110

Model: Y = 2.46992X + -23.8824
Error: 53.8003

**** Restarts Left: 9 ****
Random Model: Y = 50X + 30
Instances:
Y     X
50    25
100    50
150    73
200    90
250    110

Model: Y = 2.36849X + -14.8471
Error: 130.831

Instances:
```

```
Y     X
50    29
100    50
150    73
200    90
250    110

Model: Y = 2.46992X + -23.8824
Error: 53.8003

Instances:
Y     X
50    29
100    50
150    73
200    90
250    110

Model: Y = 2.46992X + -23.8824
Error: 53.8003

**** Restarts Left: 8 ****
Random Model: Y = 65X + 64
Instances:
Y     X
50    25
100    50
150    73
200    90
250    110

Model: Y = 2.36849X + -14.8471
Error: 130.831

Instances:
Y     X
50    29
100    50
150    73
200    90
250    110

Model: Y = 2.46992X + -23.8824
Error: 53.8003

Instances:
Y     X
50    29
100    50
150    73
200    90
250    110

Model: Y = 2.46992X + -23.8824
Error: 53.8003

**** Restarts Left: 7 ****
Random Model: Y = 10X + -7
Instances:
Y     X
50    25
100    50
150    73
200    90
250    110

Model: Y = 2.36849X + -14.8471
Error: 130.831

Instances:
Y     X
50    29
100    50
150    73
200    90
250    110

Model: Y = 2.46992X + -23.8824
Error: 53.8003

Instances:
Y     X
50    29
100    50
150    73
200    90
250    110

Model: Y = 2.46992X + -23.8824
Error: 53.8003

**** Restarts Left: 6 ****
Random Model: Y = 34X + 14
Instances:
Y     X
50    25
100    50
150    73
```

```
200    90
250    110

Model: Y = 2.36849X + -14.8471
Error: 130.831

Instances:
Y     X
50    29
100    50
150    73
200    90
250    110

Model: Y = 2.46992X + -23.8824
Error: 53.8003

Instances:
Y     X
50    29
100    50
150    73
200    90
250    110

Model: Y = 2.46992X + -23.8824
Error: 53.8003

**** Restarts Left: 5 ****
Random Model: Y = 60X + -47
Instances:
Y     X
50    25
100    50
150    73
200    90
250    110

Model: Y = 2.36849X + -14.8471
Error: 130.831

Instances:
Y     X
50    29
100    50
150    73
200    90
250    110

Model: Y = 2.46992X + -23.8824
Error: 53.8003

Instances:
Y     X
50    29
100    50
150    73
200    90
250    110

Model: Y = 2.46992X + -23.8824
Error: 53.8003

**** Restarts Left: 4 ****
Random Model: Y = 57X + -22
Instances:
Y     X
50    25
100    50
150    73
200    90
250    110

Model: Y = 2.36849X + -14.8471
Error: 130.831

Instances:
Y     X
50    29
100    50
150    73
200    90
250    110

Model: Y = 2.46992X + -23.8824
Error: 53.8003

Instances:
Y     X
50    29
100    50
150    73
200    90
250    110

Model: Y = 2.46992X + -23.8824
Error: 53.8003
```

```
**** Restarts Left: 3 ****
Random Model: Y = -1X + 84
Instances:
Y     X
50    33
100    50
150    73
200    90
250    110

Model: Y = 2.5724X + -33.1548
Error: 47.7352

Instances:
Y     X
50    33
100    50
150    73
200    90
250    110

Model: Y = 2.5724X + -33.1548
Error: 47.7352

**** Restarts Left: 2 ****
Random Model: Y = -1X + -39
Instances:
Y     X
50    25
100    50
150    73
200    90
250    110

Model: Y = 2.36849X + -14.8471
Error: 130.831

Instances:
Y     X
50    29
100    50
150    73
200    90
250    110

Model: Y = 2.46992X + -23.8824
Error: 53.8003

Instances:
Y     X
50    29
100    50
150    73
200    90
250    110

Model: Y = 2.46992X + -23.8824
Error: 53.8003

**** Restarts Left: 1 ****
Random Model: Y = 6X + -79
Instances:
Y     X
50    25
100    50
150    73
200    90
250    110

Model: Y = 2.36849X + -14.8471
Error: 130.831

Instances:
Y     X
50    29
100    50
150    73
200    90
250    110

Model: Y = 2.46992X + -23.8824
Error: 53.8003

Instances:
Y     X
50    29
100    50
150    73
200    90
250    110

Model: Y = 2.46992X + -23.8824
Error: 53.8003

[completed]
affinity(A,B) :-
   hdonor(A,C),
   lin_regress1(A,B,C, [-33.1548,2.5724],
```

```
                [[50,100,150,200,250],
                [33,50,73,90,110]],47.7352).
[5/0]
[-------------------------------------]
[found clause]
affinity(A,B) :-
   hdonor(A,C),
   lin_regress1(A,B,C,[-33.1548,2.5724],[[50,100,150,200,250],[33,50,73,90,110]],47.7352).
[pos-neg] [5]
[clause label] [[5,0,3,5]]
[clauses constructed] [5]
[-------------------------------------]
[clauses constructed] [5]
[search time] [0.02]
[best clause]
affinity(A,B) :-
   hdonor(A,C),
   lin_regress1(A, B, C, [-33.1548,2.5724], [[50,100,150,200,250], [33,50,73,90,110]], 47.7352).
[pos-neg] [5]

yes
   ?-
```