Contributing Authors

Marko Bohanec (*marko.bohanec@ijs.si*) is a research associate at the Department of Intelligent Systems, J. Stefan Institute, Ljubljana, Slovenia. His research and development interests are in decision support systems and machine learning. He has published in journals such as *Machine Learning*, *Acta Psychologica*, and *Information & Management*.

Ivan Bratko (*bratko@fri.uni-lj.si*) is professor of computer science at the Faculty of Computer and Information Science, Ljubljana University, Slovenia. He heads the AI laboratories at J. Stefan Institute and the University. He is the chairman of ISSEK, International School for the Synthesis of Expert Knowledge. He has conducted research in machine learning, knowledge-based systems, qualitative modeling, intelligent robotics, heuristic programming and computer chess. He is the author of widely adopted text PROLOG Programming for Artificial Intelligence (Addison-Wesley).

Janez Demšar (*janez.demsar@fri.uni-lj.si*) is a research assistant at the Faculty of Computer and Information Science, University of Ljubljana, Slovenia. He has graduated in computer science at the same faculty. His research interest include machine learning and intelligent data analysis.

Blaž Zupan (*blaz.zupan@ijs.si*) is a researcher at the Department of Intelligent Systems, J. Stefan Institute, Ljubljana, Slovenia. His research interests include machine learning, data mining, medical decision making, and medical informatics.

1 FEATURE TRANSFORMATION BY FUNCTION DECOMPOSITION

Blaž Zupan [1], Marko Bohanec [1],
Janez Demšar [2], and Ivan Bratko [2,1]

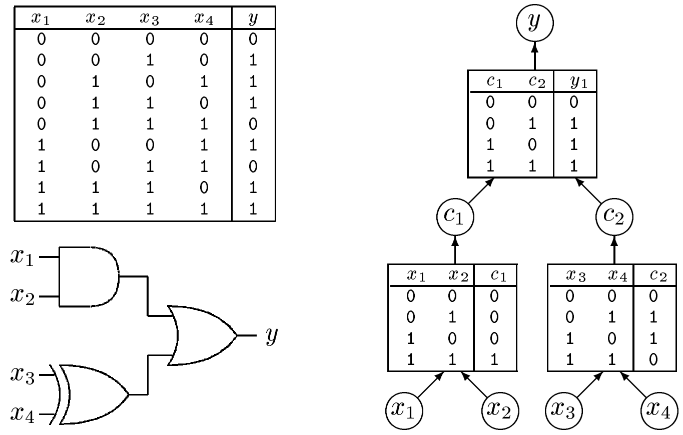
[1] J. Stefan Institute
Jamova 39, 1000 Ljubljana, Slovenia
blaz.zupan@ijs.si, marko.bohanec@ijs.si

[2] Faculty of Computer Science and Informatics
University of Ljubljana
Tržaška 25, 1000 Ljubljana, Slovenia
janez.demsar@fri.uni-lj.si, bratko@fri.uni-lj.si

Abstract: Function decomposition is a method that decomposes a dataset to an equivalent hierarchy of less complex datasets. In this chapter we show that function decomposition can contribute to feature transformation as a method for (1) discovery of a hierarchy of new features and (2) construction of features to be added to the original dataset to improve machine learning from that dataset. A particular advantage of function decomposition is its capability to identify appropriate subsets of existing features and discover functions that map each subset to a new feature. These functions are induced from examples and are not predefined in any way.

1.1 INTRODUCTION

While not explicitly intended for feature transformation, some methods for switching circuit design implicitly deal with this problem. In 1950s and 1960s, (Ashenhurst, 1952) and (Curtis, 1962) proposed a *function decomposition* method that, given a truth table of a Boolean function, develops a hierarchy of less complex tabulated Boolean functions that are preferably realizable with simple logic gates. For example, such decomposition was used to find the circuit of the Boolean function in Figure 1.1. Both the hierarchy and the functions themselves are discovered by the decomposition method and are not given in advance. This



x_1	x_2	x_3	x_4	y
0	0	0	0	0
0	0	1	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	1	1
1	0	1	1	0
1	1	1	0	1
1	1	1	1	1

c_1	c_2	y_1
0	0	0
0	1	1
1	0	1
1	1	1

x_1	x_2	c_1
0	0	0
0	1	0
1	0	0
1	1	1

x_3	x_4	c_2
0	0	0
0	1	1
1	0	1
1	1	0

Figure 1.1 A truth table of a Boolean function (upper left), its switching circuit implementation (lower left), and corresponding feature hierarchy with two intermediate features $c_1 = x_1$ AND x_2 and $c_2 = x_3$ XOR x_4, and the target concept $y = c_1$ OR c_2.

is especially important from the viewpoint of feature construction, since the outputs of such functions (e.g., c_1 and c_2 from Figure 1.1) can be regarded as new features not present in the original problem description.

The basic principle of function decomposition is the following. Let a tabulated function $y = F(X)$ use a set of input features $X = x_1, \dots, x_n$. The goal is to decompose this function into $y = G(A, H(B))$, where A and B are subsets of features in X such that $A \cup B = X$. G and H are tabulated functions that are determined by the decomposition and are not predefined. Their joint complexity, determined by some complexity measure, should be lower than the complexity of F. Such a decomposition also discovers a new feature $c = H(B)$. Since the decomposition can be applied recursively on H and G, the result in general is a hierarchy of features. For each feature in the hierarchy, there is a corresponding tabulated function, such as $H(B)$, that determines the dependency of that feature on its immediate descendants in the hierarchy.

Ashenhurst-Curtis decomposition was intended for switching circuit design of completely specified Boolean functions. Recently, the decomposition was extended to handle incompletely specified Boolean functions (Perkowski and Uong, 1987; Wan and Perkowski, 1992). In the framework of feature extraction, (Ross et al., 1994b) used a set of simple Boolean functions to show the decomposition's capability to discover and extract useful features. This chapter presents an approach to feature transformation that is based on the extension of the function decomposition method by (Zupan et al., 1997). This method allows the decomposition to deal with functions that involve nominal (i.e., not necessarily binary) features, and is implemented in a system called HINT(Hierarchy INduction Tool). Although HINT also supports noise handling (Zupan, 1997), this chapter focuses only on function decomposition mechanisms that do not explicitly deal with noise.

This chapter first introduces two basic components of function decomposition: single-step decomposition and search for the best attribute partition. Next, we show how these components can contribute to three feature transformation and selection tasks: (1) identification and removal of redundant features, (2) discovery of a hierarchy of new features and (3) construction of features to be added to the original dataset. The chapter concludes with an overview of related work and summary.

1.2 SINGLE-STEP FUNCTION DECOMPOSITION

The core of function decomposition method is a *single-step decomposition* which decomposes a tabulated function $y = F(X)$ into two possibly less complex tabulated functions G and H, so that $y = G(A, c)$ and $c = H(B)$. The resulting functions G and H have to be consistent with F. For the purpose of decomposition, a set of features X is partitioned to two disjoint subsets A and B, referred to as a *free* and *bound set*, respectively. For a given feature partition, the single-step decomposition discovers a new feature c and a tabular representation of H and G.

For example, consider a function $y = F(x_1, x_2, x_3)$ as given in Table 1.1. The input feature x_1 and the output feature y can take the values `lo`, `hi`; input features x_2 and x_3 can take the values `lo`, `med` and `hi`.

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

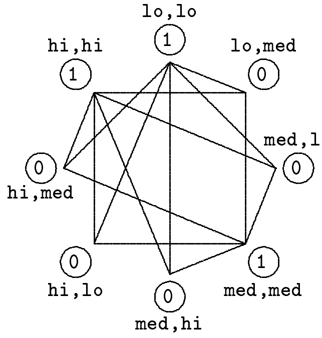
Table 1.1 Tabulated function $y = F(x_1, x_2, x_3)$.

Suppose that we want to discover a description of a new feature $c = H(x_2, x_3)$. For this purpose, we represent F by a *partition matrix* that uses the values of x_1 for row labels, and the combinations of values of x_2 and x_3 for column labels (Figure 1.2.a). Partition matrix entries with no corresponding instance in the tabulated representation of F are denoted with "-" and treated as *don't-care.*

Each column in the partition matrix denotes the behavior of F for a specific combination of x_2 and x_3. Columns that have pairwise equal row entries or at least one row entry is a don't-care are called *compatible.* The decomposition

x_2	lo	lo	med	med	med	hi	hi	hi
x_1 x_3	lo	med	lo	med	hi	lo	med	hi
lo	hi	lo	lo	hi	lo	lo	lo	hi
hi	hi	hi	-	hi	-	-	hi	-
c	1	0	0	1	0	0	0	1

(a)



(b)

Figure 1.2 Partition matrix (a) and the corresponding incompatibility graph (b) for the function from Table 1.1. The column `lo,hi` of partition matrix has no instances and is not shown. The node labels of the incompatibility graph found by graph coloring are circled.

has to assign each column a label that corresponds to a value of c. If two columns are compatible, they can be assigned the same label. To preserve the consistency of original function and its decomposition, different labels have to be used for incompatible columns.

The partition matrix column labels are found by coloring an *incompatibility graph* (Figure 1.2.b). This has a distinct node for each column of the partition matrix. Two nodes are connected if the corresponding columns of partition matrix are incompatible. For instance, the nodes `hi,hi` and `lo,med` are connected because their corresponding columns are incompatible due to the entries in the first row (hi≠lo).

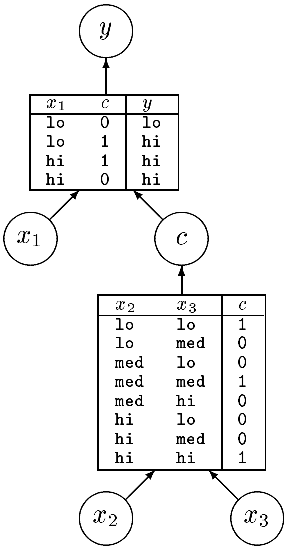
With optimal coloring of the incompatibility graph, the new feature c obtains a minimal set of values needed for consistent derivation of H and G from F. The optimal coloring of the graph from Figure 1.2.b requires two different colors, i.e., two abstract values for c. The tabulated functions G and H can then be derived straightforwardly from the labeled partition matrix.

The resulting decomposition is given in Figure 1.3. The following can be observed:

- The tabulated functions G and H are overall smaller than the original function F.

- By combining the features x_2 and x_3 we obtained a new feature c and the corresponding tabulated function H, which can be interpreted as $c = \text{EQUAL}(x_2, x_3)$. Similarly, the function G that relates x_1 and c to y can be interpreted as $y = \text{MAX}(x_1, c)$.

- The decomposition generalizes some undefined entries of the partition matrix. For example, $F($`hi,med,hi`$)$ is generalized to `hi` because the column `med,hi` has the same label as columns `lo,med` and `hi,med`.



x_1	c	y
lo	0	lo
lo	1	hi
hi	1	hi
hi	0	hi

x_2	x_3	c
lo	lo	1
lo	med	0
med	lo	0
med	med	1
med	hi	0
hi	lo	0
hi	med	0
hi	hi	1

Figure 1.3 Decomposition of tabulated function from Table 1.1 that uses a new feature $c = H(x_2, x_3)$.

There are two problems when single-step decomposition is dealing with real-world functions: the partition matrix can become very large, and because of its incompleteness the optimal coloring of the corresponding incompatibility graph can be prohibitively complex. HINT solves the first problem by avoiding the explicit construction of partition matrix and deriving the incompatibility graph directly from the tabulated function (Zupan, 1997). Partition matrix is thus only a formal construct used for the explanation and analysis of decomposition algorithm. For the problem of coloring, HINT uses a suboptimal but efficient heuristic coloring algorithm (Zupan et al., 1997; Zupan, 1997) based on the Color Influence Method by (Perkowski and Uong, 1987) and (Wan and Perkowski, 1992).

1.3 FINDING THE BEST FEATURE PARTITION

So far, we have assumed that a partition of the features to free and bound sets is given. However, for each function F there are many possible partitions, each one yielding a different feature c and a different pair of functions G and H. In feature transformation, it is important to identify partitions that lead to simple but useful new features. Typically, we prefer features with a small number of values, and those that yield functions G and H of low complexity.

The simplest measure for finding good feature partitions is the number of values required for the new feature c. That is, when decomposing F, a set of candidate partitions is examined and the one that yields c with a smallest set of possible values is selected for decomposition.

An alternative measure for the selection of partitions can be based on the complexity of functions. Let $I(F)$ denote a number of bits to encode some function $y = F(X)$:

$$I(F) = \log_2 |D_y| \prod_{x \in X} |D_x|$$

where D_x and D_y are value domains of features x and y, while $|D_x|$ and $|D_y|$ denote their cardinality. There are some other similar measures (Ross et al., 1994b; Biermann et al., 1982) that assume a different encoding for F and thus yield different $I(F)$, but they only slightly affect the performance of decomposition (Zupan, 1997).

The best partition is then the one that minimizes the overall complexity of newly discovered functions H and G, i.e., the partition with minimal $I(H) + I(G)$. The complexity of G and H should be lower than that of F, and decomposition takes place only if $I(H) + I(G) < I(F)$ for the best partition. If the latter criterion is satisfied, the function F is called *decomposable* with respect to the given attribute partition.

Table 1.2 provides an example and lists both above mentioned partition selection measures for the function from Table 1.1. This function can be decomposed in three different ways, where by both measures the partition $A = \{x_1\}$ and $B = \{x_2, x_3\}$ is favorable. Regarding the complexity measure I, this partition is also the only one for which F is decomposable.

Partition	$\|c\|$	$I(H) + I(G)$	decomposable
$A = \{x_1\}$, $B = \{x_2, x_3\}$	2	13.0	yes
$A = \{x_2\}$, $B = \{x_1, x_3\}$	4	24.0	no
$A = \{x_3\}$, $B = \{x_1, x_2\}$	4	24.0	no

Table 1.2 Partition selection measures for the function from Table 1.1. To determine decomposability, note that $I(F) = 18.0$.

The number of possible partitions increases exponentially with the number of input features. To limit the complexity of search, the decomposition should examine only a subset of partitions. In HINT, the partitions examined are only those with less than b features in the bound set, where b is a user-defined parameter usually set to 2 or 3.

1.4 REDUNDANCY DISCOVERY AND REMOVAL

A special characteristic of function decomposition is its capability to identify and remove redundant features and/or their values. Namely, whenever a feature $c = H(x_i)$ is discovered such that c uses only a single value (i.e., H is constant function), the feature x_i is redundant and can be removed from the dataset. Similarly, whenever a feature $c = H(x_i)$ uses less values than x_i, the original feature x_i can be replaced by c.

Both these dataset transformations are particularly useful for data preprocessing, since they can reduce the size of the problem and increase the coverage of the problem space by the learning examples. HINT removes redundant features in the increased order of their relevance: the less relevant features are checked first, and if any type of redundancy mentioned above is discovered, the dataset is accordingly transformed. The features are estimated by a ReliefF measure of relevance (Kononenko, 1994). ReliefF estimates the features according to how well they distinguish among the instances that are close to each other. The relevance of feature x is then

$$\begin{aligned} W(x) &= P(\text{different value of } x \mid k\text{−nearest instances with different } y) - \\ &\quad P(\text{different value of } x \mid k\text{−nearest instances with same } y) \end{aligned}$$

We use the version of ReliefF which determines the feature's relevancy based on at most 200 arbitrary selected examples, and which for every example examines $k = 5$ nearest instances of the same and of the different value for y.

1.5 DISCOVERING FEATURE HIERARCHIES

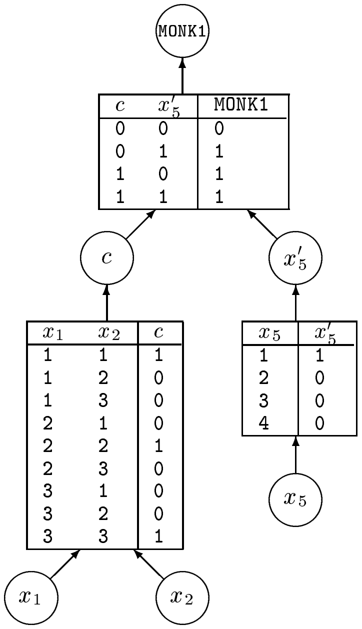
With all the above components, decomposition enables the discovery of feature hierarchies. Given a dataset, it is first pre-processed to remove redundant features and their values. Next, a single-step decomposition is used to decompose the pre-processed tabulated function F to G and H. This process is recursively repeated on G and H until they can not be decomposed further, i.e., their further decomposition would increase the overall complexity of resulting functions.

Let us illustrate this process by several examples. The first example is a well-known machine learning problem MONK1 (Thrun et al., 1991). The dataset uses six 2 to 4-valued input features (x_1 to x_6) and contains 124 (of 435 possible) distinct learning examples that partially define the target concept $x_1 = x_2$ OR $x_5 = 1$. HINT develops a feature hierarchy (Figure 1.4) that

1. correctly excludes irrelevant input features x_3, x_4, and x_6,
2. transforms x_5 to x'_5 by mapping four values of x_5 to only two values of x'_5,
3. includes a new feature c and its tabular representation for $x_1 = x_2$, and
4. relates c and x'_5 with a tabular representation of the OR function.

In other words, the resulting hierarchy correctly represents the target concept.

Similarly as MONK1, the MONK2 learning problem (Thrun et al., 1991) uses the same set of input features but defines the concept: $x_i = 1$ for exactly two choices of $i \in \{1, 2, \ldots, 6\}$. The dataset contains 172 (of 435 possible) distinct learning examples. Although the discovered structure (Figure 1.5) does not directly correspond to the original concept definition, it correctly reformulates the target concept by introducing features that count the number of ones in



c	x'_5	MONK1
0	0	0
0	1	1
1	0	1
1	1	1

x_1	x_2	c
1	1	1
1	2	0
1	3	0
2	1	0
2	2	1
2	3	0
3	1	0
3	2	0
3	3	1

x_5	x'_5
1	1
2	0
3	0
4	0

Figure 1.4 MONK1 feature hierachy as discovered by HINT.

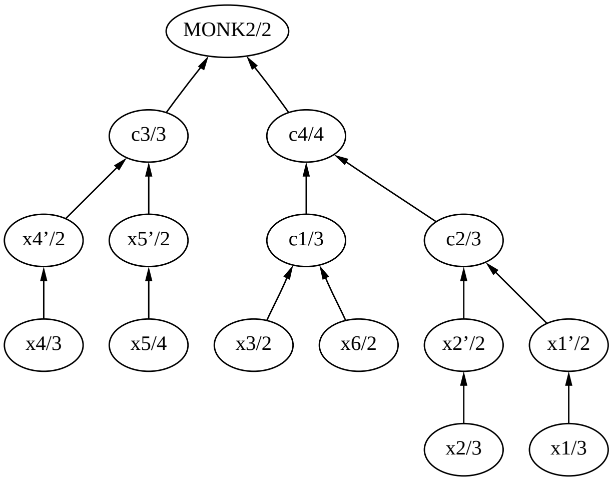


Figure 1.5 The feature hierarchy discovered for MONK2. Each node gives a name of the feature and cardinality of its set of values.

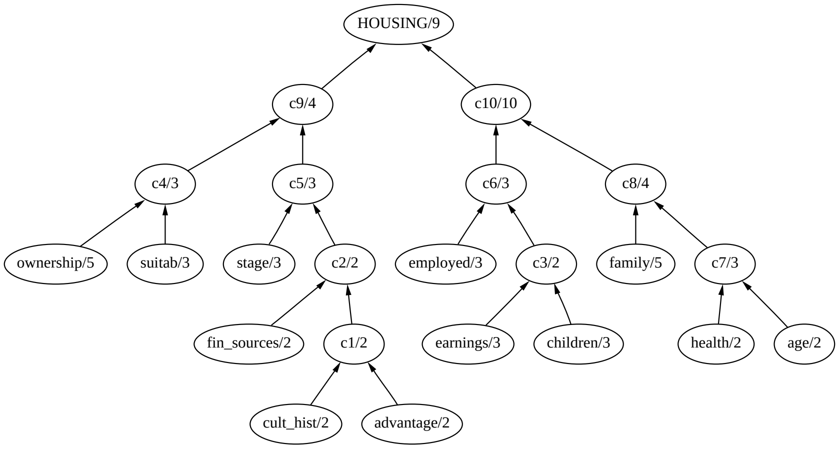


Figure 1.6 The feature structure discovered for housing loan allocation problem. Each node gives a name of the feature and cardinality of its set of values.

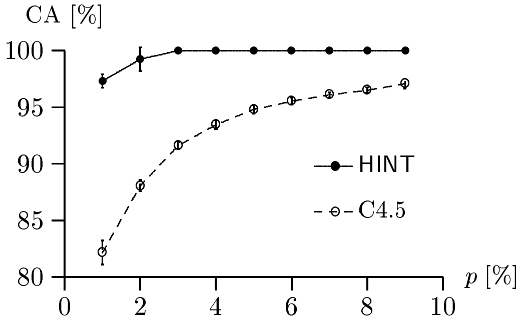


Figure 1.7 Averaged classification accuracy of HINT and C4.5 for 10 experiments with learning sets consisting of a random sample of p percents of the HOUSING problem space.

their arguments. Also note that all input features that use more than two values are replaced by new binary features.

For a real-world example, consider a HOUSING dataset from a management decision support system for allocating housing loans (Bohanec et al., 1996). This system was developed for the Housing Fund of Slovenia and used since 1991 in 13 floats of loans with a total value of approximately 90 million ECU. Basically, its task is to rank applicants into one of nine priority classes based on 12 input features. The system uses a hierarchical decision model.

For this experiment, we wanted to assess the ability of decomposition to reconstruct the system's model given a random selection of classified cases. First, it was observed that in most cases when HINT was given 8000 or more learning examples (4% coverage of problem space), it discovered the same feature hierarchy (Figure 1.6). When presented to a domain expert, the discovered features were found meaningful. For instance, the feature c_4 represents the present housing status of the applicant. Features c_2, c_6, and c_8 respectively specify the way of solving the housing problem, applicant's current status, and his/her social and health conditions. Thus, the structure of the original model was successfully reconstructed. This reconstruction takes about 2.5 minutes of processor time on a HP J210 Unix workstation.

The quality of the functions discovered for housing problem was assessed by classification of the remaining examples that completely cover this domain but were not included in the learning set. The generalization of HINT was assessed as a learning curve (Figure 1.7) and compared to that of the state-of-the-art learning tool C4.5 (Quinlan, 1993) that induces decision trees. C4.5 was run with default options except for `-m1`, by which C4.5 trees were made consistent with the original dataset. For this domain, HINT generalizes well as it achieves 100% classification accuracy by using learning sets that cover 3 or more percents of the problem space.

Housing problem domain can be viewed as a typical representative of practical domains in the field of decision making that HINT was tested on. Similar tests and experimental conclusion were for instance obtained by the rediscovery of hierarchical decision models for nursery school applications, job application evaluation, and performance evaluation of public enterprises (Zupan, 1997).

1.6 FEATURE CONSTRUCTION

Function decomposition can also be used for feature construction. This is defined as a process of augmenting the feature space by inferring or creating additional features. We here show how the single-step decomposition can be used both for finding appropriate combinations of original features and using them to construct new features which are then added to the original dataset. Again, the new features that use small number of values are preferred.

For example, consider again the dataset from Table 1.1. Suppose we want to augment it with a single new feature. For this purpose, single-step decomposition examines all pairs of original input features and for each pair derives a new feature. A new feature that has the fewest values is preferred. Such feature is $c = H(x_2, x_3)$, and is defined by the example set from Figure 1.3. Function H can now be used for each example from Table 1.1 to obtain the corresponding value for c; such augmented dataset is shown in Table 1.3.

Obviously, an original dataset can be augmented with more than just a single new feature. We propose a schema where, using the single-step decomposition, m new features are added to a dataset: a candidate set of combinations of the original features are examined, and corresponding m new features that have the fewest values are selected. Ties are resolved arbitrarily. Only original features are used to generate new features. Note also that in the process no hierarchy of features is constructed. The feature construction and corresponding aug-

x_1	x_2	x_3	c	y
lo	lo	lo	1	hi
lo	lo	med	0	lo
lo	med	lo	0	lo
lo	med	med	1	hi
lo	med	hi	0	lo
lo	hi	lo	0	lo
lo	hi	med	0	lo
lo	hi	hi	1	hi
hi	lo	lo	1	hi
hi	lo	med	0	hi
hi	med	med	1	hi
hi	hi	med	0	hi

Table 1.3 A dataset from Table 1.1 augmented with a new feature $c = H(x_2, x_3)$.

mentation of the original dataset results in a new dataset which may then be further processed by some machine learning algorithm.

We evaluate this schema on the above-mentioned domains. For new features, only those that depend only on two original input features are considered. For MONK1 and MONK2 domains, the benefit of adding new features was assessed by a 10-fold cross validation. For HOUSING, 10 experiments were performed with a learning set consisting of a random sample of 1% of the problem space, while the test set consisted of the remaining 99%. Each learning set was first augmented with m new features as described above, and then given to C4.5 (Quinlan, 1993) which was used to induce the target concept from the augmented dataset. Again, C4.5 was run with default options except for `-m1`. The induced decision tree was then tested on a test set, which was also augmented by the same set of new features, i.e., using their definition as derived from the learning set.

In all the cases, a considerable increase of classification accuracy (Figure 1.8) and considerable decrease of the size of induced decision trees (Figure 1.9) were observed when new features are added. In other words, adding new features improves both the classification accuracy and the transparency of decision trees. For example, for MONK1, HINT discovered the feature $c = \text{EQUAL}(x_1, x_2)$ which enabled C4.5 to construct the decision tree with only five nodes:

```
if c=1 then y=1 else if x5=1 then y=1 else y=0
```

This decision tree was also substantially less complex than the decision trees with an average of 77 nodes constructed from the original MONK1 data.

The results indicate that function decomposition discovers features that are relevant for these domains. It is further interesting to note how well a simple feature selection criterion based on the number of new feature's values performs in terms of yielding augmented datasets that enable C4.5 to derive classifiers with high classification accuracies.

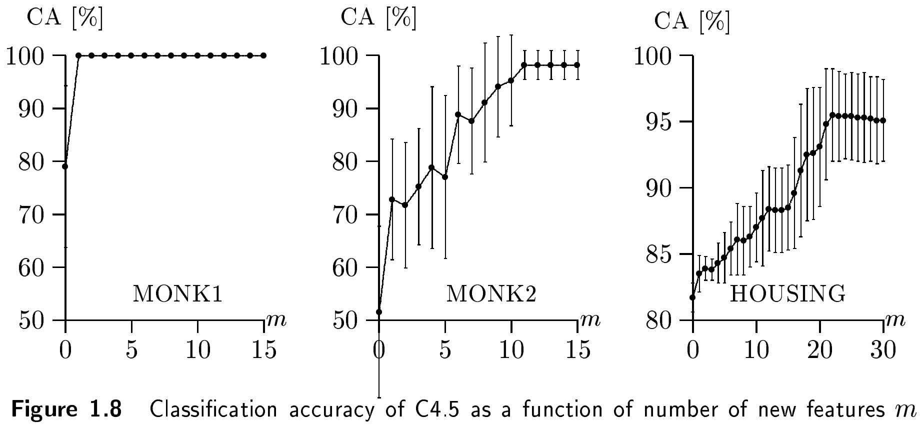


Figure 1.8 Classification accuracy of C4.5 as a function of number of new features m added to MONK1, MONK2, and housing loans dataset. Error bars indicate standard deviations.

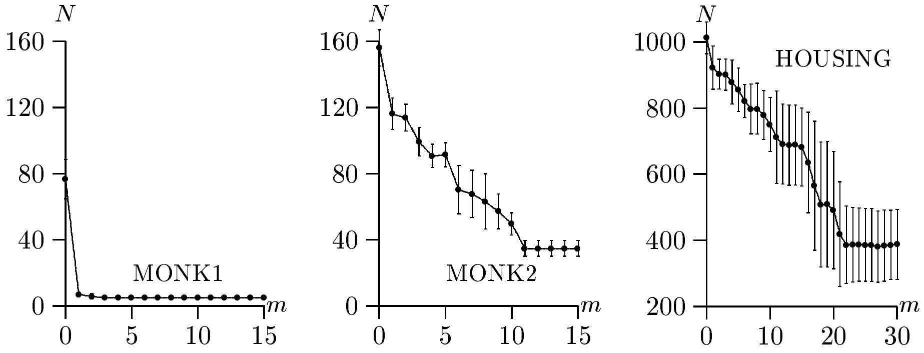


Figure 1.9 Number of nodes N of C4.5-induced decision trees as a function of number of new features m added to MONK1, MONK2, and housing loans dataset. Error bars indicate standard deviations.

1.7 RELATED WORK

A learning method that uses feature hierarchies has been proposed already by a pioneer of artificial intelligence, (Samuel, 1967). His method used a signature table system, which is essentially the same representation mechanism as the one used in this chapter. However, Samuel's method was able to learn only functions, requiring the hierarchy to be given in advance. Samuel's approach was later studied and improved by (Biermann et al., 1982) but still relied on the predefined feature hierarchy.

While, within machine learning, Samuel and Biermann et al. may be the first to realize the power of decomposition, the fundamentals of the approach were defined earlier in the area of switching circuit design. (Curtis, 1962) reports that in the late 1940's and 1950's several switching circuit theorists considered this subject and (Ashenhurst, 1952) reported on a unified theory of decomposition of switching functions. Most of related work of those times is reported and reprinted in (Curtis, 1962), where he compares the decomposition approach to other switching circuit design approaches and further formalizes and extends the decomposition theory. Curtis' method is defined over Boolean variables and requires a set of examples that completely cover the problem space.

Recently, the Ashenhurst-Curtis approach was substantially improved by research groups of M. A. Perkowski, T. Luba, and T. D. Ross. (Perkowski and Uong, 1987) and (Wan and Perkowski, 1992) propose a graph coloring approach to the decomposition of incompletely specified switching functions. A different approach is presented by (Luba and Selvaraj, 1995). Their decomposition algorithms are able to generalize. Generalization of function decomposition when applied to a set of simple Boolean functions was studied by (Ross et al., 1994b) and (Goldman, 1994). Goldman also indicates that the decomposition approach to switching function design might be termed knowledge discovery since features not previously anticipated can be discovered.

The above mentioned decomposition approaches typically deal with noiseless data and Boolean features. Various extensions towards dealing with nominal-valued features were proposed by (Biermann et al., 1982), (Luba, 1995), (Zupan et al., 1997), and (Files et al., 1997). To handle noisy data, a minimal-error decomposition was recently proposed (Zupan, 1997). It is based on a representation of learning examples with class distributions and uses successive column merging of partition matrix, so that the expected error of classification is minimized. For the decomposition of real-valued functions some preliminary methods were proposed by (Ross et al., 1994a) and (Demšar et al., 1997).

There are other machine learning approaches that either use or construct feature hierarchies. DUCE (Muggleton, 1987; Muggleton, 1990) uses transformation operators to compress the given examples by successive generalization and feature construction. Query-based PAC-learning of tabulated functions within feature hierarchies is described by (Tadepalli and Russell, 1998). Queries are also used in PAC-learning described by (Bshouty et al., 1995). Their algorithm identifies both concept structures and their associated tabulated functions, but can deal only with Boolean functions with symmetric and constant fan-in gates. Within PAC-learning, (Hancock et al., 1994) learn non-overlapping perceptron networks from examples and membership queries. A structured induction approach (Shapiro and Niblett, 1982; Michie, 1995) is based on a manual decomposition of the problem and an expert-assisted selection and classification of examples to construct functions that define intermediate features in the hierarchy.

The feature hierarchy has also been used by a multi-attribute decision support expert system shell DEX (Bohanec and Rajkovič, 1990) which has its roots in DECMAK methodology (Efstathiou and Rajkovič, 1979; Bohanec et al., 1983). There, a tree-like structure of variables is defined by an expert, and several tools assist in the acquisition of decision tables.

The above related work refers primarily to the use and discovery of feature hierarchies. As outlined in this chapter, a function decomposition may also be useful to manufacture features that are then used by some other machine learning algorithms. To the best of our knowledge, there has been no related work that would use function decomposition in this way.

1.8 SUMMARY

The distinctive property of function decomposition is that it can discover new features together with their corresponding functions that are not predefined in any way. As such and in the framework of feature transformation, function decomposition is a promising approach to discover and construct new features that are either added to the original dataset, or transform this dataset to a hierarchy of less complex datasets. The experiments show that decomposition can:

- discover useful and relevant features,
- develop relevant feature hierarchies, and
- identify and remove redundant features and/or their values.

Furthermore, we have shown that discovery of feature hierarchies by function decomposition may generalize well, which is especially important when using function decomposition within machine learning framework. Also, adding the features derived by function decomposition to the original datasets can improve the performance of other machine learning algorithms both in terms of accuracy and size of induced classifiers.

Acknowledgments

This chapter is a revised and extended version of the article ‘Feature transformation by function decomposition” by the same authors, *IEEE Intelligent Systems*, March/April 1998 (© 1998 IEEE). We acknowledge the kind permission of IEEE to use portions of this article.

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.

Bohanec, M., Bratko, I., and Rajkovič, V. (1983). An expert system for decision making. In Sol, H. G., editor, *Processes and Tools for Decision Support.* North-Holland.

Bohanec, M., Cestnik, B., and Rajkovič, V. (1996). A management decision support system for allocating housing loans. In Humphreys, P., Bannon, L., McCosh, A., and Migliarese, P., editors, *Implementing System for Supporting Management Decisions*, pages 34–43. Chapman & Hall, London.

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

Bshouty, N. H., Hancock, T. R., and Hellerstein, L. (1995). Learning boolean read-once formulas over generalized bases. *Journal of Computer and System Sciences*, 50(3):521–542.

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. In van Someren, M. and Widmer, G., editors, *Proc. European Conference on Machine Learning, ECML-97*, pages 93–107, Prague. Springer.

Efstathiou, J. and Rajkovič, V. (1979). Multiattribute decisionmaking using a fuzzy heuristic approach. *IEEE Trans. on Systems, Man and Cybernetics*, 9:326–333.

Files, C., Drechsler, R., and Perkowski, M. (1997). Functional decomposition of MVL functions using multi-valued decision diagrams. In *International Symposium on Multi-Valued Logic.*

Goldman, J. A. (1994). Pattern theoretic knowledge discovery. In *Proc. the Sixth Int'l IEEE Conference on Tools with AI.*

Hancock, T. R., Golea, M., and Marchand, M. (1994). Learning nonoverlapping perceptron networks from examples and membership queries. *Machine Learning*, 16(3):161–183.

Kononenko, I. (1994). Estimating attributes. In Bergadano, F. and Raedt, L. D., editors, *Proc. of the European Conference on Machine Learning (ECML-94)*, pages 171–182. Springer-Verlag.

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

Luba, T. and Selvaraj, H. (1995). A general approach to boolean function decomposition and its application in FPGA-based synthesis. *VLSI Design*, 3(3–4):289–300.

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

Muggleton, S. (1987). Structuring knowledge by asking questions. In Bratko, I. and Lavrač, N., editors, *Progress in Machine Learning*, pages 218–229. Sigma Press.

Muggleton, S. (1990). *Inductive Acquisition of Expert Knowledge.* Addison-Wesley, Workingham, England.

Perkowski, M. and Uong, H. (1987). Automatic design of finite state machines with electronically programmable devices. In *Record of Northcon '87*, pages 16/4.1–16/4.15, Portland, OR.

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

Ross, T. D., Goldman, J. A., Gadd, D. A., Noviskey, M. J., and Axtell, M. L. (1994a). On the decomposition of real-valued functions. In *3rd Int'l Workshop of Post-Binary VLSI Systems.*

Ross, T. D., Noviskey, M. J., Gadd, D. A., and Goldman, J. A. (1994b). Pattern theoretic feature extraction and constructive induction. In *Proc. ML-COLT '94 Workshop on Constructive Induction and Change of Representation*, New Brunswick, New Jersey.

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. and Niblett, T. (1982). Automatic induction of classificiation rules for a chess endgame. In Clarke, M. R. B., editor, *Advances in Computer Chess 3*, pages 73–92. Pergamon, Oxford.

Tadepalli, P. and Russell, S. (1998). Learning from examples and membership queries with structured determinations. *Machine Learning* (to appear).

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

Wan, W. and Perkowski, M. A. (1992). A new approach to the decomposition of incompletely specified functions based on graph-coloring and local transformations and its application to FPGA mapping. In *Proc. of the IEEE EURO-DAC '92*, pages 230–235, Hamburg.

Zupan, B. (1997). *Machine learning based on function decomposition.* PhD thesis, University of Ljubljana. Available at `http://www-ai.ijs.si/BlazZupan/papers.html`.

Zupan, B., Bohanec, M., Bratko, I., and Demšar, J. (1997). Machine learning by function decomposition. In D. H. Fisher, J., editor, *Proc. Fourteenth International Conference on Machine Learning*, pages 421–429, San Mateo, CA. Morgan Kaufmann.

Index