ON A KNOWLEDGE-BASED MODEL OF STRUCTURE LEARNING: METHODOLOGY AND IMPLEMENTATION

Dr. Ivan Bruha *) and Dr. Frantisek Franek **)
McMaster University
Department of Computer Science and Systems
Hamilton, Ont., Canada, L8S 4K1

ABSTRACT

The learning system developed by the authors emulates the learning-from-examples strategy with a perfect teacher. The system is able to process patterns represented by structures and yields a class description as a structure. It is similar in its spirit to the INDUCE family of machine learning algorithms developed by Michalski et al. Unlike INDUCE, our model is written in Prolog and is designed for future enhancement to deal with uncertainty. The paper discusses the advantages of this approach together with a concise description of our model as a knowledge-based system. In the conclusion, we discuss possible ways as to extend our (batch) model to a sequential learning system that could process both structural and numerical information within one unit.

1. INTRODUCTION

Learning systems have been investigated for many years and only recently were perceived to be a good means for automatic knowledge acquisition. Architecture of learning systems depends above all on the representation of recognized patterns and the acquired knowledge. There are two major approaches:

1. numerical (statistical, feature) approach;
2. structural approach (a relational structure, semantic network etc.).

Structure-learning systems are one of the useful means for the acquisition of a knowledge. However, most developed structure-learning systems indicate several weaknesses (see e.g. [1]):

- majority of working learning systems are not sequential, i.e. all training patterns have to be stored in the memory of the learning system;
- they are not able to process both structural and numerical data;
- most systems lack processing of uncertainty.

Therefore, we have been looking into possible models that could process both structural and numerical information within the unit. However, as the initial model, we developed and implemented a structure-learning system that is able to process structurally described concepts without any processing of numerical supporting data. Our paper describes this (strictly structural) model of learning.

We view learning as heuristic search through a space of possible class (concept) descriptions [14]. We studied some machine learning algorithms and found out that Hays-Roth's model of learning (program Sprouter, [9]), Vere's inductive learning system (program Thoth, [18]), and the family of INDUCE systems developed by Michalski et al. [1], [5], [13] were most promising vehicles for our project. The structure-learning system we developed uses the concept of 'learning-from-examples', and is similar in its spirit to Michalski's ideas employed in INDUCE. Unlike INDUCE which was implemented in Pascal, our learning system is written in Prolog. The advantage of this approach consists in that

- the inference engine of our learning system utilizes all powerful Prolog utilities like pattern matching, backtracking, list processing;
- the source code is much shorter, readable, and easily changeable;
- no special description language had to be developed because Prolog terms can be used for both pattern and class (concept) description;
- the knowledge base rules that are strictly logical need not be supplied explicitly but the Prolog logic is utilized directly.

2. TERMINOLOGY AND METHODOLOGY

Although we have been trying to follow the terminology and methodology of [8], [13], [15], we have found out that - in some cases - the terminology introduced e.g. by [6], [7], [10], [16] is more adequate and meaningful. Moreover, we are using the language Prolog (following the terminology of [4]) for describing all objects and concepts involved in learning. Therefore, we are introducing this section in order to present our terminology.

We call the objects to be recognized as *patterns* rather than facts, observable statements, situations, or events. As for the sets the patterns are to be grouped into, we prefer the 'old-fashioned' *classes* to *concepts* (but we will use both terms frequently). A set of patterns used for learning (training) which is supplied by a teacher is called *training set*, and the patterns are called *training patterns*. Similarly to [12] we denote the training patterns of the given trained class as *positive* (training) patterns (or examples), those of the other classes as *negative* (training) patterns. Therefore we distinguish two classes: z1 is the class (concept) to be learned (positive patterns), z2 indicates the other class(es) (presented by negative patterns).

As we have already mentioned, any pattern or class will be described by means of Prolog terms. More precisely, a *pattern or class description* is a *conjunction* of elementary descriptions.

149 - 154

An *elementary description* is symbolized by a Prolog structure, i.e. a functor followed by one or more its components, or an operation. All built-in Prolog predicates can be used. To reach the expressive power of Michalski's VL [11], [13], we only added the infix operation `A .. B` (a value in range `A` to `B` inclusive).

For the purposes of the learning algorithm, we distinguish two types of components of elementary descriptions:

- *individuals*, i.e. elementary objects involved in descriptions of both patterns and classes, e.g. `window1` , `window2` , `car1`;
- *properties* of functors such as `size`, `shape`, `color` .

In the same way, we distinguish two types of functors:

- *relational functors*, usually with arity 2 or more, that express relationships between (among) their components (which are individuals), or unary functors that express truthvalue statements, e.g.

```
leftof(window1, window2)
          /* window1 is left to window2 */
clear(b1)
          /*object b1 is clear*/
```

- *attribute functors* that express property values of individuals, e.g.

```
color(b1, red)
          /*individual b1 is of red color*/
length(car1, 3)
          /*length of car1 is 3 units*/
size(w1, 2 ..  3)
          /*size of the individual w1 is within
            the interval 2 to 3 */
```

We say that a class description *covers* a pattern if all elementary descriptions of the given class description match the pattern description. Furthermore, a class description is *consistent* if it does not cover any training pattern from any other class (i.e. negative patterns). A class description is *complete* if it covers all training patterns of the class being learned (i.e. all positive patterns).

Following [1], [13], [15] we define a *characteristic* class description as a complete description that characterizes the class, ideally a largest possible complete description. A *discriminant* class description is a complete and consistent description that discriminates between classes, ideally the smallest possible complete and consistent one.

Similarly, we distinguish two modes of learning:

(i) *Batch mode.* All training patterns (positive and negative) must be read in and stored in the internal memory (database) of the learning system. The class description is formed by processing all training patterns.

(ii) *Incremental mode.* A few training patterns are read in and processed altogether in order to form an initial class description or a relatively small set of such initial descriptions. The batch mode of learning can be used for that. After that, new training patterns are read subsequently and the existing class description(s) is (are) modified (refined) accordingly. There are two additional possibilities of the incremental mode:

(a) *Nonsequential mode.* All training patterns (both the initial ones and the subsequently read) must be kept in the internal memory (database).

(b) *Sequential mode.* The learning system forgets the initial training set as soon as the initial class description(s) is (are) created in the batch mode. The system remembers the latest modification(s) of the class description(s) and one or a few current training patterns. This matches the sequentiality of learning process [7], [10], [17].

3. LEARNING AS A PRODUCTION SYSTEM

Inductive learning can be characterized as a search process in a space of class descriptions. Therefore, we formulate a learning system as a production system [14] with the following parts:

1. *Database* (as a set of facts) represents specific knowledge about the concept (class) to be learned. In our system, the initial database is formed by training set of positive and negative training examples.

2. *Knowledge base* (as a set of production rules) represents general knowledge which is used for finding a required class description. Production rules of the learning system are used to form more general (or less general) class descriptions from the already existing ones. The production rules are invoked either

- in forward regime as *generalization rules*, i.e. a more general class description is created (usually if existing class description is not complete), or
- in backward regime as *specialization rules*, i.e. a more specific (less general) class description is formed (usually when existing class description is not consistent).

3. *Domain-specific knowledge* characterizes the assumptions and constrains related to both pattern and class descriptions. Such specifications cannot be comprised directly in the database since they do not have character of training examples. Nor can they be involved in the knowledge base because they are specific for a given set of problems.

4. *Inference engine* (control system) finds out which production rules are applicable, chooses one among these rules and applies it. More specifically it involves several procedures, among them the interpreter for the forward regime and one for the backward regime.

In the following, we shall have a more detail look at the components of our production system.

Database. The `N`-th training pattern belonging to the class `Z` is represented by a set of elementary descriptions expressed by the Prolog fact

```
pattern_descr(N,Z,D)
```

where `D` is an elementary description for the given training pattern. Note that in real implementation we use a set of facts (elementary descriptions) rather than their conjunction for describing patterns.

As soon as a class description begins to be formed, the elementary class descriptions will be stored in the database as the Prolog facts

```
class.descr(Z,D)
```

where `D` is an elementary description of the class `Z` . Note that in real implementation, a set of facts `class.descr(.,.)` stored in the Prolog database is used for the class description, rather than a conjunction of elementary descriptions.

Domain-specific knowledge of our learning system currently involves the following pieces of knowledge which can be processed by the inference engine:

(a) *Domain of linear functors.* If an attribute functor has an ordered set of its property values then it is called a linear functor; the interval of its property values can be specified by the (meta)functor `linear.functor` . E.g. if we consider the linear functor `size(I,V)` : the size of the individual `I` is the number `V` within the interval 1 and 100 , then we specify

```
linear.functor(size, 2, 1..100).
```

Here 2 depicts the 2nd argument of the functor is its linear property.

(b) *Hierarchical tree of structural functors.* If an attribute functor has a hierarchical set of its property values then it is called a structural functor. E.g. the functor `shape` can have property values such as `oval.window` , `rectangular.window` etc.; these values are 'subsets' (or 'subnodes') of a more general value `window`. Thus a set of property values of a structural functor can be defined by a hierarchical tree structure. We use the following (meta)functor:

```
structural.functor(shape,2).
        /* 2nd arg of shape is structural*/
```

and the Prolog operator `:-` for defining a hierarchical tree

```
            /*hierarchical tree for shape */
shape(X, oval.window) :-         shape(X, window).
shape(X, rectangular.window) :- shape(X, window).
shape(X, window) :-             shape(X, aperture).
shape(X, door) :-                shape(X, aperture).
```

(c) *Relationship among functors.* We can capture some relations of functors used in pattern and class descriptions, e.g. inverse functors:

```
rightof(X,Y) :- leftof(Y,X).
        /*functor rightof is inverse to leftof */
```

symmetric functors:

```
touch(X,Y) :- touch(Y,X).
```

definitions of functors in terms of more general functors and the like.

Knowledge base of our system involves these production rules (see [13]) that we group to two subsets:

(a) *Syntax rules*

1. The *Dropping Condition Rule*: a class description can be generalized by removing an elementary class description.

2. The *Turning Constants Into Variables Rule*: if two (or more) elementary class descriptions with the same functor involve various 'types' of individuals then they can be replaced by a single elementary description with a single individual of a 'general' (i.e. not specified) type.

(b) *Semantic rules*

1. The *Domain Extension Rule*: if two elementary class descriptions are formed by the same linear functor whose property value is `A` and `B` respectively, then they can be replaced by a single elementary class description whose property value is specified by the interval `A .. B` . The same extension rule can be applied if the existing elementary descriptions involve interval(s) already. E.g. if

```
class.descr(z1, size(w1, 2)).
        class.descr(z1, window(w1)).
class.descr(z1, size(w2, 3..4)).
        class.descr(z1, window(w2)).
```

then they can be replaced by

```
class.descr(z1, size(w, 2..4)).
        class.descr(z1, window(w)).
```

2. The *Hierarchy Climbing Rule*: if two elementary class descriptions are formed by the same structural functor whose property value is `A` and `B` respectively, then they can be replaced by a single elementary description whose property value is a more general value of both `A` and `B` . E.g. if

```
class.descr(z1, shape(w, window)).
class.descr(z1, shape(d, door)).
```

then these two elementary class description can be replaced by

```
class.descr(z1, shape(x, aperture)).
```

For the demonstration of the actual representation, let us focus e.g. on the Dropping Condition Rule which has the form [13]

If `D1` *and* `D2` *are elementary descriptions of the class* `z1` *then it can be generalized to:* `D1` *is elementary description of* `z1` *(i.e.* `D2` *is removed)*

In our Prolog-based system the Dropping Condition Rule in forward regime has the form which can be expressed as follows:

```
class.descr(z1,D1), class.descr(z1,D2), D2 \= D1 --->
                erase(class.descr(z1,D2)).
```

Here the operator `--->` denotes a generalization rule (a production rule in the forward regime), `class.descr(Z,D)` means: `D` is an elementary description of the class `Z`. The procedure `erase` is similar to built-in `retract` but it is affected by Prolog backtracking.

The Dropping Condition Rule in reverse order (backward regime) has the form:

```
class.descr(z1,D1), pattern.descr(.,z1,D2) <---
            add(class.descr(z1,D2)).
```

Here the operator `<---` denotes a specialization rule (a production rule in the backward regime), `pattern.descr(N,Z,D)` means: `D` is an elementary description of the `N`-th training pattern of the class `Z`. The procedure `add` is the backtracking-dependent version of `assertz` .

4. TOP LEVEL FLOW CHART OF THE LEARNING ALGORITHM

The rule-based structure-learning algorithm we developed and implemented in Prolog has the following attributes:

- it is a 'learning-from-examples' with a 'perfect' teacher (i.e. the teacher knows the problem to be learned completely, introduces as helpful as possible examples, and does not make mistakes),
- it is a knowledge-acquisition learning (not a skill-refinement one),
- it works in the batch mode,
- it uses both positive and negative training patterns,
- it produces a discriminant class description (either the first description generated, or a minimal one among several class descriptions generated).

Here is the top level flow chart of the learning algorithm:

1. Read in the first training pattern `x1` that has to belong to the given class `z1` (i.e. the first positive training pattern).

2. By decomposing the description of the training pattern `x1` create all possible class descriptions each consisting of one elementary description of `x1` . Insert them to the list `PLAUSIBLE` of plausible class descriptions so that a sublist of relational functors of `x1` will be followed by that of attribute functors. Furthermore, order these sublists according to their arities (the functors with highest arity come first).

3. Read in the rest of the training set, i.e. training patterns belonging to the given class `z1` (positive examples) and those belonging to other classes (negative examples).

4. Form a consistent class description:
    4.1 Take the first consistent class description from the list `PLAUSIBLE` (and remove it from this list).
    4.2 If there is no consistent one in `PLAUSIBLE`, take the first (inconsistent) class description `cd1` from `PLAUSIBLE` and extend it (i.e. add conjunction of one or more elementary descriptions to it) so that it becomes a consistent class description. However, the extended description should not have more than `Ncd` elementary descriptions (`Ncd` is a given threshold).
    4.3 If no consistent extension of the class description `cd1` can be found, or if the extension of `cd1` has more than `Ncd` elementary descriptions, then fail this process of extension; the backtracking is called to consider another extension of `cd1` or to take another plausible class description from the list `PLAUSIBLE` which will be then extended it in the same way.

    Thus, the result of this step is a consistent class description consisting of one elementary description or a conjunction of elementary descriptions.

5. If the above (consistent) class description is not complete generalize it by using the semantic production rules (in forward regime). If a complete generalization cannot be found, the backtracking is invoked to consider another extension of a previous class description, or to take a new plausible class description. Thus, the result is a consistent and complete (i.e. discriminant) description of the given class `z1`.

6. If we require a minimal discriminant class description (among the first `Ncd` generated descriptions where `Ncd` is a priori given integer) then the backtracking is invoked to consider another generalization or extension of an already processed class description, or even to take another plausible description from the list `PLAUSIBLE` which will be processed in the same way. This is done until `Ncd` consistent and complete class descriptions will be generated; afferwards, the minimum class description (i.e. that with the minimum number of functors and their components) will be chosen as the result.

Example. Let us consider two classes: `z1` is the class of houses I would buy, `z2` indicates houses I would not buy. Some of the training patterns are on Fig. 1. E.g. the 1st training pattern of the desired class `z1` can be described in our system as follows:

```
pattern_descr(1, z1, inside(w,d)).
pattern_descr(1, z1, wall(w)).
pattern_descr(1, z1, door(d)).
pattern_descr(1, z1, inside(w,wi)).
pattern_descr(1, z1, window(wi)).
pattern_descr(1, z1, leftof(d,wi)).
pattern_descr(1, z1, size(d,2)).
pattern_descr(1, z1, size(wi,1)).
pattern_descr(1, z1, above(w,r)).
pattern_descr(1, z1, touch(r,w)).
pattern_descr(1, z1, roof(r)).
```

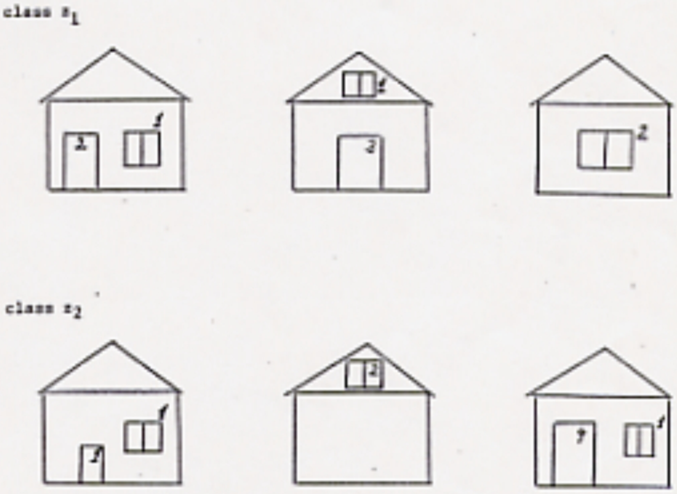
The domain-specific knowledge includes:

(a) Domain of linear functors:

```
linear_functor(size, 2, 1..100).
```

(b) Hierarchical tree of structural functors:

```
oval_window(X) :- window(X).
rectangular_window(X) :- window(X).
window(X) :- aperture(X).
door(X) :- aperture(X).
```



The learning system, after scanning the 1st training pattern of the class `z1`, will form the list of all plausible class descriptions. Since there is no consistent description among them, the system will use the Dropping Condition Rule in backward regime and the Domain Extension Rule in forward regime, and finds the following consistent description of the class `z1`:

```
class.descr(z1, inside(w,x)).
class.descr(z1, wall(w)).
class.descr(z1, door(x)).
class.descr(z1, size(x, 2..3)).
```

However, this description is not complete, therefore the system will generalize it by using the Hierarchy Climing rule:

```
class.descr(z1, inside(w,x)).
class.descr(z1, wall(w)).
class.descr(z1, aperture(x)).
class.descr(z1, size(x, 2..3)).
```

This set represents a discriminant description of the class `z1` .

5. CONCLUSION

The purpose of the development of our learning algorithm has not been, of course, just an implementation of an INDUCE-like system in the programming language Prolog. We chose the INDUCE family [1], [13] among other machine learning systems because it was a most promising vehicle for our project. As a matter of fact, we have followed in principle these ideas of the INDUCE family:

- domain-specific knowledge should be a part of the problem specification for the learning algorithm;
- a discriminant class description is obtained by the completion through generalization of a consistent description;
- we used a reasonable subset of INDUCE's generalization rules.

The principle differences between INDUCE and our learning algoithm are:

1. The entire learning algorithm has been implemented in the programming language Prolog (as a matter of fact, an extension of Prolog, called McPOPLOG [2], [3] is being used). We utilize all built-in Prolog procedures such as pattern matching, backtracking, processing of clauses, structures and lists, etc. The process of finding a discriminant class description depends on these procedures only. Therefore, the inference engine is quite simple and straightforward.

2. We have not developed any language for pattern and class description (such as VL for INDUCE). Rather, we describe patterns and classes by means of Prolog terms exclusively.

3. The knowledge base of our system is not confined to the built-in production (generalization or specialization) rules only. A user of our system can easily add any new rule he/she is convinced that it will help solve the given problem, or speed it up.

4. Last but not least, the main purpose of the development and implementation of our learning algorithm was not the batch-mode system. The entire system and representation of all objects involved had been developed and implemented in such a way as to allow an easy extension to the incremental sequential mode, and allow to process statistical (numerical) data that will be involved in both pattern and class descriptions.

Future research and development will include first of all the extension of our learning system by processing of uncertainty, statistical support of class descriptions, and forgetting/retrieving algorithm. We are going to extend our learning system in such a way as to be capable of processing both structural and numerical data. Building on our experience with processing of uncertainty in general expert systems [19], the extended system will be able

- to process structured patterns together with their (numerical) likelihood (uncertainty) by using a so-called propagation net of likelihoods;
- to form class descriptions comprising both structural and statistical (numerical) attributes, by using reinforcement learning algorithms based on the Stochastic approximation theory for modification of likelihoods;
- to forget and retrieve pieces of knowledge (elementary class descriptions) and thus it becomes a sequential learning algorithm.

ACKNOWLEDGEMENT

*) Research supported by NSERC operating research grant A8034.

**) Research supported by NSERC operating research grant OGP0025112 and by SERB 5-26397 grant.

REFERENCES

[1] Bentrup, J.A. - Mehler, G.J. - Riedesel, J.D.: "INDUCE 4: A program for incrementally learning structural descriptions from examples", Techn. Report UIUCDCS-F-87-958, Univ. of Illinois, Feb. 1987

[2] Bruha, I.: "Reference Manual of McPOPLOG", Techn. Report 87-03, Dept. Computer Science and Systems, McMaster Univ., May 1987

[3] Bruha, I.: "AI multilanguage system McPOPLOG: the power of communication between its subsystems", The Computer Journal (accepted Sept. 1988)

[4] Clocksin, W.F. - Mellish, C.S.: Programming in Prolog. Spring-Verlag, 1984

[5] Dietterich, T.: "INDUCE 1.1 - The program description and a user's guide", Techn. Report, Dept. Computer Science, Univ. Illinois, Urbana, 1978

[6] Duda, R.O. - Hart, P.E.: Pattern Classification and Scene Analysis. Wiley, New York, 1973

[7] Fu, K.S.: Sequential Methods in Pattern Recognition and Machine Learning. Academic Press, New York, 1968

[8] Garbonell, J.G. - Michalski, R.S. - Mitchell, T.M.: "An overview of machine learning", in [12]

[9] Hayes-Roth, F. - McDermott, J.: "An interference matching technique for inducing abstractions", Communications of the ACM, Vol. 21, No. 5, pp. 401-410, 1978

[10] Kotek, Z. - Bruha, I. - Jelinek, J.: Adaptive and Learning Systems. (In Czech) SNTL, Prague, 1980

[11] Michalski, R.S.: "Pattern recognition as rule-guided inductive inference", IEEE Trans. Pattern Analysis and Machine

Intelligence, Vol. PAMI-2, No. 4, pp. 349-361, July 1980

[12] Michalski, R.S. - Carbonell, J.G. - Mitchell, T.M. (eds.): Machine Learning, an Artificial Intelligence Approach. Tioga Publ. Comp., California, 1983

[13] Michalski, R.S.: "A theory and methodology of inductive learning", in [12]

[14] Nilsson, N.J.: Principles of Artificial Intelligence. Tioga Publishing Co., California, 1980

[15] Reinke, R.E. - Michalski, R.S.: "Incremental Learning of Concept Descriptions: A Method and Experimental Results", in: Machine Intelligence 11, ed. D. Michie, 1985

[16] Tsypkin, Ya.Z.: Foundation of the Theory of Learning Systems. Academic Press, New York, 1979

[17] Vapnik, V.N. - Chervonyenkis, A.J.: Theory of Pattern Recognition. (In Russian) Nauka, Moskva, 1974

[18] Vere, S.A.: "Induction of Concepts in the Predicate Calculus", Proc. of Fourth International Joint Conf. Artificial Intelligence (IJCAI), Tbilisi, USSR, pp. 281-7, 1975

[19] Franek, F. - Bruha, I.: "McESE - McMaster expert system environment", Proceedings of ICCI'89, North-Holland, 1989