The Use of Inductive Logic Programming for the Development of the Statistical Software Tool CORA

Katharina Morik Iris Pigeot[1*] Ursula Robers

[1] Institute for Statistics, University of Munich, Ludwigstr. 33, D–80539 München
e–mail: pigeot@stat.uni-muenchen.de

Abstract

This paper presents the knowledge acquisition process for the system CORA, which provides a statistical analysis of stratified case–control data and additionally supports the researcher at the different steps of this analysis. In addition to the general information given by the help system, the advice tool recommends an adequate statistical method. This recommendation depends on the actual data to be analysed. The underlying rule base containing the recommendations was discovered by inductive logic programming (ILP) methods.

1 Introduction

In epidemiology it is of interest to investigate possible associations between a potential risk factor and a certain disease. To answer such questions data on the disease and on the exposure are typically collected by conducting a case–control study. The collected data are then often analysed by stratified contingency tables where as quantifying statistical measure for the investigated association the so–called odds ratio ψ is estimated from the data. Here, a great variety of estimators with different statistical properties is available. Their behaviour depends on the characteristics of the case–control data to analyse.

Carrying out the analysis, the researcher, typically with medical or epidemiological background, has to choose an appropriate estimator from a large number of competing methods. To cope with this problem, the system CORA (Combined Odds Ratio Analysis) has been developed. In addition to a help system containing information that does not consider the characteristics of the actual data to analyse, CORA offers an advice tool which recommends an appropriate estimator depending on the properties of the underlying case–control data entered into the system. In order to build up the rule base for the advice tool, we have applied machine learning, especially inductive logic programming methods. Our approach emphasized the process of modelling the expertise. A simulation study served as knowledge source for this process. For the modelling of the expertise, we have used the system MOBAL (see Morik et al, 1993) which combines knowledge acquisition and machine learning. Therefore we had to build up a model, the knowledge base, which made the results of the simulation study available for MOBAL. The knowledge representation of MOBAL is based on a first order

*Financial support by the Bennigsen–Foerder–Grant of Nordrhein–Westfalen is gratefully acknowledged.

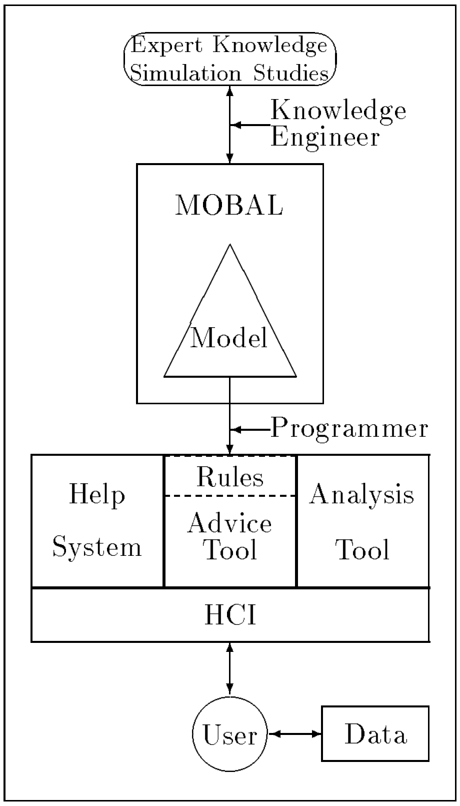

Figure 1: Integration of the modelling process into the system architecture of CORA

logic. The integrated rule discovery tool RDT then induces new rules from facts. The set of induced rules relates the characteristics of the data to the behaviour of the estimators. Figure 1 depicts the modelling process (upper part) and the integration of its results into the system architecture of CORA (lower part). Thus, with means of ILP we have the chance to discover new knowledge and therefore to improve the knowledge base.
The remainder of the paper is organized as follows. In Sections 2, 3, and 4 we give brief overviews over ILP, MOBAL, and RDT. In Section 5 we take a closer look at the modelling process. Section 6 describes the first order logic representation of the domain and Section 7 presents the learned recommendation rules which provide the basis of the advice tool of the system CORA. We close with some remarks concerning the design of CORA and suggest some possible improvements and enlargements of our approach.

2 Inductive Logic Programming

Inductive Logic Programming (ILP) has recently been developed as the intersection of inductive learning and logic programming. It investigates the induction of first order clausal theories from examples and background knowledge. ILP employs techniques from both machine learning and logic programming. The drawbacks of classical machine learning techniques are that they use a limited knowledge representation formalism (essentially a propositional logic), and that they cannot use background knowledge during the learning process. From computational logic, ILP inherits its representational formalism and extends its theory by investigating induction rather than deduction as the basic mode of inference. Whereas present computational logic theory describes deductive inference from logic formulas provided by the user, inductive logic programming theory describes the inductive inference of logic programs from instances and background knowledge.

ILP differs from other methods for inductive inference in using a universal representation. It therefore has a wider scope of applicability and allows a single representation for examples, background knowledge, and hypotheses. The general setting of ILP is as follows:

There is a background knowledge B that consists of clauses and facts. Then positive examples E^+ and negative examples E^- consisting of facts are given. The background knowledge is consistent with E^- but B does not explain E^+, i.e.

- $B \wedge E^- \not\models \Box$ (prior satisfiability)
- $B \not\models E^+$ (prior necessity)

It is of interest to derive a hypothesis H with

- $B \wedge H \wedge E^- \not\models \Box$ (posterior satisfiability)
- $B \wedge H \models E^+$ (posterior sufficiency)

That means B and H are consistent with E^- and H allows to explain E^+ relative to B.

Logic programs are easy to manipulate by machine learning algorithms. Changes can be made to the program by simply adding or deleting either complete clauses or literals within a clause. The key idea of most ILP algorithms is to structure the search space by means of the dual notions of generalization and specialization where a hypothesis G is more general than a hypothesis S, iff $G \models S$.

To construct a generalization or a specialization of a hypothesis, inductive and deductive inference rules are used. Most inductive inference rules generalize by adding a clause to a hypothesis or by dropping a negative literal from a clause. Deductive inference rules may drop a clause from a hypothesis or add a negative literal to a clause.

Generalization and specialization form the basis for pruning the search space. In case of $B \wedge H \not\models e$, where e is a positive evidence, none of the specializations H' of H will imply e. When $B \wedge H \wedge e \models \Box$, where e is a negative evidence, all generalizations H' of H will also be inconsistent with $B \wedge E^-$. Thus the space can be searched general–to–specific by applying deductive inference rules in order to remove inconsistencies with the negative examples or it can be searched specific–to–general by starting from positive examples and the background knowledge and generalize the hypothesis by applying inductive inference rules. During the search, care has to be taken that in the general–to–specific case the hypothesis remains sufficient with regard to the positive evidence and in the specific–to–general case the hypothesis remains satisfiable, i.e. does not imply negative examples.

The simplest model of induction for ILP is the Θ–subsumption as introduced by Plotkin (1970). Buntine (1988) improved this subsumption relationship, which has some serious drawbacks, by defining a generalized subsumption. According to Buntine a clause C Θ–subsumes a clause C' ($C \geq_\Theta C'$), if the more general clause C can be converted to the clause C' by repeatedly turning variables to constants or other terms, adding atoms to the body, or partially evaluating the body of C by resolving some clauses in the background knowledge. Thus there are three deductive inference operators to specialize a hypothesis.

3 MOBAL

The multi–strategy learning system MOBAL (cf. Morik et al, 1993) was developed by the GMD, Sankt Augustin. It is an environment for building up, inspecting, and changing a knowledge base. For this purpose, it integrates various tools, e.g. tools for automating the construction of rules, automatic revision and refinement, and automatic construction of new features or concepts (constructive inference). For this integration MOBAL uses a knowledge representation which is powerful enough to suit all tools and which is still tractable. The items in the knowledge base are represented within a Horn clause logic. However, negated literals are allowed in the premises and in the conclusion of the rules and some higher–order elements are added. The domain knowledge consists mainly of

- **facts**, expressing relations, properties and concept membership of objects. Within MOBAL they are represented as function–free literals without variables:

$$pred(Term_1, Term_2, ..., Term_n).$$

- **rules**, expressing inferential relations between predicates, necessary and sufficient conditions of concepts, hierarchies of properties. They are represented as Horn clauses in which the premises and the conclusion literal may be negated:

$$pred_1(Term_1^{(1)}, \ldots, Term_{n_1}^{(1)}) \;\&\ldots\&\; pred_m(Term_{p_1}^{(m)}, \ldots, Term_{p_m}^{(m)})$$
$$\rightarrow pred_{concl}(Term_1^{(concl)}, \ldots, Term_{q_{concl}}^{(concl)}).$$

  By forward chaining, new facts can be inferred from rules.

- **rule models** (metapredicates), expressing the structure of the rules to be learned. A rule model is a rule in which predicate variables are used instead of actual predicates of an application domain. A predicate variable can be instantiated by a predicate symbol of the same arity. A completely instantiated rule model, where all predicate variables have been replaced by actual predicates, is a rule.

Further, the domain knowledge can include some other items like a topology of predicates and sorts, which are not considered here in detail.
The MOBAL system includes e.g. the rule discovery tool RDT, described in Section 4, a concept formation tool which induces necessary and sufficient conditions for concepts from positive and negative examples, a sort taxonomy tool which clusters constant terms occurring as arguments in facts, a knowledge revision tool which resolves contradictions in the knowledge base, a predicate structuring tool which structures the predicates occurring in the knowledge base, and an inference engine which infers in forward– and backward–chaining.

4 The ILP tool RDT

The rule discovery tool RDT (see Kietz, Wrobel, 1992) is an ILP tool, since it induces new clauses from background knowledge and examples. The examples are entered as a set of facts. With the help of the background knowledge consisting of clauses and facts, the knowledge base is saturated. That means the inference engine performs forward inferences with the given rules, hence adds new ground facts to the knowledge base. At the end of the learning process,

the induced rules can serve as new background knowledge, i.e. RDT performs closed–loop learning.
As the necessary input, RDT additionally needs a set of rule models. With the help of rule models RDT defines a hypothesis subspace that is actually searched. This hypothesis space consists of the set of all possible instantiations of rule models with domain predicates. For efficiently searching in this hypothesis space a generalization relation on the set of rule models is defined by suitably extending the Θ–subsumption for clauses. This leads to the following definition of the generality relationship $\geq_{RS}$ between rule models where a rule model R is more general than R' ($R \geq_{RS} R'$), iff there exists a substitution σ applied to term variables, and a substitution Σ applied to predicate variables that does not unify different predicate variables such that $R\sigma\Sigma \subseteq R'$. The substitution σ turns term variables to constants or other terms and the substitution Σ renames or instantiates the predicate variables.
RDT searches this partial ordering top–down from the most general to the more specific hypotheses. Hypotheses are computed by instantiating the predicate variables of the rule models with predicate symbols. Then, these hypotheses are tested. There are three possible results for this test: 1. the hypothesis H is too general, i.e. it covers (too many) negative or unknown instances ($B \wedge H \wedge E^- \models \Box$), 2. the hypothesis H is accepted, or 3. the hypothesis H is too special, i.e. it does not cover (enough of) the positive instances ($B \wedge H \not\models E^+$). A breadth first search strategy is used and those hypotheses that have already been accepted or pruned as too special are remembered to avoid exploring their specializations. Only in the case of a hypothesis being too general, the search is continued.
The premises of a rule model are incrementally instantiated. An order of the premises (see Morik et al, 1996) allows to further prune the search of hypotheses also within a single rule model, because all partially instantiated hypotheses can be tested in the same way as the fully instantiated rule model by dropping the uninstantiated premises.
The threshold for too few instances and thus for pruning the search is computed from a user–specified acceptance criterion. Several primitives for this criterion are defined as the cardinalities of the positive instances of a hypothesis H, the negative instances of H, the unknown instances of H, i.e. the conclusion of H can neither be proved nor disproved but these instances are predicted by H, the total number of instances of H, the instances of the conclusion which are uncovered by H, and all instances of the conclusion. The acceptance criterion is a logical expression of conjunctions and disjunctions of arithmetical comparisons involving arithmetical expressions built from numbers and the above primitives. For all specializations of a hypothesis H the numbers of positive, negative, and predicted instances are smaller than those numbers for the hypothesis H itself, whereas the number of uncovered instances grows. So we can prune hypotheses which cannot be accepted.
Further effectiveness of the algorithm RDT comes from the use of a many sorted logic and a topology of predicates (see Kietz, Wrobel, 1992).

5 Modelling the domain knowledge

As mentioned above, for building up a domain model we used a simulation study (see e.g. Pigeot, 1991) in which six estimators of the common odds ratio have been examined. Such simulation studies are carried out to investigate and compare the performance of statistical methods. For the design of the simulation studies different values for the involved parameters are chosen as for instance for the most important parameter, i.e. the common odds ratio, or for

the sample sizes of cases and controls in each table. Each choice of these values characterizes a certain constellation. From these characteristics, others can easily be derived such as a coefficient for measuring the balance between the sample sizes.
As essential statistical measures for the assessment of the estimators we calculated the means of the estimated biases and mean squared errors (MSE) from 1000 simulation runs for each parameter constellation. Based on the bias and the MSE a ranking of the estimators can be given where these two criteria have to be combined appropriately.
The learned rules to characterize the estimators use qualitative criteria instead of quantitative measures. This was realized by dividing the initial criteria into appropriate categories. Fixing the bounds of the categories was the most difficult but also one of the most important tasks in modelling this domain. The definition of the categories is arbitrary, but of course the chosen categories strongly influence the rules, which are discovered, as well as the complexity of the knowledge base.
For obtaining the recommendations, we have to come back to the ranking and to the suitability of each estimator. The ranking is based on an appropriate combination of the ranks of an estimator regarding bias and MSE. The suitability is derived by combining the two categorized criteria bias and MSE. Then, we rate the ranking with respect to the suitability.
Having learned the recommendation rules, their application in a given data situation of course heavily depends on the characteristics of the specific data set. Some of these characteristics can be directly read from the data such as the sample sizes, whereas others have to be estimated such as the odds ratio. In the latter case we recommend to use a second data set to avoid a circular argument. Since a second data set is usually not at hand, a so–called pilot study from the original data should be drawn.

6 Representation of the domain model

The characteristics of the data as well as the assessment of the estimators are represented as positive examples and thus as facts. The following 18 facts with two or three arguments describe the properties of the first parameter constellation. The third place of some of the predicates is necessary for those data characteristics varying across the strata.

```
oddsratio(sit_1,1.7).               diff_prob(sit_1,0.1).
number_of_strata(sit_1,2).          prob(sit_1,1,0.2).
gini_ratio_cases(sit_1,1).          prob(sit_1,2,0.3).
gini_ratio_ratio(sit_1,0).          ratio_cc(sit_1,1,3).
gini_ratio_prob(sit_1,1).           ratio_cc(sit_1,2,3).
number_of_cases(sit_1,1,20).        mean_cases(sit_1,25).
number_of_cases(sit_1,2,30).        mean_controls(sit_1,75).
number_of_controls(sit_1,1,60).     mean_prob(sit_1,0.25).
number_of_controls(sit_1,2,90).     mean_ratio(sit_1,3).
```

`oddsratio, number_of_strata, diff_prob,` ... are predicates concerning the data characteristics. The arguments `sit_1, sit_2`, ... label the parameter constellation of the simulation study. They are the first arguments in all predicates. `1.7, 1.0, 2,` ... are the arguments in the last place. They denote the values for the corresponding data characteristics, e.g. the common odds ratio or the number of strata. The arguments `1, 2, 3,` ... in the second place

of the three–place predicates specify to which strata the corresponding values for the data characteristics belong.
The following four facts characterize the assessment of the estimators. As above the arguments in first place label the parameter constellation. The second place marks the names of the estimators. The last place marks the values for the mean squared error (mse) and the bias:

```
mse(sit_1,mh,1.12436).  bias(sit_1,mh,0.54718).
```

For all data characteristics and both assessment criteria, the classification into categories is achieved by entering rules like the following one as background knowledge:

```
number_of_strata(S,NS) & gt(NS,4) & le(NS,10) -->
moderate_number_of_strata(S).
```

The above mentioned ranking and the assessed suitabilities are derived and represented by similar rules. For further information concerning these rules see Robers (1995).
If we know the recommendations and the categorized suitabilities then we can assess the recommendations according to the suitabilities with rules like the following ones:

```
recommendation(S,E) & good_suitability(S,E) --> good_recommendation(S,E).
```

7 The learned recommendation rules

In the framework of developing CORA, the aim of learning is to gain a characterization of the estimators concerning the categorized data characteristics. Nine data characteristics were selected and classified into a total number of 22 categories. The target predicates are the four classified recommendations and additionally the unassessed recommendation. We carried out further learning steps with some different predicates, e.g. those for the suitabilities (Robers, 1995). To define the hypothesis space for learning, we have defined four metapredicates. The first metapredicate searched for one data characteristic which is sufficient to recommend an estimator. With the other ones we searched for combinations of two, three or four data characteristics.
In the represented knowledge base, we did not want to infer new facts from the learned rules. Hence, two acceptance criteria were considered. The first one does not allow predicted facts at all, where the second one allows a maximum number of 10% of the total number of predicted facts.
We learned a total of about 270 rules of the following type

```
small_oddsratio(S) & large_number_of_cases(S) & small_number_of_strata(S) -->
moderate_recommendation(S,jk).
```

The goal was then to select a rule set from the learned rules and to integrate it into the KBS. As criterium for this selection, we mainly considered the redundancy in the rule set (Morik et al, 1996).
With the help of MOBALs rule restructuring tool (RRT) (Sommer, 1995) we have evaluated the selected rule set according to different criteria. The completeness for instance was about 40%, a correctness of 93% was achieved and the rules covered four instances on an average.

To develop the advice tool, we integrated the selected rules into the KBS. For efficiently encoding the rules, we had to determine an order for querying the data characteristics (Morik et al, 1996). Additionally, we had to fix an order for the rules, because there are data constellations, where more than one rule could fire. The advice tool now analyses the actual case–control data with respect to their characteristics and selects a rule whose premises are covered by these ascertained characteristics. This rule recommends an appropriate estimator for the common odds ratio.

8 Some final remarks

One important aspect concerning the derivation of such a computer system concerns the implementation of a graphical interface. For simplifying this complex task we used the tool Delphi 1.0 (Borland, 1995). The user interface of the Windows application CORA consists of various forms which have been produced with the help of Delphi. This tool was also useful in encoding the expertise and the necessary statistical procedures.

Since CORA contains three main components we would like to briefly mention the design of the three corresponding tools.

Let us start with the advice tool. Here, we have a three–sided form. On the first page, a point estimator and an approach for correcting its bias are recommended based on the integrated rule set. The specific rule used for this recommendation, i.e. its number, is listed. This page also gives information on the suitability of this estimator. The second and the third page summarize those properties of the data set under investigation or of the pilot data, respectively, which are used for the recommendation.

Since the analysis tool offers a great variety of differing methods in that field it also consists of several forms of which only two are described here. For instance, the user can have a look on the data as stratified contingency tables where different tables can be viewed simultaneously. The corresponding estimates, their estimated variances and confidence intervals are depicted on another page of the corresponding form.

The last of the main tools of CORA, the help system created with the Windows help compiler, gives information on the statistical and epidemiological background as well as on the system itself especially including hints for using the advice tool. This help system can be entered from each form of CORA by the use of the help button which can be found on each form. It should perhaps be mentioned that the help system and the advice tool complement each other by giving general information and special data–driven recommendations to assist users being not familiar with such analyses.

Of course, all tools should and can be enlarged. The advice tool, for instance, can be extended on the one hand by adding information obtained from the asymptotic behaviour of the point estimators and on the other hand by deriving comparable rules for the other statistical methods implemented in CORA. Graphical representations of the data should be integrated in the analysis tool. Finally, each of the above components should be updated from time to time by new results published in the literature or gained from own investigations.

Nevertheless, CORA is an interesting approach for the application of AI methods, especially ILP and knowledge acquisition techniques, for building up a statistical software tool.

References

Borland GmbH (ed.) (1995). *Delphi for Windows 1.0, User Guide.*

Buntine, W. (1988). Generalized subsumption and its applications to induction and redundancy. *Artificial Intelligence* **36**, 149–176.

Kietz, J.U., Wrobel, S. (1992). Controlling the complexity of learning in logic through syntactic and task–oriented models. In: S. Muggleton (ed.). *Inductive logic programming*, A.P.I.C. Series 18, 335–360. Academic Press, London.

Morik, K., Pigeot, I., Robers, U. (1996). CORA — A knowledge based system for the analysis of case–control studies. *Research Report* **21**, Department of Artificial Intelligence, Faculty of Computer Science, University of Dortmund.

Morik, K., Wrobel, S., Kietz, J.U. and Emde W. (1993). *Knowledge acquisition and machine learning: Theory, methods and applications.* Academic Press, London.

Pigeot, I. (1991). A simulation study of estimators of a common odds ratio in several 2×2 tables. *Journal of Statistical Computation and Simulation* **38**, 65–82.

Plotkin, G. D. (1970). A note on inductive generalization. In: B. Meltzer and D. Michie (eds.) *Machine intelligence* **6**, 153–163, Edinburgh Univ. Press, Edinburgh.

Robers, U. (1995). Entwicklung eines wissensbasierten Assistentensystems zur Analyse von Fall–Kontroll–Studien, Masters thesis, Department of Artificial Intelligence, Faculty of Computer Science, University of Dortmund.

Sommer, E. (1995). MOBALs theory restructuring tool RRT, ESPRIT Project ILP (6020), ILP Deliverable GMD 2.2.