Tailoring Representations to Different Requirements

Katharina Morik[1]

Univ. Dortmund, Computer Science VIII,
D-44221 Dortmund, Germany
morik@ls8.cs.uni-dortmund.de
http://www-ai.cs.uni-dortmund.de

Abstract. Designing the representation languages for the input and output of a learning algorithm is the hardest task within machine learning applications. Transforming the given representation of observations into a well-suited language L_E may ease learning such that a simple and efficient learning algorithm can solve the learning problem. Learnability is defined with respect to the representation of the output of learning, L_H. If the predictive accuracy is the only criterion for the success of learning, the choice of L_H means to find the hypothesis space with most easily learnable concepts, which contains the solution. Additional criteria for the success of learning such as comprehensibility and embeddedness may ask for transformations of L_H such that users can easily interpret and other systems can easily exploit the learning results. Designing a language L_H that is optimal with respect to all the criteria is too difficult a task. Instead, we design families of representations, where each family member is well suited for a particular set of requirements, and implement transformations between the representations. vvIn this paper, we discuss the representation family of Horn logic. Work on tailoring representations is illustrated by a robot application.

1 Introduction

Machine learning has focused on the particular learning task of concept learning. Investigating this task has led to sound theoretical results as well as to efficient learning systems. However, some aspects of learning have not yet received the attention they deserve. Theoretical results are missing where they are urgently needed for successful applications of machine learning. If we contrast the theoretically analyzed setting of concept learning with the one found in real-world applications, we encounter quite a number of open research questions. Let me draw your attention to those questions that are related with tailoring representations.

The setting of concept learning that has been well investigated can be summarized as follows.

Concept learning: Given a set of examples in a representation language L_E, drawn according to some distribution, and background knowledge in a representation language L_B, learn a hypothesis in a representation language L_H that classifies further examples properly according to a given criterion.

Different paradigms vary assumptions about the distribution and the existence of background knowledge. In the paradigm of probably approximately correct (PAC) learning, for instance, an unknown but fixed distribution is assumed and background knowledge is ignored. In the paradigm of inductive logic programming (ILP), the distribution is ignored, but background knowledge is taken into account. Different approaches vary the particular success criterion. For instance, information gain, minimal description length, or Bayes-oriented criteria are discussed. Structural risk minimization balances the error rate and the complexity of hypotheses. The variety of criteria can be subsumed under that of accuracy of predictions (i.e. correctness of classifications of new observations). In contrast, applications of machine learning are evaluated with respect to additional criteria:

comprehensibility: How easily can the learning results be interpreted by human decision makers?

embeddedness: Most learning results are produced in order to enhance the capabilities of another system (e.g., problem solving system, natural language system, robot), the so-called performance system. The criterion is here: How easily can learning results be used by the performance system?

These criteria are most often ignored in theoretical studies. Comprehensibility and embeddedness further constrain the choice of an appropriate hypothesis language L_H. The problem is that they frequently do so in a contradictory manner. A representation that is well-suited for a human user may be hard to handle by a performance system. A representation that is well-suited for a system to perform a certain task may be hard to be understood by a human user. To make things even worse, we also consider the input of the learning algorithm. A learning algorithm A_i can be characterized by its input (i.e. L_{Ei}, L_{Bi}) and output (i.e. L_{Hi}) formats. Those pairs of input and output languages, for which an efficient and effective learning algorithm exists, are called *admissible* languages. In this way, L_H is further constrained by the given data in L_E. It is hard or even impossible to design a representation L_H which is

– efficiently learnable
– from data in a given format,
– easily understood by users,
– and can be put to direct use by a performance system.

Instead of searching for one representation that fulfills at the same time the requirements of the learning algorithm, the given data, the users, and the performance system, we may want to follow a more stepwise procedure. We divide the overall representation problem into several subproblems. Each subproblem consists of two parts, namely determining an appropriate representation and developing a transformation from a given representation into it. In order to not trade in a simple representation language for a complex transformation process, we design families of representations, where each family member is well suited for a particular set of requirements, and efficient transformations between the family members are possible. The subproblems are:

Learnability: The majority of theoretical analyses shows the learnability of concept classes under certain restrictions of the learning task, the representation languages, and the criterion for accuracy. Hence, this subject does not require justification. However, the transformation from a given representation into the desired one raises questions that have not yet been addressed frequently. The transformation here converts L_{Hi} into L_{Hj}, where L_{Hj} is better suited for learning.

Optimizing input data: The hardest task of machine learning applications is the design of a well-suited representation L_{Ej}. On one hand, the given representation *formalism* L_{Ei} must be converted into the one accepted by the learning algorithm as input. On the other hand, the representation *language* (signature) can be optimized. The formalism remaining the same, the set of features (predicates) is changed. The standard procedure is to run the algorithm on diverse feature sets until one is found that leads to an acceptable accuracy of learning. Most feature construction or selection methods aim at overcoming this trial-and-error approach.

Comprehensibility: Although stated as a primary goal of machine learning from the very beginning on, the criterion of comprehensibility never became a hot topic of learning theory. Even a clear, formal definition of the criterion is lacking.

Embeddedness or program optimization: Learning results are supposed to enhance a procedure. Most often, the procedure is already implemented as a particular performance system. The learning result in L_{Hi} must be transformed into the representation L_{Hj} of the performance system. Quite often, the performance system offers restrictions that are not presupposed by the learning algorithm. In this case, the transformation may be turned into an optimization, exploiting the application's restrictions.

In this paper, a family of subsets of Horn logic is discussed and illustrated by a robot application.

2 Learnability of restricted Horn logic

The learnability subject does not need any justification here. However, the transformation of one representation L_{Hi} into another one L_{Hj} which is better suited for learning, does. An important question is, how to recognize those parts in a given representation (or concept class) automatically, that cause negative learnability results. The user may then be asked, whether the learning task could be weakened, or at least be warned that it may take exponential time to compute a result. For instance, detecting variables that occur in the head of a clause but not in its body prohibit the clause being generative. This can easily checked automatically. Even suggestions for fixing the clause can be computed on the basis of the predicates defined so far [25] Alternatively, the learning algorithm may start with a simple representation and increase the complexity of the hypothesis space. Of course, this alternative is easier than the recognition of the complexity class.

Desired are learnability proofs that can be made operational in the sense that indicators for (non-)learnability are defined that can be recognized automatically. One example for an operational complexity criterion is the one used by the support vector machine [29]. The width of the margin separating positive and negative examples corresponds to the VC dimension of the hypothesis space, but can be calculated much easier.[1] According to this operational criterion hypothesis spaces of increasing complexity are tried by the learning algorithm. A rough indication of the complexity of hypotheses from the ILP paradigm is the number of literals in a clause. According to this criterion, the RDT algorithm searches fo increasingly complex hypotheses [8] and a similar procedure is followed in [5]. However, the number of literals is merely a heuristic measure of complexity.

Let me now illustrate operational proofs by [9]. The difficulty of first-order logic induction originates in first-order logic deduction. Hypothesis testing, saturation of examples with background knowledge, the comparison of competing hypotheses, and the reduction of hypotheses under equivalence all use the deductive inference. Since $\theta-$subsumption [23] is a correct and for not self-resolving, non tautological clauses a complete inference, most ILP systems use it.

$\theta-$subsumption: A clause D $\theta-$subsumes a clause C iff there exists a substitution θ such that $D\theta \subseteq C$.

$\theta-$subsumption is NP-complete because of the indeterminism of choosing θ in the general case.

Hence, the clauses need to be restricted. Restricting Horn clauses to k-llocal clauses allows for polynomial learning [3]. William Cohen's proof maps k-llocal clauses on monomials for which learnability proofs exist. However, the proof is not operational. It just shows where it is worth to look for theoretically well-based algorithms. What we need is an operational proof of polynomial complexity of $\theta-$subsumption of k-llocal clauses in order to solve the deductive problem within induction. We then need a learning algorithm exploiting the restriction so that learning k-llocal clauses is polynomial. Desired is an automatic check, whether a clause is k-llocal, or not.

k-llocal Horn clause: Let $D = D_0 \leftarrow D_{DET}, D_{NONDET}$ be a Horn clause, where D_{DET} is the deterministic and D_{NONDET} is the indeterministic part of its body. Let $vars$ be a function computing the set of variables of a clause. $LOC_i \subseteq D_{NONDET}$ is a local part of D, iff
$(vars(LOC_i) \setminus vars(\{D_0, D_{DET}\}))capvars(D_{NONDET} \setminus LOC_i) = \emptyset$ and there does not exist $LOC_j \subset LOC_i$ which is also a local part of D.
A local part LOC_i is k-llocal, iff
there exists a constant k such that $k \geq \mid LOC_i \mid$.

[1] Determining the VC dimension is a problem in $O(n^{O(logn)})$ where n is the size of a matrix representing the concept class [24].

A Horn clause is k-llocal, iff each local part of it is k-llocal.

J"org-Uwe Kietz presents a fast subsumption algorithm for $D = D_0 \leftarrow D_{DET}, LOC_1, ..., LOC_n$ and any Horn clause C [9]. For the deterministic part of D, there exists exactly one substitution θ. An efficient subsumption algorithm for deterministic Horn clauses is presented in [11]. In the indeterministic part of D, it is looked for local parts, i.e. for sets of literals which do not share an indeterministic variable. This is done by merging literals which have a variable in common. In this way, the algorithm checks a given clause D, whether it is k-local, or not. This can be done polynomially. If the local part is bounded by some k, its subsumption is in $O(n \cdot (k^k \cdot \mid C \mid))$. If, however, D_{Body} consists of one large local part, the subsumption is exponential. The proof of polynomial complexity in the number of literals of the clauses D and C corresponds directly to the fast subsumption algorithm. The algorithm guarantees efficient subsumption for k-local D, but can be applied to any Horn clause.

The least general generalization, LGG, of Gordon Plotkin [20] is then adjusted to k-llocal clauses and it is shown that the size of the LGG of k-llocal clauses does not increase exponentially (as does the LGG of unrestricted Horn clauses) and reduction of k-llocal clauses is polynomial, too [9]. Together with the negtive learnability results in [10], k-llocal clauses can be considered the borderline between learnable and not learnable indeterministic clauses. This concludes the illustration of what I mean by operational proofs.

3 Optimizing input data

The transformation of the given data within the same representation formalism is called feature construction/extraction, if new features are built on the basis of the given ones, and feature selection, if the most relevant subset of features is selected. An explicit construction is the invention and definition of new predicates as presented by [19], [27], and [30]. Including the transformation into the learning algorithm is called constructive induction [16], [2]. An implicit adding of new features is performed by the support vector machine [29]. Using a kernel function, the feature space is transformed such that the observations become linearly separable. A more complex input representation allows the algorithm to remain simple. Only because the enriched feature space need not be used for computation– the kernel functions are used –, transformation plus learning algorithm remain efficient. It is an open question how to make explicit, which of the added features contributed most to a good learning result.

Let me now turn to the transformation from one representation formalism to another one. A learning algorithm is selected because of both, its input and output formalisms (plus other properties). It may turn out, however, that there does not exist an algorithm with the desired input–output pair of languages. In this case, one of the options is to transform the data from the formalism in which they are available into the input format of the selected algorithm. The current interest in feature construction may stem from knowledge discovery in databases

(KDD) [14]. The given database representation has to be transformed into one which is accepted by the learning algorithm. Of course, for an ILP learning algorithm there exists a 1:1 mapping from a database table to a predicate[7]. However, this simple transformation most often is not one that eases learning. Since the arity of predicates cannot be changed through learning, a huge number of irrelevant database attributes is carried along. Therefore, the transformation from database tables into predicates should map parts the tables to predicates, where the database key is one of the arguments of the predicate. A tool that gives an ILP learner direct access to a relational database, provides different types of mappings from tables to predicates, and constructs SQL queries for hypothesis testing automatically has been developed [17]. However, the open question remains whether there are theoretically well-based indicators of the optimal mapping for a particular learning task. This is an essential question, since the number of possible mappings is only bounded by the size of the universal database relation. Therefore, it does not help that we may well compute the size of the hypothesis space for each mapping. What is required is a structure in the space of mappings that allows for efficient search.

A frequent task when applying an ILP algorithm is to transform numerical measurements into qualitative Horn logic clauses (facts). A mobile robot, for instance, reports its action and perception by multivariate time series. Each sensor and the moving engine deliver a measurement per moment. The ILP learner requires a description of a path and what has been sensed in terms of time intervals during which some assertions are true. Hence, the signal-to-symbol transformation needs to find adequate time intervals and assertions that summarize the measurements in the time interval. This task is different from time series analysis, where the curves of measurement are approximated. It corresponds to the analysis of event sequences, where events have to be automatically recognized. As opposed to current approaches to learn about event sequences (e.g., [31],[15]), the robot application asks for a representation that can be produced incrementally on-line [13]. For our robot application we have implemented a simple algorithm which constructs predicates of the form

`increasing(MissionID, Angle, Sensor, FromTime, ToTime, RelToMove)`

from measurements [18]. During the time interval

$$\texttt{FromTime}, \texttt{ToTime}$$

, the sensor preceives increasing distance while being oriented in a particular angle with respect to the global coordinates. The relation between measured distance and moved distance is expressed by the last argument. The algorithm reads in a measurement and compares it with the current summarizing assertion (i.e., a predicate with all arguments bound except for `ToTime`). Either `ToTime` is bound, the event has ended and a new one starts, or the next measurement is read in. The language L_E constructed has the nice property of fitting to general chain rules. Given an appropriate substitution σ, a literal B_i of a chain rule can be unified with a ground fact $f \in L_E$, i.e. $B_i\sigma = f$.

General chain rule: Let S be a literal or a set of literals. Let $args(S)$ be a function that returns the Datalog arguments of S. A normal clause is a general chain rule, iff its body literals can be arranged in a sequence
$B_0 \leftarrow B_1, B_2, \cdots B_k, B_{k+1}$
such that there exist Datalog terms $x, z \in args(B_0)$, $x, y_1 \in args(B_1)$, $y_1, y_2 \in args(B_2)$, ..., $y_{k-1} \in args(B_k)$, and $y_k, z \in args(B_{k+1})$.

Obviously, chain rules are well-suited for the representation of event sequences, using the time points as chaining arguments. L_E is constructed such that chain rules can be learned from them. In other words, having chain rules in mind for L_H we transform the given data into facts that correspond to literals of L_H. Note, that this transformation is not motivated by the learning algorithm in order to make learning simpler. The aim is to design L_E such that a L_H can be learned which can easily embedded in the robot application.

Open research questions:

4 Comprehensibility

It is often claimed that learning results are easier to understand than statistical results which demand a human interpreter who translates the results into conclusions which the customer of the statistical study may use. In particular, decision trees and rules were found to be easily undertandable. However, these statements lack justification. Needed are representation changes that preserve the meaning.

In a user-independent way, proposed criteria for comprehensibility most often refer to the length of a description (the minimal description length (MDL) principle [22]). For decision trees, the number of nodes was used as a guideline for comprehensibility. For logic programs, the number of literals of a clause or the number of variables was used as an operational criterion for the ease of understanding [1] [2], [4]. Irene Stahl corrected the MDL for ILP by restricting the description length of examples to positive examples only [26]. Although presented as an operationalisation of comprehensibility, compression may lead to almost incomprehensible descriptions[3]. Its value for hypothesis testing and structuring the hypothesis space not neglected, as a means to achieve comprehensible learning results it is questionable.

Ryszard Michalski has proposed to use natural language for communicating learning results. The ease of transforming L_H into natural language can then be used as a criterion for the naturalness of the representation.

Edgar Sommer introduced the notion of extensional redundancy. Removing extensional compresses a theory, but not all compressions remove redundancy. It is known from psychology that some redundancy eases understanding. Extensional redundancy aims at characterizing superfluous parts of a theory [25].

[2] The reconstruction of the examples and background knowldge from the logic theory and the example encoding by a reference Turing machine leads to a measure of compression: the length of the output tape minus the length of the input tape.

[3] Think of compressed text files!

Extensional redundancy: Let G be the set of goal concepts in a theory T. Let Q be the set of instances of goal concepts that are derivable from T. Let C be a clause in T and L be a literal in C.
A literal L is extensionally redundant in C with respect to $g \in G$, iff C is in the derivation of g and $C \setminus \{L\}$.
A clause C is extensionally redundant in T with respect to Q, iff $T \vdash Q$ and $T \setminus \{C\} \vdash Q$.

Structure is a key to comprehensibility. Structure is achieved by the folding operations which leaves the minimal Herbran modof a theory unchanged [28]. Only empirical evidence could indicate when users prefer a structured theory, and when they prefer an unfolded one. However, as mentioned above, folding/unfolding could be to users so that they have the choice.

fold: Let $C, D \in T_i$ be ordered clauses of the form
$C = C_0 \leftarrow C_1, \cdots, C_m, C_{m+1}, \cdots, C_{m+n}$ and
$D = D_0 \leftarrow D_1, \cdots, D_m$.
Let σ be a substitution satisfying the following conditions

1. $C_i = D_i\sigma, i = 1, ..., m$
2. let $X_1, ..., X_l$ be variables that occur in D_{Body} but not in D_{Head}; each $X_j\sigma, j = 1, ..., l$ does not occur in C_{Head} nor in $C_{m+1}, \cdots, C_{m+n}$; if $i \neq i$, then $X_i\sigma \neq X_j\sigma$.
3. D is the only clause in T_i whose head is unifiable with $D_0\sigma$.

If such a substitution exists, then
$C' = C_0 \leftarrow D_0\sigma, C_{m+1}, \cdots, C_{m+n}$ and
$T_{i+1} = fold(T_i, C, D) = T_i \setminus \{C\} \cup \{C'\}$.
Otherwise, T_i remains unchanged.

The FENDER program folds clauses that define a concept with an intermediate concept [25]. The intermediate concept is made of common partial premises. A common partial premise is a set of literals which most frequently occur in the given theory. The literals must share a variable and it is omitted in the folded clause. However, not all logically unnecessary variables are omitted. This is meant to not hide relevant information from the user, but only encapsulate internal details.

An alternative stratification method named prefix elimination has been developed particularly for chain programs [21], i.e. for better communicating event sequences. As opposed to FENDER which gathers common partial premises regardless of an ordering of literals, the prefix elimination method only replaces common prefixes of chained literals in a clause's body. Hence, applying prefix elimination to a chain program outputs a chain program. The prefix elimination method looks for common literal sequences in all clauses, not only ones that are used to define the same concept. It suppresses all unnecessary variables. This is meant to exhibit the time relations more clearly. An abstracted example from our robot application may illustrate the method.

Example: Let the theory T_i be

$C_1 = alongWall(S, X, Z) \leftarrow stand(V, O, X, Y_1), stable(V, O, Y_1, Y_2), decr_peak(S, O, Y_2, Z)$
$C_2 = alongDoor(S, X, Z) \leftarrow stand(V, O, X, Y_1), stable(V, O, Y_1, Y_2), incs_peak(S, V, Y_2, Z)$
Prefix elimination returns T_{i+1}:
$C_1' = alongWall(S, X, Z) \leftarrow standstable(V, O, X, Y_2), decr_peak(S, O, Y_2, Z)$
$C_2' = alongDoor(S, X, Z) \leftarrow standstable(V, O, X, Y_2), incs_peak(S, V, Y_2, Z)$
$D = standstable(V, O, X, Y_2) \leftarrow stand(V, O, X, Y_1), stable(V, O, Y_1, Y_2)$

Two time intervals are summarized, but no information is abstracted away.

Compression and stratification (structuring by intermediate concepts and and using them for folding) are two operational criteria for comprehensibility of logical theories. Whether they correspond to true needs of users should be studied empirically. Since the representation is to be understood by human users, the answer depends on general cognitive capabilities as well as on user-specific preferences which, in turn, depend on prior knowledge and training. Adjusting a representation to particular preferences is a learning task in its own right. A number of answer set equivalent representations could be presented to the user who selects his favorite one. The selections serve as positive examples and a profile of the user is learned which guides further presentations. Theoretical analysis of the user profiles could well lead to a refinement of compression which excludes incrompehensibly compact theories.

5 Embeddedness or program optimization

Optimizing learning results for their use by a performance system is easier than optimizing them for human understanding, since the requirements of the system are known. In many cases the requirements by users and performance system are conflicting. However, they can both start from the same learning result, if L_H has been chosen carefully. In our robotics application, we used the fact that chain programs correspond to definite finite automata (DFA). Since general chain programs – as opposed to elementary chain rules – are not equivalent to context free grammars, we cannot apply the transformation presented by [6]. The general chain rules can be translated into a context-free grammar, but a transformed grammar cannot be uniquely translated back into general chain rules. Hence, we may theoretically describe the learning results with reference to context-free grammars, but this analysis cannot be put to use. It is possible, however, to transform general chain rules into a DFA. Starting from a theory which has been stratified by prefix elimination, the mapping is as follows [21]:

- A clause head becomes a state of the DFA.
- A clause defining a prefix becomes a transition of the DFA.
- The clause head of an overall goal with its substitutions becomes an output.

The DFA is restricted to successful derivations. Compilation of learned theories consisting of 250 to 470 clauses took between 1 and 5 minutes CPU time [21]. A system using the learned and optimized knowledge was developed by Volker

Klingspor [12]. His system SHARC performs object-recognition, planning, and plan execution. The low-level steps of object recognition are performed in parallel, each sensor having its own process. Object recognition is performed by forward inference on and optimized clauses using a marker passing strategy. Planning and plan execution is performed by backward inference, again, on optimized clauses. The general chain rules are learned by GRDT, an ILP learner with declarative bias [12]. GRDT is not restricted to L_H being general chain rules. The top goals to be learned were moving along a door and moving through a door. Guiding the mobile robot by a joy stick through and along doors in one environment resulted in the training data. The test was that te robot actually moved along or through other doors in a different but similar environment (i.e. the university building). No map was used. Many more experiments have been made using the simulation component of the PIONEER mobile robot. Since we were told before the project, that real-time behavior is impossible on the basis of logic programs, neuro-computing or other numerical processing would be mandatory. However, the real time from sending sensor measurements to SHARC until robot's reaction in almost all cases was below 0.005 seconds and at most 0.006 seconds [12]. This shows that learned clauses can navigate a robot in real time, if they are parallelized and optimized with respect to their use. Applying the learned clauses in their most comprehensible from is not fast enough for a robot application.

6 Conclusion

This paper tries to show that theory need not be restricted to the central learning step, leaving practicians alone with the design and optimization of L_E and L_H. In contrast, developing operational indicators for the quality of a representation asks for theoretical analysis. Concerning the criteria of learnability, comprehensibility, and embeddedness, a brief overview of approaches toward well-based guidelines for the development and transformation of L_E and L_H is given. The tailoring of representations is illustrated by a robot application.

- A general ILP learner with declarative bias, GRDT, outputs general chain clauses, because L_E was made of ground chain facts.
- The original numerical data from the robot were transformed into graund chain facts by a simple procedure which uses parameter adjustment as its learning method.
- A theory can be optimized concerning comprehensibility by introducing intermediate predicates, which are then used for folding.
- A theory made of general chain rules can be optimized for real-time deductive inference using the prefix elimination method and the transformation into DFAs.

References

1. A.Srinivasan, S. Muggleton, and M. Bain. The justification of logical theories based on data compression. *Machine Intelligence*, 13, 1993.

2. Eric Bloedorn and Ryszard Michalski. Data-driven constructive induction: Metodology and applications. In Huan Liu and Hiroshi Motoda, editors, *Feature Extraction, Construction, and Selection – A Data Mining Perpective*, chapter 4, pages 51 – 68. Kluwer, 1998.
3. William W. Cohen. Learnability of restricted logic programs. In *ILP'93 Workshop*, Bled, 1993.
4. D. Conklin and I. Witten. Complexity-based induction. *Machine Learning*, 16:203 – 225, 1994.
5. Luc DeRaedt. *Interactive Theory Revision: an Inductive Logic Programming Approach*. Acad. Press, London [u.a.], 1992.
6. G. Dong and S. Ginsburg. On the decomposition of chain datalog programs into p (left-)linear l-rule components. *Logic Programming*, 23:203 – 236, 1995.
7. Saso Dzeroski. Inductive logic programming and knowledge discovery in databases. In Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, and Ramasamy Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, chapter 1, pages 117–152. AAAI Press/The MIT Press, Menlo Park, California, 1996.
8. J.-U. Kietz and S. Wrobel. Controlling the complexity of learning in logic through syntactic and task–oriented models. In Stephen Muggleton, editor, *Inductive Logic Programming.*, number 38 in The A.P.I.C. Series, chapter 16, pages 335–360. Academic Press, London [u.a.], 1992.
9. Jörg Uwe Kietz. *Induktive Analyse relationaler Daten.* PhD thesis, Technische Universität Berlin, Berlin, oct 1996.
10. Jörg-Uwe Kietz and Saso Dzeroski. Inductive logic programming and learnability. *SIGART–Bulletin*, 5(1):22–32, 1994.
11. Jörg-Uwe Kietz and Marcus Lübbe. An effecient subsumption algorithm for inductive logic programming. In W. Cohen and H. Hirsh, editors, *Proceedings of the 11th International Conference on Machine Learning IML–94*, San Francisco, CA, 1994. Morgan Kaufmann.
12. Volker Klingspor. *Reaktives Planen mit gelernten Begriffen.* PhD thesis, Univ. Dortmund, 1998.
13. P. Laird. Identifying and using patterns in sequential data. In K.Jantke, S. Kobayashi, E. Tomita, and T. Yokomori, editors, *Procs. of 4th Workshop on Algorithmic Learning Theory*, pages 1 – 18. Springer, 1993.
14. H. Liu and H. Motoda. *Feature Extraction, Construction, and Selection: A Data Mining Perspective.* Kluwer, 1998.
15. H. Mannila, H. Toivonen, and A. Verkamo. Discovering frequent episode in sequences. In *Procs. of the 1st Int. Conf. on Knowledge Discovery in Databases and Data Mining.* AAAI Press, 1995.
16. Ryszard S. Michalski and Yves Kodratoff. Research in machine learning and recent progress, classification of methods, and future directions. In Yves Kodratoff and Ryszard Michalski, editors, *Machine Learning – an Artificial Intelligence Approach*, volume III, chapter I, pages 3–30. Morgan Kaufmann, Los Altos, CA, 1990.
17. Katharina Morik and Peter Brockhausen. A multistrategy approach to relational knowledge discovery in databases. *Machine Learning Journal*, 27(3):287–312, jun 1997.
18. Katharina Morik and Stephanie Wessel. Incremental signal to symbol processing. In K.Morik, M. Kaiser, and V. Klingspor, editors, *Making Robots Smarter – Combining Sensing and Action through Robot Learning*, chapter 11, pages 185 –198. Kluwer Academic Publ., 1999.

19. Stephen Muggleton and Wray Buntine. Machine invention of first-order predicates by inverting resolution. In *Proc. Fifth Intern. Conf. on Machine Learning*, Los Altos, CA, 1988. Morgan Kaufman.
20. Gordon D. Plotkin. A note on inductive generalization. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, chapter 8, pages 153–163. American Elsevier, 1970.
21. Anke D. Rieger. *Program Optimization for Temporal Reasoning within a Logic Programming Framework.* PhD thesis, Universit"at Dortmund, Germany, Dortmund, FRG, 1998.
22. J. Rissanen. Modeling by shortest data description. *Automatica*, 14:465 – 471, 1978.
23. J. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, 1965.
24. A. Shinohara. Complexity of computing vapnik-chervonenski dimension. In K.Jantke, S. Kobayashi, E. Tomita, and T. Yokomori, editors, *Procs. of 4th Workshop on Algorithmic Learning Theory*, pages 279 – 287. Springer, 1993.
25. Edgar Sommer. *Theory Restructering: A Perspective on Design & Maintenance of Knowledge Based Systems.* PhD thesis, University of Dortmund, 1996.
26. Irene Stahl. Compression measures in ILP. In Luc De Raedt, editor, *Advances in Inductive Logic Programming*, pages 295–307. IOS Press, 1996.
27. Irene Stahl. Predicate invention in inductive logic programming. In Luc DeRaedt, editor, *Advances in Inductive Logic Programming*, pages 34 – 47. IOS Press, 1996.
28. H. Tamaki and T. Sato. Unfold and fold transformation of logic programs. In *Procs. of 2nd Int. Conf. Logic Programming*, pages 127 – 138, 1984.
29. Vladimir N. Vapnik. *The Nature of Statistical Learning Theory.* Springer, New York, 1995.
30. Stefan Wrobel. *Concept Formation and Knowledge Revision.* Kluwer Academic Publishers, Dordrecht, 1994.
31. Wei Zhang. A region-based approah to discovering temporal structures in data. In Ivan Bratko and Saso Dzeroski, editors, *Proc. of 16th Int. Conf. on Machine Learning*, pages 484 – 492. Morgan Kaufmann, 1999.