Induction of Concept Hierarchies from Noisy Data

Blaž Zupan BLAZ.ZUPAN@FRI.UNI-LJ.SI
Faculty of Computer and Information Science, University of Ljubljana, Tržaška 25, SI-1000 Ljubljana, Slovenia
Office of IT and Dept. of Family and Community Medicine, Baylor College of Medicine, Houston, TX, USA

Ivan Bratko IVAN.BRATKO@FRI.UNI-LJ.SI
Faculty of Computer and Information Science, University of Ljubljana, Tržaška 25, SI-1000 Ljubljana, Slovenia
J. Stefan Institute, Jamova 39, SI-1000 Ljubljana, Slovenia

Marko Bohanec MARKO.BOHANEC@IJS.SI
J. Stefan Institute, Jamova 39, SI-1000 Ljubljana, Slovenia

Janez Demšar JANEZ.DEMSAR@FRI.UNI-LJ.SI
Faculty of Computer and Information Science, University of Ljubljana, Tržaška 25, SI-1000 Ljubljana, Slovenia

Abstract

Function decomposition can be used as a 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. We present a new approach that allows function decomposition to learn from noisy data, where a discovery of new concepts is guided by the aim to reduce the estimated error of the resulting classifier. The method was implemented within the program HINT. The experimental evaluation of HINT demonstrates its ability to effectively decompose the problem into smaller, less complex problems, and by this discover data models of high classification accuracy that include potentially relevant and interpretable concepts.

1. Introduction

Machine learning based on function decomposition (Zupan et al., 1999; Zupan et al., 1997) is an approach that, given a set of training examples, induces a definition of the target concept in terms of (1) a hierarchy of intermediate concepts and (2) definitions of these concepts. Each intermediate concept is described by a separate example set obtained by the decomposition of original training set.

The distinguishing capabilities of function decomposition are to discover new concepts from training data, organize them into a concept hierarchy, and induce concept descriptions by decomposing the original training set into smaller and less complex example sets. For example, consider the CAR data set (Murphy & Aha, 1994). Originally, it consists of 1728 instances mapping six attributes to a four-valued outcome (Figure 1.a). Decomposition finds a concept hierarchy (Figure 1.b) with four meaningful intermediate concepts, c1 to c4 (Zupan et al., 1997). When interpreted, for instance, c2 was found to represent the overall cost of a car and c3 its technical characteristics.

Until recently, function decomposition algorithms did not include mechanisms that would handle noise in the training data. Most probably this was because for its prevailing use in switching circuit design, where the derived logic circuits have to be 100% consistent with the originally tabulated function. The process was guided by minimization of complexity, as the decomposition tried to find the least complex implementation of the original function. Within machine learning, however, appropriate treatment of noise is not only desired but required for any modern concept induction algorithm.

This paper is based on our original effort (Zupan, 1997) and reports on a particular mechanism for noise handling that extends the basic approach to machine learning based on function decomposition introduced in Zupan et al. (1997) and Zupan et al. (1999). Unlike previous approaches, here decomposition explicitly aims at minimizing the expected classification error. We refer to the proposed decomposition method as *minimal-error function decomposition*. In this paper we present the method and its experimental evaluation on a number of data sets.

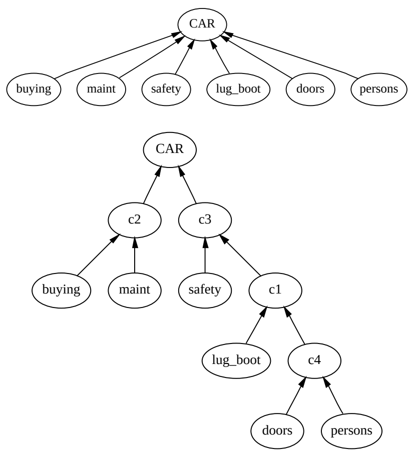


Figure 1. Attribute-class relation in undecomposed CAR data set (top) and its corresponding concept structure after decomposition (bottom).

2. Induction of Concept Hierarchies

Given a tabular representation of a function $y = F(X)$, $X = \langle x_1, \ldots, x_n \rangle$, and a partition of input attributes X to its proper subsets A and B, the decomposition employs its basic step to find tabulated functions $y = G(A, c)$ and $c = H(B)$. By recursive application of this step on G and H, the aim is to derive a hierarchy of functions.

Basic function decomposition step (see Zupan et al. (1999) for details) is the core of the overall algorithm. In the case of noisy data, for a given attribute partition $A|B$ it not only finds the corresponding functions G and H, but also estimates their classification error. The overall decomposition uses its basic step to search for the attribute partition that minimizes this error. We first describe the error estimation method, followed by a description of the basic decomposition step and overall decomposition algorithm.

2.1 Examples With Class Distributions and Estimation of Their Classification Error

We will explain the minimal-error decomposition method by an example. Consider a function $y = F(x_1, x_2, x_3)$ which is partially specified by a set of examples shown in Table 1. Note that the set is inconsistent as some examples specify different class values y for the same values of the attributes x_1 to x_3. To start, the training set is converted to the one that uses class distributions. That is, a class distribution vector is assigned to every distinct combination of attribute values in the training set. Class distribution vector $\boldsymbol{d}_i = (d_i^1, \ldots, d_i^{||y||})$ gives the counts of how many times such a combination was labeled with each of the classes $y_1, \ldots, y_{||y||}$, where $||y||$ is a cardinality of class value set (in our case equal to 3). For the training set in Table 1, an equivalent set of examples E_F with class distributions is given in Table 2.

Table 1. An example training set

x_1	x_2	x_3	y
low	low	hi	low
low	hi	low	low
low	hi	low	low
low	hi	hi	hi
med	low	low	med
med	low	low	hi
med	low	low	med
med	hi	low	med
med	hi	low	med
med	hi	hi	hi
hi	low	low	hi
hi	low	hi	med
hi	hi	low	hi
hi	hi	hi	hi

Table 2. Examples with class distribution vectors $\boldsymbol{d}_i$, probabilities of examples $p(e_i)$, and estimated errors $\varepsilon(e_i)$ when examples are used for classification.

i	x_1	x_2	x_3	$\boldsymbol{d}_i$	$p(e_i)$	$\varepsilon(e_i)$	$p(e_i)\varepsilon(e_i)$
1	lo	lo	hi	(1,0,0)	0.071	0.524	0.037
2	lo	hi	lo	(2,0,0)	0.143	0.393	0.056
3	lo	hi	hi	(0,0,1)	0.071	0.381	0.027
4	med	lo	lo	(0,2,1)	0.214	0.457	0.098
5	med	hi	lo	(0,2,0)	0.143	0.321	0.046
6	med	hi	hi	(0,0,1)	0.071	0.381	0.027
7	hi	lo	lo	(0,0,1)	0.071	0.381	0.027
8	hi	lo	hi	(0,1,0)	0.071	0.429	0.031
9	hi	hi	lo	(0,0,1)	0.071	0.381	0.027
10	hi	hi	hi	(0,0,1)	0.071	0.381	0.027
							$\Sigma = 0.404$

Suppose now we use the examples with class distributions for classification. Given a set of attribute values we look for the corresponding example $e_i \in E_F$ and use its class distribution vector $\boldsymbol{d}_i$ to predict the class. If, for instance, the predicted class is y_k, the estimated error using m-error estimate proposed by Cestnik and Bratko (1991) is

$$\varepsilon(e_i, k) = 1 - \frac{d_i^k + p(y_k)m}{\sum_j d_i^j + m} \qquad (1)$$

where $p(y_k)$ is the apriori probability of class y_k. Apriori probabilities of classes can be estimated by relative frequencies of the classes in the training set (for our data set, $p_{lo} = 0.214$, $p_{med} = 0.357$ and $p_{hi} = 0.429$), and m is a parameter of the estimation formula, which should be adjusted according to the noise level of the domain. Example e_i should thus classify to the class y_k that minimizes the estimated error $\varepsilon(e_i, k)$. The one but last column of Table 2 gives the corresponding example error estimates, when the default value of $m = 2$ is used.

Table 3. Initial partition matrix (a) and partition matrices after two successive merging of columns (c and e). Tables b, d, and f give error estimates for different candidate pairs for column merging.

a)

	x_2	lo	lo	hi	hi
x_1	x_3	lo	hi	lo	hi
lo		-	(1,0,0)	(2,0,0)	(0,0,1)
med		(0,2,1)	-	(0,2,0)	(0,0,1)
hi		(0,0,1)	(0,1,0)	(0,0,1)	(0,0,1)
c		1	2	3	4

b)

	2	3	4
1	0.423	0.363	0.415
2		0.397	0.435
3			0.442

c)

	x_2	lo hi	lo	hi
x_1	x_3	lo lo	hi	hi
lo		(2,0,0)	(1,0,0)	(0,0,1)
med		(0,4,1)	-	(0,0,1)
hi		(0,0,2)	(0,1,0)	(0,0,1)
c		1	2	3

d)

	2	3
1	0.357	0.403
2		0.394

e)

	x_2	lo hi lo	hi
x_1	x_3	lo lo hi	hi
lo		(3,0,0)	(0,0,1)
med		(0,4,1)	(0,0,1)
hi		(0,1,2)	(0,0,1)
c		1	2

f)

	2
1	0.401

We compute the probability of each row in Table 2 as the proportion of instances it represents among all the training instances, that is $\sum_j d_i^j$ divided by $\sum_i \sum_j d_i^j$. The sum of weighted errors (last column in Table 2) estimates the overall error that we would make if the data set E_F would be used for classification. In our case, this error is estimated as 0.404. Notice that this assumes only the classification of those cases that *exactly* match an entry in Table 2. Generalization to "unseen" cases results from decomposition explained in the following sections.

2.2 Basic Decomposition Step

Let us now assume that we want to decompose our example set E_F which partially defines a function $y = F(x_1, x_2, x_3)$ to sets E_G and E_H, such that these would partially define new functions $c = H(x_2, x_3)$ and $y = G(x_1, c)$ where c is a new, intermediate concept. For this, we first represent our data set from Table 2 by a *partition matrix*. This uses combinations of values of x_2 and x_3 as column labels and combinations of values of the remaining attributes (in our case, x_1) as row labels. Each instance from E_F is then placed in the corresponding cell in the partition matrix (Table 3.a).

Note that for two cells of partition matrix marked with "-" there were no corresponding examples: they can be treated as if the corresponding distribution vector is (0,0,0).

The basic decomposition step aims to derive an intermediate concept c, i.e., in our case a mapping of value combinations of x_2 and x_3 to a value of c. A trivial mapping would define a distinct value of c for each combination of x_2 and x_3 (c labels in Table 3.a). A data set represented with such partition matrix would also retain the estimated classification error of 0.404 of the undecomposed data set E_F. Suppose now we merge two partition matrix's columns, say the first and the third column. By this, we examine the case where combinations (lo,lo) and (hi,lo) of attributes x_2 and x_3 would be assigned the same value of c. The resulting partition matrix is given in Table 3.c.

Two columns are merged by summing-up the distribution vectors of corresponding columns. The new, merged data set represented by new partition matrix now consists of only eight instances. We now use the same method as described above to compute the estimated classification error of the new data set, and obtain 0.363. Compared to the original data set, the estimated error has been reduced from 0.404 to 0.363. This is essential for our decomposition method, as we can see that column merging may produce data sets of lower estimated error.

Starting with the initial partition matrix (Table 3.a), we could have merged any pair of columns. The preferred merging is the one that gives a data set with lowest error. For six possible column mergings, Table 3.b depicts the errors of corresponding resulting data sets. The merging that results in a data set with the lowest estimated error is preferred. In our case, this is the merging of columns 1 and 3.

We now continue the column merging process. For partition matrix in Table 3.c, there are three distinct mergings of which the one involving first and second column yields the lowest estimated error (Table 3.d). This further reduces the error ($0.357 < 0.363$). The resulting data set is represented by partition matrix in Table 3.e. Note that if its two columns were merged, the error would increase ($0.401 > 0.357$), so this merging is not performed.

Therefore, the resulting (final) partition matrix is the one in Table 3.e. It is now trivial to use it to derive the new data sets E_H and E_G which represent the decomposed functions $c = H(x_2, x_3)$ and $y = G(x_1, c)$, respectively. Note also that we only need two values to describe c. To derive E_H, we simply look at the c labels for columns of the final partition matrix. Distributions for E_H have all elements equal to 0 except the element for the corresponding class c, which equals the number of instances in the original data set that had a specific combination of x_2 and x_3. For E_G, a combination of x_1 and c is looked-up from the final partition matrix and entered in a data set.

The two decomposed data sets are shown in Table 4. To interpret them, if we take that c's value "1" means "low", and "2" means "high", one can easily see that $c = \text{MIN}(x_2, x_3)$ and $y = \text{MAX}(x_1, c)$. This is a typical result of the function decomposition method we are proposing: while aiming at minimizing the classification error, it can discover meaningful and interpretable concept hierarchies.

Table 4. Example sets E_H (above) and E_G (below) that are decompositions of the set E_F from Table 2.

x_2	x_3	distribution	c
lo	lo	(4 0)	1
lo	hi	(2 0)	1
hi	lo	(5 0)	1
hi	hi	(0 3)	2

x_1	c	distribution	y
lo	1	(3 0 0)	lo
lo	2	(0 0 1)	hi
med	1	(0 4 1)	med
med	2	(0 0 1)	hi
hi	1	(0 1 2)	hi
hi	2	(0 0 1)	hi

2.3 Overall Decomposition Algorithm

Given an initial (training) data set with attributes X, the overall decomposition algorithm searches through possible attribute partitions $A|B$, where $A \cup B = X$ and $A \cap B = \emptyset$. It uses the basic decomposition step to find an attribute partition and its corresponding partition matrix, for which the column merging results in a data set with the lowest estimated error. If this error is lower than the one estimated for the initial data set, the data set is decomposed accordingly. By applying this step recursively on the resulting two data sets, a concept hierarchy is discovered. Decomposition is not performed on a data set if this includes only two attributes or if no attribute partition (and corresponding partition matrix column merging) is found that would reduce the estimated error.

A special characteristic of function decomposition is its capability to identify and remove redundant attributes and/or their values. Namely, whenever a function $c = H(x_i)$ is found such that c uses only a single value (i.e., H is a constant), the attribute x_i is redundant and can be removed from the data set. Similarly, whenever c uses fewer values than x_i, it can replace the original attribute. We employ both approaches for preprocessing to remove redundancies in the increasing order of attributes' relevance: the less relevant attributes are checked first, and if any type of redundancy mentioned above is discovered, the data set is accordingly transformed. In our implementation, the attributes are estimated by the ReliefF measure of relevance (Kononenko, 1994).

2.4 Implementation and Complexity

The minimal-error decomposition algorithm was implemented within the program HINT (Hierarchy INduction Tool). To illustrate HINT's efficiency, HINT uses less then 1 second for induction in the Monk domains, and no more than two minutes of CPU time for any of the data sets in Table 6 on a medium scale UNIX workstation.

Several implementational enhancements made the algorithm more efficient. The partition matrix is encoded as a linked list of columns, where each column is represented as a linked list sorted by the values of attributes in the set A. Instead of computing *absolute errors* of the table after potential column merge, we compute *changes of errors* for each merge. Since the columns are sorted, this can be done in a single parallel pass through the values in both columns. The changes of errors for all potential merges are stored in a priority queue. After merging the two columns, the changes of errors for pairs that included one of the merged columns are replaced by the errors for pairs in the new column. The total complexity of this algorithm is $O(N(||X|| + ||y_0||^2))$, where N is the number of examples, $||X||$ is the number of attributes and $||y_0||$ number of existing combinations of values of attributes in the set B. A detailed description of the algorithm and its analysis can be found in Demšar (1999) and Zupan (1997).

To reduce computational complexity, HINT explores only k-tuples of attributes that may form a new concept. For the experiments reported in this paper, only couples and triples of attributes were explored.

For the estimation of classification error, HINT uses internal 5-fold cross-validation on the training set to determine the appropriate value of m from a set of candidate values. The value that yields classifiers with best estimated performance is then used when learning from the complete training set.

3. Experimental Evaluation

The minimal-error decomposition as implemented in HINT was evaluated on well-known Monk data sets, selected DEX data sets that were derived from already structured classifiers (where the task was reconstruction of original models), and selected data sets from the UCI machine learning repository.

3.1 Monk Data Sets

The study of 25 machine learning algorithms in (Thrun et al., 1991) used three artificial domains, of which MONK3 included 122 examples where 5% of examples were subject to noise. To test the classifier, a set of 432 examples that covered the complete attribute space and were consistent with the target concept definition were provided. MONK3 uses two two-valued (`c` and `f`), three three-valued (`a`, `b`, and `d`), and a four-valued attribute `e` of which only three are used to define the target concept `e=3 AND d=1 OR e`$\neq$`4 AND b`$\neq$3.

Using cross-validation on the training set, HINT found $m = 0.4$ to be the most appropriate for this data set. The concept structure induced with this value of m (Figure 2.b) correctly classified all the examples in the test set. The data sets discovered by HINT are given in Table 5. First, note that the irrelevant attributes `a`, `c`, and `f` are not present and as they were removed by HINT's preprocessing. Next, the example sets for `b'`, `d'`, and `e'` reveal that HINT successfully found the groups of values which are relevant to the target concept. The interpretation of examples for the intermediate concept `c1` reveals that `c1=2` if `e=4`, `c1=1` if `d=1` and `e=3`, and `c1=0` otherwise. Finally, `MONK3=1` if `c1=2`, or if `c1=1` and `b`$\neq$`3`, which is equivalent to the original concept for MONK3.

Concept hierarchies that HINT derives when using $m = 0.2$ and $m = 0.7$ are given in Figures 2.a and c. Note that with $m = 0.2$, the hierarchy additionally includes an irrelevant attribute `a`, while with $m = 0.7$ a relevant attribute `d` is missing. These two concept hierarchies misclassify 4.6% and 2.8% of the examples in the test set, respectively.

We have also tested HINT on noise-free domains of MONK1 and MONK2 (Thrun et al., 1991). For both, HINT found the best value of m to be 0.0 (which was expected due to noise-free domains) and with discovered concept structure correctly classified all examples in corresponding test sets. Note that in the study by Thrun et al. (1991) there is no single machine learning tool that would achieve this performance on all three Monk domains.

Table 5. Resulting example sets for the concept structure from Figure 2.b for MONK3 as discovered by HINT (class distributions are not shown).

b	b'
1	0
2	0
3	1

d	d'
1	0
2	1
3	1

e	e'
1	0
2	0
3	1
4	2

d'	e'	c1
1	2	2
1	1	0
1	0	0
0	2	2
0	1	1
0	0	0

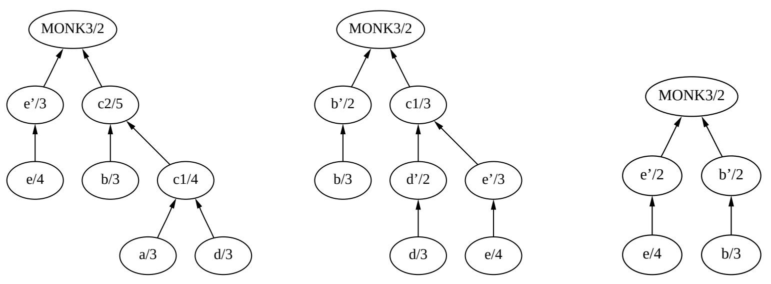
b'	c	MONK3
1	2	0
1	1	1
1	0	0
0	2	0
0	1	1
0	0	1

3.2 DEX Data Sets

We have considered HOUSE and ENTERP domains for which the hierarchical classifiers in DEX (Bohanec & Rajkovič, 1990) formalism already existed. The HOUSE model was used by the Slovenia's Housing Fund to assess the priority of applications for housing loans (Bohanec et al., 1998). Specifically, among several Fund's models we took the one that was used in the Fund's eighth float of loans for applications related to housing renovation and maintenance. ENTERP is one of the models developed at the International Center for Public Enterprises in Ljubljana aimed at performance evaluation of public enterprises (Bohanec & Rajkovič, 1990). The two models were used to obtain a set of examples from which HINT tried to reconstruct the original hierarchies. HOUSE includes nine attributes and uses 4319 examples that are each classified to one of nine class values. These exhaustively define the domain. The majority class appears in 22.8% of examples. ENTERP uses seven attributes, 960 examples and a four-valued class. Some 61.5% of examples in ENTERP are classified into the majority class.

We have investigated the behavior of minimal-error function decomposition through the construction of learning curves, where a variant of stratified ten-fold cross-validation was used. In ten-fold cross validation, the data is divided to ten subsets, of which nine are used for training and the remaining one for testing. The experiment is repeated ten times, each time using a different testing subset. In our case, instead of learning from all examples from nine subsets, only a certain percentage (from 10% to 100% in 10% steps) of training instances from nine subsets are randomly selected for learning. This adaptation of the standard method was necessary to keep test sets independent and compare classifiers as proposed by Salzberg (1997).

The training data was spoiled by noise: 10%, 20% or 30% of instances were assigned a random class. HINT first used the resulting training data to determine the value of m (29 different values were used, ranging from



(a) $m = 0.2$ (b) $m = 0.4$ (c) $m = 0.7$

Figure 2. Concept structures for MONK3 discovered by HINT when different values of m were used.

0 to 20; see Sec. 2.4) and then used the obtained value to derive the concept hierarchy. Results of HINT were compared to those of C4.5 (Quinlan, 1993), where internal 5-fold cross validation on a training data was used to select among 20 different option settings (running C4.5 with and without option -s, -c ranged from 10 to 100 step 10).

The learning curves are given in Figure 3. Overall, HINT performed better than C4.5. For ENTERP, it appears that HINT requires a sufficiently large number of examples to be significantly better than C4.5. When we have examined the concept hierarchies HINT induces, we found that for both domains with increased size of training set HINT converged to hierarchies that were equivalent to the original DEX hierarchies, the only difference being that some concepts from the original hierarchies were further decomposed. These results are consistent with experiments on DEX domains with our previous decomposition method that could not handle noise (Zupan et al., 1999).

3.3 Data Sets from UCI ML Repository

Ten-fold cross validation was used to compare the performance of HINT and C4.5 on selected data sets from UCI machine learning repository (Murphy & Aha, 1994). The m-value for HINT and option settings for C4.5 were determined as in the DEX domains. Table 6 summarizes the results. The null hypothesis (the two classifiers perform equally) that was tested, as proposed in Salzberg (1997), was rejected with probabilities p given in the last column of the table. The two algorithms perform rather similarly in terms of classification accuracy, C4.5 being slightly but insignificantly better. Because of the unavailability of expertise for these data sets, we could not further analyze the relevance of the concepts HINT discovered. Notice that the data sets in Table 6 only contain discrete data. HINT can not directly handle continuous attributes and classes.

Table 6. Classification accuracy, standard errors and significance levels (p) on UCI ML-Repository data sets.

data set	HINT	C4.5	p
shuttle	98.4 ± 2.8	98.8 ± 2.6	0.664
zoo	91.0 ± 9.9	91.2 ± 8.5	0.951
crx	86.1 ± 5.9	85.9 ± 5.0	0.886
soybean	97.5 ± 7.9	98.3 ± 5.3	0.798
led	72.2 ± 4.1	73.6 ± 3.7	0.379
led17	73.1 ± 3.8	72.1 ± 4.3	0.073
breast	79.0 ± 9.5	78.3 ± 8.9	0.698
rheuma	65.0 ± 6.5	65.5 ± 7.6	0.515
diab	70.3 ± 6.4	72.5 ± 5.3	0.146
hear	75.6 ± 7.2	81.1 ± 5.6	0.002
lymp	75.1 ± 10.4	78.3 ± 13.3	0.435

4. Related Work

Originally, function decomposition was intended to support switching circuit design (Ashenhurst, 1952; Curtis, 1962). Given a completely specified tabulated Boolean function, the aim was to derive a hierarchy of functions that can be implemented using simple logic gates. Recent enhancement enabled function decomposition to handle incompletely specified functions (Wan & Perkowski, 1992; Luba & Selvaraj, 1995) and multi-valued attributes and classes (Zupan et al., 1999; Zupan et al., 1997). By these, it became an interesting new paradigm for machine learning.

Within machine learning, Samuel (1967) was the one who first realized the utility of concepts hierarchies. His hierarchies were predefined, but functions were determined from data. Biermann et al. (1982) pointed out the correlation of Samuel's approach and function decomposition. While, unfortunately, their work largely escaped the attention of machine learning community, concepts hierarchies were later used within structured induction (Shapiro, 1987) and DEX (Bohanec & Rajkovič, 1990) approaches that involved experts in crafting both hierarchies and functions. While most of related approaches only use concept hierar-

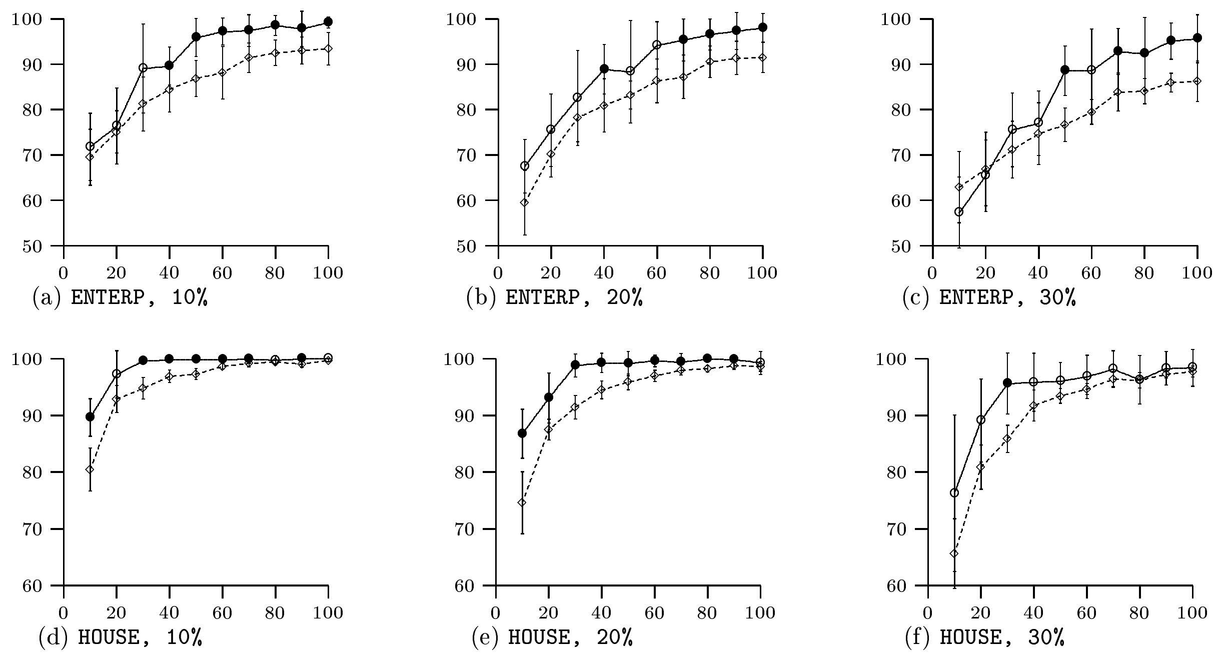


Figure 3. Learning curves for three DEX datasets and three noise levels (10%, 20%, and 30%). Symbol —○— is used for HINT and symbol —●— for HINT when significantly ($p < 0.05$) better than C4.5. Symbol --◇-- is used for C4.5, which was never significantly better than HINT for these two DEX domains.

chies, there are a few including (Tadepalli & Russell, 1998; Bshouty et al., 1995) that derive them from data. When compared to function decomposition as presented in Zupan et al. (1999), these approaches are often limited to Boolean or some other special kind of functions.

The work presented in this paper is based on our original effort described in Zupan (1997). The only other work we are aware of that aims at extending function decomposition to handle noise is that of Perkowski et al. (1998). Their function decomposition program Scomin is limited to binary attributes and classes and it is not clear how its heuristic decides on an appropriate balance between the error on training data and the simplicity of the induced hypothesis. Experimental results presented in that paper do not give any indication how successful Scomin is in terms of prediction accuracy.

5. Conclusions

We have presented a minimal-error function decomposition approach to machine learning. Our work extends the existing function decomposition method (Zupan et al., 1999) with mechanisms that allow appropriate treatment of noise in data. Both approaches aim at decomposing initial set of examples to smaller and more manageable example sets that are organized in a concept hierarchy. Whereas the older approach aimed at reducing the complexity, the approach proposed here aims at minimizing the estimated error when the decomposition example sets are used for classification.

The proposed decomposition method relies on an estimation of classification error that uses m-estimates (Cestnik & Bratko, 1991). Finding the "right" value of m allows decomposition to adjust to the specific noise level in the data. In general, using higher values for m yields less complex concepts and concept hierarchies. This is because with higher m more columns in partition matrices are allowed to be merged. The effect is analogous to decision tree punning, where with higher m's decision trees become simpler.

The proposed decomposition approach is computationally expensive since it has to examine all couples, triples, etc., of attributes in the training set. Heuristic approaches are needed to select and examine only a subset of attribute tuples. Some possibilities for further research include examining only the tuples of most relevant attributes, or, for instance, the use of decision trees as in FRINGE (Pagallo & Haussler, 1990) to determine which attributes may be related.

Experimental evaluation of HINT– the system within which we have implemented the proposed method – shows that decomposition may generalize well and, perhaps more importantly, may discover useful and interpretable concepts. We strongly believe, however, that decomposition should not be used alone and in the "completely automatic" fashion, but should rather be incorporated within interactive data mining systems to propose new concepts that are then reviewed and interpreted by experts. Another potential use is for constructive induction (Liu & Motoda, 1998), where the particular advantage of decomposition over other approaches is discovery of arbitrary functions that relate attributes and form new features.

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., & Beres, T. (1982). Signature table systems and learning. *IEEE Transactions on Systems, Man and Cybernetics*, *12*, 635–648.

Bohanec, M., Cestnik, B., & Rajkovič, V. (1998). Context sensitive decision support systems. In D. Berkeley, G. R. Widmeyer, P. Brezillon and V. Rajkovič (Eds.), *Context sensitive decision support systems*, 174–189. London: Chapman & Hall.

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

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

Cestnik, B., & Bratko, I. (1991). On estimating probabilities in tree prunning. *Proceedings of the Fifth European Working Session on Learning.*

Curtis, H. A. (1962). *A new approach to the design of switching functions.* Van Nostrand, Princeton, N.J.

Demšar, J. (1999). Use of function decomposition for attribute space transformation (in Slovene). Master's thesis, University of Ljubljana, Ljubljana.

Kononenko, I. (1994). Estimating attributes: Analysis and extensions of RELIEF. *Proceedings of the Seventh European Conference on Machine Learning* (pp. 171–182). Springer-Verlag.

Liu, H., & Motoda, H. (1998). *Feature extraction, construction and selection: A data mining perspective.* Kluwer Academic Publishers.

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

Murphy, P. M., & Aha, D. W. (1994). UCI Repository of machine learning databases. Department of Information and Computer Science, University of California at Irvine, Irvine, CA.

Pagallo, G., & Haussler, D. (1990). Boolean feature discovery in empirical learning. *Machine Learning*, *5*, 71–99.

Perkowski, M., Jozwiak, L., & Mohamed, S. (1998). New approach to learning noisy boolean functions. *Proceedings of the Second International Conference on Computational Intelligence and Multimedia Applications* (pp. 693–706). World Scientific. Australia.

Quinlan, J. R. (1993). *C4.5: Programs for machine learning.* Morgan Kaufmann, San Francisco.

Salzberg, S. L. (1997). On comparing classifiers: Pitfalls to avoid and a recommended approach. *Data Mining and Knowledge Discovery*, *1*, 317–328.

Samuel, A. (1967). Some studies in machine learning using the game of checkers II: Recent progress. *IBM Journal of Research and Development*, *11*, 601–617.

Shapiro, A. D. (1987). *Structured induction in expert systems.* Structured Induction in Expert Systems. Addison-Wesley, Wokingham, UK.

Tadepalli, P., & Russell, S. (1998). Learning from examples and membership queries with structured determinations. *Machine Learning*, *32*, 245–295.

Thrun, S. B., et al. (1991). *A performance comparison of different learning algorithms* (Technical Report). CMU-CS-91-197, Dep. of Computer Science, Carnegie Mellon University, Pittsburgh, USA.

Wan, W., & 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. *Proceedings of European Design Automation Conference* (pp. 230–235). Hamburg.

Zupan, B. (1997). *Machine learning based on function decomposition.* Doctoral dissertation, University of Ljubljana, Ljubljana, Slovenia.

Zupan, B., Bohanec, M., Bratko, I., & Demšar, J. (1997). Machine learning by function decomposition. *Proceedings of the Fourteenth International Conference on Machine Learning* (pp. 421–429). San Mateo, CA: Morgan Kaufmann.

Zupan, B., Bohanec, M., Demšar, J., & Bratko, I. (1999). Learning by discovering concept hierarchies. *Artificial Intelligence*, *109*, 211–242.