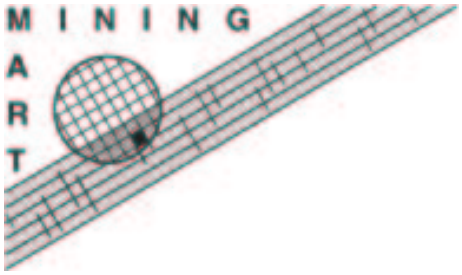

Enabling End-User Datawarehouse Mining
Contract No. IST-1999-11993
Deliverable No. D16.1

Discretization and Grouping operators

Petr Berka[1]

[1] University of Economics
Laboratory for Intelligent Systems
CZ-13065 Prague, Czech Republic
berka@vse.cz

1. December 2002

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, discretization into given number of categories with the same cardinality (equifrequent discretization), or categorization based on mean and standard deviation. All these approaches are "class-blind" (or unsupervised), since they deal only with the discretized attribute (Fig. 1).

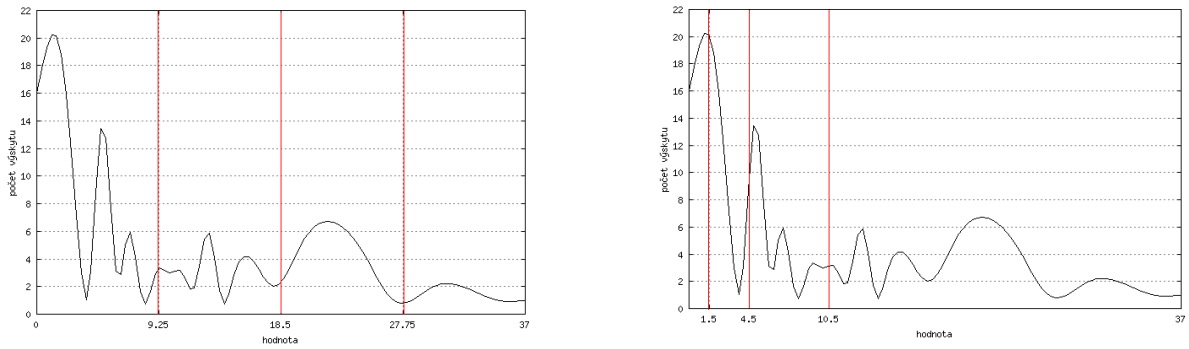

Fig. 1 Equidistant and equifrequent discretization

In the framework of machine learning, the fact that the examples (objects) belong to different classes is taken into account. So the discretization algorithms are "class sensitive" (or supervised) - see Fig. 2. A variety of algorithms has been developed in the last decade. Most newer versions of machine learning algorithms 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 [Catlett, 1991], [Fayyad, Irani, 1993]. 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 [Biggs et al., 1991]. Another interesting approach to discretization can be found in [Lee, Shin, 1994]. As pointed out by Elomaa and Rousu [Elomaa, Rousu, 2002], most algorithms are guided by the criterion of minimal training error – they search for intervals with one dominant class. The differences between the algortihms are in

- integration with machine learning algorithms (integrated or stand-alone as preprocessing tool),
- search strategy (top-down by splitting intervals or bottom-up by merging intervals),
- the impurity measure for evaluating potential intervals (entropy, information gain, χ^2 test, minimum classification error),
- number of intervals (binarization or creating more intervals),
- stopping criterion (number of intervals, frequency of intervals),
- type of intervals (most algorithms create crisp intervals, algorithms for creating fuzzy intervals can be found e.g. in [Bruha, Berka, 2000] or [Peng, Flach, 2001]),
- number of processed attributes (most algorithms are univariate and consider only one numeric attribute a time, a multivariate algorithm is described e.g. in [Bay, 2000]).

For implementation within the MiningMart project, we choose discretization algorithms that are univariate, create crisp intervals, perform merging of intervals and have different stopping criterions.

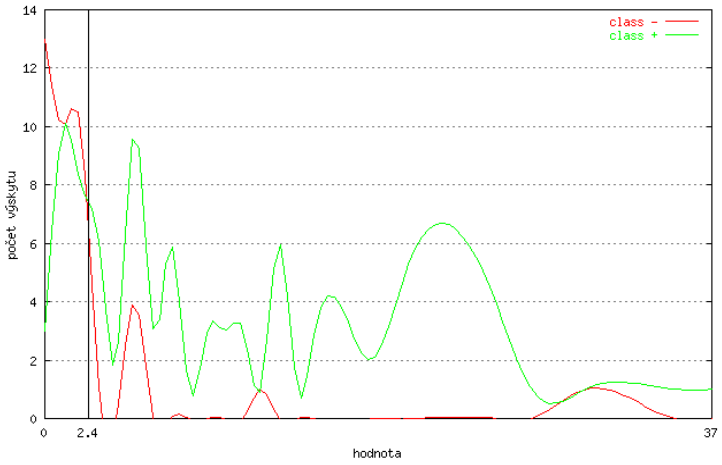

Fig. 2 Class sensitive discretization

Grouping values of a nominal attribute becomes important if the number of these values is too large (e.g. hundreds of ZIP codes or profession codes). To deal with each value separately can bring problems both during computation (e.g. branching a node) and interpretation. While discretization is a standard preprocessing operation, grouping of values of categorial attribute is less common. Some examples of grouping algorithms can be found in [Biggs et al., 1991] or [Berka, Bruha, 1998].

For implementation within the MiningMart project, we choose grouping algorithms that are cunterpart to the discretization ones.

2 Discretization operators

The idea of discretization is to divide the range of a numeric or ordinal attribute (i.e.attribute with values encoded using numbers) into intervals according to given CutPoints. The CutPoints can be given directly by the user (*UserDefinedDiscretization*), can be computed from the request to create equidistant intervals (*EquidistantDiscretizationGivenWidth*, *EquidistantDiscretizationgivenNoOfInt*), from the request to create equifrequent intervals (*EquifrequentDiscretizationGivenCardinality*, *EquifrequentDiscretizationGivenNoOfInt*), or can be computed from the request to create intervals that will help to classify examples (rows) into classes given in ClassAttribute (*ErrorBasedDiscretizationGivenMinCard*, *ErrorBasedDiscretizationGivenNoOfInt*). To assign the $CutPoint_k$ to one of the intervals, two types of intervals are assumed to be created: ClosedToLeft - i.e. [$CutPoint_{k-1}$, $CutPoint_k$), [$CutPoint_k$, $CutPoint_{k+1}$), and ClosedToRight ($CutPoint_{k-1}$, $CutPoint_k$] ($CutPoint_k$, $CutPoint_{k+1}$]. The Labels for the intervals can be given by the user or can be created automatically.

If some out-of-range values (i.e. values not in the interval $[u_0, v_0]$ = [TheTargetAttribute.min, TheTargetAttribute.max] which is given in the COLSTATIST1_T) will occur in the data, the discretization should deal with such situation. There are basically three possibilities: (1) create new `boundary` intervals i.e. intervals $(-\infty, u_0)$ and (v_0, ∞), (2) report out-of-range error, or (3) merge boundary intervals with first resp. last interval – i.e. create intervals $(-\infty, CutPoint_1)$ and $(CutPoint_{LAST}, \infty)$. In the specification of operators, I assume the third option (this option corresponds also to the case that no out-of-range values can occur).

The operators described bellow share a common part that creates a specification of intervals (SQL statement) for TheTargetAttribute according to CutPoints $\{u_k\}$ and parameter ClosedTo.

Discretize
if Label=NULL then
for k=1 to kmax $Label_k$= k
$Label_{kmax+1}$= kmax+1
case ClosedTo
'LEFT': Create View T as Select *, $Label_1$ | $T_0.A < u_1$ $\{Label_k \mid T_0.A \in [u_k, u_{k+1})\}_{k=1+1 \text{ to } kmax}$ $Label_{kmax+1}$ | $T_0.A \geq u_{kmax}$ as A' From T_0
'RIGHT': Create View T as Select *, $Label_1$ | $T_0.A \leq u_1$ $\{Label_k \mid T_0.A \in (u_k, u_{k+1}]\}_{k=1+1 \text{ to } kmax}$ $Label_{kmax+1}$ | $T_0.A > u_{kmax}$ as A' From T_0

2.1 Manual discretization operators

Manual discretization operators use only the information about discretized attribute itself (min, max, frequencies of distinct values ….). Three basic types of operators will be implemented: (1) *equidistant* operators divide the range of the numeric attribute into intervals with the same width, (2) *equifrequent* operators divide the range of the numeric attribute into intervals with the same number of examples, and (3) *user defined* operator divides the range of the numeric attribute into intervals according to cutpoints given by the user. Remember, that *k* cutpoints will create *k+1* intervals.

2.1.1 EquidistantDiscretizationGivenWidth

This operator divides the range of TheTargetAttribute into intervals with given width IntervalWidth starting at StartPoint. The first and the last interval cover also the out of range values[1].

- **Operator**
  - *name* EquidistantDiscretizationGivenWidth
  - *loopable* – yes
  - *multistepable* – no
  - *manual* – yes

- **Parameters**

Name	minarg	maxarg	IO	type
TheInputConcept	1	1	IN	CON
TheTargetAttribute	1	1	IN	BA
StartPoint	0	1	IN	V
IntervalWidth	1	1	IN	V
ClosedTo	1	1	IN	V
TheOutputAttribute	1	1	OUT	BA

- **Constraints**
  - TheTargetAttribute is in TheInputConcept
  - TheOutputAttribute is in TheInputConcept
  - TheTargetAttribute is NUMERIC
  - ClosedTo one-of 'LEFT,RIGHT'
  - IntervalWidth is NUMERIC
  - IntervalWidth > 0
  - StartPoint is NUMERIC

- **Conditions**

- **Assertions**
  - TheOutputAttribute is CATEGORIAL

[1] It would be nice to have a possibility to add this (and all the others) comments to the operators (like the DOCU field in tables OP_PARAMS_T, OP_CONSTR_T, OP_COND_T).

- **OPChecks**
  - IntervalWidth < (TheTargetAttribute.max – TheTargetAttribute.min)
  - StartPoint ≥ TheTargetAttribute.min
  - StartPoint ≤ TheTargetAttribute.max

Algorithm

u_0 := TheTargetAttribute.min, v_0 := TheTargetAttribute.max
Find CutPoints $\{u_k\}$: $k \geq 1$, $u_k < u_{k+1}$, $u_{kmax+1} \geq v_0$, $u_{k+1} - u_k$ = IntWidth
Discretize

Where the Find_CutPoints procedure can be described as follows:

Find_Cutpoints

k=1
If StartPoint == NULL then $CutP_1 = u_0$ else $CutP_1$ = StartPoint
repeat until $CutP_k > v_0$
    $CutP_{k+1} = CutP_k + IntWidth$
    k=k+1
k=k-1

2.1.2 EquidistantDiscretizationGivenNoOfIntervals

This operator divides the range of TheTargetAttribute into given number of intervals NoOfIntervals with the same width. The first and the last interval cover also the out of range values. Values of TheOutputAattribute can be specified in Label.

- **Operator**
  - *name* EquidistantDiscretizationGivenNoOfIntervals
  - *loopable* – yes
  - *multistepable* - no
  - *manual* – yes

- **Parameters**

Name	minarg	maxarg	IO	type
TheInputConcept	1	1	IN	CON
TheTargetAttribute	1	1	IN	BA
NoOfIntervals	1	1	IN	V
ClosedTo	1	1	IN	V
Label	0	NULL	IN	V
TheOutputAttribute	1	1	OUT	BA

- **Constraints**
  - TheTargetAttribute is in TheInputConcept
  - TheOutputAttribute is in TheInputConcept
  - TheTargetAttribute is NUMERIC
  - ClosedTo one-of ‘LEFT,RIGHT’
  - NoOfIntervals is NUMERIC
  - NoOfIntervals > 1
  - Label_ is CATEGORIAL

- **Conditions**

- **Assertions**
  - TheOutputAttribute is CATEGORIAL

- **OPChecks**
  - NoOfIntervals < number of unique values of TheTargetAttribute

Algorithm

u_0 := TheTargetAttribute.min, v_0 := TheTargetAttribute.max
IntWidth := (v_0 - u_0) / NoOfInt
Find CutPoints $\{u_k\}$: $k \geq 1$, $u_k < u_{k+1}$, $u_{kmax+1} = v_0$, $u_{k+1} - u_k$ = IntWidth
Discretize

Where the Find_CutPoints procedure can be described as follows:

Find_Cutpoints

k=1
$CutP_1 = u_0$
repeat until $CutP_k > v_0$
    $CutP_{k+1} = CutP_k$ + IntWidth
    k=k+1
k=k-1

2.1.3 EquifrequentDiscretizationGivenCardinality

This operator divides the range of TheTargetAttribute into intervals with given cardinality Cardinality (number of examples with values within the interval). The first and the last interval cover also the out of range values. Values of TheOutputAattribute can be specified in Label (this makes sense only if CardinalityType is relative.

- **Operator**
  - *name* EquifrequentDiscretizationGivenCardinality
  - *loopable* – yes
  - *multistepable* – no
  - *manual* – yes

- **Parameters**

Name	minarg	maxarg	IO	type
TheInputConcept	1	1	IN	CON
TheTargetAttribute	1	1	IN	BA
Cardinality	1	1	IN	V
CardinalityType	1	1	IN	V
ClosedTo	1	1	IN	V
TheOutputAttribute	1	1	OUT	BA

- **Constraints**
  - TheTargetAttribute is in TheInputConcept
  - TheOutputAttribute is in TheInputConcept
  - TheTargetAttribute is NUMERIC
  - ClosedTo one-of 'LEFT,RIGHT'
  - Cardinality is NUMERIC
  - CardinalityType one-of 'ABSOLUTE,RELATIVE'
  - Cardinality >
- **Conditions**
  - TheTargetAttribute NOT_NULL

- **Assertions**
  - TheOutputAttribute is CATGORIAL

- **OPChecks**
  - If CardinalityType is RELATIVE, then Cardinality < 0.5
  - If CardinalityType is ABSOLUTE then Cardinality < no_of_rows_in_the_table / 2 (TheInputConcept.allNumber / 2 ??)

Algorithm

u_0 := TheTargetAttribute.min, v_0 := TheTargetAttribute.max
if IntCardType = 'ABSOLUTE' then IntCardA := IntCard, else IntCardA := TheInputConcept.alNumber * IntCard
Case ClosedTo
'LEFT': Find CutPoints $\{u_k\}$: $k \geq 1$, $u_k < u_{k+1}$, $u_{kmax+1} = v_0$, card($[u_k, u_{k+1})$) = IntCardA
'RIGHT': Find CutPoints $\{u_k\}$: $k \geq 1$, $u_k < u_{k+1}$, $u_{kmax+1} = v_0$, card($(u_k, u_{k+1}]$) = IntCardA
Discretize

Where the Find_CutPoints procedure can be described as follows[2]:

Find_CutPoints

sort the values of the numeric attribute in ascending order;
for each value a_i compute the frequency $n(a_i)$ in the data (available in COLSTAT?)
k=1: i=0: MaxDif=0
$CutP_1 = u_0$ //$u_0 - 0.001u_0$
repeat until $a_i > v_0$
count = 0
repeat until count > IntCardA
count = count + $n(a_i)$
i = i+1
k = k + 1: i=i-1
if count – IntCardA > IntCardA – (count – $n(a_i)$)
then MaxDif = max(MaxDif, IntCardA – (count – $n(a_i)$))
i = i-1
$CutP_k = (a_i + a_{i+1})/2$
else MaxDif = max(MaxDif, count – IntCardA)
$CutP_k = (a_i + a_{i+1})/2$
if (MaxDif / IntCardA) > 0.125 then print (error message)

2.1.4 EquifrequentDiscretizationGivenNoOfIntervals

This operator divides the range of TheTargetAttribute into given number of intervals NoOfIntervals. The intervals have the same cardinality (number of examples with values within the interval). The first and the last interval cover also the out of range values. Values of TheOutputAattribute can be specified in Label.

- **Operator**
  - *name* EquifrequentDiscretizationGivenNoOfIntervals
  - *loopable* – yes
  - *multistepable* - no
  - *manual* – yes

- **Parameters**

[2] This is a locally optimal method. A globally suboptimal method will require to keep track of all possible variants (two possibilities for each CutPoint), e.g. for k CutPoints 2^k different discretizations.

Name	minarg	maxarg	IO	type
TheInputConcept	1	1	IN	CON
TheTargetAttribute	1	1	IN	BA
NoOfIntervals	1	1	IN	V
ClosedTo	1	1	IN	V
Label	0	NULL	IN	V
TheOutputAttribute	1	1	OUT	BA

- **Constraints**
  - TheTargetAttribute is in TheInputConcept
  - TheOutputAttribute is in TheInputConcept
  - TheTargetAttribute is NUMERIC
  - ClosedTo one-of 'LEFT,RIGHT'
  - NoOfIntervals is NUMERIC
  - NoOfIntervals > 1
  - Label_ is CATEGORIAL

- **Conditions**
  - TheTargetAttribute NOT_NULL

- **Assertions**
  - TheOutputAttribute is CATEGORIAL

- **OPChecks**
  - NoOfIntervals < number of unique values of TheTargetAttribute

Algorithm

u_0 := TheTargetAttribute.min, v_0 := TheTargetAttribute.max
IntCardA := TheInputConcept.allNumber / NoOfInt
Case ClosedTo
'LEFT': Find CutPoints $\{u_k\}$: $k \geq 1$, $u_k < u_{k+1}$, $u_{kmax+1} = v_0$, card($[u_k, u_{k+1})$) = IntCardA
'RIGHT': Find CutPoints $\{u_k\}$: $k \geq 1$, $u_k < u_{k+1}$, $u_{kmax+1} = v_0$, card($(u_k, u_{k+1}]$) = IntCardA
Discretize

Where the Find_CutPoints procedure can be described as follows:

Find_Cutpoints

sort the values of the numeric attribute in ascending order;
for each value a_i compute the frequency $n(a_i)$ in the data (available in COLSTAT?)
k=1: i=0: MaxDif=0
$CutP_1 = u_0$ //$u_0 - 0.001u_0$
repeat until $a_i > v_0$
    count = 0
    repeat until count > IntCardA
        count = count + $n(a_i)$
        i = i+1
    k = k + 1
    if count – IntCardA > IntCardA – (count – $n(a_i)$)
        then MaxDif = max(MaxDif, IntCardA – (count – $n(a_i)$))
            i = i-1
            $u_k = (a_i + a_{i+1})/2$
        else MaxDif = max(MaxDif, count – IntCardA)
            $u_k = (a_i + a_{i+1})/2$
if (MaxDif / IntCardA) > 0.125 then print (error message)

2.1.5 UserDefinedDiscretization

This operator divides the range of TheTargetAttribute into intervals according to user given cutpoints TheCutpoints. Values of TheOutputAattribute can be specified in Label.

- **Operator**
  - *name* UserDefinedDiscretization
  - *loopable* – yes
  - *multistepable* - no
  - *manual* – yes

- **Parameters**

Name	minarg	maxarg	IO	type
TheInputConcept	1	1	IN	CON
TheTargetAttribute	1	1	IN	BA
ClosedTo	1	1	IN	V
TheCutpoints	1	NULL	IN	V
Label	0	NULL	IN	V
TheOutputAttribute	1	1	OUT	BA

- **Constraints**
  - TheTargetAttribute is in TheInputConcept
  - TheOutputAttribute is in TheInputConcept
  - TheTargetAttribute is NUMERIC
  - ClosedTo one-of 'LEFT,RIGHT'
  - TheCutpoints_ is NUMERIC
  - TheCutpointsK+1 > TheCutpointsK
  - Label_ is categorial
- **Conditions**

- **Assertions**
  - TheOutputAttribute is CATEGORIAL
- **OPChecks**
  - TheCutpoints_ < TheTargetAttribute.max
  - TheCutpoints_ > TheTargetAttribute.min

Algorithm

Discretize

2.2 ML discretization operators

The algorithms proposed for implementation in MiningMart discretize a single attribute into crisp intervals, perform bottom-up search, can create more intervals and are based on minimum classification error criterion. The training data are taken as random sample of given size (default size is 10 000).

2.2.1 ImplicitErrorBasedDiscretization

This operator divides the range of TheTargetAttribute into intervals by merging subsequent values with the same majority class (or classes) given in TheClassAttribute. The resulting intervals minimize the classification error. If FullMerge is set to yes, then an interval with two or more majority classes is merged with its neighbour, if both intervals share the same majority class.

- **Operator**
  - *name* ImplicitErrorBasedDiscretization
  - *loopable* – yes
  - *multistepable* - no
  - *manual* – no

- **Parameters**

Name	minarg	maxarg	IO	type
TheInputConcept	1	1	IN	CON
TheClassAttribute	1	1	IN	BA
TheTargetAttribute	1	1	IN	BA
FullMerge	1	1	IN	V
ClosedTo	1	1	IN	V
SampleSize	0	1	IN	V
TheOutputAttribute	1	1	OUT	BA

- **Constraints**
  - TheTargetAttribute is in TheInputConcept
  - TheOutputAttribute is in TheInputConcept
  - TheTargetAttribute is NUMERIC
  - TheClassAttribute is in TheInputConcept
  - TheClassAttribute is CATEGORIAL
  - FullMerge one-of 'YES,NO'
  - ClosedTo one-of 'LEFT,RIGHT'
  - SampleSize is NUMERIC
  - SampleSize > 0
- **Conditions**
  - TheTargetAttribute NOT_NULL
  - TheClassAttribute NOT_NULL
- **Assertions**
  - TheOutputAttribute is CATEGORIAL

Algorithm

Create stratified sample of TheInputConcept
u_0 := TheTargetAttribute.min, v_0 := TheTargetAttribute.max
Find_CutPoints $\{u_k\}$: $k \geq 1$, $u_k < u_{k+1}$, $u_{kmax+1} = v_0$, card((u_k, u_{k+1})) $\geq$ MinIntCardA
Discretize

Where the Find_CutPoints procedure can be described as follows:

Find_CutPoints

1. sort the values of the numeric attribute in ascending order;
2. for each value a_i
   2.1. $LBound_i := (a_{i-1}+a_i)/2$, $UBound_i := (a_i+a_{i+1})/2$
   2.2. count the frequencies of each class and store the max frequency into *maxfreq*;
   2.3. assign class label using procedure ASSIGN;
3. create intervals using procedure INTERVAL;

ASSIGN: //unique label corresponds to each combination of most frequent classes that can occur
label =0
for k = 1 to No_of_classes
  if $n_k(a) = maxfreq$ then $label = label + 2^k$ // $n_k(a)$ is #.examples with value *a* belonging to class *k*

INTERVAL:
1. create interval $INT = [LBound, UBound]$ for a sequence of values with the same class label; *//first pass*
2. if FullMerge='YES' then
   2.1 for each interval INT_i *//second pass*
   2.1.1 if INT_i has no single majority class //i.e. *label equals* 2^x
   then create interval $INT_{i-1} \cup INT_i$ or INT_i or $INT_i \cup INT_{i+1}$ that will not increase the min. error
   if both merges are possible, then prefer merge with single majority class interval, if both merges are possible then prefer merge with interval with smaller frequency
// a merge will not increase the error, if both intervals share same majority class, i.e. *labelA bitAND labelB > 0*

Min. error for an interval is computed as $n(Int) - n_k(Int)$;
where $n(Int)$ is the no. of examples having the value in the interval
$n_k(Int)$ is the no. of examples of the majority class having the value in the interval

2.2.2 ErrorBasedDiscretizationGivenMinCardinality

This operator divides the range of TheTargetAttribute into intervals with cardinality greater or equal to MinCardinality. The numeric attribute is divided into intervals with respect to TheClassAttribute, but unlike the implicit discretization, intervals with single majority class are further merged if they do not have the required cardinality. This will increase the classification error.

- **Operator**
  - *name* ErrorBasedDiscretizationGivenMinCardinality
  - *loopable* – yes
  - *multistepable* - no
  - *manual* – no

- **Parameters**

Name	minarg	maxarg	IO	type
TheInputConcept	1	1	IN	CON
TheClassAttribute	1	1	IN	BA
TheTargetAttribute	1	1	IN	BA
MinCardinality	1	1	IN	V
MinCardinalityType	1	1	IN	V
ClosedTo	1	1	IN	V
SampleSize	0	1	IN	V
TheOutputAttribute	1	1	OUT	BA

- **Constraints**
  - TheTargetAttribute is in TheInputConcept
  - TheOutputAttribute is in TheInputConcept
  - TheTargetAttribute is NUMERIC
  - TheClassAttribute is in TheInputConcept
  - TheClassAttribute is CATEGORIAL
  - ClosedTo one-of 'LEFT,RIGHT'
  - MinCardinality is NUMERIC
  - MinCardinalityType one-of 'ABSOLUTE,RELATIVE'
  - MinCardinality > 0 SampleSize is NUMERIC
  - SampleSize > 0
- **Conditions**
  - TheTargetAttribute NOT_NULL
  - TheClassAttribute NOT_NULL

- **Assertions**
  - TheOutputAttribute is CATEGORIAL

- **OPChecks**
  - If MinCardinalityType is RELATIVE, then MinCardinality < 0.5
  - If MinCardinalityType is ABSOLUTE then MinCardinality < no_of_rows_in_the_table / 2

Algorithm

Create stratified sample of TheInputConcept
u_0 := TheTargetAttribute.min, v_0 := TheTargetAttribute.max
if MinCardinalityType = 'ABSOLUTE' then MinCardA := MinCardinality, else MinCardA := TheInputConcept.alNumber * MinCardinality
Find_CutPoints $\{u_k\}$: $k \geq 1$, $u_k < u_{k+1}$, $u_{kmax+1} = v_0$, $card((u_k, u_{k+1})) \geq$ MinIntCardA
Discretize

Where the Find_CutPoints procedure can be described as follows:

Find_CutPoints

1. sort the values of the numeric attribute in ascending order;
2. for each value a_i
   2.1. $LBound_i := (a_{i-1}+a_i)/2$, $UBound_i := (a_i+a_{i+1})/2$
   2.2. count the frequencies of each class and store the max frequency into *maxfreq*;
   2.3. assign class label using procedure ASSIGN;
3. create intervals using procedure INTERVAL;

ASSIGN: //unique label corresponds to each combination of most frequent classes that can occur
label =0
for k = 1 to No_of_classes
  if $n_k(a) = maxfreq$ then label = label + 2^k // $n_k(a)$ is #.examples with value *a* belonging to class *k*

INTERVAL:
1. create interval *INT = [LBound, UBound]* for a sequence of values with the same class label; //*first*
2. do while the frequencies of intervals are smaller then *MinIntCardA* //*second*
   2.1 take the interval with smallest frequency as INT_i
   2.2 create either interval $INT_{i-1} \cup INT_i$ or $INT_i \cup INT_{i+1}$ according to min. error
      if both merges are possible, then prefer merge with single majority class interval, if both merges are possible then prefer merge with interval with smaller frequency
      // a merge will not increase the error, if both intervals share same majority class, i.e. *labelA bitAND labelB > 0*
   2.3. update frequency of the new interval

2.2.3 ErrorBasedDiscretizationGivenNoOfInt

This operator divides the range of TheTargetAttribute into at most NoOfIntervals intervals. The numeric attribute is divided into intervals with respect to TheClassAttribute, but unlike the implicit discretization, if the number of interval exceeds NoOfInt, intervals are further merged. This will increase the classification error. Values of TheOutputAattribute can be specified in Label.

- **Operator**
  - *name* ErrorBasedDiscretizationGivenNoOfInt
  - *loopable* – yes
  - *multistepable* - no
  - *manual* – no

- **Parameters**

Name	minarg	maxarg	IO	type
TheInputConcept	1	1	IN	CON
TheClassAttribute	1	1	IN	BA
TheTargetAttribute	1	1	IN	BA
NoOfIntervals	1	1	IN	V
ClosedTo	1	1	IN	V
SampleSize	0	1	IN	V
Label	0	NULL	IN	V
TheOutputAttribute	1	1	OUT	BA

- **Constraints**
  - TheTargetAttribute is in TheInputConcept
  - TheOutputAttribute is in TheInputConcept
  - TheTargetAttribute is NUMERIC
  - TheClassAttribute is in TheInputConcept
  - TheClassAttribute is CATEGORIAL
  - ClosedTo one-of 'LEFT,RIGHT'
  - NoOfIntervals is NUMERIC

- NoOfIntervals > 1
- SampleSize is NUMERIC
- SampleSize > 0
- Label_ is CATEGORIAL

- **Conditions**
  - TheTargetAttribute NOT_NULL
  - TheClassAttribute NOT_NULL
- **Assertions**
  - TheOutputAttribute is CATEGORIAL
- **OPChecks**
  - NoOfIntervals < number of unique values of TheTargetAttribute

Algorithm

Create stratified sample of TheInputConcept
u_0 := TheTargetAttribute.min, v_0 := TheTargetAttribute.max
Find CutPoints $\{u_k\}$: $k \geq 1$, $u_k < u_{k+1}$, $u_{kmax+1} = v_0$, k_{max} = NoOfInt
Discretize

Where the Find_CutPoints procedure can be described as follows:

Find_CutPoints

1. sort the values of the numeric attribute in ascending order;
2. for each value a_i
   2.1. $LBound_i := (a_{i-1}+a_i)/2$, $UBound_i := (a_i+a_{i+1})/2$
   2.2. count the frequencies of each class and store the max frequency into *maxfreq*;
   2.3. assign class label using procedure ASSIGN;
3. create intervals using procedure INTERVAL;

ASSIGN: //unique label corresponds to each combination of most frequent classes that can occur
label =0
for k = 1 to No_of_classes
  if $n_k(a) = maxfreq$ then label = label + 2^k // $n_k(a)$ is #.examples with value *a* belonging to class *k*

INTERVAL:
1. create interval $INT = [LBound, UBound]$ for a sequence of values with the same class label; *//first*
2. for each interval INT_i *//second*
   2.1 if INT_i has no single majority class //i.e. *label bitAND* $2^{No\_of\_classes} - 1 > 1$
      2.1.1 then create either interval $INT_{i-1} \cup INT_i$ or INT_i or $INT_i \cup INT_{i+1}$ according to min. error
      2.1.2 if *NoOfInt* is reached then terminate
3. do while the no. of intervals is greater then *NoOfInt* *//third*
   take the interval with smallest frequency as INT_i
   create either interval $INT_{i-1} \cup INT_i$ or $INT_i \cup INT_{i+1}$ according to min. error

3 Grouping operators

The idea of grouping is to merge values of a categorial attribute to have less but more frequent values. The values of TheTargetAttribute that will belong to a group can be given directly by the user (*UserDefinedGrouping*), can be found from the request to have a minimal cardinality (no of examples – rows) of the groups (*GroupingGivenMinCardinality*), can be found from the request to have a given number of groups[3] (*GroupingGivenNoOfGroups*), or can be found from the request to have groups that that will help to classify examples (rows) into classes given in ClassAttribute. The Labels for the groups can be given by the user or can be created from the original values of TheTargetAttribute within a group.

[3] Since there is no ordering between the values of a categorial attribute, an equivalent to equidistant discretization makes no sense.

As for discretization, there is again a common part of all operators that creates the SQL statement for defining the groups:

Group
if Label=NULL then $Label_k$='G ' + k
Create View T as Select *, {$Label_k$ | T_0.A ∈ $\{u_{k,i}\}_i\}_k$ as A' From T_0

3.1 Manual grouping operators

The proposed manual grouping operators are a counterpart to equifrequent and user defined discretization operators.

3.1.1 GroupingGivenMinCardinality

This operator groups values of TheTargetAttribute by iteratively merging in each step two groups with the lowest frequencies until all groups have the cardinality (number of examples with values within the interval) at least MinCardinality. The algorithm has been inspired by hierarchical clustering.

- **Operator**
  - *name* GroupingGivenMinCardinality
  - *loopable* – yes
  - *multistepable* – no
  - *manual* – yes

- **Parameters**

Name	minarg	maxarg	IO	type
TheInputConcept	1	1	IN	CON
TheTargetAttribute	1	1	IN	BA
MinCardinality	1	1	IN	V
MinCardinalityType	1	1	IN	V
TheOutputAttribute	1	1	OUT	BA

- **Constraints**
  - TheTargetAttribute is in TheInputConcept
  - TheTargetAttribute is CATEGORIAL
  - TheOutputAttribute is in TheInputConcept
  - MinCardinality is NUMERIC
  - MinCardinalityType one-of 'ABSOLUTE,RELATIVE'
  - MinCardinality > 0

- **Conditions**
  - TheTargetAttribute NOT_NULL
- **Assertions**
  - TheOutputAttribute is CATEGORIAL
- **OPChecks**
  - If MinCardinalityType is RELATIVE, then MinCardinality < 0.5
  - If MinCardinalityType is ABSOLUTE then MinCardinality < no_of_rows_in_the_table / 2 (TheInputConcept.allNumber / 2 ??)

Algorithm
if MinCardinalityType = 'ABSOLUTE' then MinCardA := MinCardinality, else MinCardA := TheInputConcept.alNumber * MinCardinality
forall $u_{k,i}$ such that card ($u_{k,l}$) < MinCardA
Find Groups {U_k, U_k =$\{u_{k,i}\}_i$} : k≥1, card(U_k) ≥ MinCardA
Group

Where the Find_Groups procedure can be described as follows:

Find_Groups

1. create one group for each value
2. until each group has the frequency at least MinCardA do
   2.1. sort the values in ascending order of their frequencies
   2.2. merge first two groups into new group(i.e. the groups with lowest frequencies)

3.1.2 GroupingGivenNoOfGroups

This operator groups values of TheTargetAttribute by iteratively merging in each step two groups with the lowest frequencies until the number of groups NoOfGroups is reached. The algorithm has been inspired by hierarchical clustering. Values of TheOutputAttribute can be specified in Label.

- **Operator**
  - *name* GroupingGivenNoOfGroups
  - *loopable* – yes
  - *multistepable* - no
  - *manual* – yes
- **Parameters**

Name	minarg	maxarg	IO	type
TheInputConcept	1	1	IN	CON
TheTargetAttribute	1	1	IN	BA
NoOfGroups	1	1	IN	V
Label	0	NULL	IN	V
TheOutputAttribute	1	1	OUT	BA

- **Constraints**
  - TheTargetAttribute is in TheInputConcept
  - TheTargetAttribute is CATEGORIAL
  - TheOutputAttribute is in TheInputConcept
  - NoOfGroups is NUMERIC
  - NoOfGroups > 1
  - Label_ is CATEGORIAL

- **Conditions**
  - TheTargetAttribute NOT_NULL

- **Assertions**
  - TheOutputAttribute is CATEGORIAL

- **OPChecks**
  - NoOfGroups < number of unique values of TheTargetAttribute

Algorithm

Find Groups $\{U_k, U_k = \{u_{k,i}\}_i\}$: k=NoOfGroups
Group

Where the Find_Groups procedure can be described as follows:

Find_Groups

1. create one group for each value
2. repeat until the number of groups is NoOfGroups
   2.1. sort the values in ascending order of their frequencies
   2.2. merge first two groups into new group(i.e. the groups with lowest frequencies)

3.1.3 UserDefinedGrouping

This operator creates groups of TheTargetAttribute according to specifications given by the user in TheGroupings. Values not specified for grouping retain their original values. Unlike in the operator Mapping, more groups can be defined (TheGroupings is for each group a list of values). Values of TheOutputAttribute can be specified in Label.

- **Operator**
  - *name* UserDefinedGrouping
  - *loopable* – yes
  - *multistepable* - no
  - *manual* – yes

- **Parameters**

Name	minarg	maxarg	IO	type
TheInputConcept	1	1	IN	CON
TheTargetAttribute	1	1	IN	BA
TheGroupings	1	NULL	IN	V
Label	0	NULL	IN	V
TheOutputAttribute	1	1	OUT	BA

- **Constraints**
  - TheTargetAttribute is in TheInputConcept
  - TheTargetAttribute is CATEGORIAL
  - TheOutputAttribute is in TheInputConcept
  - TheGroupings_ is CATEGORIAL
  - Label_ is CATEGORIAL

- **Conditions**
- **Assertions**
  - TheOutputAttribute is CATEGORIAL

Algorithm

Find Groups
Group

Where the Find_Groups procedure can be described as follows:

Find_Groups

1. create one group for each list of values given by the user
2. create one group for each remaining value not assigned to any group

3.1.4 UserDefinedGroupingWithDefaultValue

This operator creates groups of TheTargetAttribute according to specifications given by the user in TheGroupings. Values not specified for grouping are grouped into default group Default. Unlike in the operator MappingWithDefaultValue, more groups can be defined (TheGrouping is for each group a list of values). Values of TheOutputAttribute can be specified in Label.

- **Operator**
  - *name* UserDefinedGroupingWithDefaultValue
  - *loopable* – yes
  - *multistepable* - no
  - *manual* – yes

- **Parameters**

Name	minarg	maxarg	IO	type
TheInputConcept	1	1	IN	CON
TheTargetAttribute	1	1	IN	BA
Default	1	1	IN	V
TheGroupings	1	NULL	IN	V
Label	0	NULL	IN	V
TheOutputAttribute	1	1	OUT	BA

- **Constraints**
  - TheTargetAttribute is in TheInputConcept
  - TheTargetAttribute is CATEGORIAL
  - TheOutputAttribute is in TheInputConcept
  - TheGroupings_ is CATEGORIAL
  - Label_ is CATEGORIAL
  - Default is CATEGORIAL

- **Conditions**

- **Assertions**
  - TheOutputAttribute is CATEGORIAL

Algorithm

Find Groups
Group

Where the Find_Groups procedure can be described as follows:

Find_Groups

1. create one group for each list of values given by the user
2. create one group for all remaining value not assigned to any group and assign the Default to this group

3.2 ML grouping operators

The machine learning grouping operators proposed for implementation in MiningMart were designed as an extension of machine learning discretization. They group values of a single attribute using the minimum classification error criterion. The training data are taken as random sample of given size (default size is 10 000).

3.2.1 ImplicitErrorBasedGrouping

This operator merges the values of TheTargetAttribute into groups with the same majority class (or classes) given in TheClassAttribute. If FullMerge is set to yes, then a group with two or more majority classes is merged with a group that have the same majority class. The resulting grouping minimize the classification error.

- **Operator**
  - *name* ImplicitErrorBasedGrouping
  - *loopable* – yes
  - *multistepable* - no
  - *manual* – no

- **Parameters**

Name	minarg	maxarg	IO	type
TheInputConcept	1	1	IN	CON
TheTargetAttribute	1	1	IN	BA
TheClassAttribute	1	1	IN	BA
FullMerge	1	1	IN	V
SampleSize	0	1	IN	V
TheOutputAttribute	1	1	OUT	BA

- **Constraints**
  - TheTargetAttribute is in TheInputConcept
  - TheTargetAttribute is CATEGORIAL
  - TheClassAttribute is in TheInputConcept
  - TheClassAttribute is CATEGORIAL
  - FullMerge one-of 'YES,NO'
  - TheOutputAttribute is in TheInputConcept
  - SampleSize is NUMERIC
  - SampleSize > 0

- **Conditions**
  - TheTargetAttribute NOT_NULL
  - TheClassAttribute NOT_NULL

- **Assertions**
  - TheOutputAttribute is CATEGORIAL

Algorithm

Create Groups $\{U_k, U_k = \{u_{k,i}\}_i\} : k \geq$ TheClassAttribute.distinct Group

Where the Create_Groups procedure can be described as follows:

Create_Groups

1. for each value *a*
   1.1. count the frequencies of each class and store the max frequency into *maxfreq*;
   1.2. assign class label using procedure ASSIGN;
2. create groups using procedure GROUP;

ASSIGN: //unique label corresponds to each combination of most frequent classes that can occur
label =0
for k = 1 to No_of_classes
if $n_k(a) = maxfreq$ then label = label + 2^k // $n_k(a)$ is #.examples with value *a* belonging to class *k*

GROUP:
1. create one group for values with the same class label; *//first*
2. if FullMerge='YES' then *//second*
   2.1. sort the groups having single majority class in ascending order into list *A*
   2.2. sort the groups having no single majority class in ascending order into list *B*
   2.3. repeat until each group from list *B* has been processed
      2.3.1. take first group from the list *B* //i.e. group with lowest frequency
      2.3.2. add this group to the first group from the list A that will not increase the classification error
      //i.e. to a group form *A* that has the same majority class and that has the lowest frequency
      2.3.3. resort list *A*

3.2.2 ErrorBasedGroupingGivenMinCardinality

This operator merges the values of TheTargetAttribute into groups with the cardinality above given threshold MinCardinality. The grouping is performed with respect to TheClassAttribute, but unlike implicit grouping, groups with single majority class are further merged if they do not have the required cardinality. This will increase the classification error.

- **Operator**
  - *name* ErrorBasedGroupingGivenMinCardinality
  - *loopable* – yes
  - *multistepable* - no
  - *manual* – no

- **Parameters**

Name	minarg	maxarg	IO	type
TheInputConcept	1	1	IN	CON
TheTargetAttribute	1	1	IN	BA
TheClassAttribute	1	1	IN	BA
MinCardinality	1	1	IN	V
MinCardinalityType	1	1	IN	V
SampleSize	0	1	IN	V
TheOutputAttribute	1	1	OUT	BA

- **Constraints**
  - TheTargetAttribute is in TheInputConcept
  - TheTargetAttribute is CATEGORIAL
  - TheClassAttribute is in TheInputConcept
  - TheClassAttribute is CATEGORIAL
  - TheOutputAttribute is in TheInputConcept
  - MinCardinality is NUMERIC
  - MinCardinalityType one-of 'ABSOLUTE,RELATIVE'
  - MinCardinality > 0
  - SampleSize is NUMERIC
  - SampleSize > 0

- **Conditions**
  - TheTargetAttribute NOT_NULL
  - TheClassAttribute NOT_NULL

- **Assertions**
  - TheOutputAttribute is CATEGORIAL

- **OPChecks**
  - If MinCardinalityType is RELATIVE, then MinCardinality < 0.5
  - If MinCardinalityType is ABSOLUTE then MinCardinality < no_of_rows_in_the_table / 2 (TheInputConcept.allNumber / 2 ??)

Algorithm

if MinCardinalityType = 'ABSOLUTE' then MinCardA := MinCardinality, else MinCardA := TheInputConcept.alNumber * MinCardinality
Create Groups $\{U_k, U_k = \{u_{k,i}\}_i\} : n(U_k) > \text{MinCardA}$
Group

Where the Create_Groups procedure can be described as follows:

Create_Groups

1. for each value *a*
   1.1. count the frequencies of each class and store the max frequency into *maxfreq*;
   1.2. assign class label using procedure ASSIGN;
2. create groups using procedure GROUP;

ASSIGN: //unique label corresponds to each combination of most frequent classes that can occur
label =0
for k = 1 to No_of_classes
  if $n_k(a)$ = *maxfreq* then label = label + 2^k // $n_k(a)$ is #.examples with value *a* belonging to class *k*

GROUP:
1. create one group for values with the same class label; *//first*
2. sort the groups in ascending order of their frequencies into list *A*
3. for all groups from *A* with frequency bellow *MinCardA* *//second* i.e. processing of ordered list A starting with lowest frequencies
   3.1. create a merge (new group) that will minimize the increase of classification error
   3.2. update frequency of the new group
   3.3. resort list *A*

3.2.3 ErrorBasedGroupingGivenNoOfGroups

This operator merges the values of TheTargetAttribute into at most NoOfGroups groups. The grouping is performed with respect to TheClassAttribute, but unlike the implicit discretization, if the number of groups exceeds NoOfGroups, groups are further merged. This will increase the classification error. Values of TheOutputAttribute can be specified in Label.

- **Operator**
  - *name* ErrorBasedGroupingGivenNoOfGroups
  - *loopable* – yes
  - *multistepable* - no
  - *manual* – no

- **Parameters**

Name	minarg	maxarg	IO	type
TheInputConcept	1	1	IN	CON
TheTargetAttribute	1	1	IN	BA
TheClassAttribute	1	1	IN	BA
NoOfGroups	1	1	IN	V
SampleSize	0	1	IN	V
Label	0	NULL	IN	V
TheOutputAttribute	1	1	OUT	BA

- **Constraints**
  - TheTargetAttribute is in TheInputConcept
  - TheTargetAttribute is CATEGORIAL
  - TheClassAttribute is in TheInputConcept
  - TheClassAttribute is CATEGORIAL
  - TheOutputAttribute is in TheInputConcept
  - Label_ is CATEGORIAL
  - NoOfGroups is NUMERIC
  - NoOfGroups > 1
  - SampleSize is NUMERIC
  - SampleSize > 0

- **Conditions**
  - TheTargetAttribute NOT_NULL
  - TheClassAttribute NOT_NULL

- **Assertions**
  - TheOutputAttribute is CATEGORIAL

Algorithm

Create Groups $\{U_k, U_k = \{u_{k,i}\}_i\} : n(U_k) = NoOfGropus$
Group

Where the Create_Groups procedure can be described as follows:

Create_Groups

1. for each value *a*
   1.1. count the frequencies of each class and store the max frequency into *maxfreq*;
   1.2. assign class label using procedure ASSIGN;
2. create groups using procedure GROUP;

ASSIGN: //unique label corresponds to each combination of most frequent classes that can occur
label =0
for k = 1 to No_of_classes
if $n_k(a) = maxfreq$ then label = label + 2^k // $n_k(a)$ is #.examples with value *a* belonging to class *k*

GROUP:
1. create one group for values with the same class label; *//first*
2. sort the groups having single majority class in ascending order into list *A*
3. sort the groups having no single majority class in ascending order into list *B*
4. repeat until each group from list *B* has been processed *//second*
   4.1. take first group from the list *B* //i.e. group with lowest frequency
   4.2. add this group to the first group from the list A that will not increase the classification error
   //i.e. to a group form *A* that has the same majority class and that has the lowest frequency
   4.3. resort list *A*
   4.4. if NoOfGroups is reached then terminate
5. repeat until the number of groups in *A* is NoOfGroups *//third*
   //processing of ordered list A starting with lowest frequencies
   5.1. take first group from the list A
   5.2. create a merge that will minimize the increase of classification error
   5.3. resort list *A*

References:

[Bay, 2000] Bay,S.D: Multivariate Discretization of Continuous Variables for Set Mining. In: Proc. of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2000.

[Berka, Bruha, 1995] Berka,P. - Bruha,I.: Various discretization procedures of numerical attributes: Empirical comparisons. In: (Kodratoff, Nakhaeizadeh, Taylor eds.) Proc. MLNet Familiarization Workshop on Statistics, Machine Learning and Knowledge Discovery in Databases, Herakleion, 1995, p.136-141.

[Berka, Bruha, 1998] Berka,P. - Bruha,I.: Discretization and Grouping: Preprocessing Steps for Data Mining. In: (Zytkow, Quafafou eds.) Proc. Principles of Data Mining and Knowledge Discovery PKDD'98, LNAI 1510, Springer, 1998, 239-245.

[Biggs et al., 1991] Biggs,D. - de Ville,B - Suen,E.: A~method of choosing multiway partitions for classification and decision trees. Journal of Applied Statistics, Vol. 18, No. 1, 1991, 49-62.

[Bruha, Berka, 2000] Bruha,I, - Berka,P.: Discretization and Fuzzification of Numerical Attributes in Attribute-Based Learning. In: (Szcepaniak,P.S. - Lisboa,P.J.G. - Kacprzyk,J, eds.) Fuzzy Systems in Medicine. Physica Verlag, 2000, 112-138. ISBN 3-7908-1263-3.

[Catlett, 1991] Catlett,J.: On changing continuous attributes into ordered discrete attributes. In: Y. Kodratoff, ed.: Machine Learning - EWSL-91, Springer-Verlag, 1991, 164-178.

[Dougherty et al. 1995] Dougherty,J. - Kohavi,R. –Sahami,M.: Supervised and unsupervised discretization of continuous features. In Proc. 12th Int. Conf. on Machine Learning, 1995.

[Elomaa, Rousu, 2002] Elomaa,T. – Rousu,J.: Fast Minimum Error Discretization. In: Proc. 19th Int. Conf. On Machine Learning ICML2002, Morgan Kaufman, 2002, 131-138.

[Fayyad, Irani, 1993] Fayyad,U.M. - Irani,K.B.: Multi-Interval Discretization ofContinuous-Valued Attributes for Classifiacation Learning. In: Proc.IJCAI'93, 1993.

[Lee, Shin, 1994] Lee,C. - Shin,D.: A Context-Sensitive Discretization of Numeric Attributes for Classification Learning. In: (Cohn,A. ,ed.), ECAI'94, Amsterdam, John Wiley \& Sons, 1994, 428-432.

[Peng, Flach, 2001] Peng,Y. – Flach,P.A.: Soft Discretization to Enhance the Continuous Decision Tree Induction. Int. Workshop on Integration and Collaboration Aspects of Data Mining, Decision Support and Meta-Learning, ECMLPKDD2001, Freiburg, 2001