Decision-Tree Instance-Space Decomposition with Grouped Gain-Ratio

Shahar Cohen[1], Lior Rokach[2], Oded Maimon[1]
[1]*Department of Industrial Engineering, Tel-Aviv University*
[2]*Department of Information Systems Engineering, Ben-Gurion University of the Negev*
SHAHARCO@POST.TAU.AC.IL, LIORRK@BGU.AC.IL, MAIMON@ENG.TAU.AC.IL

Abstract

This paper examines a decision-tree framework for instance-space decomposition. According to the framework, the original instance-space is hierarchically partitioned into multiple subspaces and a distinct classifier is assigned to each subspace. Subsequently, an unlabeled, previously-unseen instance is classified by employing the classifier that was assigned to the subspace to which the instance belongs. After describing the framework, the paper suggests a novel splitting-rule for the framework and presents an experimental study, which was conducted, to compare various implementations of the framework. The study indicates that using the novel splitting-rule, previously presented implementations of the framework, can be improved in terms of accuracy and computation time.

Keywords: Classification, Multiple-Classifier Systems, Instance-Space Decomposition, Decision-Trees

1 INTRODUCTION

Classification is an important task in data mining and machine learning. Given a set of training instances, the classification objective is to induce a classifier, that is, a function that assigns instances to classes based on the realization of some explaining attributes. The classification performance is typically measured in terms of accuracy. Seeking to increase the classification performance, numerous authors have tackled the classification task with multi-classifier methods. Sharkey ([34]) distinguished between two basic multi-classifier methodologies: decomposition and ensemble.

Ensemble methods ([1], [2], [7], [9], [11], [14]) combine multiple classifiers in order to obtain a more accurate and reliable result than when using just a single classifier. In ensemble methods, each classifier is typically trained on data, selected or sampled, from a single, common dataset. Since the training sets are taken from a common dataset, each classifier can entirely solve the original problem.

In the decomposition methodology ([21]), on the other hand, the single classifiers cannot provide a solution to the original problem and the classification requires the combination of all the classifiers. From a different perspective, decomposition can be seen as a methodology that breaks down a classification problem into multiple sub-problems, solves each sub-problem with a unique classifier, and then combines the sub-solutions. In contrast to the ensemble methodology, the sub-classifiers solve the sub-problems rather than the original problem. Combining the multiple classifiers is central in both methodologies and an issue that several authors have addressed ([13], [16], [22], [23], [38]). Decomposition methods can be utilized for solving multi-class problems with binary classifiers ([10], [20], [33]). Combining multiple binary classifiers, rather than using a single (multi-class) classifier can reduce the overall computational complexity and allow the incorporation of methods that are inherently binary (e.g., support vector machines).

Instance-Space Decomposition (ISD) is a specific approach to decomposition. In ISD, the instance space of the original problem is partitioned into several subspaces. A distinct classifier is assigned to each subspace. Subsequently, an unlabeled, previously unseen instance is classified by employing the classifier that was assigned to the subspace to which the instance belongs.

Kusiak ([19]) proposed three ISD strategies (although he did not use the term ISD): object-content decomposition, decision-value decomposition and feature-value decomposition. In these strategies, the partition is not induced automatically. Instead, the analyst must determine the partition prior to the training phase. Brodley ([4]) proposed the model class selection (MCS) system. MCS fits different classifiers to different subspaces of the instance space, by employing one of three classification methods (a decision-tree, a discriminant function or an instance-based method). In

order to select the classification method, MCS uses the characteristics of the underlined training-set, and a collection of expert rules. Brodley's expert-rules were based on empirical comparisons of the methods' performance (i.e., on prior knowledge). In the neural-networks community, Nowlan and Hinton ([26]) examined the mixture of experts (ME) approach, which partitions the instance space into several subspaces and assigns different experts (classifiers) to the different subspaces. The subspaces, in ME, have soft boundaries (i.e., they are allowed to overlap). A gating network then combines the experts' outputs and produces a composite decision. Jordan and Jacobs ([17]) extended ME by considering hierarchical partitions. ME variations have been developed for solving specific-domain problems, such as speech recognition ([12], [28]), time-series analysis ([36]), predicting the survival of AIDS patients ([27]) and handwriting recognition ([30]).

Decision-tree methods ([3], [29]) are frequently employed in classification problems. In an approach, which we term decision-tree ISD, the partition of the instance-space is attained by a decision-tree. Along with the decision-tree, a decision-tree ISD method employs another classification method, which classifies the tree's leaves (the tree's leaves represent the different subspaces). Namely, decision-tree ISD methods produce decision-trees, in which the leaves are assigned with classifiers rather than with simple class-labels. When a non-decision-tree method produces the leaves' classifiers, the composite classifier is sometimes termed a decision-tree hybrid classifier. The term decision-tree hybrid classifier is, however, used also in a broader context, such as in the case where a sub-classification method makes the decisions regarding the growth of the tree and its pruning ([31]). There are two basic techniques for implementing decision-tree ISD. The first technique is to use some decision-tree method to create the tree and then, in a post-growing phase, to attach classifiers to the tree's leaves. The second technique is to consider the classifiers as part of the tree-growing procedure. Potentially, the latter technique can achieve more accurate composite classifiers. On the other hand, it usually requires more computationally-intensive procedures.

Carvalho and Freitas ([5]) proposed a hybrid decision-tree\genetic-algorithm classifier, which tackles decision-tree ISD with the first technique. Their method grows a decision-tree and assigns some of the leaves with class labels and the others with genetic-algorithm classifiers. The leaves that are assigned with the classifiers are the ones that have a small number of corresponding instances. A previously unseen instance is subsequently either directly assigned with a class label or is sub-classified by a genetic-algorithm classifier (depending on the leaf to which the instance is sorted). Zhou and Chen ([40]) suggested a method, called hybrid decision-tree (HDT). HDT uses the binary information gain-ratio criterion, to grow a binary decision-tree, in an instance-space that is defined by the nominal explaining-attributes only. A feed-forward neural network, subsequently classifies the leaves, whose diversity exceeds a pre-defined threshold. The network uses the ordinal explaining-attributes only.

In this paper, we focus on the second decision-tree ISD technique, which considers the classifiers as part of the decision-tree's growth. Employing this technique, Kohavi ([18]) proposed NBTree, a method which produces a decision-tree\naive-Bayes hybrid classifier. In order to decide when to stop the recursive partition of the instance-space (i.e., stop growing the tree), NBTree compares two alternatives: partitioning the instance-space further on (i.e., continue splitting the tree) versus stopping the partition and producing a single naive-Bayes classifier. The two alternatives are compared in terms of their error estimations, which are calculated by a cross-validation procedure. Naive-Bayes classification, by itself, is very efficient in terms of its processing time. However, using cross-validation significantly increases the overall computational complexity. Although Kohavi has used naive-Bayes, to produce the classifiers, other classification methods are also applicable. However, due to the cross-validation estimations, NBTree becomes computationally expensive for methods that are more time-consuming than naive-Bayes (e.g., neural networks).

Although different researchers have targeted decision-tree ISD, there is still no algorithmic framework that is common to all the decision-tree ISD methods. An algorithmic framework helps the analyst to focus on the specific characteristics that differentiate one method from another and to compare different methods. This paper

describes a simple framework for decision-tree ISD, termed decision-tree framework for instance-space decomposition (DFID). The framework hierarchically partitions the instance-space using a top-down (pruning-free) decision-tree procedure. Various implementations of DFID use different stopping-rules, split-validation examinations and splitting-rules. Our work aims to improve the quality of currently available decision-tree ISD methods. We suggest a novel DFID method that can reduce the processing time while keeping the composite classifier accurate. The new method is termed contrasted populations miner (CPOM). CPOM uses a novel splitting-rule, termed grouped gain-ratio. Grouped gain-ratio combines the well-accepted gain-ratio criterion with a heuristic grouping procedure. An experimental study shows that the proposed method outperforms previous decision-tree ISD methods (NBTree and HDT).

The rest of this paper is organized as follows: Section 2 formally defines the objective of ISD. Section 3 describes the DFID algorithmic framework. Section 4 proposes the novel splitting-rule and describes the CPOM algorithm. Section 5 describes the experimental study and discusses the study's results. Finally, Section 6 concludes the work and suggests issues for future research.

2 THE OBJECTIVE OF ISD

We begin this section by presenting our main notations. Consider the training set $S=\{<\boldsymbol{x_1},y_1>,<\boldsymbol{x_2},y_2>,\ldots,<\boldsymbol{x_n},y_n>\}$, where the j^{th} instance (for some $j\in\{1,\ldots,n\}$) consists of $\boldsymbol{x_j}$, a vector of the concrete values of the k explaining attributes, A_1, A_2,…, A_k, and y_j, the instance's class relation (label). There are M possible classes, which are denoted by c_1, c_2,…, c_M. We are only considering discrete explaining attributes, and we denote the possible $d(i)$ values that the attribute A_i can receive by $a_{i,1}$, $a_{i,2}$,…,$a_{i,d(i)}$. The set of these possible values is called the domain of A_i, and is denoted by $dom(A_i)$. The instance-space, denoted by X, is defined by $X=dom(A_1)\times dom(A_2)\times\ldots\times dom(A_k)$. Classification methods are trained to produce classifiers, based on a training set. Let I be a classification method, and $H=I(S)$ the classifier (hypothesis) that was produced by training I based on S. Given an instance $x\in X$, the classifier returns a class relation

$H(x) \in \{c_1, c_2, \ldots, c_M\}$. Many classification methods produce classifiers that return a vector of M probability estimations, describing the likelihood that the classifier associates with the M possible class relations. However, we assume that all the classifiers return a crisp class-relation. Crisp classification can be potentially gained through maximization over the probability estimations.

An instance-space partition breaks down X into multiple (mutually-exclusive and collectively-exhaustive) subspaces $X_1, X_2, \ldots, X_L$. This definition of instance-space partition does not impose any restrictions on its structure. This paper, however, focuses on hierarchical partitions that can be represented by a univariate decision-tree. An ISD method receives S and I; finds an instance-space partition; and returns a composite classifier, that is, a classifier that classifies instances from X_l (for some $l \in \{1,2,\ldots,L\}$) according to the classifier $I(S_l)$, where S_l is the subset of the training instances from S that belong to X_l. The objective of ISD is to find an optimal instance-space partition. Three optimality criteria are considered in this work: generalized accuracy (which is estimated by the proportion of test instances that are classified correctly by the composite classifier); time-complexity; and comprehensibility (which is measured by the number of leaves in the composite classifier).

3 DFID: A DECISION-TREE FRAMEWORK FOR ISD

Implementations of the new decision-tree framework for instance-space decomposition (DFID) consist of a decision-tree (as a wrapper) and another embedded classification method (this method can, in principle, also be a decision-tree). The embedded classification method generates the multiple classifiers for the tree's leaves. The DFID sequence is illustrated by the pseudo code in Figure 1. DFID's inputs are: training instances; a list of attributes (which will be examined as candidates for splitting the decision-tree); a classification method; and optionally (depending on the specific implementation), some additional parameters.

The procedure begins by creating the decision-tree's root node. The root represents the entire instance-space X. When constructed, each node is attached with a

rule, which defines the subspace of X that the node represents. The DFID framework considers rules that can be expressed in a conjunctive normal form. A rule may be, for example: "(A_1=3 $\vee$ A_1=4) $\wedge$ A_2=1". DFID then checks whether there should be a split from the root node (i.e., whether X should be partitioned). This check, which is done using some stopping-rules, is represented, in Figure 1, by the general function `check_stopping_rule`. The function receives some inputs (depending on the specific implementation) and returns a Boolean value that indicates whether the stopping-rules are met. If the stopping-rules are met, then I is trained using all of the training instances, the classifier that results, is attached to the root node, and the procedure terminates. If however, the stopping-rules are not met, then DFID searches for a split, according to some splitting-rule, represented, in Figure 1, by the general function `split`. Splits in DFID are based on the values of a certain candidate attribute. We assume that there exists at least a single attribute that can create a split (or otherwise the stopping-rules would have indicated that there should be no more splits).

The function `split` receives a training set, a set of candidate attributes and optionally some additional inputs, and returns the attribute, on which values the split is based, and a set of descendents nodes. Recall that upon its creation, each node is attached with a rule, which defines the subspace of X that the node represents. The rules for the descendent nodes are conjunctions of the root's rule and restrictions on the values of the selected attribute. The split that was found may be then subjected to a validation examination, represented, in Figure 1, by the general function `validate`. If a split is found to be invalid, then DFID will search for another split (another attribute). If there are no more candidate attributes, I will be trained using all the training instances, and the classifier that results will be attached to the root node. As soon as a valid split is found, the descendent nodes that were created by the split are recursively considered for further splits. Further splits are achieved by the recurrence of DFID. In the recurrence, only a subset of the training instances is relevant (the instances that are actually sorted to the certain descendent node). In addition, the attribute, which defined the current split, is removed from the list of

candidate attributes. The descendents are finally linked to their parent (the root). Different DFID implementations may differ in all or some of the procedures that implement the three main framework's components – stopping-rules (the function `check_stopping_rule`), splitting-rules (the function `split`) and split validation examinations (the function `validate`).

3.1 STOPPING-RULES

Stopping-rules are checked by the general function `check_stopping_rule` (Figure 1). However, it should be noticed that a negative answer by this function is not the only condition that stops the DFID recurrence; another, and even more natural, condition, is the lack of any valid split.

NBTree ([18]) uses a simple stopping-rule, according to which no splits are considered, when there are 30 instances or less in the examined node. Splitting a node with only few training instances will hardly affect the final accuracy and will lead, on the other hand, to a complex and less comprehensible decision-tree (and hence a complex and less comprehensible composite classifier). Moreover, since the classifiers are required to generalize from the instances in their subspaces, they must be trained on samples of sufficient size. Kohavi's stopping-rule can be revised to a rule that never considers further splits in nodes that correspond to $\beta|S|$ instances or less, where $0<\beta<1$ is a proportion and $|S|$ is the number of instances in original training set, S. When using this stopping rule (either in Kohavi's way or in the revised version), a threshold parameter must be provided to DFID as well as to the function `check_stopping_rule`. Another heuristic stopping-rule is never to consider splitting a node, if a single classifier can accurately describe the node's subspace (i.e., if a single classifier which was trained by all of the training instances, and using the classification method appear to be accurate). Practically, this rule can be checked by comparing an accuracy estimation of the classifier to a pre-defined threshold (thus using this rule requires an additional parameter). The motivation for this stopping-rule is that if a single classifier is good enough, why replace it with a more complex tree

that also has less generalization capabilities. Finally, as mentioned above, another (inherent) stopping-rule of DFID is the lack of even a single candidate attribute.

3.2 SPLITTING-RULES

The core question of DFID is how to split nodes. The answer to this question lies in the general function `split` (Figure 1). It should be noted that any splitting-rule that is used to grow a pure decision-tree, is also suitable in DFID. In Section 4 we propose a novel splitting rule, which combines the well-known gain-ratio splitting rule with a grouping heuristic.

Kohavi ([18]) has suggested a new splitting-rule, which is to select the attribute with the highest value of a measure, which he referred to as the "utility". Kohavi defined the utility as the 5-fold cross-validation accuracy estimation, of using a naive-Bayes method for classifying the subspaces, which will be generated by the considered split.

3.3 SPLIT VALIDATION EXAMINATIONS

Since splitting-rules, are heuristic, it may be beneficial to regard the splits they produce as recommendations that should be validated. Kohavi ([18]) validated a split by estimating the reduction in error, which is gained by the split, and comparing it to a predefined threshold of 5% (i.e., if it is estimated that the split will reduce the overall error rate by only 5% or less, the split is regarded as invalid). In an NBTree, it is enough to examine only the first proposed split in order to conclude that there are no valid splits, if the one examined is invalid. This follows since in an NBTree, the attribute according to which the split is done, is the one that maximizes the utility measure, which is strictly increasing with the reduction in error. If a split, in accordance to the selected attribute cannot reduce the accuracy by more than 5%, then no other split can.

This work suggests a new split validation procedure. More details are provided in Section 4, but in very general terms, a split according to the values of a certain

attribute is regarded as invalid, if the subspaces that result from this split are similar enough to be grouped together.

```
DFID (S,A,I)
Get
   S – a training set
   A – a set of candidate input attributes
   I – a classification method
Return
   A classifier

 Create a tree with a root node;
 IF check_stopping_rule(S,A,I)
   Attach the classifier I(S,A) to the root;
 ELSE
   A*←A;
   valid←FALSE
   WHILE A*≠∅ and NOT(valid)
       (split_att,nodes)←split(S,A*);
       IF validate(nodes,split_att,S)
          valid←TRUE;
          A←A \ split_att;
          FOR each node ∈ nodes
            node_instances←the instances that belong to node;
            attach the classifier DFID(node_instances,A,I) to
               node;
            link node to root;
          END FOR
       ELSE
          A*←A* \ split_att;
       END IF
   END WHILE
   IF NOT (valid)
       Attach the classifier I(S,A) to the root;
   END IF
 END IF
 RETURN tree;
```

Figure 1: DFID outline. A DFID implementation recursively partitions the instance space of the training set, according to the values of the candidate attributes. As the recursive partition ends, classifiers are attached to the leaves, by employing the embedded classification method.

4 THE CPOM ALGORITHM

This section presents a new DFID implementation, termed contrasted population miner (CPOM), which splits nodes according to a novel splitting-rule, termed grouped gain-ratio. Generally speaking, this splitting-rule is based on the gain-ratio criterion ([29]), followed by a grouping heuristic. The gain-ratio criterion selects a single attribute from the set of candidate attributes, and the grouping heuristic thereafter groups together subspaces, which correspond to different values of the selected attribute.

4.1 CPOM OUTLINE

CPOM uses two stopping-rules. First, the algorithm compares the number of training instances to a pre-defined ratio of the number of instances in the original training-set. If the subset is too small, CPOM stops (since it is undesirable to learn from a too small training subset). Secondly, CPOM compares the accuracy estimation of a single classifier to a pre-defined threshold. It stops if the accuracy estimation exceeds the threshold (if a single classifier is accurate enough, there is no point in splitting further on). Therefore, in addition to the inputs in Figure 1, CPOM must receive two parameters: β, the minimal ratio of the training instances and *acc*, the maximal accuracy estimation, that will still result in split considerations.

CPOM's split validation procedure is directly based on grouped gain-ratio. The novel rule is described in detail, in the following subsection; however, in general terms, the rule returns the splitting attribute and a set of descendent nodes. The nodes represent subspaces of X that are believed to be different. If the procedure returns just a single descendent node, the split it has generated is regarded as invalid.

4.2 THE GROUPED GAIN-RATIO SPLITTING-RULE

Grouped gain-ratio is based on the gain-ratio criterion ([29]), followed by a grouping heuristic. The gain-ratio criterion selects a single attribute from a set of candidate attributes. The instance-subspace, whose partition we are now considering, may, in

principle, be partitioned so that each new sub-subspace will correspond to a unique value of the selected attribute. Group gain-ratio avoids this alternative, through heuristically grouping sub-subspaces together. By grouping sub-subspaces together, grouped gain-ratio increases the generalization capabilities, as there are more instances in a group of sub-subspaces than there are in the individual sub-subspaces.

Before introducing the grouping heuristic, we provide some intuition. During each split, we are considering a node of the decision-tree, which represents an instance-subspace, and have an associated subset of the original training set. For expository reasons we will focus on the split from the tree's root (that is, we assume that we are searching for a split that will partition the entire instance-space X into several subspaces). The following intuition can be applied to the non-root nodes of the tree as well. Let us further assume that there will be a single split at most (i.e., we either split the root node and attach classifiers to its descendants, or we attach a single classifier directly to the root). If we decide to split, it means that there are several subspaces of X that will be assigned with different classifiers. The only reason for us to prefer this alternative is if we believe that the entire set, S, cannot be accurately described by the single classifier $I(S)$, and we think that it is better to train I separately, on the training subsets that correspond to the subspaces of X. Clearly, if we separately train I on each subset and obtain the same exact classifier from each subset, then there is no point in the split, since using this single classifier for the entire instance-space is as accurate as using the multiple classifiers; it is also much simpler and understandable, and it can generalize better. The other direction of this argument is slightly less straightforward. If the classifiers that were trained over the training subsets are very different from one another, then none of them can classify X as one, and we may believe that the split is beneficial. Based on this observation, the grouped gain-ratio splitting-rule groups together subspaces that have similar classifiers.

The intuition regarding the classifier comparisons raises questions of what is similar, what is different and how to compare classifiers? Although there may be multiple classifiers, all of which must be simultaneously compared to each other, we begin answering these questions with the simpler case of exactly two classifiers, using

a comparison heuristic, which we refer to as cross-inspection (see Figure 2). Cross-inspection is based on two mutually-exclusive training subsets and a classification method as inputs. The comparison begins by randomly partitioning each subset into a training sub-subset and a test sub-subset. Then, two classifiers are produced, by training the input method, once over each training sub-subset. After producing the two classifiers, the cross-inspection heuristic calculates the error rates of each classifier over each of the test sub-subsets. If the error rate of the first classifier over the first test sub-subset is significantly (with confidence level `alpha`) different from the error of the first classifier over the second test sub-subset, or vice versa, then the two classifiers are regarded as different. As Dietterich suggests ([6]), the errors are compared by testing the hypothesis that the errors are generated by the same binomial random variable. The confidence level of `alpha`=0.01 was used throughout this paper.

The cross-inspection heuristic compares only two distinct classifiers. However, in the DFID framework more than two classifiers must be compared at a time (if the attribute, which was selected by the gain-ratio criterion, has more than two possible values). For example, if it is believed that graduate students from different schools behave differently, one may consider splitting according to the school's name. The attribute 'school' can receive multiple values, all of which will have to be compared simultaneously. A successful split will group similar schools together, while different schools will be in different groups. Since an exhaustive search, over all the possible groupings, is unacceptable in terms of complexity, grouped gain-ratio (see Figure 3) uses a greedy grouping heuristic, which is based on cross-inspection. The procedure begins by using cross-inspection, to compare all the distinct pairs of classifiers (if there are q classifiers, there are $q(q\text{-}1)/2$ comparisons). For each instance-subspace, the procedure computes the number of instances that belong to subspaces that are similar to it (by definition the similarity by cross-inspection is defined with regard to classifiers rather than subspaces; each subspace, however, is described by a classifier). The classifier that represents the subspace with the largest such number is regarded as the classifier that covers the maximal number of instances. The subspaces of all the

instances which are covered by this classifier are grouped together, and the procedure iterates. The greedy aspect in grouped gain-ratio is similar to the considerations that Harries and Horn presented ([15]). The heuristic does not explicitly guarantee that any two classifiers in a group are equivalent, but equivalence is assumed to be a transitive relation. The greedy grouping procedure is a simple clustering method and other clustering methods, like graph coloring ([41]) may also be suitable here. Alternatively one could use the Warshall algorithm ([35]) for finding the transitive closure of the comparison matrix, which can be used for calculating $\mathtt{sup}_j$. However, this form of calculation will not be convenient in this case because it will tend to group too much as it is illustrated in the following example.

We demonstrate grouped gain-ratio with a simple example. Assume that we are considering a split from the root node, and that the gain-ratio criterion has selected the attribute A_1, which has six possible values. The training set S is consequently partitioned into six mutually-exclusive subsets, and the embedded classification method is trained six times, once over each subset. The six classifiers that result are then compared in pairs, and each pair is marked as either similar or different. Let the result of this comparison be as described by Figure 4 (A) (notice that each classifier is by definition equivalent to itself), and assume that the six subsets, corresponding to $a_{1,1}$ through $a_{1,6}$, have 100, 120, 150, 90, 80 and 200 instances respectively. The classifier, which was trained on the first subset, therefore covers 100+150+200=450 instances. In the same way, the remaining classifiers cover 320, 450, 170, 170 and 570 instances respectively. Therefore, the instance-subspace, which is associated with the classifier that covers the maximal number of instances, is the subspace in which $A_1=a_{1,6}$. Grouped gain-ratio will group this subspace with the subspaces in which $A_1=\mathrm{OR}(a_{1,1}, a_{1,2}, a_{1,3})$. Since the two subspaces that remain can be seen to have equivalent classifiers, there will be another group, and the split will be as in Figure 5. Notice that the subspaces in which $A_1=a_{1,2}$ and $A_1=a_{1,3}$ were grouped together although their corresponding classifiers were marked as non-equivalent. In this example, using the transitive closure will leads to the same results. However, If the comparison matrix looked like in Figure 4 (B) (notice that the only difference between

the two matrices is in the cell $a_{1,3}$-$a_{1,6}$) then the six classifiers would have covered 450, 320, 150, 170, 170 and 420 instances respectively. Consequently, there would have been three subgroups: $\{a_{1,1}; a_{1,3}; a_{1,6}\}$, $\{a_{1,2}\}$ and $\{a_{1,4;} a_{1,5}\}$. Notice however, that the transitive closure, if used, would leads to the same results that were obtained in the case of Figure 4 (A).

```
cross_inspection (S1,S2,I,alpha)
Get
   S1,S2 – mutually-exclusive training sets
   I – a classification method
   alpha – a confidence level
Return
   a Boolean value reflecting equivalence

 S11 ← a random sample from S1;
 S12 ← S1 \ S11;
 S21 ← a random sample from S2;
 S22 ← S2 \ S21;
 H1 ← I(S11);
 H2 ← I(S21);
 FOR i,j∈{1,2}
   εi,j ← accuracy estimation of Hi over Sj,2;
 END FOR
 IF ε1,2 is different from ε1,1 with a confidence level alpha,
       or ε2,1 is different from ε2,2 with confidence level alpha
   return FALSE;
 ELSE
   return TRUE;
 END IF
```

Figure 2: The cross-inspection procedure outline. The procedure compares the accuracy estimations of two distinct classifiers, searching for statistical significance.

```
grouped_gain_ratio (S,A,I,root,alpha)
Get
   S – a training set
   A – a set of candidate input attributes
   I – a classification method
   root – the node from which the split is considered
   alpha – confidence level
Return
   split_att – the attribute that splits the current node
   nodes – the set of nodes that results from the split

 A_i ← the attribute from A with the maximal gain-ratio;
 S_1,S_2,…,S_d(i) ← a partition of S, according to values of A_i;
 FOR all j and k in {1,2,…,d(i)} so that j≤k
    E_j,k ← cross_inspection(S_j,S_k,I,alpha);
    E_k,j ← E_j,k;
 END FOR
 FOR all j ∈ {1,2,…,d(i)}
    sup_j ← the number of instances in the subsets S_k for which
           E_j,k=TRUE;
 END FOR
 L ← a list of the subsets indices sorted descending by sup_j;
 nodes ← an empty set of nodes
 WHILE L is not empty
    create a new node;
    Attach the rule which is a conjecture of the root's rule
      and a disjoint of the values that correspond to S_j the
      first member of L and the members S_k for which E_j,k=TRUE;
    Remove from L any member that is described by the new node;
    Add node to nodes;
 END WHILE
 RETURN (A_i,nodes)
```

Figure 3: The grouped gain-ratio procedure outline. The procedure groups together similar values of a candidate attribute. Similarity is based on the cross-inspection heuristic.

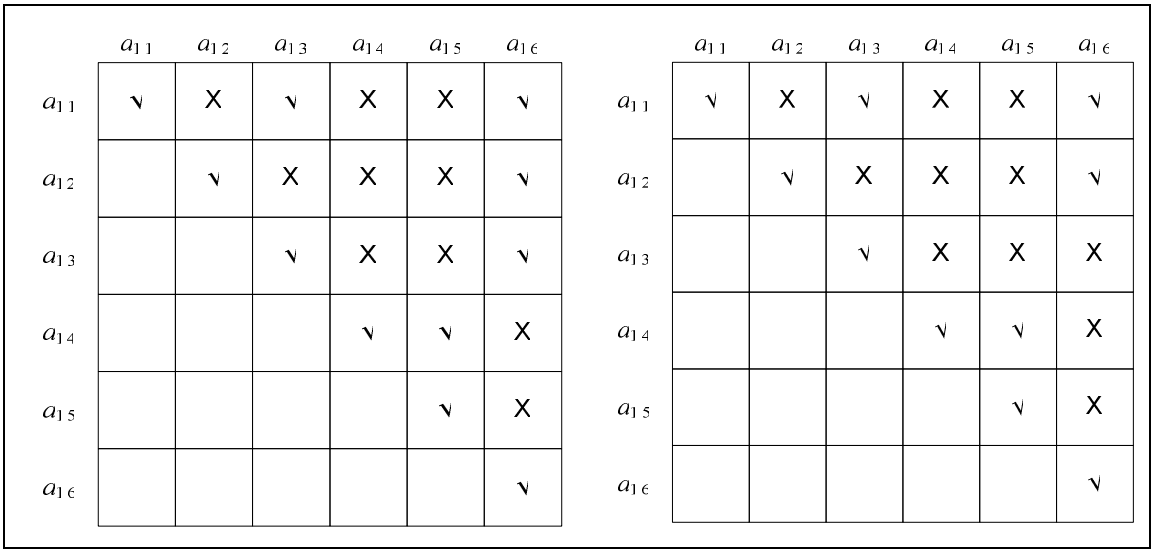

Figure 4: An illustration of the pair-wise comparison results. √ represents equivalence and X represents non-equivalence.

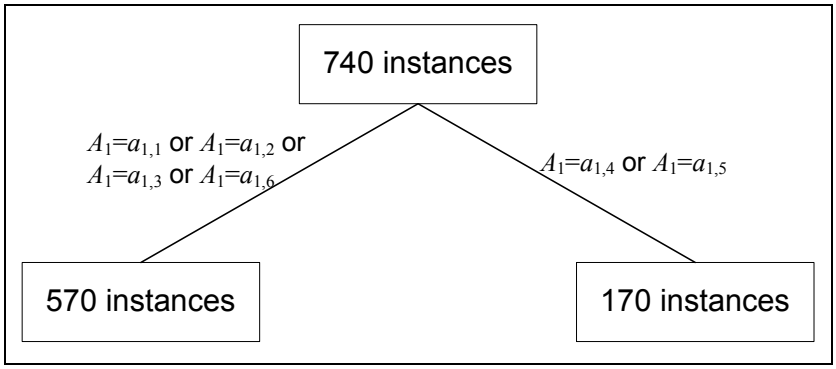

Figure 5: The split that results from grouped gain-ratio.

5 EXPERIMENTAL STUDY

A comparative experimental-study was carried out, using mainly benchmark data sets (three synthetic datasets were handcrafted for the experiments in Section 5.10). The primary objective of the study was to evaluate the potentials of the DFID framework, and especially of the CPOM algorithm. The following subsections describe the experimental set-up and discuss the obtained results.

5.1 THE EMBEDDED CLASSIFICATION-METHODS

The experimental study compared the performance of CPOM, when using the following embedded classification methods: naive Bayes, backpropagation (to train artificial neural networks) and C4.5. The naive Bayes method was chosen in order to compare CPOM with NBTree, and backpropagation was chosen in order to compare CPOM with the HDT algorithm. The C4.5 method was chosen because it is considered to be a state-of-the-art decision-tree algorithm, and is widely used in many other comparative studies.

All the experiments were made with the WEKA environment ([37]). The experiments with C4.5 took place using J48, the Java version of C4.5. We have used the NBTree implementation, which is included in WEKA for simulating Kohavi's original work. We also implemented HDT in WEKA. The original implementation of HDT has utilized a specific multi-layer, feed-forward neural-network named FANNC ([38]). However, in our implementation, we employed backpropagation, that is already available in the WEKA environment, and which is more widely-used in the literature. CPOM was, of course, also implemented in WEKA. In all the CPOM executions, 95% was chosen as the maximal accuracy estimation that would still be considered for further splits and the minimal training-subset size was chosen to be one-fifth of the initial training-set size.

5.2 THE BENCHMARK DATASETS

All the compared methods were trained over 20 datasets, which were manually selected from the UCI Machine Learning Repository ([24]). Although this repository’s limitations for comparing algorithms are known ([32]), it is still considered to be objective since the published results can be validated. The selected datasets vary across several dimensions: the number of classes, the number of instances, the number of explaining attributes and the type of attributes. Table 1 describes the datasets' characteristics.

Table 1: The datasets' characteristics

Dataset	No. of Attributes	No. of Instances	No. of Classes	% of Numeric Attributes
Audiology	70	200	23	0
Australian	15	690	2	40
Breast Cancer	10	699	2	100
Car	7	1,728	4	0
Hayes-Roth	5	132	3	0
Iris	5	150	3	100
Labor	17	57	2	50
Led-17	25	220	10	0
Letter	17	15,000	26	100
Monk-1	7	124	2	0
Monk-2	7	169	2	0
Monk-3	7	122	2	0
Mushroom	22	8,124	2	0
Nurse	9	12,960	5	0
Sonar	61	208	2	100
Soybean	36	683	19	0
Tic-Tac-Toe	10	958	2	0
Vote	17	435	2	0
Wine	14	178	3	100
Zoo	17	101	7	12

The datasets went through a simple preprocessing stage. In this stage, missing values were replaced by a distinctive value, and numeric attributes were made discrete by dividing their original range into ten equal-sized intervals (or one per observed value, whichever was least). Accuracy results could have been improved by using a more robust way for treating the missing values (see for example [29]).

5.3 THE EVALUATION CRITERIA

The following list describes the evaluation criteria that were measured in each execution of each of the evaluated methods.

i. The Generalized Accuracy

The generalized accuracy is the probability that an unlabeled, previously-unseen instance will be classified correctly, by the output (possibly composite) classifier. In order to estimate this probability, a 10-fold cross-validation procedure has been

implemented. In 10-fold cross-validation, the dataset is randomly partitioned into 10 disjoint, equal-sized subsets. Each subset is used once as a test set and nine times as part of the training set. The partition (i.e., the same folds) was used in all of the methods. Furthermore, a single-tailed paired t-test, with a confidence level of 95%, was used in order to verify the statistical significance of the differences between the accuracy estimations, of the examined methods.

ii. The Number of Leaves in the Composite Decision-Tree

The complication of the output-classifier was measured in terms of the total number of leaves in the composite decision-tree. The lower this number, the simpler and potentially more comprehensible and general is the composite classifier.

iii. The Overall Number of Sub-Classifiers that were Induced

This criterion indicates the number of times, in which the embedded classification methods was trained, in order to produce the composite classifier. This criterion may have a dramatic effect on the computational complexity. This effect is especially important when the computational complexity of the embedded method is more than linear (such as in the case of C4.5 or backpropagation).

iv. Execution Time

The execution time is the actual time (in seconds), required for producing the composite classifier. We conducted all of our experiments on the following hardware configuration: A desktop computer with an Intel Pentium 4-2.8GHz, Windows XP operating system, and 1GB of physical memory.

5.4 CPOM WITH NAIVE BAYES: RESULTS AND DISCUSSION

Table 2-A compares the performance of CPOM, with naive Bayes as the embedded classification method (CPOM-NB) to NBTree and simple naive Bayes. The results indicate that the average accuracy of CPOM-NB (86.74%) is higher, by about 3%, than the average accuracy of NBTree (83.75%), and by about 4% than the average accuracy of simple naive Bayes (82.99%). Examining the statistical significance of

the results indicates that CPOM-NB is significantly more accurate than NBTree in eight datasets, while NBTree shows significantly higher accuracy in only three datasets. The "+" superscripts, next to some of the accuracy measures, in Table 2-A, indicate that the accuracy of CPOM-NB was significantly higher (with a confidence level of 5%) than the accuracy of the method under which the "+" is superscripted. The "–" superscripts, similarly, indicate that the accuracy of CPOM-NB was significantly lower.

Table 2-A also presents the number of leaf nodes in the composite classifier, and the number of (inner) classifiers that were needed, as part of the construction of this composite classifier. It should be recalled that the number of leaves is a way of assessing the classifier's comprehensibility, and the number of inner-classifiers is a way of assessing the computational complexity. The table indicates that CPOM-NB required only 10% of the leaves and 2% of the inner classifiers, compared to NBTree. The reduced number of inner-classifiers is partly due to the more compact trees that CPOM-NB builds, but it is also (and probably mainly) due to the splitting rule that NBTree uses. In order to select the attribute with the highest utility, NBTree estimates the accuracies of all the possible splits, where each such split is evaluated using a 5-fold cross-validation procedure. The number of inner-classifiers that are needed for this splitting-rule becomes a significant burden when the number of attributes and overall splits increases. CPOM, on the other hand, does not use cross-validation but builds inner-classifier only for the attributes that are selected by gain-ratio splitting criterion. The last observation is also supported by the actual execution time of the two algorithms. The execution of CPOM-NB took, on average, only 13% of the time that NBTree required.

We conclude that CPOM outperforms NBTree in all the important criteria: accuracy, model complexity (as measured by number of leaves) and execution time. The high number of inner-classifiers that NBTree produces implies that this method may be impractical, when employed with more computational-intensive embedded methods, such as neural networks. On the other hand, as subsequent sections demonstrate, CPOM, can be used with other embedded methods.

The results also indicate that there was not a single dataset on which simple naive Bayes was significantly more accurate than CPOM-NB. On the other hand, CPOM-NB was significantly more accurate than naive Bayes in nine of the datasets. In terms of the training time, naive Bayes is, clearly, faster than CPOM-NB. However, we believe that the speed of CPOM is acceptable.

Table 2-B provides the root mean square error (RMSE) of the compared methods. For every dataset, we provide both the RMSE of the training set and the RMSE, which is based on 10-folds cross validation. The results indicate that on average CPOM-NB obtains the best RMSE values (with a mean of 0.213685) followed by NBTree (with a mean of 0.228955) and naive Bayes (with a mean of 0.25192). As expected in most of the cases the RMSE of the training set is smaller than the RMSE calculated over the 10 folds cross validation. Nevertheless it can be seen that the lowest difference between these two values is obtained for the naive Bayes algorithm with a mean of 0.03187. The largest difference is obtained for the NBTree algorithm with a mean of 0.08986. The mean difference obtained for the CPOM-NB is 0.05561. These results can be explained by the complexity of the classifier (that can be measured by the number of leaves presented in Table 2-A). As the number of leaves increases, so does the RMSE difference.

Table 2-A: Comparison of CPOM-NB to NBTree and naive Bayes

Dataset	NBTree				CPOM-NB				Naive Bayes	
	Accuracy	# leaves	# inner-classifiers	Execution time (in sec.)	Accuracy	# leaves	# inner-classifiers	Execution time (in sec.)	Accuracy	Execution time (in sec.)
Audiology	72.5±3.16	27	7210	27.94	72±5.84	14	247	1.92	+65.5±7.39	0.01
Australian	84.78±2.64	24	5620	4.72	87.24±3.96	6	606	0.49	84.93±2.7	0.01
Breast Cancer	96.56±1.46	28	2250	0.83	97.42±1.16	1	92	0.05	97.42±1.16	0.01
Car	+85.30±0.92	1	105	0.3	93.92±0.74	8	90	0.13	+85.30±0.92	0.01
Hayes-Roth	+67.42±7.11	10	225	0.32	77.27±8.56	4	54	0.06	81.06±9.61	0.01
Iris	94±5.17	4	220	0.14	96±6	2	8	0.05	95.33±5.05	0.02
Labor	+87.71±4.73	3	510	0.44	94.73±4.61	2	32	0.01	92.98±4.52	0.01
Led-17	58.63±4.16	6	1815	4.27	60.90±5.12	8	28	0.22	63.18±8.7	0.01
Letter	-81.13±0.82	916	52180	234.61	77.70±0.64	25	892	32.34	+73.29±1	0.13
Monk-1	+91.12±4.16	5	340	0.2	97.58±3.97	7	72	0.02	+73.39±6.7	0.01
Monk-2	58.58±2.21	6	420	0.34	59.53±5.43	4	57	0.03	+56.21±6.1	0.01
Monk-3	92.62±3.61	1	30	0.06	92.62±3.61	1	6	0.02	92.62±3.61	0.01
Mushroom	-99.95±0.07	18	12420	9.02	99.37±0.17	10	262	1.12	+95.48±0.9	0.08
Nurse	-95.92±0.96	139	45500	10.11	94.21±0.54	15	77	0.83	+90.26±0.49	0.05
Sonar	+62.98±7.68	19	9000	17.95	76.44±7.62	2	772	4.21	75.48±7.3	0.02
Soybean	+91.51±1.27	34	9500	55.61	93.41±1.49	18	122	0.24	91.95±1.99	0.02
Tic-Tac-Toe	75.67±5.82	51	2135	2.64	76.51±1.87	7	18	0.08	+69.27±3.2	0.01
Vote	94.06±3.24	35	4080	1.17	96.2 ±3.31	2	8	0.11	+90.34±3.44	0.01
Wine	+93.44±5.69	10	1950	2.72	96.62±3.35	1	368	0.01	96.62±3.35	0.01
Zoo	+91.08±6.25	8	540	0.42	95.04±5.05	5	36	0.03	+89.11±7	0.01

Table 2-B: Comparison of the RMSE of CPOM-NB, NBTree and naive Bayes

Dataset	NBTree		CPOM-NB		Naive Bayes	
	RMSE (Training Set)	RMSE (10 Fold CV)	RMSE (Training Set)	RMSE (10 Fold CV)	RMSE (Training Set)	RMSE (10 Fold CV)
Audiology	0.1088	0.1447	0.1173	0.1359	0.1209	0.148
Australian	0.2065	0.348	0.2866	0.3271	0.3261	0.3363
Breast Cancer	0.1557	0.1704	0.1557	0.1593	0.1557	0.1593
Car	0.2218	0.2263	0.115	0.1577	0.2218	0.2263
Hayes-Roth	0.3114	0.3762	0.3021	0.3624	0.3066	0.3541
Iris	0.1251	0.1621	0.1268	0.1482	0.1495	0.155
Labor	0.0751	0.2961	0.1041	0.251	0.1532	0.2637
Led-17	0.1188	0.2413	0.1499	0.2294	0.1718	0.226
Letter	0.0674	0.1058	0.0881	0.126	0.1166	0.121
Monk-1	0.1447	0.2952	0.1399	0.2761	0.3836	0.4089
Monk-2	0.3368	0.5052	0.3905	0.4895	0.4711	0.5048
Monk-3	0.2644	0.2891	0.2644	0.2851	0.2644	0.2851
Mushroom	0.0106	0.006	0.0111	0.015	0.181	0.1853
Nurse	0.0893	0.1	0.117	0.1364	0.1762	0.1767
Sonar	0.0712	0.4655	0.2092	0.4361	0.2733	0.4425
Soybean	0.0543	0.0885	0.0641	0.0707	0.0765	0.0817
Tic-Tac-Toe	0.173	0.3152	0.2413	0.3073	0.423	0.4298
Vote	0.1552	0.183	0.1496	0.1669	0.2924	0.2992
Wine	0.0822	0.1643	0.0895	0.1326	0.0895	0.1326
Zoo	0.0478	0.0962	0.0393	0.061	0.0478	0.1021

5.5 CPOM WITH NEURAL NETWORKS: RESULTS AND DISCUSSION

Table 3 compares the performance of CPOM, with backpropagation as the embedded method (CPOM-NN) to the performance of HDT and simple backpropagation. Backpropagation was trained on 500 epochs and the number of hidden units that was used in each dataset is described in Table 3. The results indicate that the average accuracy of CPOM-NN is higher by about 1% (92.76%) than the average accuracy of HDT (91.69%) and by about 2% from the average accuracy of a single neural network (90.63%). Although the mean differences are relatively moderate, some of the per-dataset differences are statistically significant. Specifically, CPOM-NN is significantly more accurate than HDT in five datasets, and significantly more accurate than a single neural network in eight datasets. On the other hand, HDT and a single neural network were not found to be significantly more accurate than CPOM-NN in any of the datasets.

The improved accuracy of CPOM-NN required an execution time that is, on average, 14 times greater than the execution time of simple backpropagation. When compared with the execution time of HDT, it can be seen that, on average, CPOM-NN

required an execution time that is four times greater than that of HDT. Notice that in HDT the leaf-classifiers are trained in a post-growing phase (see the discussion on the two decision-tree ISD techniques in Section 1). Still, there are four datasets, in which HDT required more time than CPOM-NN.

Table 3 also presents the number of leaf nodes in the composite classifier and the number of inner-classifiers that were needed, by the two methods. CPOM-NN requires more than two times of inner-classifiers, than HDT requires. It should be noted, again, that in HDT, the leaf-classifiers are trained only after the final tree structure is decided upon. Moreover, not all leaves in HDT have a classifier (it depends on the leaf-node's diversity). Thus the increased number of inner-classifiers that CPOM-NN requires is not surprising. On the other hand, the table indicates that CPOM-NN tends to build trees with fewer leaves. It can be seen that HDT on average results in five times more leaves than CPOM-NN. Namely, CPOM-NN creates more compact and comprehensible composite classifiers.

5.6 CPOM WITH DECISION-TREES: RESULTS AND DISCUSSION

At first sight it seems pointless to use CPOM with C4.5 as the embedded method (CPOM-C4.5), since the result of this configuration is a pure decision-tree. Nonetheless, this section describes experiments with CPOM-C4.5, with a twofold motivation. First, CPOM-C4.5 can be seen as a kind of lookahead-based method for producing decision-trees. Lookahead-based algorithms attempt to predict the profitability of a split at some node by estimating the effect of this split on deeper decedents of the node ([8], [25]). By using CPOM-C4.5, one actually examines the effect of a certain split with the depth of at least two levels. The second motivation for using CPOM-C4.5 can be explained by the grouped gain-ratio splitting rule that suggests a new way to branch the tree. The combination of the new splitting rule together with C4.5's splitting-rule extends the tree's search-space.

Table 3: Comparison of CPOM-NN to HDT and backpropagation

Dataset	HDT				CPOM-NN					Backpropagation	
	Accuracy	# leaves	# inner-classifiers	Execution time (in sec.)	Accuracy	# leaves	# inner-classifiers	Execution time (in sec.)	# Hidden Units	Accuracy	Execution time (in sec.)
Audiology	95.11±3.22	31	12	8911.62	96.01±2.78	6	74	9798.06	47	+93±1.22	203.52
Australian	85. 8±2.94	23	2	18.66	86.31±2.18	5	55	212.17	9	85.2±2.66	12.87
Breast Cancer	95.6±1.31	55	48	4720	95.28±1.16	3	621	6409.38	6	95.70±1.28	446.66
Car	99.25±0.44	131	35	966.05	99.25±0.44	4	18	562.14	6	99.65±0.35	91.92
Hayes-Roth	81.94±8.3	19	6	22.15	81.81±9.06	3	34	29.66	4	81.0±8.18	4.08
Iris	+93.98±4.72	4	1	5.91	96.1± 4.1	2	8	20.85	4	95.7±3 .2	5.12
Labor	+93.71±4.73	3	1	24.11	96.48±3.83	4	42	294.82	10	+93.9± 3.1	22.18
Led-17	55.71±1.77	35	35	152.08	56.71±1.17	8	102	432.59	18	+50±0.96	20.63
Letter	+89.5±4.38	9562	281	3996.72	96.2±1.73	8	45	1281.11	22	+82.51±1.43	32.18
Monk-1	95.17±6.39	12	1	4.77	94.35±6.91	3	10	10.86	5	95.97±6.44	3.03
Monk-2	+96.16±0.93	18	2	8.61	100±0	1	3	9.51	5	100±0	4
Monk-3	92.62±3.61	9	0	0.61	91.95±9.28	3	8	10.38	5	+89.34±6.19	2.97
Mushroom	100±0	23	7	25432.21	100±0	6	32	98471.95	12	100±0	7858.17
Nurse	99.32±0.37	355	66	41268.64	99.32±0.37	6	22	13268.64	7	+98.76±0.42	949.64
Sonar	+84.24±5.72	42	26	644.8	87.1±4.16	4	67	1121.58	32	+84.14±3.84	62.19
Soybean	93.2±1.27	56	46	396.86	93.7±1.73	8	98	646.8	28	90.21±2.68	33.17
Tic-Tac-Toe	95.67±4.95	95	24	755.11	96.87±4.22	7	18	717.94	6	+92.11±5.17	61.45
Vote	96.12±3.82	11	1	57.89	96.17±2.50	2	8	226.08	10	96±1.05	55.22
Wine	95.66±2.11	37	6	981.76	96.62±3.35	3	62	6712.09	9	94.44±3.22	228.69
Zoo	95.04±5.05	13	2	8.63	95.04±5.05	1	6	11.05	12	95.04±5.05	6.53

Table 4 compares the performance of CPOM-C4.5 with the performance of simple C4.5. The table indicates that, on average, CPOM-C4.5 is almost 2% (85.07%) more accurate than C4.5 (83.29%). It can be seen that in nine of the datasets the two methods have obtained the same accuracy results. In all of these datasets, CPOM has decided, in the light of the C4.5 inner-classifier's performance, not to split the instance-space. In three out of the remaining 11 datasets, CPOM-C4.5 was significantly more accurate than C4.5.

Table 4: Comparison of CPOM-C4.5 to C4.5

Dataset	CPOM-C4.5				C4.5	
	Accuracy	# leaves	# inner-classifiers	Execution time (in sec.)	Accuracy	Execution time (in sec.)
Audiology	74±5.54	10	226	1.11	76±7.87	0.06
Australian	85.36±2.26	4	582	0.67	86.81±2.37	0.02
Breast Cancer	94.42±2.62	3	482	0.5	93.41±2.86	0.02
Car	94.1±0.97	7	35	0.2	$^{+}$91.78±1.93	0.03
Hayes-Roth	78.79±9.21	3	34	0.14	$^{+}$68.18±7.41	0.01
Iris	95.33±5.97	1	12	0.11	95.33±5.97	0.02
Labor	73.68±12.3	1	5	0.09	73.68±12.3	0.02
Led-17	61.81±3.14	1	8	0.36	61.81±3.14	0.03
Letter	73.45±0.67	1	21	16.8	73.45±0.67	1.19
Monk-1	98.39±3.77	2	15	0.13	$^{+}$75±7.82	0.01
Monk-2	63.91±8.86	2	11	0.19	61.54±7.82	0.01
Monk-3	92.62±5.82	2	11	0.14	93.44±5.34	0.01
Mushroom	100±0	1	5	0.28	100±0	0.13
Nurse	97.43±0.31	1	4	0.52	97.43±0.31	0.41
Sonar	70.67±6.82	8	3967	3.33	71.15±8.74	0.11
Soybean	91.51±1.68	1	7	0.41	91.51±1.68	0.13
Tic-Tac-Toe	87.79±2.07	2	8	0.13	86.01±1.71	0.03
Vote	96.21±2.45	1	12	0.09	96.21±2.45	0.01
Wine	79.78±3.25	2	769	0.47	80.89±2.70	0.02
Zoo	92.08±6.57	1	4	0.08	92.08±6.57	0.05

5.7 MEASURING THE EFFECT OF THE GROUPING HEURISTIC

This section focuses on the contribution of the grouped gain-ratio splitting-rule and the grouping heuristic, in particular. For this purpose we compared the performance of CPOM-NB with that of a method which is similar to CPOM-NB, except for its splitting rule. This second method, henceforth termed CPOM-NB with no grouping, uses the simple gain-ratio splitting-rule (instead of grouped gain-ratio).

Table 5 compares the performance of the two methods. The table indicates that the grouping heuristic tends to improve the accuracy of the composite classifier

(86.74% versus 83.81%). As a matter of fact, in none of the datasets does the gain-ratio rule significantly outperform grouped gain-ratio. Thus, it is probable to assume that the grouping heuristic is a beneficial (i.e., subspaces that are described by similar classifiers should be grouped together).

The grouping heuristic has also a positive effect on the number of leaves and on the execution time. Grouping makes the hierarchical structure more compact. More specifically, the average number of leaves with the grouped gain-ratio rule was 7.1, where the average number of leaves with the simple gain-ratio rule was 9.85. This observation can be explained by the fact that without the grouping, one branch is built for each distinct value of the splitting attribute. On the other hand, when grouping is employed then one branch in the tree may represent several values.

Table 5: The Effect of grouping heuristic on the CPOM's performance

Dataset	CPOM-NB - no Grouping				CPOM-NB			
	Accuracy	# leaves	# inner-classifiers	Execution time (in sec.)	Accuracy	# leaves	# inner-classifiers	Execution time (in sec.)
Audiology	71.80±4.84	28	129	0.78	72±5.84	14	247	1.92
Australian	84.32±2.92	8	712	0.52	87.24±3.96	6	606	0.49
Breast Cancer	+93.84±1.86	10	2250	0.83	97.42±1.16	1	92	0.05
Car	+89.64±0.96	7	102	0.17	93.92±0.74	8	90	0.13
Hayes-Roth	+64.39±9.43	13	96	0.15	77.27±8.56	4	54	0.06
Iris	96±6	2	8	0.05	96±6	2	8	0.05
Labor	94.73±4.61	2	32	0.01	94.73±4.61	2	32	0.01
Led-17	61.81±7.22	8	32	0.25	60.90±5.12	8	28	0.22
Letter	77.70±0.64	25	892	32.34	77.70±0.64	25	892	32.34
Monk-1	+70.16±8.49	9	564	0.09	97.58±3.97	7	72	0.02
Monk-2	61.53±4.35	8	126	0.08	59.53±5.43	4	57	0.03
Monk-3	92.97±3.57	1	6	0.02	92.62±3.61	1	6	0.02
Mushroom	99.28±0.23	12	326	1.56	99.37±0.17	10	262	1.12
Nurse	+92.80±0.36	9	36	0.75	94.21±0.54	15	77	0.83
Sonar	+70.19±9.09	28	328	12.62	76.44±7.62	2	772	4.21
Soybean	93.99±1.03	7	167	0.31	93.41±1.49	18	122	0.24
Tic-Tac-Toe	78.70±1.43	7	45	0.25	76.51±1.87	7	18	0.08
Vote	+91.03±3.32	7	12	0.13	96.2 ±3.31	2	8	0.11
Wine	96.01±4.24	1	368	0.05	96.62±3.35	1	368	0.01
Zoo	95.04±5.05	5	36	0.03	95.04±5.05	5	36	0.03

5.8 MEASURING THE EFFECT OF THE NUMBER OF INTERVALS

Recall that the numeric attributes, in all previous experiments, were made discrete by dividing their original range into ten equal-sized intervals. The selection of the value ten was arbitrary. In this section we examine the effect of the number of equal-sized intervals on the performance of the CPOM-NB. Table 6 presents the performance obtained, when using two intervals and five intervals, in all datasets that include numeric attributes. The results indicate that the differences between 5 and 10 intervals are usually negligible. Still, in the Wine dataset, the 10 intervals alternative has led to a significantly better accuracy (based on a single-tailed paired t-test, with a confidence level of 95%). The difference in accuracy is more remarkable when the 10-intervals discretization is compared to binary discretization: in three datasets (Iris, Sonar and Wine) the binary discretization have led to loss of information. This indicates that converting numeric attributes to binary intervals might be too rough.

It should be noticed that while the 10-interval discretization increases the search space, there is minimal affect on the final classifier complexity (measured by the number of leaves). This supports the observation that CPOM with grouping usually creates compact trees.

Table 6: Comparison of the accuracy measures, when using 10-interval discretization, 5-interval discretization and binary discretization

Dataset	Two Intervals		Five Intervals		Ten Intervals	
	Accuracy	# leaves	Accuracy	# leaves	Accuracy	# leaves
Australian	88.68±3.51	3	86.80±3.12	2	87.24±3.96	6
Breast Cancer	96.56±1.72	6	97.27±1.91	13	97.42±1.16	1
Iris	$^{+}$76.53± 4.7	1	94.65± 5.72	2	96±6	2
Labor	94.65±3.95	2	95.12±5.03	3	94.73±4.61	2
Letter	78.75±0.82	1	76.74±0.66	10	77.70±0.64	25
Sonar	$^{+}$69.44±7.53	1	76.21±6.34	2	76.44±7.62	2
Wine	$^{+}$89.27±5.12	1	$^{+}$93.79±2.91	1	96.62±3.35	1
Zoo	95.04±5.05	5	95.04±5.05	5	95.04±5.05	5

5.9 THE PERFORMANCE ACROSS VARIOUS METHODS: RESULTS AND DISCUSSION

When comparing the number of leaves, the various CPOM implementations obtained, it can be seen that the simpler the embedded classification method, the greater the number of leaves. More specifically, with naive Bayes, the average number of leaves

is 7.1, while with backpropagation this average drops to 4.3. Moreover, the simpler the embedded method, the greater the potential contribution of CPOM to the overall accuracy (in naive Bayes, CPOM improved the accuracy by about 4%, but in backpropagation it improved the accuracy by only about 2%). In general, therefore, one should consider using CPOM, when the base classification method is relatively weak.

Moreover, we have examined the correlations between the accuracy-gain of using the various decision-tree ISD methods. The accuracy gain of a certain decision-tree ISD method, with respect to a certain dataset, is defined by dividing the accuracy estimation of the method over the dataset, by the accuracy estimation of the embedded method alone over the same dataset. Table 6 presents the correlation coefficients between the accuracy-gains. A high correlation (a value near 1.0) between two methods indicates that the two methods are consistent over the datasets. The table provides some interesting insights: the neural network-based methods are strongly correlated with one another and the naive Bayes methods are strongly correlated with one another. This may suggest that the effectiveness of decision-tree ISD methods depends more on the base inducer algorithm than on the dataset characteristics (although the correlation between CPOM-C4.5 and CPOM-NB cannot be neglected).

Table 6: Performance correlation coefficients across decision-tree ISD methods. The performance is measured in terms of the accuracy-gain

	NBTree	CPOM-NB	HDT	CPOM-NN	CPOM-C4.5
NBTree	1.00				
CPOM-NB	0.82	1.00			
HDT	-0.03	-0.23	1.00		
CPOM-NN	-0.05	-0.24	0.90	1.00	
CPOM-C4.5	0.34	0.66	-0.20	-0.27	1.00

5.10 THE CAPABILITY OF CPOM TO DIFFERENTIATE BETWEEN POPULATIONS

This section demonstrates an interesting ability of the CPOM algorithm: identifying different populations within the underlined datasets. For the purpose of this subsection, three synthetic datasets were created. In the first dataset we have merged the three Monk datasets into a single dataset, and added an additional attribute (denoted by a_7) which indicates on the original Monk dataset (i.e., an instance in

which $a_7=i$ belongs to the dataset monk-i, for i=1,2,3). The Monk datasets are suitable to this experiment because they all have the same input attributes, but the target attribute represents a completely different function. The selection of the new a_7 attribute in the root of the decision-tree is the indication for the correct instance-space decomposition. The second syntactic dataset (denoted as Art1) consists of 10 Boolean input attributes a_1-a_{10}, and a Boolean target attribute. The target attribute was set according to:

If a_1=0, $Y=(a_2 \wedge a_3 \wedge a_4) \vee (a_5 \wedge a_6 \wedge a_7) \vee a_{10}$

Else (If a_1=1), $Y=(\neg a_2 \wedge a_3 \wedge a_8) \vee (\neg a_5 \wedge a_6 \wedge a_9) \vee (\neg a_{10})$.

The attributes $a_2 - a_{10}$ were drawn from a uniform distribution. The dataset consists of 400 records, while in exactly 200 (randomly selected) instances, a_1 was set to 0, and in the remaining instances a_1=1. The third dataset (denoted as Art2) consists of 1000 records, and 10 input attributes. Nine of the input attributes were Boolean (a_2–a_{10}), and the remaining attribute (a_1) had 4 possible values. The target attribute was set according to:

If a_1=1, $Y=(a_2 \vee a_3) \wedge (a_4 \vee a_5) \wedge a_{10}$,

If a_1=2, $Y=(\neg a_2 \vee a_3) \wedge (a_6 \vee a_7) \wedge (\neg a_{10})$,

If a_1=3, $Y=(a_6 \vee \neg a_8) \wedge (\neg a_6 \vee a_7) \wedge a_4$.

If a_1=4, $Y=(\neg a_6 \vee a_8) \wedge (\neg a_9 \vee a_{10}) \wedge a_4$.

Table 7 presents the accuracy estimations of CPOM, with the three embedded classification methods, over the three synthetic datasets. Each accuracy measure is compared to the accuracy which was obtained by executing the embedded method alone (without wrapping it by CPOM.) The table presents some interesting insights. It can be seen that the CPOM has improved the accuracy, relative to the embedded method, in all the datasets. In the C4.5 and naive Bayes implementations, this improvement is statistically significant in all the datasets. Moreover, when analyzing the tree structures that the CPOM implementations obtained, it was seen that CPOM tended to succeed in selecting the differentiating attribute at the root node. For example, in the case of the "Monk-All" dataset, the attribute a_7 has been selected at the root node in 26 of the 30 executions (10 folds times 3 embedded methods). All the four executions, in which the attribute a_7 has not been selected at the root node, occurred when the embedded method was backpropagation. It may be interesting to note that in the single decision-tree that was obtained by the simple C4.5 algorithm,

the attribute a_7 was selected at the root node in only three of the ten executions. This observation probably explains why C4.5 has not succeeded in obtaining accuracy measures comparable to those of CPOM-C4.5.

Table 7: The CPOM performance on the datasets with the contrast populations

Classification Method		Datasets		
		Monk-All	**Art1**	**Art2**
Naive Bayes	Simple NB	$^{+}$67.08 ±3.59	$^{+}$58.81±0.02	$^{+}$70.67±0.03
	CPOM-NB	77.58±4.23	88.72±0.01	79.53±0.04
Neural Network	Simple Backprop.	96.93 ±1.91	$^{+}$92.38±0.08	94.11±0.05
	CPOM-NN	97.10±1.92	97.35±0.02	94.23±0.04
Decision-tree	Simple C4.5	$^{+}$80.47 ±2.56	$^{+}$81.51±0.06	$^{+}$91.24±0.05
	CPOM-C4.5	85.35±2.89	96.92±0.01	94.26±0.04

6 CONCLUSION

This paper introduced a decision-tree framework for instance-space decomposition (DFID) – an automatic, general, decision-tree based framework for instance-space decomposition and contrasted populations miner (CPOM) – an implementation of the DFID framework that uses a new splitting rule, termed grouped gain-ratio. DFID recursively partitions the underlined instance-space according to the values of the explained attributes until some pre-determined stopping rules are met. Subsequently, for each subspace that was formed by the partition, a unique classifier is attached using an embedded classification method. The CPOM algorithm implements the DFID framework by incorporating a new splitting rule, termed grouped gain-ratio. In the grouped gain-ratio, an attribute is first selected according to the gain-ratio criterion. Thereafter, a greedy grouping heuristic groups-together similar subspaces that correspond to different values of the selected attribute.

With datasets that were manually selected from the well-known UCI Machine Learning repository, CPOM improved the obtained accuracy compared to the examined embedded methods (naive Bayes, backpropagation and C4.5). CPOM has been found to be more accurate than other decision-tree ISD methods. Moreover, the grouping heuristic was shown to significantly improve the accuracy results, compared to a variation of CPOM which does not group. Finally, using three synthetic datasets, CPOM was able to distinguish between different populations in an underlined dataset.

As to future research, the CPOM algorithm can be extended in various ways. An essential part of the algorithm lies in grouping together similar instance subspaces. The grouping heuristic in this paper was based on the cross-inspection procedure (see Figure 2). We suggest examining different heuristics, for determining what similarity is. In addition, it is well-known that the accuracy of decision-trees can benefit from a pruning capability. The fact that the proposed algorithm has no pruning capabilities is considered to be a limitation. Thus the algorithm should be extended to include such a capability. Moreover, due to the explosive increase of data volumes, incremental (online) learning has become a very important capability in machine learning methods, which are designed for solving real-world problems. Developing an incremental version of CPOM is not necessarily simple, because it requires incremental adaptation of the hierarchical structure as well as incremental adaptation of the inner-classifiers. Additional issues to be further studied include examining how the proposed algorithm can be implemented using other classification methods, such as support vectors machines or Bayesian networks. Along with improving the practical framework, a further theoretical investigation is required in order to better understand under what circumstances the proposed approach is advantageous.

REFERENCES

[1] E. Bauer & R. Kohavi, An empirical comparison of voting classification algorithms: Bagging, boosting, and variants, Machine Learning 36 (1999) 105-139.

[2] L. Breiman, Bagging predictors, Machine Learning 24 (1996) 123-140.

[3] L. Breiman, J.H. Olshen & C.J. Stone, Classification and Regression Trees, Chapman Hall, New York, 1984.

[4] C.E. Brodley, Recursive automatic bias selection for classifier construction, Machine Learning 20 (1995) 63-94.

[5] D.R. Carvalho & A.A. Freitas, A hybrid decision-tree/genetic algorithm method for data mining, Information Science 163 (2004) 13-35.

[6] T.G. Dietterich, Approximate statistical tests for comparing supervised classification learning algorithms, Neural Computation 10 (1998) 1895-1923.

[7] T.G. Dietterich, An experimental comparison of three methods for constructing ensembles of decision trees: bagging, boosting and randomization, Machine Learning 40 (2000), 139-157.

[8] S. Esmeir & S. Markovitch, Lookahead-based algorithms for anytime induction of decision trees. In Proceedings of the Twenty-First International Conference on Machine Learning, pp. 257-264, Morgan Kaufmann, 2004.

[9] Y. Freund & R. Schapire, A short introduction to boosting, Journal of Japanese Society for Artificial Intelligence 14 (1999), 771-780.

[10] J. Fürnkranz, Round robin rule learning, in: Proc. Of the 18th International Conference on Machine Learning, Williamstown, MA, 2001, pp. 146-153.

[11] J. Fürnkranz, Round robin ensembles, Intelligent Data Analysis 7 (2003) 385-403.

[12] J.B. Hampshire & A. Waibel, The meta-pi network – building distributed knowledge representation for robust multisourse pattern recognition, IEEE Trans. on Pattern Analysis and Machine Intelligence 14 (1992) 751-769.

[13] J.V. Hansen, Combining predictors: comparison of five meta machine learning methods, Information Science 119 (1999) 91-105.

[14] L. Hansen & P. Salamon, Neural network ensembles, IEEE Trans. on Pattern Analysis and Machine Intelligence 12 (1990) 993-1001.

[15] M.B. Harries & K. Horn, Learning stable concepts in domains with hidden changes in context, in M. Kubat and G. Widmer (Eds.), Learning in context-sensitive domains (Workshop Notes), 13th International Conference on Machine Learning, Bari, Itali, 1996 pp. 106-122.

[16] T. Horton & B. Lausen, Bundling classifiers by bagging trees, Computational Statistics and Data Analysis 49 (2005) 1068-1078.

[17] M.I. Jordan & R.A. Jacobs, Hierarchical mixture of experts and the EM algorithm, Neural Computation 6 (1994) 181-214.

[18] R. Kohavi, Scaling up the accuracy of naive-Bayes classifiers: a decision-tree hybrid, in: Proc. Of the 2nd International Conference on Knowledge Discovery and Data Mining, Portland, OR, 1996, pp. 202-207.

[19] A. Kusiak, Decomposition in data mining: an industrial case study, IEEE Trans. on Electronics Packaging Manufacturing 23 (2000), 345-353.

[20] O. Lezoray & H. Cardot, Combining multiple pairwise neural networks classifiers: a comparative study, in: International Workshop on Artificial Neural Networks and Intelligent Information Processing, Barcelona, Spain 2005, pp. 52-61.

[21] O. Maimon & L. Rokach, Decomposition Methodology for Knowledge Discovery and Data Mining: Theory and Applications, World Scientific, 2005.

[22] C.J. Mertz, Dynamical selection of learning algorithms, in: C. Fisher & H. Lenz (Eds.), Learning from Data: Artificial Intelligence and Statistics, Springer-Verlag, 1996, pp. 281-290.

[23] C.J. Mertz, Using correspondence analysis to combine classifiers, Machine Learning 36 (1999) 33-58.

[24] C.J. Mertz & P.M. Murphy, UCI repository of machine learning databases, Available at: http://www.ics.uci.edu/~mlearn/MLRepository.html.

[25] S. Murthy & S. Salzberg, Lookahead and pathology in decision tree induction, in Proceedings of the 14th International Joint Conference on Artificial Intelligence, pp. 1025-1031, Morgan Kaufmann, 1995.

[26] S.J. Nowlan & G.E. Hinton, Evaluation of adaptive mixtures of competing experts, in: Advances of Neural Information Processing Systems 3, Denver CO, 1990, pp. 774-780.

[27] L. Onho-Machado & M.A. Musen, neural networks for medical prognosis: Quantifying the benefits of combining neural networks for survival prediction, Connection Science 9 (1997) 71-86.

[28] F. Peng, R.A. Jacobs & M.A. Tanner, Bayesian inference in mixture-of-experts and hierarchical mixture-of-experts models with an application to speech recognition, Journal of the American Statistical Association 91 (1996) 953-960.

[29] J.R. Quinlan, C4.5: programs for machine learning, Morgan Kaufmann, San Francisco, CA, 1993.

[30] A.F.R. Rahman & M.C. Fairhurst, A new hybrid approach in combining multiple experts to recognize handwritten numerals, Pattern Recognition Letters 18 (1997) 781-790.

[31] A. Sakar & R.J. Mammone, Growing and pruning neural tree networks, IEEE Trans. on Computers 42 (1993) 291-299.

[32] S.L. Salzberg, On comparing classifiers: pitfalls to avoid and a recommended approach. Data Mining and Knowledge Discovery, 1, 317-328, 1997.

[33] P. Savicky & J. Fürnkranz, Combining pairwise classifiers with stacking, in Advances on Intelligent Data Analysis V, Berlin, Germany, 2003, pp. 219-229.

[34] A.J.C. Sharkey, Multi-net systems, in: A.J.C. Sharkey (Ed.), Combining Artificial Neural Nets: Ensemble and Modular Multi-Net Systems, Springer-Verlag, 1999, pp. 1-30.

[35] S. Warshall, A theorem on Boolean matrices, Journal of the ACM 9 (1962) 11–12.

[36] A.S. Weigend, M. Mangeas & A.N. Srivastava, Nonlinear gated experts for time-series - discovering regimes and avoiding overfitting, International Journal of Neural Systems 6 (1995) 373-399.

[37] I.H. Witten & E. Frank, Data Mining: Practical machine learning tools and techniques, 2nd Edition, Morgan Kaufmann, San Francisco, 2005.

[38] D.H. Wolpert, Stacked generalization, Neural Networks 5 (1992) 241-259.

[39] Z. Zhou, S. Chen & Z. Chen, FANNC: A fast adaptive neural network classifier, Knowledge and Information Systems 2 (2000) 115-129

[40] Z. Zhou & C. Chen, Hybrid decision tree, Knowledge-Based Systems 15 (2002) 515-528.

[41] B. Zupan, M. Bohanec, J. Demsar & I. Bratko, Feature transformation by function decomposition, IEEE Intelligent Systems 13 (1998) 38-43.