ISSN 1392 – 124X INFORMATION TECHNOLOGY AND CONTROL, 2008, Vol. 37, No. 1

HIERARCHICAL CLASSIFICATOR: A COGNITIVE APPROACH TO DECISION TREE BUILDING

Jurgita Kapočiūtė-Dzikienė, Arimantas Raškinis

Vytautas Magnus University
Vileikos 8, LT-3035 Kaunas, Lithuania

Abstract. We present a new algorithm that follows "divide and conquer" machine learning approach and exhibits a few interesting cognitive properties. The algorithm aims at building the decision tree with only one terminal node per class. Splits of tree nodes are constrained to functions that take identical values (true or false) for every instance within the same class. Appropriate splits are found through an exhaustive search in the attribute-value-based function space. Simple single-attribute functions are considered before complex multi-attribute k-DNF type ones. Redundant functions are also being incorporated into the decision tree. The unique structure of the decision tree results in that semantic interpretation can be attached to both terminal and non-terminal nodes, the task-specific set of classes is structured within a hierarchy of similarity relationships, and sources of recognition errors can be traced back (localized) to some particular function/split in the decision tree. The new algorithm was implemented, experimentally evaluated and compared with the well-known machine learning techniques Ripper and C4.5. Though limited in scope the experiments showed that the new algorithm can perform at least as well as Ripper and C4.5. Redundant knowledge incorporated into the decision tree helped to improve the recognition accuracy.

1. Introduction

This paper describes the initial phase of our research, which has the ultimate objective of constructing adaptive computer-based system that behaves rationally in an artificial grid-world environment. The ability of learning, generalizing, recognizing repeated patterns in this environment are of crucial importance to any adaptive system. Under the architectural assumptions of the "classical AI", learning abilities may be thought of as constituting independent learning subsystem that interacts with the subsystems of planning, execution, and monitoring. Though learning subsystem must be of incremental nature, we are addressing the simplified non-incremental version of it in this paper. Though the algorithm presented in this paper is designed to solve the standard supervised learning/classification problem, it is different from analogous approaches in two respects. First, it is biased for detecting reliable regularities in the data. Second, it learns knowledge structures that are easily amenable to error localization, which means that the incremental version of the proposed algorithm would have the ability of finding and modifying inconsistent parts of its knowledge without need to invalidate the entirety of knowledge it has.

2. Related work

The classification problem[1] is the problem of assigning one of the pre-defined class labels to some instance, the decision being based on the training set, i.e. on the set of other related instances with pre-assigned class labels. There are a lot of machine learning (ML) techniques that address the classification problem. The majority of ML techniques are operating in two phases: learning phase and recognition phase. During the learning phase, computational efforts are spent on inductive reasoning, i.e. on generalizing and discovering regularities within the training set. During the recognition phase, regularities are exploited for making decisions about unknown instances. Machine learning techniques are based on statistical, neural/ distributed or symbolic approaches [8][6]. Symbolic machine learning techniques have the advantage of inducing knowledge that can be understood and interpreted by humans. We believe that symbolic knowledge might be more appropriate for computer-based manipulations as well. Thus we concentrate on symbolic machine learning methods in this paper.

Symbolic machine learning methods can be divided in two broad categories: those resulting in a rule set and those resulting in a decision tree/list (top-down induction methods). Rule induction algorithms

[1] This problem is also called "instance based learning" or "supervised learning".

(Charade proposed by Ganascia [4]) do not eliminate covered instances in the training process. Therefore created rules can cover the same instances repeatedly. Charade results in a set of conjunctive if-then rules. Rules are discovered through an exhaustive search by trial-and-error starting from the simplest rules and gradually going to more complex. The method is based on the formalism of two distributive lattices: one for the set of training instances, and another for the description space. Properties of these lattices are used for limiting the search scope.

Top-down induction methods recursively split training set into disjoint subsets until some stopping condition is achieved. Usually stopping condition means that every subset contains instances belonging to the same class. Every subset can be associated to a node, thus the whole recursive process can be associated to the hierarchical structure called decision tree/list. Tree nodes associated to a single class and having no descendants are usually called terminal nodes. Decision tree/list can be also rewritten as a set of rules such that every instance is covered by a single rule. Top-down induction techniques differ on the criterion for the "optimum" split and on the search strategies how to find it in the description space.

Some methods allow splits into more than two constituent subsets and consider optimality to be related to the decrease in average class "impurity" of all the constituent subsets with respect to the original set. ID3 (the precursor to C4.5) proposed by Quinlan[12] is one of such algorithms. The split is found through an exhaustive search in the space of possible tests of a single attribute value using class entropy as an optimality criterion. GID3 proposed by Cheng et al. [1] can build more general trees than ID3. Instead of creating one branch per each possible value of a nominal attribute, GID3 generates branches only for those outcomes that are relevant to the classification depending on the information entropy and a tolerance level, determined by the user. LFC algorithm proposed by Ragavan and Rendell [10] replaces exhaustive search for a single attribute with the beam search for multi-attribute functions. Therefore it is better adjusted to overcome the problems associated with greediness of decision tree building approach. Methods of this group result in a knowledge structure known as a decision tree.

Some other methods allow just binary splits that separate a small subset of instances belonging to the same class from the rest of the training set. Thus optimality is measured on a single subset instead of all subsets. Test for such a split usually takes the form of a rule having multi-attribute conjunctions. Such rules can be sought using two alternative techniques: “specific-to-general” and “general-to-specific”. “Specific-to-general” training process starts from the most specific rule having many conjuncts and covering only one positive instance. It ends with the most general rule that covers no negative instances (AQ proposed by Michalski et al. [7] [5]). “General-to-specific” technique (CN2 proposed by Clark and Niblett [2] and Ripper proposed by Cohen [3]) is opposite to “specific-to-general”. CN2 algorithm uses depth-first search together performing beam search of k best candidates while working with the full set of instances. Whereas Ripper algorithm randomly divides the training data into rule growing and pruning sets. Using these two sets, Ripper builds up a rule set by repeatedly adding rules to an empty rule set. The single rule-growing algorithm is based on a hill-climbing search. It begins with an empty "rule", and greedily adds conjuncts until the rule no longer makes incorrect prediction on the growing set. Next, the learned rule is simplified by deleting conjuncts so as to improve performance of the rule on the pruning set. Methods of this group result in a knowledge structure known as a decision list.

3. Motivation for a new approach

The k-DNF learning problem may be considered out of the focus of the mainstream ML research. Indeed, many solutions have been proposed and developed during the last 20 years in this domain. Nevertheless, we consider our proposal important, as it addresses the question of what the generalization (induction) mechanism could be if we have perfect attributes. This question has not received much attention as it is usually implicitly assumed that the feature space is hard-fixed to the task and cannot be changed. Thus, any good induction algorithm must cope with noisy parameter spaces.

In fact, majority of the most popular symbolic machine learning techniques were designed with the real-world tasks in mind. Real-world tasks are hard in the sense that attributes are not perfect and instances of different classes occupy overlapping areas in the attribute space. Imperfect attributes and the necessity of returning some classification decision (albeit imperfect but better than a random guess), has biased symbolic learning techniques towards weak constraints imposed on the top-down splitting process. Top-down splitting based on continuous "impurity" measures does not discard any single split but assigns every possible split a rating. This way, existing symbolic ML approaches can adapt to the wide continuum of tasks ranging from hard to easy.

As a consequence of weakly constrained top-down splitting process symbolic ML techniques have the potential of discovering "regularities" in randomly generated training sets[2] whereas humans are expected to find no regularities in similar cases. Error localization, i.e. the ability of a system to find and modify inconsistent parts of its knowledge without a need to

[2] This is true given that contradictory labeling, i.e. instances having the same description but assigned to different classes, is avoided. Pruning of the decision tree on the held out instances may limit the discovery of "regularities" in random training sets.

invalidate the entirety of knowledge it has, is another cognitive property missed by the decision trees/lists constructed by the widespread ML techniques[3]. Error localization is very important to incremental ML techniques if they are considered for usage within adaptive systems.

The idea underlying our approach is that ML techniques could benefit from separately addressing two different problems they currently address *a la fois*: the problem of finding good attribute space in which any particular ML task becomes "easy", and the problem of finding good generalizations/regularities in that space. This paper is addressing just the second problem. It focuses on the questions of how generalized class descriptions have to be induced in order to avoid discovering random "regularities", and on how multiple-class descriptions have to be organized and structured in order to be error-localization-friendly. The symbolic machine learning technique named Hierarchical Classifier (HL) is presented in this paper. It represents the implementation of the ideas originally developed by Raškinis [9][11].

4. Method

Any supervised symbolic machine learning algorithm takes the set of training instances as an input and outputs class descriptions consistent with the training instances and stated in the form of logical functions.

Let $\{S_1, S_2, ..., S_N\}$ be the set of training instances described by the same set of attributes $A = \{A_1, A_2, \ldots A_M\}$ and the class attribute C. Let $D(A_j)$ be the finite domain of the attribute A_j and $D(C)$ be the finite domain of the class attribute C.

Let the term function refer to any statement of prepositional logic based on one or more attribute value tests over A. Single-variable function would have the form $A_j = \zeta$, where $A_j \in A$ and $\zeta \in D(A_j)$. Multi-variable functions could be constructed by combining single-variable functions using logical connectives (negation, conjunction and disjunction).

Let $f(S) = true$ denote the statement that the function f covers the instance S.

Let $f(c) = true$ denote the statement that the function f covers the class c, i.e. it covers at least one instance belonging to the class c:

$$f(c) = true \Leftrightarrow \exists\ S \text{ such that } S \in c \text{ and } f(S) = true \quad (1)$$

[3] Suppose the system is facing the fact that its decision tree/list has misclassified some instance. In this case, any section (attribute value test) of the entire classification path from the root node to the wrong terminal node may be held responsible for the error. In addition, all paths leading to the terminal nodes labeled with the correct class may be held responsible for not covering the instance in question, i.e. for being collectively too specific. In practice, this means that the whole decision tree needs to be erased and entirely reconstructed.

Let cl be the mapping $\mathtt{F}\rightarrow\mathtt{C}$, where $\mathtt{F}$ is the space of all possible functions and $\mathtt{C}$ is the space of all subsets of class attribute values from $D(C)$, such that for any function $f \in \mathtt{F}$ the mapping cl returns the subset of classes covered by f:

$$cl(f) \equiv \{c_i \text{ such that } c_i \in D(C) \text{ and } f(c_i) = \text{true}\}. \quad (2)$$

The function f will be called *stable* with respect to the training set if

$$cl(f) \cap cl(\neg f) = \varnothing. \quad (3)$$

Otherwise f will be called *unstable* (Figure 1).

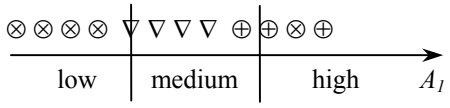


Figure 1. Stable and unstable functions. The function $f_1 \equiv (A_1=\text{low})$ is unstable because of $cl(f_1) \cap cl(\neg f_1) = cl(A_1=low) \cap cl(A_1 \neq low) = \{\otimes\} \cap \{\otimes,\oplus,\nabla\} = \{\otimes\} \neq \varnothing$. The function $f_2 \equiv (A_1=\text{medium})$ is stable because of $cl(f_2) \cap cl(\neg f_2) = cl(A_1=medium) \cap cl(A_1 \neq medium) = \{\nabla\} \cap \{\otimes,\oplus\} = \varnothing$

Let $F = asp(A')$ refer to the Attribute Subspace Partition (ASP) over A', $A' \subseteq A$, i.e. to the largest possible set of stable, class discriminating, and pairwise exclusive functions that could be constructed over A':

$$|cl(f_i)| > 0, \text{ for all } f_i \in F \quad (4)$$

$$cl(f_i) \cap cl(\neg f_i) = \varnothing, \text{ for all } f_i \in F \quad (5)$$

$$f_i \wedge f_j = \text{false, for all } f_i, f_j \in F, \text{ given } i \neq j \quad (6)$$

The definition above implies that for an arbitrary subset of attributes $A' \subseteq A$, if there exists an ASP it must be unique (Figure 2).

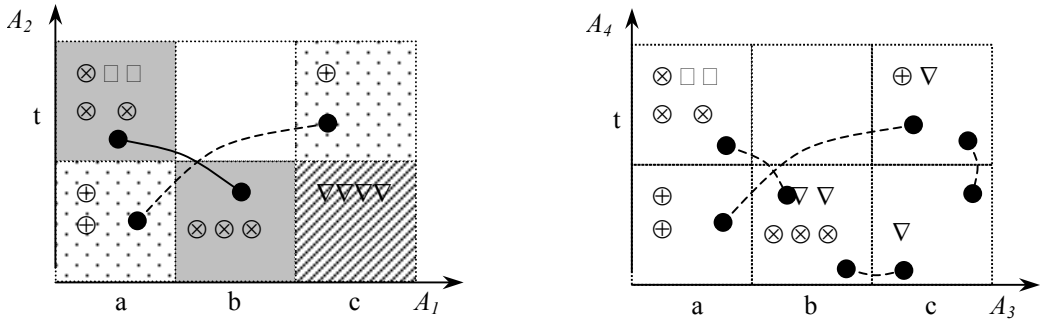
Let two attribute subspace partitions F_a and F_b be constructed over different subsets of A. F_a and F_b will be called alternative ASPs if for every $f_i \in F_a$ there exists $f_j \in F_b$ such that $cl(f_i) = cl(f_j)$ (fig. 3). Alternative ASPs may be imagined as forming clusters. Let $altset(F)$ denote the cluster of alternative ASPs comprising the ASP F.

Composite partitions can be formed by combining two or more attribute subspace partitions. Let F_a and F_b be two ASPs (or clusters of alternative ASPs). Then the composite ASP $F_a \otimes F_b$ is defined as:

$$F_a \otimes F_b = \{f_i \wedge f_j, \text{ for every } f_i \in F_a \text{ and } f_j \in F_b \text{ such that } cl(f_i \wedge f_j) \neq \varnothing\}. \quad (9)$$

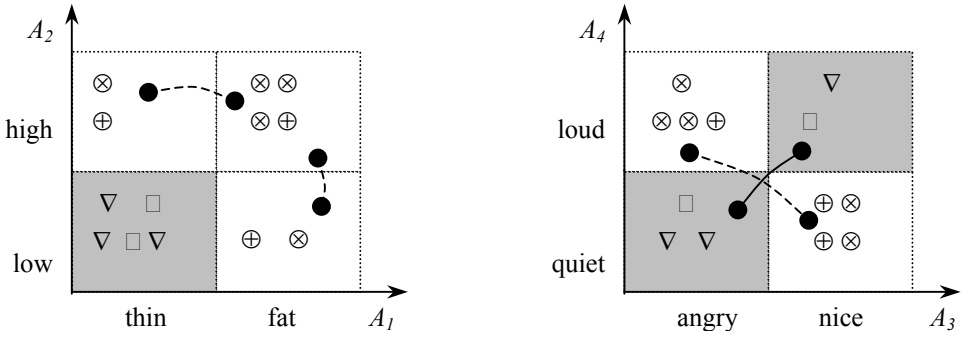
For instance, $asp(\{A_1\}) \otimes asp(\{A_2\}) = \{f_1 \wedge f_2, f_1 \wedge \neg f_2, \neg f_1 \wedge \neg f_2\}$ in the case shown in fig. 4. The term $\neg f_1 \wedge f_2$ is not included to this composite partition as $cl(\neg f_1 \wedge f_2) = \varnothing$.

Composite partitions of more than two ASPs can be constructed incrementally in the same way defined by (9).



$asp(\{A_1, A_2\}) \equiv \{f_1, f_2, f_3\}$, where:
$f_1 \equiv (A_1=a \wedge A_2=t) \vee (A_1=b \wedge A_2=f)$
$f_2 \equiv (A_1=a \wedge A_2=f) \vee (A_1=c \wedge A_2=t)$
$f_3 \equiv (A_1=c \wedge A_2=f)$

Figure 2. The attribute subspace partition. There exists an ASP $\{f_1, f_2, f_3\}$ for the subspace $\{A_1, A_2\} \subseteq A$ (picture to the left). The functions in $\{f_1, f_2, f_3\}$ are stable and pairwise exclusive. The members of the set $\{f_1, f_2 \vee f_3\}$ are stable and pairwise exclusive as well. However $\{f_1, f_2 \vee f_3\}$ is not an ASP because it is not as large as the set $\{f_1, f_2, f_3\}$. It could be noticed that $f_1 \vee f_2 \vee f_3 \neq true$. There exists no ASP for the subspace $\{A_3, A_4\}$ (picture to the right)



$asp(\{A_1, A_2\}) \equiv \{f_1, \neg f_1\}$, where
$f_1 \equiv (A_1=thin \wedge A_2=low)$

$asp(\{A_3, A_4\}) \equiv \{f_2, \neg f_2\}$, where
$f_2 \equiv (A_3=angry \wedge A_4=quiet) \vee (A_3=nice \wedge A_4=loud)$

Figure 3. The alternative ASPs. Attribute subspace partitions $\{f_1, \neg f_1\}$ and $\{f_2, \neg f_2\}$ are alternative ASPs as $cl(f_1)=cl(\neg f_2)=\{\square,\nabla\}$ and $cl(\neg f_1)=cl(f_2)=\{\otimes,\oplus\}$

$asp(\{A_1\}) \equiv \{f_1, \neg f_1\}$, where
$f_1 \equiv (A_1=empty)$
$inf(\{f_1, \neg f_1\}) = 2$

$asp(\{A_2\}) \equiv \{f_2, \neg f_2\}$, where
$f_2 \equiv (A_1=educated)$
$inf(\{f_2, \neg f_2\}) = 1$

Figure 4. The informativity of an attribute subspace partition
$inf(\{f_1, \neg f_1\}) = |D(C)|\text{-}\max(|cl(f_1)|, |cl(\neg f_1)|) = 4 - \max(|\{\nabla,\square\}|, |\{\oplus,\otimes\}|) = 4\text{-}\max(2,2) = 2$ (left).
$inf(\{f_2, \neg f_2\}) = |D(C)|\text{-}\max(|cl(f_2)|, |cl(\neg f_2)|) = 4 - \max(|\{\nabla\}|, |\{\oplus,\otimes,\square\}|) = 4\text{-}\max(1,3) = 1$ (right)

5.Algorithm description

The symbolic machine learning technique addressing cognitive challenges described in section 4 was implemented and entitled Hierarchical Classifier (HC).

During the phase of training, HC builds a decision tree according to the "divide and conquer" strategy. It recursively splits the set of training instances into two or more subsets and stops when all instances in all subsets belong to the single class. HC algorithm uses no pruning. Therefore, the resulting decision tree is consistent with all training instances. Though, the standard top-down induction scheme is used for discovering class descriptions, strongly constraining top-down splitting criterion (stable function) is used as a

replacement to continuous "impurity" measure proper to the family of decision-tree inducing algorithms. The stability criterion discards splits that result in instances of the same class being divided among different subsets of the split. If this criterion is consistently applied, the decision tree with one terminal node per class is obtained.

The scheme of decision tree building by HC algorithm is presented below.

```
PROCEDURE BuildSubTree(Node)   // Node is the current node
IF |Classes(Node)| = 1         // Instances(Node) belong to the single class
THEN RETURN(success)
ELSE
   F←SearchForASP(Classes(Node))
   IF F=∅                      // There's no Attribute Subspace Partition
   THEN RETURN(failure)
   ELSE
      FOR every f_k∈F DO
         Node_k←CreateNewNode(Node)
         Classes(Node_k) ← Classes(Node) ∩ cl(f_k)
         BuildSubTree(Node_k)
END PROCEDURE
```

This recursive procedure is invoked by the call to `BuildSubTree(RootNode)`, where `RootNode` is the initial node associated with the entire training set.

The requirement of stability of a function is very constraining. Thus, simple ASPs corresponding to single attribute value tests, which are so common to other decision tree builders, would rarely pass this requirement. The number of ASPs corresponding to complex multi-variable splits grows exponentially with the growing dimensionality of the subspace. This increases chances of discovering stable partitions but suffers from combinatorial explosion. `SearchForASP` is the core procedure of HC that performs an exhaustive search in the space of subsets of A checking each subset for the presence of an attribute subspace partition. Simpler ASPs (functions having fewer variables) are preferred over complex ones (Occam's razor). This procedure clusters alternative ASPs, performs ASP ranking with respect to their informativity and redundancy, and selects top ranked ASPs to combine them into a composite ASP. The scheme of the `SearchForASP` procedure is given below

```
PROCEDURE SearchForASP(C)  // C is the set of classes to be discriminated
 ASPs=∅                    // Let ASPs be the initially empty set of ASPs
 FOR λ*=1..n** DO
    FOR all subsets A′⊆A such that |A′|=λ
       Let F={f_i} be the set of all possible conjunctions over A′ having
          exactly λ conjuncts
       FOR all pairs f_i,f_j∈F DO
          IF cl(f_i)∩cl(f_j)∩C ≠ ∅      // Does f_i and f_j cover the same class?
          THEN F ← F / f_i / f_j        // Join f_i and f_j together
               F ← F ∪ (f_i ∨ f_j)
       UNTIL F remained unchanged
       IF |F|>1                         // Is F an ASP?
       THEN ASPs ← ASPs ∪ F             // Remember it
    IF ASPs≠∅
    THEN BREAK                          // Don't look for more complex ASPs
 Rank elements of ASPs according to their informativity and redundancy
 RETURN(F_1⊗...⊗F_z) where F_i are top ranked members of ASPs***
 END PROCEDURE
```

During the phase of recognition, HC exploits the decision tree built in the training phase. Any instance S that needs to be classified is examined at the root node. It is passed to the descendant of the root code Node_k for which $\text{Function}(\text{Node}_k)$ covers S. Such a recursive process continues until S reaches some terminal node. Then, S is assigned the same class as the terminal node.

---

* λ is a subspace dimensionality.

** n=3 is used in all our experiments.

*** Maximum number of members is 3 used in all our experiments.

In case where current node is associated with a non-composite set of alternative ASPs, the voting approach is taken. Every function in the set of alternative ASPs vote for some descendant of the node in question. The instance *S* is passed to the descendant node receiving maximum number of votes. If several descendant nodes receive equal number of votes, *S* is transferred to the descendant node covering the largest number of training instances.

In case where current node is associated with composite ASP, the voting approach is taken too. Every component of composite ASP votes for some descendant node in question. The decision of each component is taken in the way described above. The instance *S* is passed to the descendant node receiving maximum number of votes. If several descendant nodes receive equal number of votes, *S* is transferred to the descendant node covering the largest number of training instances.

If instance *S* have unseen attributes or their combinations, the algorithm eliminates ASPs that do not cover those attributes, further manipulating with remaining in the way described above.

5.1. Complexity

Let: *N* be the number of instances in the training set,

M be the number of attributes,

V be the average number of different discrete values per attribute,

n be the maximum dimensionality (number of attributes) of the ASP (n=3 in all our experiments).

Then the upper bound on the number of ASPs that are tested for a node split is $\frac{M!}{n!\cdot(M-n)!}\cdot V^{(2^n)} = C_M^n \cdot V^{(2^n)}$ and the time complexity of HC is $O(M^n \cdot V^{2\cdot n} \cdot N)$. Thus, HC is more costly in computation time than Charade or C4.5 which have node splitting time complexity $O(M^n \cdot V^n \cdot N)$ and $O(M \cdot N)$ respectively.

6. An experimental evaluation

The HC algorithm was compared with two other algorithms belonging to the same family of symbolic machine learning techniques: Ripper and C4.5. Two aspects were considered during this comparison: recognition error rate and inductive bias. Comparisons were based on the ZOO dataset from the UCI[4] machine-learning repository [13]. This dataset consists of 101 instances distributed among 7 classes and described by 17 attributes. It should be noted that our choice of a dataset was very limited even though UCI machine learning repository contained many datasets. ZOO dataset was probably the only one that had instances distributed among more than two classes, described by nominal attributes without missing attribute values and had instances belonging to the same class compactly distributed in the parameter space.

6.1. Recognition error rate

The recognition error rate was measured by the leave-one-out cross validation technique (100 instances were used for training, the remaining 1 instance was used for recognition; error rate estimate represented an average over 101 runs, each run with the different instance to recognize). Average recognition error rate *ER* is defined by:

$$ER = \frac{1}{Runs \cdot All} \sum_{i=1}^{Runs} Err_i \cdot 100\%,$$

where *Runs* denotes the total number of runs, *All* denotes the number of instances recognized in one run, and Err_i denotes erroneously classified instances in the i^{th} run[5]. Average recognition error rates thus obtained are presented in Table 1.

Table 1. Recognition error rates of HC, Ripper and C4.5 estimated by means of the leave-one-out cross validation technique and shown with the 95% confidence intervals.

Average recognition error rate, %			
HC		Ripper	C4.5
Without redundancy	With redundancy		
3.96±0	1.98±0	2.97±0	1.98±0

All three algorithms had their settings adjusted in order to yield the lowest recognition error rates. Both Ripper and C4.5 were informed they are working with the noise-free training dataset. Both algorithms were allowed to cover any single training instance; their decision trees were not pruned. In addition, Ripper used MDL class ordering and had coding cost of the theory set to zero. HC was tested in two operational modes: standard mode exploiting redundancy of the cluster of alternative ASPs as explained in the previous section and simplified mode, in which the redundancy was ignored by arbitrary choosing one ASP from the cluster of alternative ASPs.

Inductive bias represents the "preferences" of a given machine learning algorithm to induce certain knowledge structures over others when multiple generalizations are possible. As quantitative comparisons of inductive bias are not feasible we limited ourselves to showing and comparing different decision trees induced by HC (Figure 5), Ripper (Figure 6) and C4.5 (Figure 7) for the entire Zoo dataset. Decision trees correspond to the training of the abovementioned algorithms with their best settings in the sense of the lowest recognition error rate.

[4] University of California Irvine.

[5] Leave-one-out cross validation technique meant *Runs* = 101, *All* = 1 and $Err_i \in \{0, 1\}$ in case of ZOO dataset.

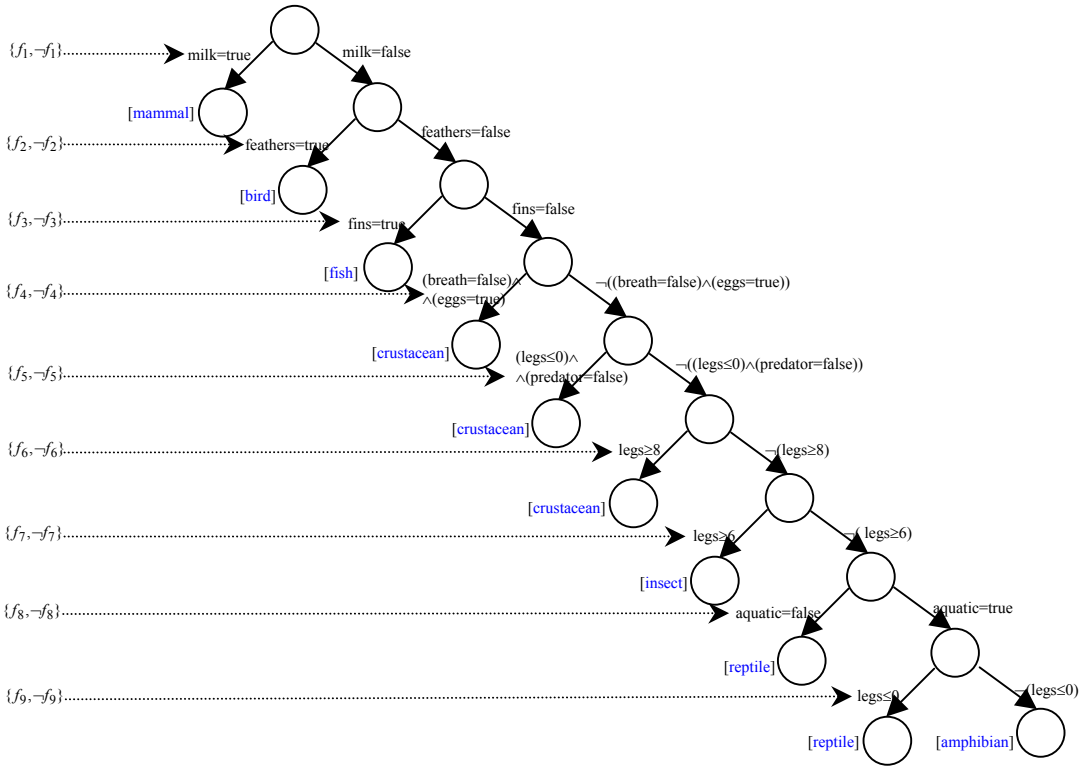


Figure 5. Decision list built by Ripper algorithm. Nodes are represented by circles. Terminal nodes are labeled with class names (in brackets). ASPs are shown as function sets resulting in branching of non-terminal nodes

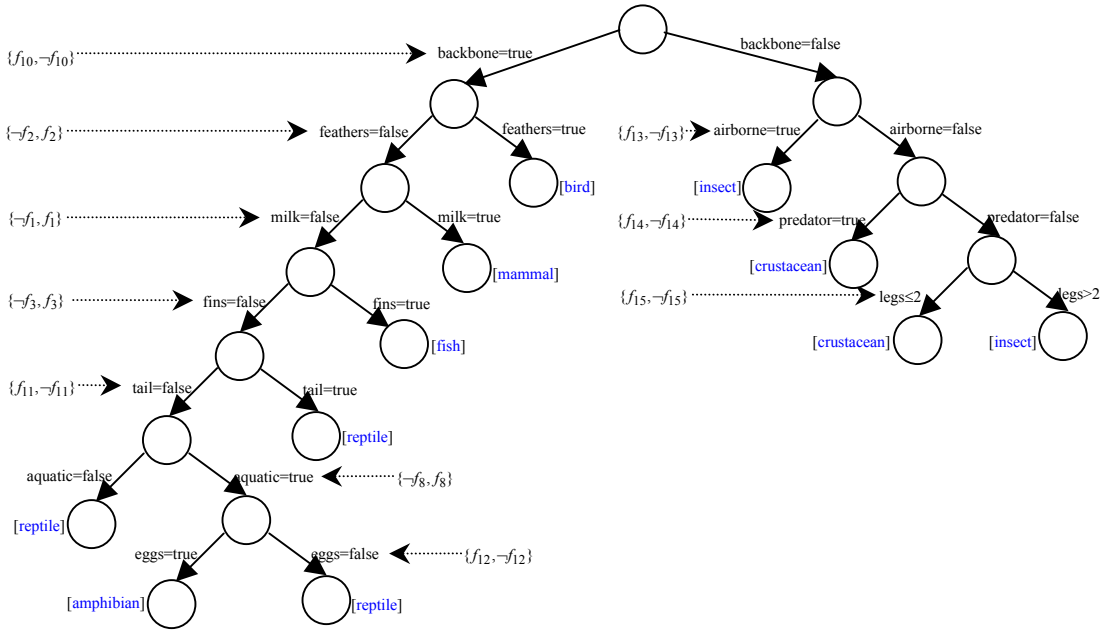


Figure 6. Decision tree built by C4.5 algorithm. Nodes are represented by circles. Terminal nodes are labeled with class names (in brackets). ASPs are shown as function sets resulting in branching of non-terminal nodes

The comments of the process of HC (Figure 7) are presented below.

The algorithm described in section 5 was trained on the entire set of instances S taken from the database ZOO and started in the root node number 0.

For λ=1 (λ is a subspace dimensionality, defined in section 5) three ASPs were found: $asp(feathers)\equiv\{f_2,\neg f_2\}$, $asp(backbone)\equiv\{f_{10},\neg f_{10}\}$, $asp(milk)\equiv\{f_1,\neg f_1\}$. These three ASPs were combined into the composite partition $\{f_2,\neg f_2\}\otimes\{f_{10},\neg f_{10}\}\otimes\{f_1,\neg f_1\}$, which divided the entire set of classes into four subgroups.

Thereafter, the algorithm attempted to partition the subgroup associated to node "2". For λ=1 only one

ASP was found $asp(fins) \equiv \{f_3, \neg f_3\}$ that separated class "fish" from other classes. The subgroup associated to the node "1" was skipped because it contained a single class "bird".

Thereafter, the algorithm attempted to partition the subgroup associated to the node "5". For λ=1 no ASPs were found, for λ=2 three ASPs were found: $asp(aquatic,legs) \equiv \{f_{16}, \neg f_{16}\}$, $asp(eggs,aquatic) \equiv \{f_{17}, \neg f_{17}\}$ and $asp(aquatic,breath) \equiv \{f_{18}, \neg f_{18}\}$. As $\{f_{17}, \neg f_{17}\}$, $\{f_{18}, \neg f_{18}\}$ appeared to be alternative partitions, the composite partition $\{f_{16}, \neg f_{16}\} \otimes (\{f_{17}, \neg f_{17}\}, \{f_{18}, \neg f_{18}\})$ was constructed. It separated "amphibian" and "reptile" classes.

Finally, the algorithm selected the subgroup associated to the last "impure" node "4". For λ=2 two ASPs were found: $asp(aquatic,legs) \equiv \{f_{19}, \neg f_{19}\}$ and $asp(breath,legs) \equiv \{f_{20}, \neg f_{20}\}$, which separated "crustacean" and "insect" classes. ASPs $\{f_{19}, \neg f_{19}\}$, $\{f_{20}, \neg f_{20}\}$ appeared to be alternatives so they were included into the same set of alternatives.

During recognition phase, error localization is easy with the decision tree shown in Figure 7. Suppose that some instance S was recognized as "amphibian", but actually it belongs to the class "fish". The reason of choosing the incorrect path can be clearly identified in the tree, which is the the ASP $\{\neg f_3, f_3\}$ in case of our example.

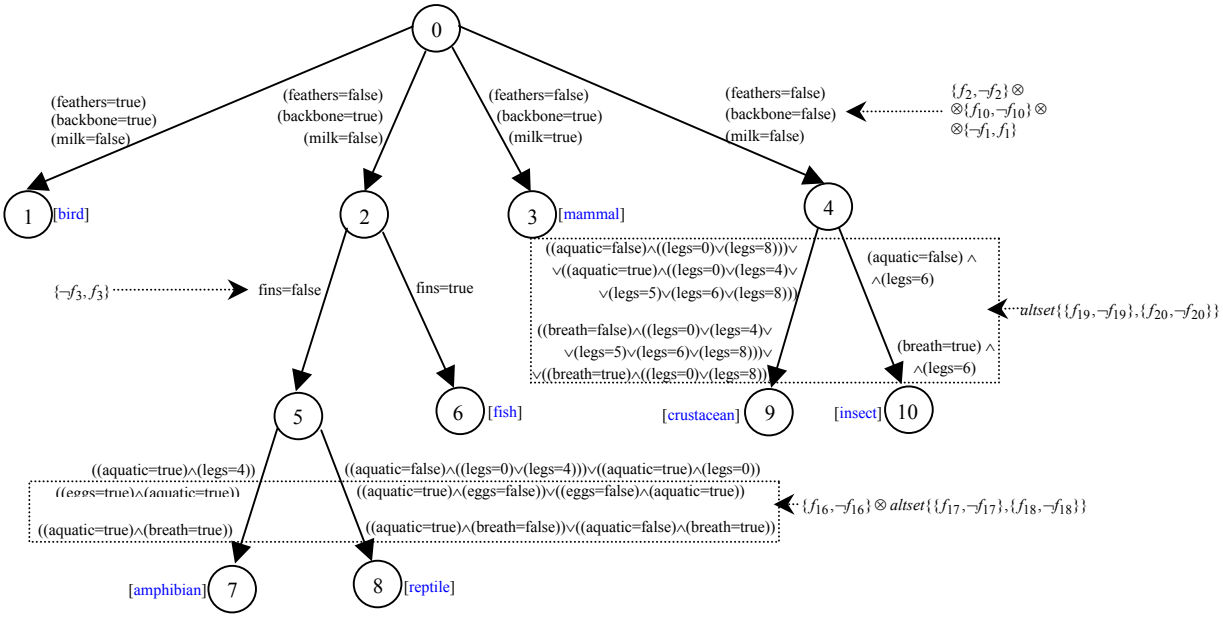


Figure 7. Decision tree built by HC algorithm. Nodes are represented by circles. Terminal nodes are labeled with class names (in brackets). ASPs are shown as function sets resulting in branching of non-terminal nodes. Sets of Alternative ASPs are enclosed within doted rectangles

The results of all decision trees are summarized in Table 2.

Table 2. Knowledge structure comparison considering given features.

Feature \ Method	HC	Ripper	C4.5
Nodes	11	19	21
Terminal nodes	7	11	9
ASPs	9	9	10
Number of different clauses	19	15	20

7 Conclusions

The new symbolic machine learning technique named Hierarchical Classifier (HC) belonging to the family of top-down decision tree induction methods is presented in this paper. HC is based on the strongly constraining node splitting criterion (stability) which discards splits that result in instances of the same class being divided among the descendant nodes. An exhaustive search for multi-attribute functions in the attribute space is performed for splitting tree nodes. This results in higher time complexity of HC with respect to other ML techniques. It was shown that if stability criterion is consistently applied, the decision tree with one terminal node per class is obtained. The decision tree built by HC has an important property of error localization, i.e. the ability of a system to find inconsistent parts of its knowledge and possibly correct them without a need to invalidate the entirety of knowledge. Error localization is very important to incremental ML which is envisioned for our future research. It was experimentally shown that HC achieves the same or better recognition accuracy than other state-of-the-art symbolic ML techniques, such as Ripper and C4.5, for ML tasks that are "easy" in the sense of good attribute space. It was also shown that

the redundant knowledge improved HC recognition accuracy by ~2%.

References

[1] **J. Cheng, U.M. Fayyad, K.B. Irani, Z. Qian.** Improved Decision Trees: A Generalized Version of ID3. *Fifth Int. Conf. Machine Learning, Ann Arbor, MI*, 1988, 100-107.

[2] **P. Clark, T. Niblett.** The CN2 induction algorithm. *Machine Learning*, *Vol*.3, 1989, 261-283.

[3] **W.W. Cohen.** Fast Effective Rule Induction. *From Machine Learning: Proceedings of the Twelfth International Conference*, 1995.

[4] **J.G. Ganascia.** CHARADE : A rule System Learning System. 10*th IJCAI*, *Milan*, 1987.

[5] **J. Hong, I. Mozetic, R.S. Michalski.** AQ15: Incremental Learning of Attribute-Based Descriptions from Examples, The Method and User's Guide. *Reports of the Intelligent Systems Group, ISG* 86-5, *UIUCDCS-F*-86-949*, Department of Computer Science, University of Illinois, Urbana, May* 1986.

[6] **I.R. Johnson.** Machine Learning. *Department of Computer Science, University of Bristol*.

[7] **R.S. Michalski, I. Mozetic, J. Hong, N. Lavrac.** The Multi-Purpose Incremental Learning System AQ15 and its Testing Application to Three Medical Domains. *Proceedings of the National Conference on Artificial Intelligence, AAAI*, Philadelphia, August 11-15, 1986.

[8] **T.M. Mitchell.** Machine learning. *McGraw Hill*, 1997, 52-80.

[9] **А. Рашкинис, К. Станёнис.** Система метапризнаков для формирования устойчивых логических разделяющих функций в задачах распознавания. *Труды Академий наук Литовской ССР. Серия Б*, 1988, *Т*. 3 (166), 141-148.

[10] **H. Ragavan, L. Rendell.** Lookahead Feature Construction for Learning Hard Concepts. *ICML*, Amherst, *MA, USA*, 1993. 252-259.

[11] **A.J. Raškinis.** Metapožymių metodikos ypatumai sprendžiant hierarchinio mokymo uždavinį. *Lietuvių katalikų mokslo suvažiavimo darbai* XV. 1991 m. *birželio* 9-16*d*., 767-775.

[12] **J.R. Quinlan.** Induction of decision trees. *Machine Learning*, *Vol*.1, 1986, 81–106.

[13] Donald Bren School of Information and Computer Sciences. *Zoo DB. http://www.ics.uci.edu/~frost/past/ics171-pastAssignments/P2-2002spring/P2-2419506/.*

Received December 2007.

DOI: 10.5755/j01.itc.37.1.11902