To appear in *Proc. ICML-97*

Machine Learning by Function Decomposition

Blaž Zupan
Jožef Stefan Institute
Ljubljana, Slovenia
blaz.zupan@ijs.si

Marko Bohanec
Jožef Stefan Institute
Ljubljana, Slovenia
marko.bohanec@ijs.si

Ivan Bratko
Faculty of Computer and
Information Sciences, and
Jožef Stefan Institute
Ljubljana, Slovenia
ivan.bratko@fri.uni-lj.si

Janez Demšar
Faculty of Computer
and Information Sciences
University of Ljubljana
Ljubljana, Slovenia
janez.demsar@fri.uni-lj.si

Abstract

We present a new machine learning method that, given a set of training examples, induces a definition of the target concept in terms of a hierarchy of intermediate concepts and their definitions. This effectively decomposes the problem into smaller, less complex problems. The method is inspired by the Boolean function decomposition approach to the design of digital circuits. To cope with high time complexity of finding an optimal decomposition, we propose a suboptimal heuristic algorithm. The method, implemented in program HINT (HIerarchy Induction Tool), is experimentally evaluated using a set of artificial and real-world learning problems. It is shown that the method performs well both in terms of classification accuracy and discovery of meaningful concept hierarchies.

1 INTRODUCTION

To solve a complex problem, one of the most general approaches is to decompose it into smaller, less complex and more manageable subproblems. In machine learning, this principle is a foundation for structured induction (Shapiro 1987): instead of learning a single complex classification rule from examples, define a goal-subgoal hierarchy and learn the rules for each of the subgoals. Originally, Shapiro used structured induction for the classification of a fairly complex chess endgame and demonstrated that the complexity and comprehensibility ("brain-compatibility") of the obtained solution was superior to the unstructured one. Typically, applications of structured induction involve a manual development of the hierarchy and a manual selection of examples to induce the classification rules; usually this is a tiresome process that requires active availability of a domain expert over long periods of time. Considerable improvements in this respect may be expected from methods that automate or at least actively support the user in the problem decomposition task.

In this paper we present a method for developing a problem decomposition hierarchy from examples and investigate its applicability in machine learning. The method is based on function decomposition, an approach originally developed for the design of digital circuits (Ashenhurst 1952, Curtis 1962). The goal is to decompose a function $y = F(X)$ into $y = G(A, H(B))$, where X is a set of input attributes $x_1, \ldots, x_n$, and y is the class variable. F, G, and H are functions partially specified by examples, i.e., by sets of attribute-value vectors with assigned classes. A and B are subsets of input attributes such that $A \cup B = X$. The functions G and H are determined in the decomposition process and are not predefined in any way. Their joint complexity (determined by some complexity measure) should be lower than the complexity of F. Such a decomposition also discovers a new intermediate concept $c = H(B)$. Since the decomposition can be applied recursively on H and G, the result in general is a hierarchy of concepts. For each concept in the hierarchy, there is a corresponding function (such as $H(B)$) that determines the dependency of that concept on its immediate descendants in the hierarchy.

The proposed decomposition method is limited to nominal-valued attributes and classes. It was implemented in program HINT (HIerarchy Induction Tool). In this paper we do not describe the specific noise handling mechanism in HINT.

The reminder of the paper is organized as follows. Section 2 overviews the related work. The learning method is described in detail in section 3, and experimentally evaluated in section 4 on several domains of different complexity. The paper is concluded by a summary and possible directions of further work.

2 RELATED WORK

The decomposition approach to machine learning was used by a pioneer of artificial intelligence, A. Samuel. He proposed a method based on a signature table system (Samuel 1967) and successfully used it as an evaluation mechanism for his checkers playing programs. This approach was later improved by Biermann et al. (1982). Their method, however, did not address the problem of deriving the structure of concepts.

A similar approach had been defined even earlier within the area of switching circuit design. Ashenhurst (1952) reported on a unified theory of decomposition of switching functions. His decomposition method was essentially the same as that of Biermann et al., except that it was used to decompose a truth table of a specific Boolean function to be then realized with standard binary gates. Most of other related work of those times is reported and reprinted by Curtis (1962).

Recently, the Ashenhurst-Curtis approach was substantially improved by research groups of M. A. Perkowski, T. Luba, and T. D. Ross. Perkowski et al. (1995) report on the decomposition approach for incompletely specified switching functions. Luba (1995) proposes a method for the decomposition of multi-valued switching functions in which each multi-valued variable is encoded by a set of Boolean variables. The authors identify the potential usefulness of function decomposition for machine learning. Goldman et al. (1995) evaluate FLASH, a Boolean function decomposer, on a set of eight-attribute binary functions and show its robustness in comparison with C4.5 decision tree inducer.

Feature discovery has been at large investigated by constructive induction (Michalski 1986). Perhaps closest to the function decomposition method are the constructive induction systems that use a set of existing attributes and a set of predefined constructive operators to derive new attributes (Pfahringer 1994, Ragavan and Rendell 1993).

Within machine learning, there are other approaches that are based on problem decomposition, but where the problem is decomposed by the expert and not discovered by a machine. A well-known example is structured induction (a term introduced by Donald Michie) applied by Shapiro (1987). Their approach is based on a manual decomposition of the problem and an expert-assisted selection of examples to construct rules for the concepts in the hierarchy. In comparison with standard decision tree induction techniques, structured induction exhibits about the same classification accuracy with the increased transparency and lower complexity of the developed models. Michie (1995) emphasized the important role of structured induction and listed several real problems that were solved in this way.

The concept hierarchy has also been used by a multi-attribute decision support expert system shell DEX (Bohanec and Rajkovič 1990). There, a tree-like structure of variables is defined by a domain expert. DEX has been successfully applied in more than 50 realistic decision making problems.

The method presented in this paper therefore borrows from three different research areas: it shares the motivation with structured induction and structured approach to decision support, while the core of the method is based on Ashenhurst-Curtis function decomposition. In comparison with related work, the present paper is original in the following aspects: new method for handling multi-valued attributes and classes, improved decomposition heuristics, emphasis on generalization effects of decomposition, paying strong attention to the discovery of meaningful concept hierarchies, and experimental evaluation on machine learning problems.

3 DECOMPOSITION METHOD

This section presents the decomposition method. First, we introduce the method by an example. Next, we formally present the decomposition algorithm and conclude with a note on the implementation.

3.1 INTRODUCTORY EXAMPLE

Suppose a function $y = F(x_1, x_2, x_3)$ is given where x_1, x_2, and x_3 are attributes and y is the target concept. y, x_1, and x_2 can take the values `lo`, `med`, `hi`; x_3 can take the values `lo`, `hi`. The function F is partially specified with a set of examples in Table 1.

There are three non-trivial partitions of the attributes: $\langle x_1\rangle|\langle x_2, x_3\rangle$, $\langle x_2\rangle|\langle x_1, x_3\rangle$, and $\langle x_3\rangle|\langle x_1, x_2\rangle$, and three corresponding decompositions: $y = G_1(x_1, H_1(x_2, x_3))$, $y = G_2(x_2, H_2(x_1, x_3))$, and $y = G_3(x_3, H_3(x_1, x_2))$. These decompositions are given in Figure 1. The comparison shows that:

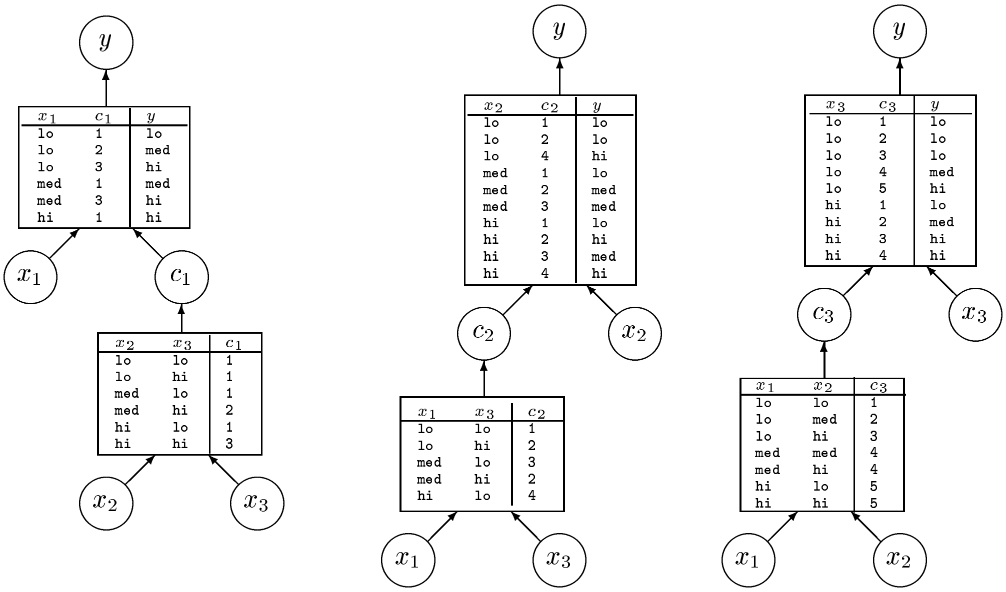


Figure 1: Three different decompositions of the example set from Table 1.

x_1	x_2	x_3	y
lo	lo	lo	lo
lo	lo	hi	lo
lo	med	lo	lo
lo	med	hi	med
lo	hi	lo	lo
lo	hi	hi	hi
med	med	lo	med
med	hi	lo	med
med	hi	hi	hi
hi	lo	lo	hi
hi	hi	lo	hi

Table 1: Set of examples that partially describe the function $y = F(x_1, x_2, x_3)$.

- Example sets in the decomposition $y = G_1(x_1, H_1(x_2, x_3))$ are overall smaller than those for the other two decompositions.
- The new concept $c_1 = H_1(x_2, x_3)$ uses only three values, whereas that for $H_2(x_1, x_3)$ uses four and that for $H_3(x_1, x_2)$ uses five.
- By inspecting the example sets for H_1 and G_1 it is easy to see that c_1 corresponds to $\mathrm{MIN}(x_2, x_3)$ and y to $\mathrm{MAX}(x_1, c_1)$. It is harder to interpret the sets of examples for G_2, H_2, G_3, and H_3.

Among the three attribute partitions it is therefore beneficial to decide for $\langle x_1\rangle|\langle x_2, x_3\rangle$ and decompose $y = F(x_1, x_2, x_3)$ to $y = G_1(x_1, c_1)$ and $c_1 = H_1(x_2, x_3))$.

3.2 SINGLE-STEP DECOMPOSITION

The core of the decomposition algorithm is a *single-step decomposition* which, given a set of examples E_F that partially specify the function $c_i = F(X)$ and a partition of attributes X to sets A and B, decomposes F into $c_i = G(A, c_j)$ and $c_j = H(B)$. This is done by constructing the example sets E_G and E_H that partially specify G and H, respectively. X is a set of attributes $x_1, \ldots, x_m$, and c_j is a new, intermediate concept. A is called a *free set* and B a *bound set*, such that $A \cup B = X$ and $A \cap B = \emptyset$. E_G and E_H are discovered in the decomposition process and are not predefined in any way.

The single-step decomposition starts with the derivation of partition matrix.

Definition 1 Given a disjoint partition of X to $A|B$, a *partition matrix* $\mathcal{P}_{A|B}$ is a tabular representation of example set E_F with all combinations of values of attributes in A as row labels and of B as column labels. Each example $e_i \in E_F$ has its corresponding entry in $\mathcal{P}_{A|B}$ with a row index $A(e_i)$ and a column index $B(e_i)$. $\mathcal{P}_{A|B}$ entries with no corresponding examples in E_F are denoted with "-". A column a of $\mathcal{P}_{A|B}$ is called non-empty if there exists $e_i \in E_F$ such that $B(e_i) = a$.

Each column in the partition matrix denotes the behavior of F when the attributes in the bound set are constant. Columns that exhibit the same behavior are called compatible and can be represented with the same value of c_j. An example partition matrix is given in Figure 2.a.

Definition 2 Columns a and b of partition matrix $\mathcal{P}_{A|B}$ are *compatible* if $F(e_i) = F(e_j)$ for every pair of examples $e_i, e_j \in E_F$ with $A(e_i) = A(e_j)$ and $B(e_i) = a$, $B(e_j) = b$. The number of such pairs is denoted $d(a, b)$.

Note that according to this definition the unspecified $\mathcal{P}_{A|B}$ entries are compatible with any value. The number of values for c_j corresponds to the number of groups of mutually compatible columns. The lowest number of such groups is called *column multiplicity* and denoted by $\nu(A|B)$. It is derived by the coloring of column incompatibility graph.

Definition 3 *Column incompatibility graph* $\mathcal{I}_{A|B}$ is a pair (V, E), where each non-empty column i of $\mathcal{P}_{A|B}$ is represented with a vertex $v_i \in V$, and an edge $(v_i, v_j) \in E$ connects two vertices if the corresponding columns of v_i and v_j are incompatible.

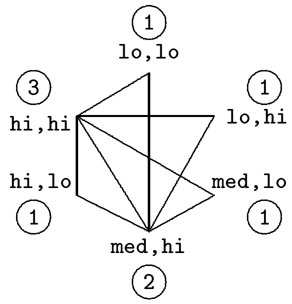
Then, $\nu(A|B)$ is the number of colors needed to color $\mathcal{I}_{A|B}$. Namely, the proper coloring guarantees that two vertices representing incompatible columns are not assigned the same color. The same colors are only assigned to the columns that are compatible. Therefore, the optimal coloring discovers the lowest number of groups of compatible $\mathcal{P}_{A|B}$ columns. An example of colored incompatibility graph is given in Figure 2.b.

Graph coloring is an NP-hard problem and the computation time of an exhaustive search algorithm is prohibitive even for small graphs with about 15 vertices. Instead, Perkowski et al. (1995) suggested a Color Influence Method of polynomial complexity and showed that the method performed well compared to the optimal algorithm. The Color Influence Method sorts the vertices to color by their decreasing connectivity and then assigns to each vertex a color that is different from the colors of its neighbors so that a minimal number of colors is used. We use the same coloring method, with the following improvement: when a color is to be assigned to vertex v and several compatible vertices have already been colored with different colors, the color is chosen that is used for a group of colored vertices $v_1, \ldots, v_k$ that are *most compatible* to v. The degree of compatibility is estimated as $\sum_1^k d(v, v_i)$ (see Definition 2 for d).

Each vertex in $\mathcal{I}_{A|B}$ denotes a distinct combination of values of attributes in B, and its label (color) denotes the value of c_j. It is therefore straightforward to derive an example set E_H from the colored $\mathcal{I}_{A|B}$. Attribute set for these examples is B. Each vertex in $\mathcal{I}_{A|B}$ is an example in set E_H. Color c_j of the vertex is the class of the example.

x_2	lo	lo	med	med	hi	hi
x_1 x_3	lo	hi	lo	hi	lo	hi
lo	lo	lo	lo	med	lo	hi
med	-	-	med	-	med	hi
hi	hi	-	-	-	hi	-
c_1	1	1	1	2	1	3

(a)



(b)

Figure 2: Partition matrix with column labels (c_1) for the attribute partition $\langle x_1\rangle|\langle x_2, x_3\rangle$ and set of examples from Table 1 (a) and corresponding column incompatibility graph (b). Colors (labels) of the vertices are circled.

E_G is derived as follows. For any value of c_j and combination of values of attributes in A, $c_i = G(A, c_j)$ is determined by looking for an example e_i in row $A(e_i)$ and in any column labeled with the value of c_j. If such example exists, an example with attribute set $A \cup \{c_j\}$ and class $c_i = F(e_i)$ is added to E_G.

Decomposition generalizes every undefined ("-") entry of $\mathcal{P}_{A|B}$ in row a and column b, if a corresponding example e_i with $a = A(e_i)$ and column $B(e_i)$ with the same label as b is found. For example, an entry $\mathcal{P}_{A|B}$[<hi>,<lo,hi>] of partition matrix in Figure 2.a was generalized to hi because the column <lo,hi> has the same label as columns <lo,lo> and <hi,lo>.

In our implementation, the incompatibility graph is constructed directly from the set of examples, avoiding the construction of partition matrix for efficiency reasons. The algorithm first sorts the examples E_F based on the values of attributes in A and values of c_i. The

Input: Initial set of examples describing a single output concept
Output: Its hierarchical decomposition

get an initial example set E_{F0} and mark it decomposable
$j \leftarrow 1$
while $\exists$ decomposable example set E_{Fi} that partially specifies $c_i = F_i(x_1, \ldots, x_m)$ with $m > 2$ **do**
  evaluate all possible partitions $A|B$ of $X = \langle x_1, \ldots, x_m \rangle$ such that $A \cup B = X$, $A \cap B = \emptyset$, and $||B|| \leq b$
  select the best partition $A|B$
  if E_{Fi} is decomposable using $A|B$ **then**
    decompose E_{Fi} to E_G and E_H, such that $c_i = G(A, c_j)$ and $c_j = H(B)$, where G and H are partially specified by E_G and E_H
    mark E_G and E_H decomposable
    $j \leftarrow j + 1$
  else mark E_{Fi} non-decomposable

Algorithm 1 The decomposition algorithm

examples with the same $A(e_i)$ constitute groups that correspond to rows in partition matrix $\mathcal{P}_{A|B}$. Within each group, examples with the same value of c_i constitute subgroups. Two examples that are in the same group but in different subgroups have a corresponding edge in $\mathcal{I}_{A|B}$.

3.3 DECOMPOSITION ALGORITHM

The decomposition aims to discover a hierarchy of concepts described with example sets that are overall less complex than the initial one. Since an exhaustive search is prohibitively complex, the decomposition uses a suboptimal iterative algorithm (Algorithm 1).

In each step the algorithm tries to decompose a single example set of the evolving structure. It evaluates all possible disjoint partitions of the attributes and selects the best one. This step requires a so-called *partition selection measure.* A possible measure is the number of values of the new concept $\nu(A|B)$. The best partition $A|B$ is the one with the lowest $\nu(A|B)$.

An alternative measure for the selection of partitions is based on the complexity of function F. Let F be defined on attributes $x_i \in X_F$ with class variable y_F. In this attribute-class space, there are a total of $N(X_F, y_F) = ||y_F||^{\prod_{x_i \in X_F} ||x_i||}$ possible functions, where $||y_F||$ and $||x_i||$ represent the cardinalities of value sets of y_F and x_i, respectively. The number of bits to encode F is therefore $\Theta(F) = \log_2 N(X_F, y_F) = (\log_2 ||y_F||) \prod_{x_i \in X_F} ||x_i||$. Decomposition prefers to discover functions of low complexity, so the measure is therefore defined as $\Theta(A|B) = \Theta(G) + \Theta(H)$.

The decomposition algorithm will decompose E_F and the function F it partially represents only if its decomposed functions G and H are overall less complex than F. Therefore, the partition $A|B$ can be used to decompose E_F to E_G and E_H if and only if $\Theta(A|B) < \Theta(F)$. We say that example set E_F is decomposable if there exists a partition $A|B$ with this property.

3.4 COMPLEXITY OF DECOMPOSITION ALGORITHM

The time complexity of single step decomposition of E_F to E_G and E_H, which consists of sorting of E_F, deriving the incompatibility graph and coloring it, is $O(N \log N) + O(Nk) + O(k^2)$ where N is the number of examples in E_F and k is the number of vertices in $\mathcal{I}_{A|B}$. For any bound set B, the upper bound of k is $k_{max} = (\max_{x_i \in X} ||x_i||)^b$ where $b = ||B||$. The number of disjoint partitions considered by decomposition when decomposing E_F with m attributes is

$$\sum_{j=2}^{b} \binom{m}{j} \leq \sum_{j=2}^{b} \left(\frac{\mathrm{e}m}{j}\right)^j = O(m^b)$$

The highest number of $n-2$ decompositions is required when the hierarchy is a binary tree, where n is the number of attributes in the initial example set. The running time of the decomposition algorithm is thus

$$O\Big((N \log N + N k_{max} + k_{max}^2) \sum_{m=3}^{n} m^b\Big) = $$
$$= O\Big(n^{b+1}(N \log N + N k_{max} + k_{max}^2)\Big)$$

Therefore, the algorithm's complexity is polynomial in N, n, and k_{max}. Note that the bound b is a user-defined constant. This analysis clearly illustrates the benefits of setting b to a sufficiently low value. In our experiments, b was set to 3.

3.5 IMPLEMENTATION

The machine learning method based on function decomposition was implemented in the C language as a system called HINT (Hierarchy INduction Tool). The system runs on several UNIX platforms, including HP-UX, SGI Iris, and SunOS. The definition of domain names and examples, and the guidance of the decomposition is managed through a script language.

4 EXPERIMENTAL EVALUATION

We experimentally evaluated the decomposition method using the following datasets:

MM4 A function $y = \text{MIN}(x_1, \text{AVG}(x_2, \text{MAX}(x_3, x_4), x_5))$ with 4-valued attributes and class. While the definition of MIN and MAX is standard, the function AVG computes the average of its arguments and rounds it to the closest integer.

LENSES A small domain taken from UCI machine learning repository (Murphy and Aha 1994). Using patient age, spectacle prescription, astigmatism, and tear production rate each example describes whether the patient should wear soft or hard contact lenses or no lenses at all.

MONK1 and **MONK2** Well-known six-attribute binary classification problems taken from the same repository (Murphy and Aha 1994, Thrun et al. 1991). Attributes are 2 to 4-valued. MONK1 has an underlying concept $(x_1 = x_2)$ OR $x_5 = 1$ and MONK2 the concept $x_i = 1$ for exactly two choices of $i \in \{1, \ldots, 6\}$.

CAR and **NURSERY** For these two domains hierarchical classifiers in DEX (Bohanec and Rajkovič 1990) formalism already existed. These were used to obtain a set of examples from which decomposition tried to reconstruct the original hierarchies. CAR evaluates cars based on their price and technical characteristics. This simple model was developed for educational purposes and is described in (Bohanec and Rajkovič 1988). NURSERY is a real-world model developed to rank applications for nursery schools (Olave et al. 1989).

The original datasets are noiseless. They completely cover the attribute space for all domains other than MONK1 and MONK2, where the coverage is 28.7% and 39.1%, respectively. Some other domain characteristics are given in Table 2.

The decomposition used column multiplicity as a partition selection measure. When the Θ complexity measure was used instead, the results were similar and are not shown here.

The bound set size b was limited to the maximum of three elements. The decomposition times on HP J210 workstation were all below 2 seconds for all the domains other than NURSERY, for which HINT required about 20 seconds for the largest training sets.

The experimental evaluation addressed the classification accuracy of HINT and its ability to derive a comprehensible and meaningful structure, possibly similar to the anticipated one. The classification accuracy learning curves were computed, where the datasets were split to training and test sets of sizes p and $1-p$, respectively, for p from 10% to 90%. HINT derived a concept hierarchy and corresponding classifier using the examples in the training set and was tested for classification accuracy on the test set. For each p, the results are the average of 10 randomly chosen splits. The learning curve is compared to the one obtained by C4.5 inductive decision tree learner (Quinlan 1993) run on the same data. C4.5 used the default options except for `-m1`, which was observed to obtain a better classification accuracy than the default `-m2`. Accuracy is measured on unpruned decision trees for the same reason. For each p, the significance of the difference between C4.5 and HINT is determined using a paired t-test with $\alpha = 0.01$ (99% confidence level).

Domain	n	N	Class names and their relative frequencies
MM4	4	1024	`0`/0.27, `1`/0.42, `2`/0.29, `3`/0.02
LENSES	4	24	`hard`/0.17, `soft`/0.21, `no`/0.62
MONK1	6	124	`0`/0.5, `1`/0.5
MONK2	6	169	`0`/0.621, `1`/0.379
CAR	6	1728	`unacc`/0.70, `acc`/0.22, `good`/0.04, `v-good`/0.04
NURSERY	8	12960	`unacc`/0.33, `acc`/0.0001, `v-acc`/0.03, `prior`/0.33, `h-prior`/0.31

Table 2: Some characteristics of domains used in the experiments. n is the number of attributes and N the dataset size.

The learning curves are given in Figure 3. For all the domains other than LENSES, HINT outperforms C4.5. With more than 20% of examples in the training set, this difference is always significant. Moreover, HINT's learning curves converge faster to the desired 100%, which is in turn never reached by C4.5. For LENSES, there are no significant differences in the classification accuracy of the two learners. It is also interesting to note that in MM4 C4.5's accuracy decreases with higher coverage of example space, which may be explained with decreased generalization.

HINT was further tested on the data sets for MONK1 and MONK2 used in the detailed study of 25 machine learning algorithms (Thrun et al. 1991). For both MONK1 and MONK2, the training set was the same as our original data set described above. The two test sets used in the study consisted of 432 examples that completely covered the attribute space. For MONK1, the accuracy of HINT is 100%. In the study (Thrun et al. 1991), this score was achieved by 9 learners: three variants of AQ17, Assistant Professional, mFOIL, CN2, two variants of Backpropagation,

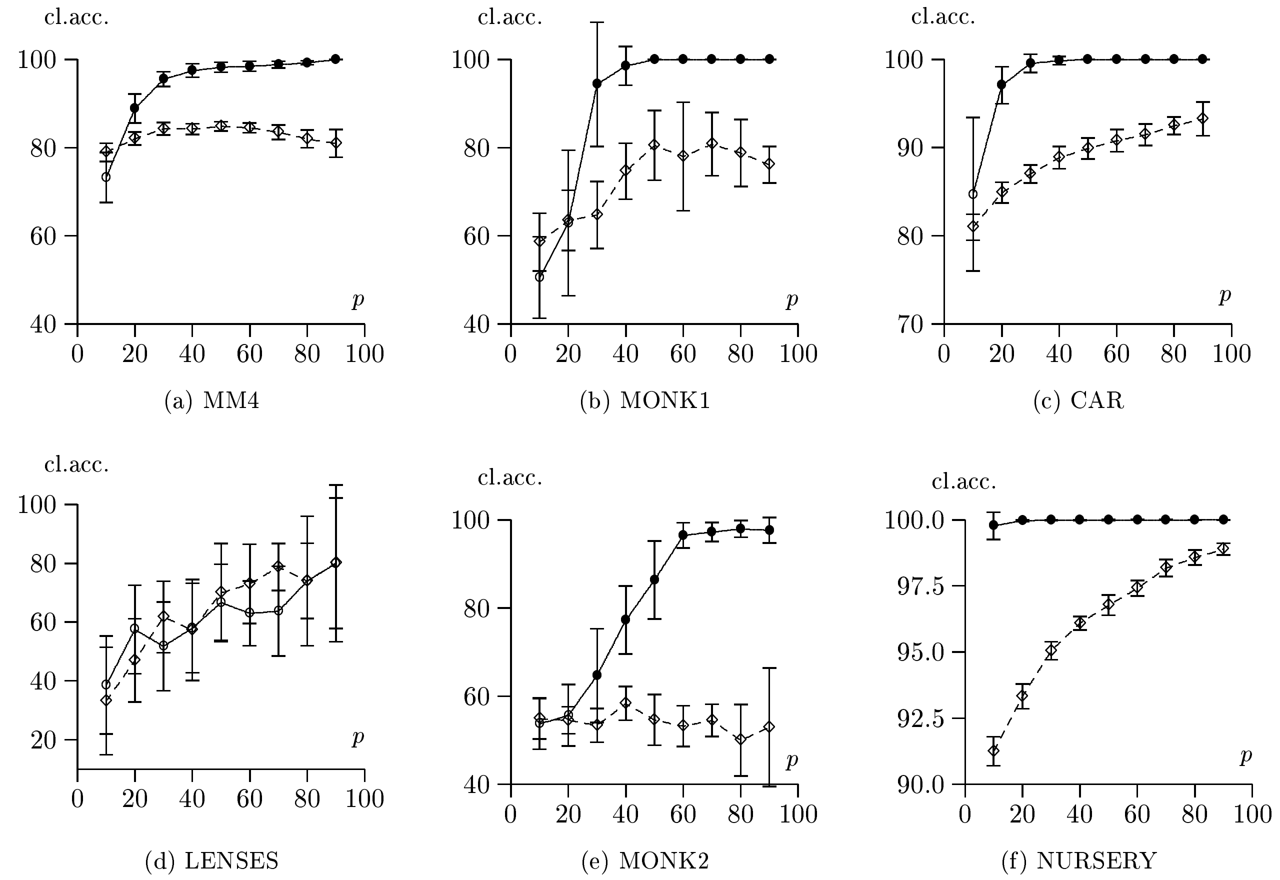


Figure 3: Learning curves for HINT (solid line with $\circ$) and for C4.5 (dashed line with $\diamond$). When, for a specific relative training set size p, the classification accuracy of HINT is significantly better than that of C4.5, HINT's data points are marked with $\bullet$.

and Cascade Correlation. For MONK2, the accuracy of HINT is 97.7%. In the same study, four learners performed better: AQ17-DCI, two variants of Backpropagation, and Cascade Correlation. It should be noted that these results were obtained by HINT without tuning in less than 0.3 seconds of CPU time on HP J210 workstation.

For each of the domains and with increasing p, HINT converged to a single concept structure. These are shown in Figures 4 to 6, with the names of attributes and concepts, and cardinality of their value sets. For MM4, this is the anticipated structure except for the concept $\mathrm{AVG}(x_2, \mathrm{MAX}(x_3, x_4), x_5))$, which HINT additionally decomposed by introducing an intermediate concept c_3. For MONK1, HINT discovered the anticipated hierarchy MONK1 $= F_1(c_1, x_5)$ $c_1 = F_2(x_1, x_2)$ with F_1 and F_2 matching the expected disjunctive and equality functions. For MONK2, because of disjunctive condition on a bound and free set it was impossible to derive concepts comparable to the original concept definition. However, the discovered concept hierarchy is a reformulation of the target concept using functions that count 1's. For LENSES, HINT discovered the structure in Figure 4 which we did not try to interpret without the domain expert. For CAR and NURSERY (Figures 5 and 6), the structures discovered were very similar to the original DEX models. In fact, they were the same except that some original DEX intermediate concepts were further decomposed. It should be emphasized that we consider this similarity of concept structures as a most significant indicator of success of our decomposition-based learning method.

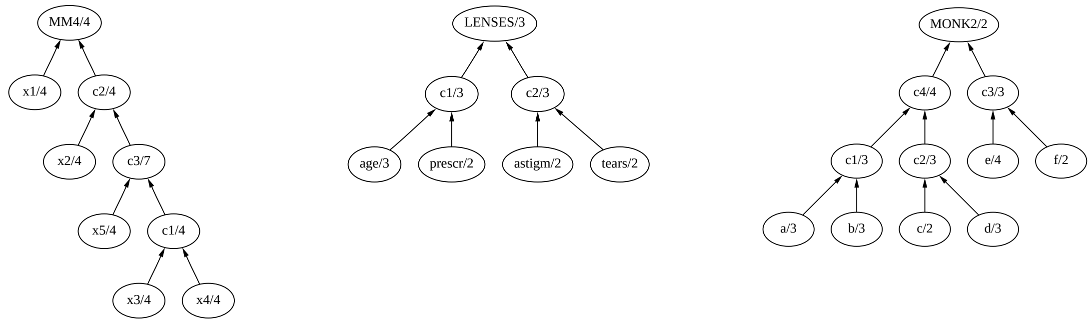


Figure 4: Structures discovered for MM4, LENSES, and MONK2 domains.

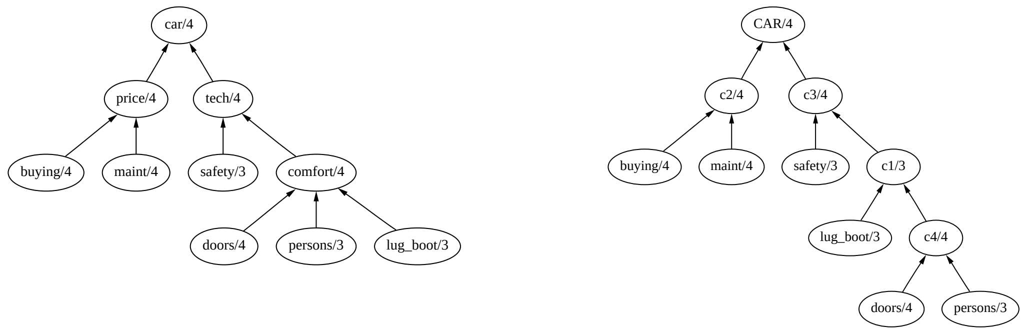


Figure 5: Original (left) and discovered structure (right) for CAR

5 CONCLUSION

We introduced a new machine learning approach based on function decomposition. A distinguishing feature of this approach is its capability to discover new, intermediate concepts, organize them into a hierarchical structure, and define the relationships between the attributes, newly discovered concepts, and target concept. In their basic form, these relationships are specified by newly constructed example sets. In a way, the learning process can thus be viewed as a process of generating new, equivalent example sets, which are consistent with the original example set. The new sets are smaller, have smaller number of attributes, and introduce intermediate concepts. Generalization also occurs in this process.

We have evaluated the decomposition-based learning method on six datasets. In terms of classification accuracy, the decomposition significantly outperformed C4.5 in all but one domain. The examples also show that the decomposition is useful for discovery of new intermediate concepts. For example, the decomposition was able to discover an appropriate concept hierarchy approved by domain experts for a rather complex real-world NURSERY domain.

The classification accuracy results may be biased because we have mostly used the domains where we anticipated the hierarchies discoverable by decomposition. However, MONK2 is a counter example where decomposition was not able to discover the original definition of the target concept, but rather unexpectedly its reformulation.

The decomposition approach as presented in this paper is limited by that there is no special mechanism for handling noise and continuous attributes. However, preliminary results on using an extended version of decomposition for continuously-valued data sets in (Demšar et al. 1997) and preliminary results on noise-handling extension strongly encourage further developments in this direction.

References

Ashenhurst, R. L.: 1952, The decomposition of switching functions, *Technical report*, Bell Laboratories BL-1(11), pages 541–602.

Biermann, A. W., Fairfield, J. and Beres, T.: 1982, Signature table systems and learning, *IEEE Trans. Syst. Man Cybern.* **12**(5), 635–648.

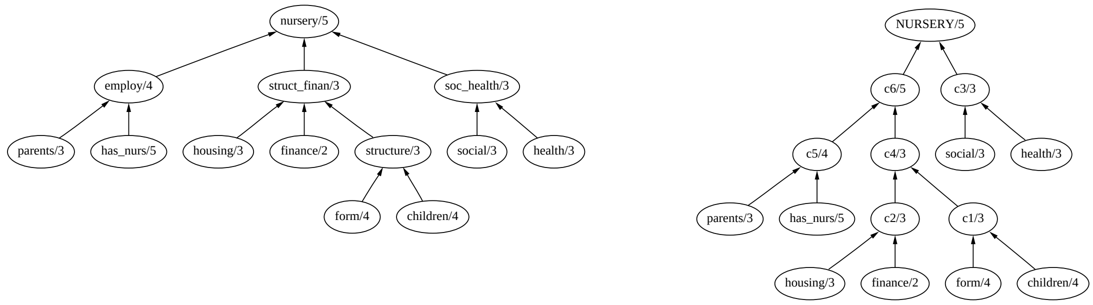


Figure 6: Original (left) and discovered structure (right) for NURSERY

Bohanec, M. and Rajkovič, V.: 1988, Knowledge acquisition and explanation for multi-attribute decision making, *8th Intl Workshop on Expert Systems and their Applications*, Avignon, France, pp. 59–78.

Bohanec, M. and Rajkovič, V.: 1990, DEX: An expert system shell for decision support, *Sistemica* **1**(1), 145–157.

Curtis, H. A.: 1962, *A New Approach to the Design of Switching Functions*, Van Nostrand, Princeton, N.J.

Demšar, J., Zupan, B., Bohanec, M. and Bratko, I.: 1997, Constructing intermediate concepts by decomposition of real functions, *Proc. European Conference on Machine Learning, ECML-97*, Prague.

Goldman, J. A., Ross, T. D. and Gadd, D. A.: 1995, Pattern theoretic learning, *AAAI Spring Symposium Series: Systematic Methods of Scientific Discovery*.

Luba, T.: 1995, Decomposition of multiple-valued functions, *25th Intl. Symposium on Multiple-Valued Logic*, Bloomigton, Indiana, pp. 256–261.

Michalski, R. S.: 1986, Understanding the nature of learning: issues and research directions, *in* R. Michalski, J. Carbonnel and T. Michell (eds), *Machine Learning: An Artificial Intelligence Approach*, Kaufmann, Los Atlos, CA, pp. 3–25.

Michie, D.: 1995, Problem decomposition and the learning of skills, *in* N. Lavrač and S. Wrobel (eds), *Machine Learning: ECML-95*, Notes in Artificial Intelligence 912, Springer-Verlag, pp. 17–31.

Murphy, P. M. and Aha, D. W.: 1994, UCI Repository of machine learning databases `[http://www.ics.uci.edu/~mlearn/mlrepository.html]`. Irvine, CA: University of California, Department of Information and Computer Science.

Olave, M., Rajkovič, V. and Bohanec, M.: 1989, An application for admission in public school systems, *in* I. T. M. Snellen, W. B. H. J. van de Donk and J.-P. Baquiast (eds), *Expert Systems in Public Administration*, Elsevier Science Publishers (North Holland), pp. 145–160.

Perkowski, M. A. et al.: 1995, Unified approach to functional decompositions of switching functions, *Technical report*, Warsaw University of Technology and Eindhoven University of Technology.

Pfahringer, B.: 1994, Controlling constructive induction in CiPF, *in* F. Bergadano and L. D. Raedt (eds), *Machine Learning: ECML-94*, Springer-Verlag, pp. 242–256.

Quinlan, J. R.: 1993, *C4.5: Programs for Machine Learning*, Morgan Kaufmann Publishers.

Ragavan, H. and Rendell, L.: 1993, Lookahead feature construction for learning hard concepts, *Proc. Tenth International Machine Learning Conference*, Morgan Kaufman, pp. 252–259.

Samuel, A.: 1967, Some studies in machine learning using the game of checkers II: Recent progress, *IBM J. Res. Develop.* **11**, 601–617.

Shapiro, A. D.: 1987, *Structured induction in expert systems*, Turing Institute Press in association with Addison-Wesley Publishing Company.

Thrun, S. B. et al.: 1991, A performance comparison of different learning algorithms, *Technical report*, Carnegie Mellon University CMU-CS-91-197.