Discretization and Fuzzification of Numerical Attributes in Attribute-Based Learning

Ivan Bruha[1] and Petr Berka[2]

[1] McMaster University, Dept. Computing & Software,
Hamilton, Ont., Canada L8S4K1
Email: bruha@mcmaster.ca
[2] University of Economics, Laboratory of Intelligent Systems,
Prague, Czech Republic CZ-13067
Email: berka@vse.cz

Abstract. Machine learning (ML) algorithms have been capable of processing symbolic, categorial data only. Real-world problems, particularly in medicine, comprise not only symbolic, but also numerical attributes. There are several approaches to discretize (categorize) numerical attributes. This article describes two newer algorithms for such a discretization.

The first one has been designed and implemented in KEX (Knowledge Explorer) as its preprocessing procedure. The other discretization procedure was designed for the CN4 algorithm, a large extension of the well-known CN2. The discretization procedure in CN4 works on-line, i.e., it dynamically (within the induction) discretizes numerical attributes.

A large drawback of these discretization procedures, either off-line or on-line, is that they generate sharp bounds between intervals. One way how to eliminate an impurity around the interval borders is to fuzzify them. Here we introduce the newest empirical procedures for fuzzification, both off-line (within KEX) and on-line (CN4).

This chapter first surveys the methodology of empirical machine learning (Section 1), then attribute-based rule-inducing learning from examples (Section 2). Section 3 briefly introduces the KEX algorithm and Section 4 surveys CN4. The last Section focuses on discretization and fuzzification procedures, includes empirical results that compare performance of KEX, CN4, and other well-known machine learning algorithms as for discretization and fuzzification, and concludes with analysis.

Keywords. Empirical learning, inductive learning, attribute-based learning, rule induction, covering paradigm, numerical attributes, discretization, fuzzification

P. S. Szczepaniak et al. (eds.), *Fuzzy Systems in Medicine*

1 Empirical Learning: a Survey

1.1 Informal Definitions

Psychology defines learning as a process of changes in a system that enable the system to do the same task or tasks drawn from the same population more efficiently and more effectively the next time [25]. Learning involves assimilating information about objects, phenomena, and its generalizing. A learner increases his knowledge or skill in accomplishing certain tasks. He applies inferences to some material in order to construct an appropriate representation of some relevant aspect (concept) of reality. The process of constructing such a representation is a crucial step in any form of learning.

Human learning is one of the most important characteristics of human intelligence. In the similar way, *machine learning* (ML) is one of the most significant fields of artificial intelligence. Machine learning is a considerably old discipline; it was a founding branch of artificial intelligence in the late 1950's. However, it has become very attractive, especially recently, thanks to the Expert Systems (ES), Knowledge Discovery from Databases (KDD), Data Mining (DM), and their developments and applications: machine learning is one of the feasible means for automatic knowledge acquisition.

One can distinguish two basic activities of learning: knowledge acquisition and skill refinement. *Knowledge acquisition* consists in inferring and assimilating new material, composed of concepts, general laws, procedures, etc. The knowledge acquired should allow to solve a problem, perform a new task, or improve the performance of an existing task, explain a situation, predict behaviour, etc. *Refinement of skills through practice* consists in gradually correcting deviations between observed and desired behaviour through repeated practice. This activity of human learning covers mental, motor, and sensory processes. It should be noticed that the current research in ML focuses mostly on knowledge acquisition.

It should be also pointed out that most of the research done so far in machine learning has taken assumptions that are oversimplified with respect to human learning:

- The concepts learned by a machine learning system are usually described in more or less expressive formalism, e.g., in terms of conjunctions or disjunctions of elementary properties and relations; many real-world concepts cannot be described in such a naive manner.
- The concepts learned by a learner often belong to a unique level of abstraction whereas humans tend to organize their concepts into multiple levels of abstraction.
- Many learning systems rely on the assumption that a teacher is supervising their learning process. Human learning, however, does not always require such supervision.

- Most machine learning systems exploit so-called *batch-mode*, i.e. non-incremental (non-sequential) one in the sense that the entire training data must be available before any learning can take place. In contrast, humans use incremental (sequential) learning.

1.2 Taxonomy of Learning

There exists vast number of machine learning algorithms at present. One can categorize machine learning systems along many different dimensions. However, the following dimensions seem to be widely accepted [12]:

1. categorization on the basis of the amount of effort or inference required by the learner,
2. categorization according to underlying paradigms, and
3. categorization on the basis of the representation of input data and that of acquired concepts.

The taxonomy of learning systems according to the amount of effort (inference) required by the learner to perform its learning task looks as follows [12]:

1. *Rote learning* consists in just recording data or pieces of knowledge supplied by the teacher. No inference nor transformation is applied by the learner to this input.
2. *Learning from instruction* (*learning by being told*) consists in acquiring knowledge from an external source and integrating it with the knowledge previously acquired. The learner transforms the acquired knowledge from the input language into an internally usable representation.
3. *Learning by analogy* consists in acquiring new facts or pieces of knowledge by transforming and augmenting an existing knowledge from a similar source idea to a target analogous concept.
4. *Explanation-based learning* forms a general characterization of a concept from a very small number of examples and large background knowledge.
5. *Learning from examples* consists in acquiring a general concept description (characterization) from a given training set of examples and counterexamples of the concept. The teacher supplies the training set and provides the desired concept for each example. The learner applies induction and can eventually use the background knowledge specific for the given domain. This form of learning seems to be the most investigated one in artificial intelligence.
6. *Learning from observation and discovery* is a very general form of inductive learning that consists in clustering input examples or observations into different conceptual classes, discovering new concepts and their hierarchical structure. The learner is not supervised by an external teacher, and does not have any information about desired classes (concepts) of input examples; therefore, this form is also called *unsupervised learning* or *learning without a teacher*. It corresponds to typical human learning, where the learner elaborates a

conceptual structure that will enable him to organize his observations and classify future experience.

Induction is the most appropriate form of inference in artificial intelligence, therefore - if we are talking about learning - we usually mean only the *inductive learning* (i.e. learning from examples and from observation and discovery). Inductive learning can be viewed as a heuristic search of the space of concept descriptions.

Two contrasting groups of learning can be distinguished among the above learning forms:

- *Empirical* (*similarity-based*) *learning* involves examining multiple training examples of one or more concepts (classes) in order to determine the characteristics they have in common. Such systems have usually limited background (domain-specific) knowledge bases. The learner acquires a concept by generalizing these examples through a search process based on inductive inference. The empirical learning comprises both learning from examples and from observation and discovery.
- *Analytic learning* formulates a generalization by observing only a single example (or with absence of examples at all) and by exploiting a large background knowledge about the given domain. Explanation-based learning and learning by analogy belong to this group.

Empirical learning systems can be also categorized according to underlying paradigms they utilize:

- *Statistical methods* are mostly used if input data have numerical character. They are more or less based on statistical techniques. We mention just a few among many methods: discriminant functions (non-parametric methods), k-nearest-neighbour, Bayesian methods, parametric (distribution-parameters estimate) methods and distribution-structure estimate methods, cluster analysis, regression methods, belief functions, etc.
- *Symbolic (logical) methods* are used for input data represented by (symbolic) attributes, relational structures, semantic nets, formulas of predicate calculus, Prolog terms, strings of terminals, etc. They can exploit both statistical techniques and knowledge-intensive approaches.
- *Connectionist (neural-net) methods* are applied to numerical data, and encompass various topologies of neurons and a wide variety of learning techniques, some of which are statistically based. They are designed to work in a manner analogous to the neural networks found in animal life.
- *Genetic algorithms* emulate biological processes and have become quite attractive in machine learning by applying various selection, crossover, and mutation procedures.

From now on, we will focus on *symbolic empirical learning from examples*. In most problems, both positive (belonging to a given concept) and negative (not belonging to the given concept) examples are available. The positive examples force

generalization of the concept description and the negative ones prevent its overgeneralization.

Another taxonomy of empirical learning is based on the way of processing of training examples:

- *Batch (one-trial) mode* assumes all training examples to be presented to the learner at once since all of them are required for acquiring a concept description. Therefore, they must be stored in its database during the entire acquisition process.
- In *incremental mode*, the learner reads training examples subsequently; it forms one or more hypotheses of the concept, and gradually refines these hypotheses when reading the next available example.

2 Attribute-Based Rule-Inducing Learning from Examples

2.1 Learning from Examples: Definitions

Learning from examples (called also *concept acquisition*) is one of the most researched form of learning. It has been applied in many domains, e.g., to learn medical diagnoses, to predict weather, in speech recognition, chemistry, geology. The obvious kind of its applications is associated with knowledge acquisition for Expert Systems, Knowledge Discovery from Databases, and Data Mining. Learning from examples exploits the known empirical observation that experts (i.e. 'teachers') find it easier to produce good examples than to provide explicit general theories or concept descriptions (so-called the knowledge-engineering paradox).

Let us try to define it more formally. A learning-from-examples system is provided with a set of training examples (objects) of one or more concepts (classes) by a teacher (designer, domain expert), and based on it it forms an abstract generalization of these examples, i.e. acquires underlying common characteristics (theory) and infers formal descriptions of these concepts. The teacher can help the learner by selecting good, representative examples, and also by describing the examples in a language that permits formulating elegant, efficient, and powerful general descriptions of concepts. Therefore, the task of the teacher is not trivial, although he need not specify the targeted concepts in complete detail but just provide illustrative, typical examples (objects) instead.

An *object* (also called *case, event, evidence, fact, instance, manifestation, pattern, observation, statement*) is a unit of the given problem. E.g., if a child learns different car makes, then each distinct car is an object. Let X be the space of all objects of the given task. A concept (class) can be characterized as a set $c \subseteq X$ of objects having common properties in the given task. E.g., Opel Kadett is one concept (class) of car makes.

A *training (learning) example (object)* is actually a pair

```
[ object, desired_concept ]
```

Both object and its desired class (concept) are provided by the teacher. (If the learner is supposed to infer just one concept, then all training examples of this concept are called positive ones, those of other concepts are negative examples.) A set $T \subseteq X$ of training (learning) examples given to the learner is called training (learning) set.

The problem of learning a concept c from examples can be specified as follows: Given a training set $T \subseteq X$, find an expression (formula) `Descr` (called concept description, concept hypothesis, or inductive assertion) in a concept description language, such that

- the concept description is complete, i.e., each positive example of c in T matches `Descr`, and
- the concept description is consistent, i.e., no negative example of c in T matches `Descr`.

The above (still more or less informal) definition of learning from examples needs some additional explanations.

(1) To learn the concept c means to learn to recognize all objects of this concept, i.e. for each object $x \subseteq X$ to recognize whether x is in c or not. However, the training set T is always finite, but the object space X contains usually quite a huge number of objects. Thus, the majority of objects to be recognized will be those from X not belonging to the training set (called unseen, new, unknown objects). Hence, we have to realize that the concept description `Descr` is just the learner's understanding of the concept c, i.e. a hypothesis of c. Consequently, we would like to have a guarantee that the learner generalizes the information from the training set to the entire object space X. By other words, Descr should be more general than the positive examples in T. Such generalization is the essence of inductivity of learning.

Statistics claims that the necessary condition for inductivity of learning (i.e. generalization of information in a training set) is representativeness of the training set, i.e., the training set should comprise representative, typical examples that reflect common characteristics of the concepts to be inferred.

(2) Besides inductivity, learning should be also sequential (incremental). Sequentiality (incrementality) exhibits these two characteristics:

- training examples are gradually perceived by the learner, and most (or all) of them are forgotten;
- after forming one or more hypotheses of the concept, the learner is capable of *refining* those hypotheses successively.

However, as we mentioned above, majority of ML algorithms exploit batch mode and hence, do not exhibit incrementality of learning.

The result of inductive inference, a concept description, is usually interpreted in so-called *classification fashion*: a concept description is interpreted as a *decision scheme* (e.g., a set of decision rules, or a decision tree) for a classifier that will be able (more or less correctly) to recognize unseen objects of the given task. An unseen object is usually compared with conditions of such rules by invoking a kind of matching algorithm.

2.2 Representation of Objects

Any learner or classifier does not perceive its input objects directly, but through their representations in an *object description language* which is more suitable for understanding. Since most learning systems are implemented as computer programs, such a description language involves numbers and symbols only. As a consequence, input objects have to be formally represented (described) for both the learner and classifier. The description (representation) is called formal because the actual (physical) interpretation of its elements is not important for learning or classification. The learner learns something without having the imagination about the tangible meaning of actual objects: the learner processes just numbers, symbols, or structures, and it does not care whether they actually represent a chemical process or proportions of a pretty girl.

An object description language is fully designed by the system designer (teacher). He has to select all elementary descriptions on objects before the actual inductive learning takes place. In general, we distinguish two major kinds of object description languages: attribute description and structural (relational) description. Since the first one is used in medicine, geology, and other applications, we will focus here on *attribute description* only.

Elementary properties, called *attributes*, are selected on objects. Such an attribute can be e.g. height of an object, its colour, its weight, etc. Each attribute has its domain of possible values. Depending on the organization of the attribute's domain, we distinguish two basic types of attributes: discrete (symbolic) and numerical (continuous) ones:

- The value domain of a *discrete (symbolic)* attribute consists of independent symbols or names. For instance, `colour` is a nominal attribute since its value domain does not reflect any structure. (One could object to it declaring that colours are ordered in the spectrum; this fact, however, is not relevant in inductive learning.) *Binary* attributes whose values are yes or no also belong to discrete attributes; e.g., the binary attribute raining has two values: `raining = yes, raining = no.`

- The value set of a *numeric (continuous)* attribute is a totally ordered set; e.g. `weight, temperature.` They are often called *features.*

A *selector* is an attribute name followed by an attribute value, e.g., `weight=80.` An object is thus represented by a list of selectors (attributes and their values), called *attribute list*, e.g.,

```
[ hair = black & eyes = blue & height = 165 ]
```

2.3 Representation of Concepts (Classes)

Besides an object description language, a specification of a learning task has to comprise a *language for concept descriptions (hypotheses).* Such a language is of types similar to those that can be used for representing knowledge in general. There exist various formalisms, but the following two ones are mostly used in machine learning: decision trees [21], [24] and decision rules. Since both KEX and CN4 algorithms process decision rules, we concentrate here on the latter concept description, a *set of decision (production) rules.* A decision (production) rule is of the form

```
if Condition then class is c
```

The condition on the left-hand side is usually of the form of a conjunction of selectors.
Each class (concept) can be characterized by more than one decision rule. The reason follows from a usual way of the constructing the rules: a learning algorithm infers consistent conditions that need not be complete. Therefore, a set of consistent conditions has to be generally inferred.
A condition of a decision rule is formed by a conjunction of selectors (i.e., in a similar way to an attribute list that represents an object) with the following extension: The right-hand size of any numerical selector can comprise also an interval of its values, e.g., `weight > 69, 25 < temperature < 35.` Thus, a decision rule can look as follows:

```
if colour=red && weight>69 then class is Normal
```

Now there is time for more precise definition of matching an object with a rule condition. A selector of a rule condition matches according to its type:
(i) Let `A` be a discrete attribute and `V` one of its values, then the selector `A = V` is *satisfied* by the object `x` if the value of `A` in `x` equals to `V.`
(ii) The selector `A ∈ V`, where `A` is a numerical attribute and V an interval, is *satisfied* by the object `x` if the value of `A` in `x` lies within the interval `V.`

A rule condition is *satisfied* by (or *covers*) the object `x` (`or: x` matches the condition) if each selector of the condition is satisfied by `x`. We say that a rule is *satisfied* by (or *is fired* by, or *covers*) the object `x` if its condition is satisfied by `x`.

As we already noted, a concept description is *complete* if it covers (is satisfied by) all positive training examples of this concept. A concept description is *consistent* if it does not cover any negative example. (The same definitions hold for single rule conditions.)

Section 1.2 has introduced several paradigms of Machine Learning. Here we focus on the symbolic (logical) paradigms. Now, it is the time to declare that there exist a few symbolic paradigms of learning from examples. One commonly used paradigm is called *divide-and-conquer*. It is widely utilized by the family of TDIDT (*T*op-*D*own *I*nduction of *D*ecision *T*rees) learning algorithms that induce decision trees from examples. One of its first pioneers is the ID3 algorithm [21], but currently mostly used and well-known member is the C4.5 algorithm [24]. Training examples are portrayed by attribute lists and the output induced (the concept description) is represented by a decision tree.

Another widely used stream in symbolic learning uses the *covering rule-inducing* paradigm. The AQx family [17], [18] is one of the first algorithms utilizing this paradigm. Since KEX and CN4 exploit this paradigm we will center on it. In the following two sections, we briefly introduce two of these sophisticated attribute-based rule-inducing machine learning algorithms.

3 KEX: Knowledge Explorer

A Survey of the System

KEX (Knowledge EXplorer) performs symbolic empirical multiple concept learning from examples, where the induced concept description is represented as weighted decision rules in the form

```
Ant ==> C(w)
```

where: `Ant` is a combination (conjunction) of selectors,
`C` is a single category (class), and
weight `w` from the interval [0,1] expresses the uncertainty of the rule.

During knowledge acquisition, KEX works in an iterative way, in each iteration testing and expanding an implication `Ant ==> C` This process starts with an

„empty rule" weighted with the relative frequency of the class C and stops after testing all implications created according to the user defined criteria[1].

During testing, the validity (conditional probability $P(C|Ant)$) of an implication is computed. If this validity significantly differs from the composed weight (value obtained when composing weights of all subrules of the implication Ant ==> C, then this implication is added to the knowledge base. To test the difference between validity and composed weight, we use the $\chi 2$ goodness-of-fit test. The weight of this new rule is computed from the validity and from the composed weight using inverse composing function. For composing weights we use a pseudobayesian (Prospector-like) combination function [14]:

$$x \oplus y = \frac{xy}{xy \oplus (1-x)(1-y)}.$$

Since the knowledge base can contain both a rule Ant ==> C(weight) and its subrule Ant' ==> C (weight') where Ant' $\subset$ Ant, this operation is used with respect to the correction principle suggested by Hájek in [15].

When expanding, new implications are created by adding single selector to *Ant*. New implications are stored according to frequencies of *Ant* in an ordered list. Thus, for any implication in question during testing, all its subimplications have been already tested.

To clarify the KEX algorithm, let us consider the following simple example. Let the implication in question is `7a11a ==> 1+`[2] with the following four-fold contingency table:

	C	*non* C
Ant	11	14
non Ant	c	d

Table 1. Contingency Table for `7a11a ==> 1+`

So, the validity of this implication is 11/(11+14) = 0.44. Suppose, there are the following rules in the KB, which are applicable for the combination *Ant*:

```
  ∅  ==>  1+ (0.6800)
11a  ==>  1+ (0.2720)
 7a  ==>  1+ (0.3052)
```

[1] The algorithm is shown in **Fig. 1**.

[2] The implication has the meaning: IF attribute 7 has value *a* AND attribute 11 has value *a* THEN attribute 1 has value +.

Input
Data D, goal C, maximal length l_{max} of the left-hand side of a rule, minimal frequency f_{min} of the left-hand side of a rule, and minimal validity P_{min} of the implication.

Initialisation:
1. Let KB be a list consisting of empty implication ∅ ==> `C(weight)` with the weight equal to the relative frequency of C in the data;
2. Let CAT be a list of selectors jc sorted in descending order of $\| jc \|$[3];
3. Let OPEN be a list of implications jc ==> `C` sorted in descending order according to $\| jc \|$;

Computation:
while OPEN is not empty **do**
1. select the first implication `Ant ==> C` from OPEN;
2. compute its validity $P(C \mid Ant)$;
3. **if** $P(C \mid Ant) > P_{min}$ **then**
   3.1. compute composed weight *CW(C,Ant)* from the weights of all subrules of `Ant ==> C` which are already in KB, using composition function ⊕;
   3.2. if the validity significantly differs[4] from the composed weight then add `Ant ==> C` to KB with the weight w such that
   $w \oplus composed\ weight = validity$;
4. **if** length(Ant) < l_{max} **then**
for each jc from CAT such that jc is in CAT before any selector from Ant **do**
   4.1. generate new combination $jc \wedge Ant$;
   4.2. **if** $\| jc \wedge Ant \| \geq f_{min}$ **then**
      insert `jc ∧ Ant ==> C` into OPEN just after the last implication `Comb ==> C` such that $\| Comb \| \geq \| jc \wedge Ant \|$;
enddo;
5. delete `Ant ==> C` from OPEN;
enddo;

Fig. 1. KEX knowledge acquisition algorithm

From these three rules, we can compute the composed weight $CW(C, Ant) = 0.6800 \oplus 0.2720 \oplus 0.3052 = 0.2586$. Since this composed weight significantly (according to χ^2) differs from the validity, we must add the

[3] $\| jc \|$ is the number of objects that fulfil the category jc.
[4] We test the difference using χ^2 goodnes-of-fit test.

implication `7a11a ==> 1+` into KB with weight w such, that $w \oplus 0.2586 = 0.44$.

Unlike C4.5 or CN2, KEX doesn't remove covered examples from the data. So one example can be covered with more rules. During consultation for a particular case, all applicable rules (e.g. rules whose `Ant` corresponds to characteristics of the case) are fired and their weights are combined using $\oplus$. An unseen example is assigned to the class with highest composed weight.

4 CN4: Exploiting Covering Paradigm

4.1 An Overview

The family of CN*x* learning algorithms is another set of attribute-based rule-inducing algorithms that exploits so-called covering paradigm (see the next section). The most known member of this family is the CN2 algorithm [13], [11]. It induces a set of decision rules of the form

`if` *Condition* `then class is` *c*

It handles noisy data by applying statistical techniques in order to select the most predictive and reliable conditions. Due to the relaxation on consistency and completeness of the decision set inferred, the algorithm exhibits quite promising performance in processing noisy data. CN2 uses for that purpose a heuristic to terminate the rule construction, based on an estimate of the noise present in the training data. The original version uses either entropy[13], [11] or a Laplacian error estimate [11]. It should be also pointed out that CN2 induces either *ordered* or *unordered* set of rules.

To include some extensions, the first author of this contribution has decided to design and implement his own version of this inductive algorithm which is called CN4. Firstly, he has enhanced the algorithm itself by adding more robust stopping conditions when generating the best conditions, and by improving numerical attribute processing. Then, following in spirit of the extensions of ID3 that handle attribute cost [20], [26], he has incorporated routines and heuristics that are able to process attribute cost and thus to economize the classification of unseen objects. Last but not least, inspired by processing unknown attribute values in ID3 [23], [3], several routines for processing unknown attribute values to CN4 have been added, too. CN4 also deals with rule quality and various schemes for combining these qualities [5], [7]. The entire algorithm has been written in C and runs under the Unix or MS-DOS systems.

4.2 The Covering Paradigm

A learning algorithm using the *covering paradigm* works in sweeps. In each sweep, it tries to find a reliable and powerful rule which covers a subset of training examples; the covered examples are then removed from the training set and the algorithm tries to cover the remaining examples in the next sweeps. The entire process is terminated if all examples are covered or a certain stopping condition is satisfied.

Covering learning algorithms exhibit two modes: ordered and unordered. In the ordered mode, the order of rules generated by a covering learning algorithm is vital, i.e,. the rules create a 'cascade' for testing that cannot be changed. Ordered-mode covering paradigm may be generally formalized by the procedure `ORDERED_COVER(E)` where E is a given training set (**Fig. 2**).

procedure ORDERED_COVER(E)

ListOfRules := NIL
until E = 0 **do**
1. Find a rule condition Cond according to a certain technique
2. Let E' be the subset of training examples covered by Cond
   E := E \ E'
3. Add the rule
   `if Cond then class is C`
   to the end of ListOfRules where C is the desired class of examples in E'

enduntil
return ListOfRules

Fig. 2. CN4 ordered-mode algorithm

All the learning algorithms utilizing this paradigm differ namely in the step 1 and in the stopping condition which is here (for simplicity) expressed as $E = 0$, i.e. the process stops if the entire training set is covered. Some algorithms induce rule conditions that may cover examples from more than one class; in this case, the class C in the step 3 is the *majority* class of examples in E'. A technique for finding a suitable rule condition (step 1) usually evaluates a condition *Cond* according to a user-specified heuristic which is determined either by (negative) entropy:

$$NegEntr(Cond) = \sum_r \frac{K_r(Cond)}{K(Cond)} \log_2 \frac{K_r(Cond)}{K(Cond)}$$

or by Laplacian criterion for expected accuracy (for the class C_r):

$$Lapl(C_r, Cond) = \frac{K_r(Cond) + 1}{K(Cond) + R}$$

where $K_r(Cond)$ is the number of training examples of the class C_r covered by the condition *Cond*, $K(Cond)$ is the total number of examples covered by the condition, R is the number of classes involved in the given task. Here the larger value of the evaluation function means the better condition. It should be also noticed that the entropy may be exploited in the ordered mode only.

In the unordered mode, the rules are generated for each class independently, i.e., rules can be tested within classification in any fashion. A general procedure for unordered-mode covering paradigm looks as shown in **Fig. 3**.

```
procedure UNORDERED_COVER(E)

ListOfRules := NIL
for each class C do
        Let E+ be examples of the class C in E
        until E+ = 0 do
                1. Find a rule condition Cond with respect to C
                2. Let E' be the subset of training examples covered by Cond
                   E+ := E+ \ E'
                3. Add the rule
                        if Cond then class is C
                   to ListOfRules
        enduntil
endfor
return ListOfRules
```

Fig. 3. CN4 unordered-mode algorithm

Again, the covering algorithms with unordered mode differ in the step 1 and the stopping condition. The term „with respect to *C*" is mostly interpreted as „where *C* is the majority class".[5]

Covering learning algorithms of the CNx family use the star methodology and the beam search in a way similar to the AQx family [17], [18], but they handle noisy data by applying statistical techniques, analogous to those used for tree pruning in TDIDT [10], [24]. They relax the constraint of both consistency and completeness. A rule induced may cover examples from different classes, i.e., some training examples need not be classified perfectly. Moreover, some examples need not be covered by any rule

[5] In literature, one may find an alternative procedure `UNORDERED_COVER_1(E, C)` which induces rules for the explicitly specified class *C*; its body is identical to the core of the **for**-loop above.

induced. Due to this relaxation, the algorithms exhibit quite promising performance in processing noisy data.

A rule condition (see step 1) is searched for by a procedure that selects conditions according to a certain heuristic; moreover, the condition has to satisfy a so-called significance test, i.e., it must be a promising and reliable one. If no such condition is found, the inductive process terminates.

To illustrate the performance of CN4 we will at least present its output (set of decision rules) both for the ordered and unordered mode. For simplicity, let us consider the well-known set of the weather problem [21] with 14 examples, two classes, and 4 symbolic attributes (`windy`, `humidity`, `outlook`, `temperature`). The ordered mode of CN4 yields:

```
if outlook=overcast then class is +;
Kr=[ 4 0]; signif=5.099; quality=0.889; cost=1
else if windy=false && humidity=normal then class is +;
Kr=[ 3 0]; signif=3.825; quality=0.867; cost=2
else if humidity=high && outlook=sunny then class is -;
Kr=[ 0 3]; signif=8.913; quality=0.920; cost=0
else if windy=true && outlook=rain then class is -;
Kr=[ 0 2]; signif=5.942; quality=0.880; cost=0
else if true then class is +;
Kr=[ 2 0]; signif=2.550; quality=0.844; cost=0
```

Here `Kr` represents the (absolute) frequency of examples covered by each rule, `signif` is the rule significance, `cost` is its cost if we consider economy learning (by default cost of each attribute is 1), and `quality` depicts the quality of each rule. Note that the structure if - else if --- forms the cascade of the rules whose order cannot be changed. The unordered mode of CN4 yields:

```
if humidity=high && outlook=sunny then class is -;
Kr=[ 0 3]; signif=8.913; quality=0.920; cost=2
if outlook=overcast then class is +;
Kr=[ 4 0]; signif=5.099; quality=0.889; cost=1
if windy=false && humidity=normal then class is +;
Kr=[ 4 0]; signif=5.099; quality=0.889; cost=2
if windy=true && outlook=rain then class is -;
Kr=[ 0 2]; signif=5.942; quality=0.880; cost=2
if windy=false && outlook=rain then class is +;
Kr=[ 3 0]; signif=3.825; quality=0.867; cost=2
if humidity=normal && temperature=mild then class is +;
Kr=[ 2 0]; signif=2.550; quality=0.844; cost=2
if true then class is +;
Kr=[ 9 5]; signif=0.000; quality=0.714; cost=0
```

Notice that the algorithm always generates the default rule with its condition equal to true. It is used only if no rule matches an unseen object to be classified; the object is then classified to the majority class of the given task.

5 Discretization and Fuzzification of Numerical Attributes

5.1 Introduction

The genuine symbolic machine learning algorithms were able to process symbolic, categorial data only. However, real-world problems, particularly in medicine, involve both symbolic and numerical attributes. Therefore, there is an important issue of machine learning to discretize numerical attributes. The assignment of discretization of numerical variables is well known to statisticians. Different approaches are used; for instance, discretization into a given number of categories using equidistant cutpoints, or categorization based on mean and standard deviation. All these approaches are 'class-blind', since they do not take into account that objects belong to different classes. Therefore, they are not very efficient for machine learning algorithms.

Most newer versions have been designed and enhanced by adding the possibility to deal also with numerical data. In ID3 or C4.5, the algorithms for discretization are based mostly on *binarization* within a subset of training data created during tree generation [9], [22]. KNOWLEDGESEEKER, a commercial system of the TDIDT family, uses F-statistics instead of χ^2-statistic to test the dependence when processing a numerical attribute during tree induction [4]. Another interesting approach to discretization can be found in [16].

Discretization of numerical attributes into crisp intervals does not correspond to real situations in many application areas, particularly in medicine. Small difference in the value of an attribute cannot completely change the class of an object. For instance, if commonly accepted threshold of body temperature of a patient with flu is 37°C, then 36.9°C does not mean definitely a healthy situation and 37.1°C an ill one. Thus, it seems more realistic to use *fuzzy* intervals for low as well as high body temperatures with respect to the state of patient.

This section introduces two newer algorithms for discretization and fuzzification of numerical attributes. They are „class-sensitive", which means that the procedures do the discretization/fuzzification according to the class-membership of the training objects. This section first discusses the discretization/fuzzification procedure which is implemented in the KEX [2] as its preprocessing (off-line) procedure. Its idea is to discretize the numerical attributes so that the resulting categorization fits the way how KEX creates a knowledge base. Nevertheless, the resulting categorization is suitable also for other machine learning algorithms. Next, we present another discretization/fuzzification procedure that is implemented in the covering learning algorithm CN4 [6], [8], a large extension of the well-known CN2

machine learning algorithm [13]. Finally, experiments of applying various scenarios to several ML databases, exploiting both KEX, CN4, and C4.5, are introduced, and corresponding results are discussed.

5.2 Discretization and Fuzzification in KEX

We categorize each numerical attribute separately [1]. The basic idea is to create intervals for which the a-posteriori distribution of classes P(C|interval) significantly differs from the a-priori distribution of classes P(C) in the whole training data. This can be achieved by simply merging such values, for which „most" objects belong to the same class. Within the KEX knowledge acquisition approach, this will lead to rules with the description of interval on the left-hand side of the rule:

```
interval ==> C
```

The algorithm for discretization is shown in **Fig. 5**. The number of resulting intervals is „controlled" by giving a threshold for minimal number of objects within one interval; less frequent intervals are labeled as „UNKNOWN" in the step 3.1.

The algorithm does not resolve the situation, when a large amount of intervals with rare occurrences is created. This can be solved by giving a threshold for minimal number of objects within one interval, and in step 3.1 by assigning less frequent intervals with the label „UNKNOWN". So it may happen, that the result of the discretization is a single interval. Such situation indicates, that the attribute is irrelevant for the classification task and thus can be omitted during induction.

The fuzzification was introduced in the steps 3.2 and 3.3 of the discretization procedure. Instead of merging „uncertain" interval with its most promissing neighbour (see [1]), fuzzy borders (with linearly increasing or decreasing characteristic functions) between „certain" intervals were used.

When discretizing values of one numerical attribute of an object using this fuzzy discretization procedure, we can obtain up to two categories (discretized values) with the sum of characteristic functions up to 1 (see **Fig. 4**). Thus, the original object with one numerical attribute can be split (according to the value of this attribute) into two objects, each having the global characteristic function[6] lower then 1. Let us call such objects *incomplete*. This splitting of objects can be repeated for every numerical attribute[7].

---

[6] Global characteristic function is a product of characteristic functions of values of each attribute; for discrete attribute the characteristic function is equal to 1.

[7] Global characteristic functions of all objects related to one original object sum up to 1.

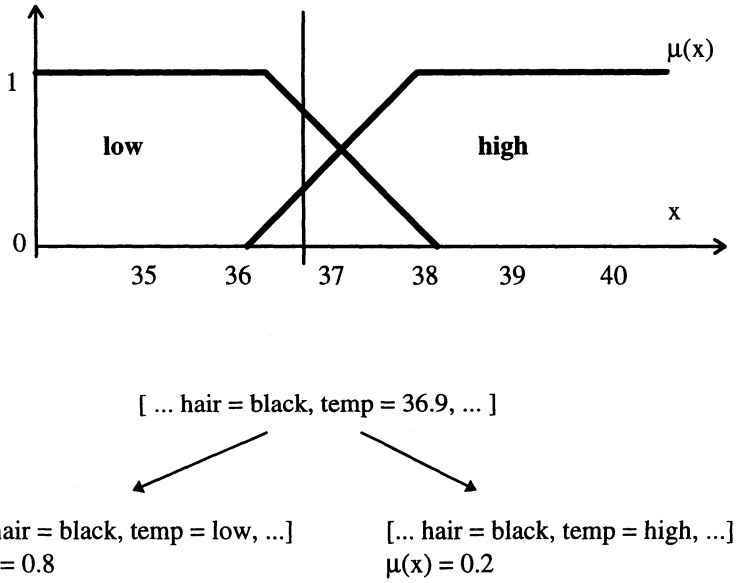

Fig. 4. Body temperature

To handle incomplete objects yielded by the fuzzy-discretization step, the induction algorithm of KEX has been modified only so as to work with the global characteristic functions. It selects and processes only one incomplete object with the largest global characteristic function from the set of such objects generated from a single original object. Hence, the induction algorithm processes „incomplete" objects in the same way as „complete" ones (not split). The global characteristic function is also taken into consideration during classification when counting number of successfully classified objects.

During categorization of an attribute, we can loose some information hidden in the data. We can measure this loss by the number of contradictions before and after the categorization. By contradictions we understand situations when objects described by same attribute values belong to different classes. Any learning algorithm will classify such objects as objects belonging to the same (usually the majority) class and objects belonging to other classes will be classified wrongly. We can count such errors and thus estimate the maximum possible accuracy as:

$$1 - \frac{\#.\ \text{errors}}{\#.\ \text{objects}} .$$

MAIN LOOP:
1. create ordered list of values of the attribute;
2. **for each** value **do**
   2.1. compute frequencies of occurence of objects with respect to each class;
   2.2. assign the class indicator to every value using procedure ASSIGN;
3. create the intervals from values using procedure INTERVAL;

ASSIGN:
if for the given value all objects belong to same class,
then assign the value to that class
else if for the given value the distribution of objects with respect to class membership significantly differs (according to χ^2 goodness-of-fit test) from frequencies of goal classes,
then assign that value to the most frequent class
else assign the value to the class "UNKNOWN";

INTERVAL:
3.1. **if** a sequence of values belongs to the same class,
then create the interval $INT_i = [LBound_i, UBound_i]$ from these values (with characteristic function set to 1 in the entire range $[LBound_i, UBound_i]$;
3.2. **if** the interval INT_i belongs to the class "UNKNOWN"
then
if its neighbouring intervals INT_{i-1} , INT_{i+1} belong to the same class
then create the interval by joining $INT_{i-1} \cup INT_i \cup INT_{i+1}$ with characteristic function set to 1 in the whole range $[LBound_{i-1}, UBound_{i+1}]$;
else create one interval by joining $INT_{i-1} \cup INT_i$ with characteristic function set to 1 in the range $[LBound_{i-1}, UBound_{i-1}]$ and set to $\frac{x - LBound_i}{UBound_i - LBound_i}$ for x in the range $[LBound_i, UBound_i]$, and second interval by joining $INT_i \cup INT_{i+1}$ with characteristic function set to 1 in the range $[LBound_{i+1}, UBound_{i+1}]$ and set to $\frac{LBound_i - x}{UBound_i - LBound_i}$ for x in the range $[LBound_i, UBound_i]$;
3.3. create continuous coverage of the attribute by treating „gaps" between intervals as intervals of class "UNKNOWN" (in the same way as in the step 3.2)

Fig. 5. Algorithm for discretization and fuzzyfication in KEX

5.3 Discretization and Fuzzification in CN4

The other discretization procedure is embedded in the machine learning algorithm CN4 [6], [8], a large extension of the well-known CN2 [13]. It exploits a procedure for splitting continuous ranges of numerical attributes generally to more than two intervals. Unlike [9], that calls the splitting procedure recursively, promising bounds (thresholds) are found here within 'one shot', iteratively [1], [8].

Since CN4 exploits the covering paradigm (i.e., it tries to find in each sweep a relevant rule condition which covers a certain portion of training data), the relative frequencies of classes as well as the distributions of values of each (not only numerical) attribute may change in each sweep. Hence, the discretization procedure is invoked at the beginning of each sweep of the covering algorithm. Because of this characteristic, we call this type of discretization as 'dynamic', 'on-line' one, unlike the discretization procedure of KEX which is done only once ('off-line'), before the actual inductive process.

To discretize a numerical attribute A actually means to find suitable (*promising*) bounds V_j which divide the entire range of this attribute values into intervals that are as consistent as possible. Promising upper [lower] bounds correspond to non-increasing [non-decreasing] local maxima of a heuristic evaluation function $H(V_j)$ (entropy or Laplacian estimate in CN4). Hence, the discretizing procedure goes along the entire range of the attribute A and considers each value of A that occurs in the training set as a potential promising bound. It invokes the heuristic evaluation function for the above intervals to find promising upper [lower] bounds. The inner intervals are generated from the bounds found above and immediately tested. As the last step, a pre-specified number of promising intervals (ones with lower bounds, upper bounds, and inner intervals) are then selected according to their evaluation.

The entire procedure for discretizing a numerical attribute A is shown in **Fig. 6**. The function $Rank(D_1, ..., D_R)$; corresponds to the user-selected heuristic evaluation function, for instance, for entropy

$$Rank(D_1 ; ...; D_R) = \sum_r \frac{D_r}{D} \log_2 \frac{D_r}{D}$$

where D is the sum of all D_r's.

procedure `SetBounds (A)`
1. **Let** *ArrayOfBounds* be *NIL*;
2. **For each** value V_j of the attribute *A* **do**
   2.1. Calculate the frequencies $Dleft_r$ [$Dright_r$] of the values *V* for which $V \leq V_j$ [$V > V_j$] for each class C_r, $r=1,...,R$ (R is the number of classes);
   2.2. Calculate the heuristic degree $Hleft(V_j) = Rank(Dleft_1, ..., Dleft_R)$ which considers V_j as a potential *upper* bound, i.e. the selector $A \leq V_j$;
   similarly calculate $Hright(V_j) = Rank(Dright_1, ...,Dright_R)$ for potential *lower* bound, i.e. the selector $A > V_j$;
   2.3. **If** $Hleft(V_j)$ is a non-increasing local maximum, i.e.,
   $$Hleft(V_{j-1}) \leq Hleft(V_j) > Hleft(V_{j+1})$$
   then insert the selector $A \leq V_j$ to *ArrayOfBounds* according to its degree $Hleft(V_j)$, similarly,
   if $Hright(V_j)$ is a non-decreasing local maximum,
   then insert the selector $A > V_j$;
enddo;
3. **For each** possible pair $V_1 < V_2$ of bounds inserted already in *ArrayOfBounds* **do**
   3.1. Calculate the frequencies D_r of the values *V* within the interval $V_1 < V \leq V_2$ for each class C_r ;
   3.2. Calculate the degree $H(V_1, V_2) = Rank(D_1, ..., D_R)$;
   3.3. Insert the selector $V_1 < A_n \leq V_2$ to *ArrayOfBounds* according to its degree $H(V_1,V_2)$;
enddo;

Fig. 6. Algorithm for discretization in CN4

Let us introduce an illustrative example; consider the task with two classes (+ and -) and a single numerical attribute `ww` . Let the training set consist of 7 +ve examples (45, 46, 50, 50, 100, 100, 120) and 5 -ve examples (51, 51, 51, 99, 99). The graph of the entropy (as a user-specified heuristic evaluation function) for the lower and upper bounds is on **Fig. 7**. The promising lower bounds are 45, 50, and 99, the only upper bound was found 50. Consequently, the three inner intervals were generated. As the result, the following array of promising intervals has been produced (starting with the best one):

```
50<ww<=99;      entropy=0.000;  maxfreq=5
ww<=50;         entropy=0.000;  maxfreq=4
ww>99;          entropy=0.000;  maxfreq=3
45<ww<=50;      entropy=0.000;  maxfreq=3
ww>50;          entropy=-0.954; maxfreq=5
45<ww<=99;      entropy=-0.954; maxfreq=5
ww>45;          entropy=-0.994; maxfreq=6
```

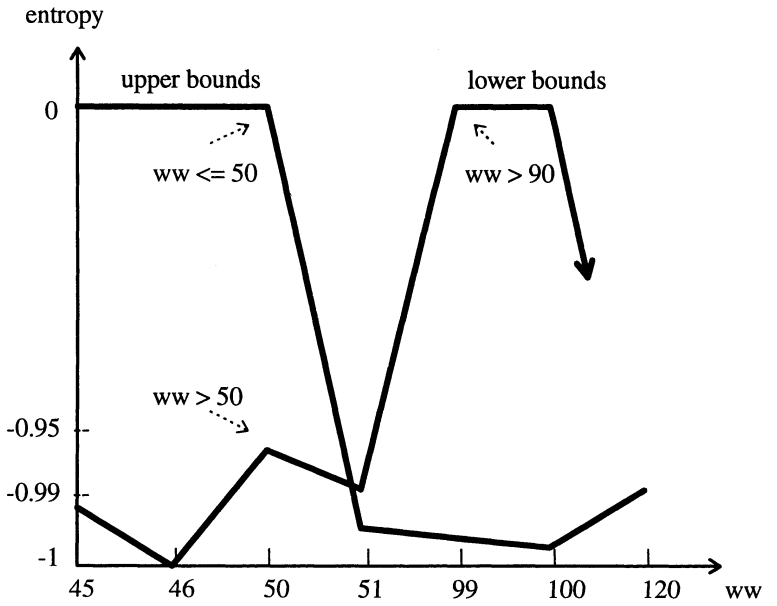

Fig. 7. Local maxima of the entropy and corresponding promising bounds

The inductive algorithm asks the discretization procedure for promising intervals always when it creates new rule conditions and evaluates them. Thus, the above discretization is performed dynamically, 'on-line', always when the training set has been reduced (see step 2 in the **until**-loop of the procedure *ORDERED_COVER*). The other possibility of discretizing numerical attributes for the purposes of symbolic learning algorithms is to perform discretization 'off-line', i.e., by a discretization preprocessor, before the actual inductive algorithm is invoked. The learning algorithm thus processes discretized (categorized) numerical attributes as if they were symbolic ones.

If an interval generated by the above discretization procedures is not genuinely consistent, then processing of a numerical value that occurs quite close to its bound is to be done sensitively, since the bounds generated by noisy data need not be reliable. Therefore, in such cases, the behaviour of the discretization might be improved by *fuzzification*. This enhancement has been added to the newer version of CN4.

In a learning stage of CN4, the fuzzy-discretization procedure provides not only promising intervals but each interval is attached to by its consistency factor *cons* and completeness factor[1] *compl*. A membership function of a fuzzified numerical

[8] Consistency factor expresses the reliability of each subset (interval); it is defined as a relative frequency of examples of the majority class covered by this interval. Completeness factor characterizes the power of the give examples; it is defined as the

interval has the trapezoidal shape indicated in Fig. 8 (the bold line). The original membership function is drawn as dotted line. Here V_1 and V_2 are lower and upper bounds of the interval, respectively. Following [27], the membership values of both bounds equal to 0.5 and the length of both oblique edges is 2ε, where

$$\varepsilon = compl\ (1 - cons)\ (V_2 - V_1)$$

The membership functions of intervals thus overlap. In most cases, however, a numerical value will fall into one function, but values closed to borders will fit into two functions.
Hence, the classification procedure with such fuzzified numerical intervals has to check for each numerical value the membership value for two adjacent intervals, and eventually to add both membership values together. Therefore, the classifier has to go along the entire set of decision rules and check which rules fires. For this reason, the fuzzy-discretization may be utilized only in the unordered mode.

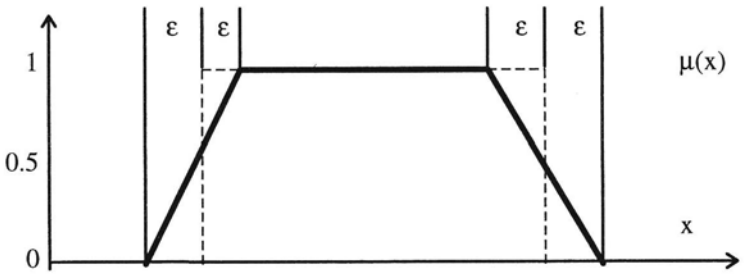

Fig. 8. Characteristic functions for original and fuzzified intervals

5.4 Empirical Results

This section describes the experiments that have been carried out in order to compare the off-line and on-line discretization procedures, both 'genuine' and fuzzy versions. In order to make the comparison of various discretization procedures even more exhibitory, we run KEX, CN4 with the ordered mode, CN4 with the unordered mode, and C4.5, both on original data, discretized data, and fuzzified data. Moreover, CN4 was also utilizing its genuine as well as fuzzy version of discretization.

---

number of majority-class examples covered by the interval over the size of the entire class. For detailes see e.g. [4], [5].

We have been exploiting the following medical databases:

- **Thyreosis** This task of diagnosis of thyroid gland disease has been provided by the Institute of Nuclear Medicine of Inselspital, Bern, Switzerland. The database has been used at the Dept. of Advanced Mathematics, University of Bern, and also in the project CN4. The entire task in fact consists of five subsets, their characteristics are shown in **Table 2**.
- **Onco** The oncological data were used for testing in the Czech Academy of Sciences, Prague, Czech Republic, and also in the project CN4 [6]. The entire task involves in fact two subsets, see **Table 3**.

subset	*# class.*	*frequency of majority class*	*# examp.*	*# attrib.*	*# numer. attrib.*	*aver .# of values per symb attrib.*	*% unknown values*
thyr21	2	72%	269	21	5	5.4	23%
thyr5	2	72%	269	5	2	5.7	29%
thyr7	3	48%	73	7	6	2.0	36%
thyr14	3	44%	72	14	1	2.8	12%
thyr15	3	53%	32	15	1	4.0	4.8%

Table 2. Data Thyreosis

subset	*# class.*	*frequency of majority class*	*# examp.*	*# attrib.*	*# numer. attrib.*	*aver. # of values per symb. attrib.*	*% unknown values*
onco7	3	50%	127	7	6	3.0	0%
onco8	4	47%	121	8	7	3.0	0%

Table 3. Data Onco

Table 4 exhibits both the classification accuracy and the numbers of decision rules (in case of C4.5, nodes of decision trees) induced, which may interpreted a simple measures for sizes of concept descriptions. In each cell, the first number represents the classification accuracy, the latter the number of decision rules induced. Here CN4 (D) means that the genuine on-line discretization is applied if necessary; (F) is for the on-line fuzzy-discretization of CN4. For each dataset we show the results on original data (O), discretized data (D) and fuzzified data (F).

database	*KEX*		*CN4 ord. (D)*		*CN4 unord. (F)*		*C4.5*	
	accur.	*#rules*	*accur.*	*# rules*	*accur.*	*# rules*	*accur.*	*size*
thyr21 (O)	--		97%	20	92%	15	92%	15
thyr21 (D)	90%	18	97%	17	--		93%	18
thyr21 (F)	91%	10	95%	10	--		--	
thyr5 (O)	--		91%	10	91%	12	91%	12
thyr5 (D)	89%	12	89%	11	--		90%	15
thyr5 (F)	89%	8	90%	7	--		--	
thyr7 (O)	--		96%	5	88%	6	89%	7
thyr7 (D)	95%	54	93%	5	--		85%	4
thyr7 (F)	96%	42	90%	4	--		--	
thyr14 (O)	--		93%	14	79%	10	75%	19
thyr14 (D)	73%	84	85%	9	--		72%	17
thyr14 (F)	69%	33	78%	8	--		--	
thyr15 (O)	--		96%	6	93%	6	83%	4
thyr15 (D)	86%	15	81%	3	--		81%	4
thyr15 (F)	87%	15	82%	3	--		--	
onco7 (O)	--		98%	23	84%	20	76%	27
onco7 (D)	69%	51	77%	15	--		73%	29
onco7 (F)	61%	33	64%	5	--		--	
onco8 (O)	--		93%	18	84%	19	75%	34
onco8 (D)	57%	32	66%	13	--		69%	41
onco8 (F)	60%	12	57%	2	--		--	

Table 4. Classification results

5.5 Analysis

We came to the following conclusion when having analyzed the above experiments applied to the above seven datasets:

• When comparing the *on-line* discretization executed by the inductive algorithms (CN4 and C4.5 in this project) and the *off-line* discretization done by the KEX preprocessor, one may find that the own (on-line) discretization works always better. The on-line discretization reflects in fact the current distribution of attribute values that changes when new rules (or decision nodes) are generated. Moreover, the *off-line fuzzy*-discretization is substantially worse (when comparing the classification accuracy) than the *off-line genuine* discretization.

• Since KEX does not remove covered examples from training dataset, the induced knowledge base usually comprises more rules than those derived by other learning algorithms. However, this redundancy corresponds to the compositional approach (combining weights of more applicable rules) and can be more powerful in situations when new objects (examples) portray incomplete descriptions.
• KEX creates substantially smaller knowledge bases using fuzzy-discretization procedure than using the genuine discretization (without fuzzy intervals). This can sometimes result in small decrease of classification accuracy.
• The comparison of the on-line *genuine* and *fuzzy* discretization of CN4 is not so obvious from the first view, since one has to realize that fuzzification can be carried out only in the unordered mode, and - as many experiments revealed - the unordered mode is always about 5% worse in performance than the ordered one. Realizing this fact, we may conclude that fuzzy-discretization is slightly better (1 to 2% in classification accuracy) than the genuine one.

References

1. P. Berka, I. Bruha (1995): Various discretizing procedures of numerical attributes: Empirical comparisons. *European Conf. on Machine Learning, Workshop Statistics, Machine Learning, and Knowledge Discovery in Databases*, Heraklion, Crete, 136-141.
2. P. Berka, J. Ivanek (1994): Automated knowledge acquisition for PROSPECTOR-like expert systems. *ECML-94*, Springer-Verlag, 339-342
3. P.B. Brazdil, I. Bruha (1992): A note on processing missing attribute values: a modified technique. *Workshop on Machine learning, Canadian Conf. AI*, Vancouver.
4. D. Biggs, B. de Ville, E. Suen (1991): A method of choosing multiway partitions for classification and decision trees. *J. Applied Statistics*, **18**, 1, 49-62.
5. I. Bruha, S. Kockova (1993): Quality of decision rules: empirical and statistical approaches. *Informatica*, **17**, 233-243.
6. I. Bruha, S. Kockova (1993): A covering learning algorithm for cost-sensitive and noisy environments. *European Conf. Machine Learning, Workshop on Learning Robots*, Vienna.
7. I. Bruha (1996): Quality of decision rules: Definitions and classification schemes for multiple rules. In: G. Nakhaeizadeh, C.C. Taylor (eds.): *Machine Learning and Statistics: The Interface*. John Wiley, 107-131.
8. I. Bruha, S. Kockova (1994): *A support for decision making: Cost-sensitive learning system*. Artificial Intelligence in Medicine, **6**, 67-82.
9. J. Catlett (1991): On changing continuous attributes into ordered discrete attributes. *EWSL-91*, Porto, Springer-Verlag, 164-178.

10. B. Cestnik, I. Kononenko, I. Bratko (1988): Assistant 86: A knowledge-elicitation tool for sophisticated users. In: I. Bratko, N. Lavrac (eds.): *Progress in machine learning. Proc. EWSL'88*, Sigma Press, 31-46.
11. P. Clark, R. Boswell (1991): Rule Induction with CN2: Some Recent Improvements. *EWSL-91*, Porto.
12. J.G Carbonell, R.S. Michalski, T.M. Mitchell (1983): *An overview of machine learning*. In [19]
13. P. Clark, T. Niblett (1989): The CN2 induction algorithm. *Machine Learning*, **3**, 261-283
14. R.O. Duda, J.E. Gasching (1979): Model Design in the PROSPECTOR Consultant System for Mineral Exploration. In: Michie, D. (ed.), *Expert Systems in the Micro Electronic Age*, Edinburgh University Press, UK.
15. P. Hajek (1985): Combining Functions for Certainty Factors in Consulting Systems. *Int.J. Man-Machine Studies*, **22**, 59-76.
16. C. Lee, D. Shin (1994): A context-sensitive discretization of numeric attributes for classification learning. *ECAI-94*, Amsterdam, John Wiley, 428-432.
17. R.S. Michalski (1980): Pattern recognition as rule-guided inductive inference. *IEEE Trans. PAMI-2*, **4**, 349-361.
18. R.S. Michalski et al. (1986): The multi-purpose incremental learning system AQ15 and its testing application to three medical domains. *Proc. 5th AAAI*, 1041-5.
19. R.S. Michalski, J.G Carbonell, T.M. Mitchell (eds.) (1983): *Machine learning: An artificial intelligence approach, I.* Tioga Publ.
20. M. Nunez (1988): *Economic induction: a case study*. EWSL'88, Glasgow, 139-145.
21. J.R. Quinlan (1986): Induction of decision trees. *Machine Learning*, **1**, 81-106.
22. J.R. Quinlan (1987): Simplifying decision trees. *Internl. J. on Man-machine Studies*, **27**, 221-234.
23. J.R. Quinlan (1989): Unknown attribute values in ID3. *Intrn'l Conf. ML*, 164-168.
24. J.R. Quinlan (1994): *C4.5: Programs for machine learning*. Morgan Kaufmann Publ.
25. H.A. Simon (1983): *Why should machines learn?* In [19].
26. M. Tan, J.C. Schlimmer (1990): Two case studies in cost-sensitive concept acquisition. *8th Conf. AI.*
27. J. Zeidler, M. Schlosser (1995): Fuzzy handling of continuous-valued attributes in decision trees. *8th European Conf. on Machine Learning, Workshop Statistics, Machine Learning, and Knowledge Discovery in Databases*, Heraklion, Crete, 41-46.