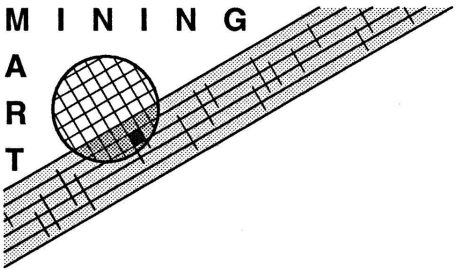

Enabling End-User Datawarehouse Mining
Contract No. IST-1999-11993
No. TR 12-02

Operator Specifications

Timm Euler

Dortmund, April 9, 2003

Contents

1 What this document is about

This document explains two things in detail: Firstly, section 2 describes some details about how the MiningMart compiler expects the metadata for a case description to be set up. Secondly, section 3 describes the current operators and their parameters.

2 Compiler constraints on metadata

This section explains in detail some issues in describing a case in such a way that it is operational for the MiningMart compiler.

2.1 Naming conventions

2.1.1 Operator names

The name of an operator (entry `op_name` in M4 table `Operator_T`) corresponds exactly (respecting case!) to the Java class that implements this operator in the compiler. This is only important to know if you want to implement additional operators. What is more generally important is that the names of the parameters of an operator are also fixed, because the compiler recognizes the type of a parameter by its name. This is described in more detail in section 3.1.

2.1.2 BaseAttribute names

Some operators have as their output on the conceptual level a Concept rather than a BaseAttribute (see section 3.1). This output Concept will generally be similar to the input Concept, in the sense that it copies some of the input BaseAttributes without changing them. To find out which BaseAttribute in the output Concept corresponds to which BaseAttribute in the input concept, their names are used. They must match exactly, ignoring case. This also means that it is necessary to give the output BaseAttribute in a feature construction operator (see section 3.1) a name which is different from all BaseAttribute names in the input Concept, so that no names are mixed up. If the output of the operator is a Concept, and a BaseAttribute in this output concept has no corresponding BaseAttribute in the input concept, it will be ignored by the compiler, because it may be needed for later steps. Ignoring means that no Column is created for it.

A similar mechanism is applied when Relations are used (see following section 2.2).

2.2 Relations

Relations are defined by the user between the initial Concepts of a case. In a case, the Concepts may then be modified. If later in the chain an operator is applied that makes use of relations, it must be able to find the Columns that realize the

keys. To this end, again the names of the BaseAttributes are used. Currently only `MultiRelationalFeatureConstruction` (MRFC; see section 3.2.1) uses relations. This means that in the Concepts used by MRFC, the BaseAttributes that correspond to the key BaseAttributes in the initial Concepts must have the same name (ignoring case).
Example: Suppose there are initial Concepts *Customer* and *Product* linked by a relation *buys* which is realized by a foreign link from the *Customer* to the *Product* table. The foreign key Column in the *Customer* table is named `fk_prod` and its BaseAttribute is named *CustomerBuys*. The Concept *Customer* may be the input to a chain which results in a new Concept *PrivateCustomer*. This new Concept must still have a BaseAttribute named *CustomerBuys*, which must not be the result of a feature construction, but must be copied from Concept to Concept in the chain[1]. Then the compiler can find the Column `fk_prod` by comparing the BaseAttributes of the current input concept *PrivateCustomer* and of the Concept which is linked to the relation *buys* (this relation is an input to the MRFC operator). The Column can be used to join the two Concepts *PrivateCustomer* and *Product*, although the first is a subconcept of *Customer*.

3 Operators and their parameters

This section explains the current MiningMart operators and the exact way of setting their parameters.

3.1 General issues

There are two kinds of operators, distinguished by their output on the conceptual level: those that have an output Concept (*Concept Operators*, listed in section 3.2), and those that have an output BaseAttribute (*Feature Construction Operators*, listed in section 3.4).

All operators have parameters, such as input Concept or output BaseAttribute. The name of such a parameter is fixed, for instance *TheInputConcept* is used for the input Concept for all operators. This means that the entry for this parameter in `par_name` in the M4 table `Parameter_T` must be *TheInputConcept*, respecting case. The parameter specification for each operator is stored in the M4 table OP_PARAMS_T (see MiningMart technical report TR18.1 and TR18.2).

Some operators have an unspecified number of parameters of the same type. For example, the learning operators take as input a number of BaseAttributes of the same concept and use them to construct their training examples. All these BaseAttributes use the same prefix for their parameter name (here *ThePredictingAttributes*) in `Parameter_T`. Since all parameters for one step are expected to have different names (for HCI use), number suffixes are added to these prefixes (*ThePredictingAttributes1, ThePredictingAttributes2*, etc). The

[1] Copying is done by simply having a BaseAttribute of this name in every output Concept in the chain.

compiler uses `ORDER BY par_nr` when reading them. Such parameters, which may contain a list, are marked with the word *List* in the operator descriptions in sections 3.2 and 3.4.

Special attention is needed if an operator is applied in a loop. All feature construction operators are loopable; further, the concept operator `RowSelectionBy-Query` is loopable. Feature construction operators are applied to one target attribute of an input concept and produce an output attribute. Looping means that the operator is applied to several target attributes (one after the other) and produces the respective number of output attributes, but the input concept is the same in all loops.

To decide whether an operator must be applied in a loop, the compiler checks the field `st_loopnr` in the M4 table `Step_T`, which gives the number of loops to be executed. If 0 or NULL is entered here, the operator is still executed once! If a number x (greater than 0) is entered here, the compiler looks for x sets of parameters for this operator in `Parameter_T`, excluding the parameters that are the same for all loops, which need to be entered only once. Thus, the parameter *TheInputConcept* must be declared only once, with the field `par_stloopnr` in the table `Parameter_T` set to 0, while the other parameters are given for every loop, with the respective loop number set in the field `par_stloopnr`, starting with 1. If no looping is intended, this field must be left NULL or 0. **Note**: Again, all parameters that are given for more than one loop must have a number suffix to their name, like the *List* parameters, to ensure that parameter names are uni que within one step.

For the concept operator `RowSelectionByQuery`, looping means that several query conditions are formulated using the parameters of this operator (one set of parameters for each condition), and that they are connected with AND. See the description of this operator.

In the following sections, all current operators are listed with their exact name (see section 2.1.1), a short description and the names of their parameters. In general, all input BaseAttributes belong to the input Concept, and all output BaseAttributes belong to the output Concept.

3.2 Concept operators

All Concept operators take an input Concept and create at least one new ColumnSet which they attach to the output Concept. The output Concept must have all its Features attached to it before the operator is compiled. All Concept operators have the two parameters *TheInputConcept* and *TheOutputConcept*, which are marked as *inherited* in the following parameter descriptions.

3.2.1 MultiRelationalFeatureConstruction

Takes a list of concepts which are linked by relations, and selects specified Features from them which are collected in the output Concept, via a join on the concepts of the chain. To be more precise: Recall (section 2.2) that Relations are only defined by the user between initial Concepts of a Case. Suppose

there is a chain of initial Concepts $C_1, \ldots, C_n$ such that between all C_i and $C_{i+1}, 1 \leq i < n$, C_i is the *FromConcept* of the i-th Relation and C_{i+1} is its *To-Concept*. These Concepts may be modified in the Case being modelled, to result in new Concepts $C'_1, \ldots, C'_n$, where some C'_i may be equal to C_i. However, as explained in section 2.2, the BaseAttributes that correspond to the Relation keys are still present in the new Concepts C'_i. By using their names, this operator can find the key Columns and join the new Concepts C'_i.

The parameter table below refers to this explanation. Note that all input Concepts are the new Concepts C'_i, but all input Relations link the original Concepts C_i.

ParameterName	ObjectType	Type	Remarks
TheInputConcept	CON	IN	Concept C'_1 (inherited)
TheConcepts	CON *List*	IN	Concepts $C'_2, \ldots, C'_n$
TheRelations	REL *List*	IN	they link $C_1, \ldots, C_n$
TheChainedFeatures	BA or MCF *List*	IN	from $C'_1, \ldots, C'_n$
TheOutputConcept	CON	OUT	inherited

3.2.2 JoinByKey

Takes a list of concepts, plus attributes indicating their primary keys, and joins the concepts. In *TheOutputConcept*, only one of the keys must be present. Each `BaseAttribute` specified in *TheKeys* must be a primary key of one of *TheConcepts*; thus, the number of entries in *TheConcepts* and *TheKeys* must be equal.

If several of the input concepts contain a `BaseAttribute` (or a `MultiColumnFeature`) with the same name, a special mapping mechanism is needed to relate them to different features in *TheOutputConcept*. For this, the parameters *MapInput* and *MapOutput* exist. Use *MapInput* to specify any feature in one of *TheConcepts*, and use *MapOutput* to specify the **corresponding** feature in *TheOutputConcept*. To make sure that for each *MapInput* the right *MapOutput* is found by this operator, it uses the looping mechanism. Although the parameter is not looped, the loop numbers in the parameter table in M4 are used to ensure the correspondence between *MapInput* and *MapOutput*. However, these two parameters only need to be specified for every pair of equally-named features in *TheConcepts*. So there are not necessarily as many "loops" as there are features in *TheOutputConcept*.

The field `par_stloopnr` in the M4 parameter table must be set to the number of pairs of *MapInput*/*MapOutput* parameters (may be 0). Each of these pairs gets a different loop number while all the other parameters get loop number 0.

ParameterName	ObjectType	Type	Remarks
TheConcepts	CON *List*	IN	no *TheInputConcept*!
TheKeys	BA *List*	IN	
MapInput	BA or MCF	IN	"looped"!
MapOutput	BA or MCF	OUT	"looped"!
TheOutputConcept	CON	OUT	inherited

3.2.3 UnionByKey

Takes a list of concepts, plus attributes indicating their primary keys, and unifies the concepts. In contrast to the operator JoinByKey (section 3.2.2), the output columnset is a union of the input columnsets rather than a join. For each value occuring in one of the key attributes of an input columnset a tuple in the output columnset is created. If a value is not present in all key attributes of the input columnsets, the corresponding (non-key) attributes of the output columnset are filled by *NULL* values.

In *TheOutputConcept*, only one of the keys must be present. Each `BaseAttribute` specified in *TheKeys* must be a primary key of one of *TheConcepts*; thus, the number of entries in *TheConcepts* and *TheKeys* must be equal.

If several of the input concepts contain a `BaseAttribute` (or a `MultiColumnFeature`) with the same name, a special mapping mechanism is needed to relate them to different features in *TheOutputConcept*. For this, the parameters *MapInput* and *MapOutput* exist. Use *MapInput* to specify any feature in one of *TheConcepts*, and use *MapOutput* to specify the **corresponding** feature in *TheOutputConcept*. To make sure that for each *MapInput* the right *MapOutput* is found by this operator, it uses the looping mechanism. Although the parameter is not looped, the loop numbers in the parameter table in M4 are used to ensure the correspondence between *MapInput* and *MapOutput*. However, these two parameters only need to be specified for every pair of equally-named features in *TheConcepts*. So there are not necessarily as many "loops" as there are features in *TheOutputConcept*.

The field `par_stloopnr` in the M4 parameter table must be set to the number of pairs of *MapInput*/*MapOutput* parameters (may be 0). Each of these pairs gets a different loop number while all the other parameters get loop number 0.

ParameterName	ObjectType	Type	Remarks
TheConcepts	CON *List*	IN	no *TheInputConcept*!
TheKeys	BA *List*	IN	
MapInput	BA or MCF	IN	"looped"!
MapOutput	BA or MCF	OUT	"looped"!
TheOutputConcept	CON	OUT	inherited

3.2.4 SpecifiedStatistics

An operator which computes certain statistical values for the *TheInputConcept*. The computed values appear in a `ColumnSet` which contains exactly one row with the statistical values, and which belongs to *TheOutputConcept*.

The sum of all values in an attribute can be computed by specifying a `BaseAttribute` with the parameter *AttributesComputeSum*. There can be more such attributes; the sum is computed for each. *TheOutputConcept* must contain a `BaseAttribute` for each sum which is computed; their names must be those of the input attributes, followed by the suffix "`_SUM`".

The total number of entries in an attribute can be computed by specifying a `BaseAttribute` with the parameter *AttributesComputeCount*. There can be

more such attributes; the number of entries is computed for each. *TheOutputConcept* must contain a `BaseAttribute` for each count which is computed; their names must be those of the input attributes, followed by the suffix "`_COUNT`".

The number of unique values in an attribute can be computed by specifying a `BaseAttribute` with the parameter *AttributesComputeUnique*. There can be more such attributes; the number of unique values is computed for each. *TheOutputConcept* must contain a `BaseAttribute` for each number of unique values which is computed; their names must be those of the input attributes, followed by the suffix "`_UNIQUE`".

Further, for a `BaseAttribute` specified with *AttributesComputeDistrib*, the distribution of its values is computed. For example, if a `BaseAttribute` contains the values 2, 4 and 6, three output `BaseAttributes` will contain the number of entries in the input where the value was 2, 4 and 6, respectively. For each `BaseAttribute` whose value distribution is to be computed, the possible values must be given with the parameter *DistribValues*. One entry in this parameter is a comma-separated string containing the different values; in the example, the string would be "`2,4,6`". Thus, the number of entries in *AttributesComputeDistrib* and *DistribValues* must be equal. *TheOutputConcept* must contain the corresponding number of `BaseAttributes` (three in the example); their names must be those of the input attributes, followed by the suffix "`_<value>`". In the example, *TheOutputConcept* would contain the `BaseAttributes` "`inputBaName_2`' ', "`inputBaName_4`" and "`inputBaName_6`".

ParameterName	ObjectType	Type	Remarks
TheInputConcept	CON	IN	inherited
AttributesComputeSum	BA *List*	IN	numeric
AttributesComputeCount	BA *List*	IN	(see
AttributesComputeUnique	BA *List*	IN	
AttributesComputeDistrib	BA *List*	IN	text)
DistribValues	V *List*	IN	
TheOutputConcept	CON	OUT	inherited

3.2.5 UnSegment

This operator is the inverse to any segmentation operator (see 3.2.9, 3.2.10, 3.2.11). While a segmentation operator segments its input concept's `ColumnSet` into several `ColumnSets`, UnSegment joins several `ColumnSets` into one. This operator makes sense only if a segmentation operator was applied previously in the chain, because it exactly reverses the function of that operator. To do so, the parameter *UnsegmentAttribute* specifies indirectly which of the three segmentation operators is reversed:

If a SegmentationStratified operator is reversed (section 3.2.9), this parameter gives the name of the `BaseAttribute` that was used for stratified segmentation. Note that this `BaseAttribute` must belong to *TheOutputConcept* of this operator, because the re-unified `ColumnSet` contains different values for this attribute (whereas before the execution of this operator, the different `ColumnSets`

did not contain this attribute, but each represented one of its values).

If a SegmentationByPartitioning operator is reversed (section 3.2.10), this parameter must have the value "(Random)".

If a SegmentationWithKMean operator is reversed (section 3.2.11), this parameter must have the value "(KMeans)".

Note that the segmentation to be reversed by this operator can be any segmentation in the chain before this operator.

ParameterName	ObjectType	Type	Remarks
TheInputConcept	CON	IN	inherited
UnsegmentAttribute	BA	OUT	see text
TheOutputConcept	CON	OUT	inherited

3.2.6 RowSelectionByQuery

The output Concept contains only records that fulfill the SQL condition formulated by the parameters of this operator. This operator is **loopable**! If applied in a loop, the conditions from the different loops are connected by AND. Every condition consists of a left-hand side, an SQL operator and a right-hand side. Together, these three must form a valid SQL condition. For example, to specify that only records (rows) whose value of attribute `sale` is either 50 or 60 should be selected, the left condition is the BaseAttribute for `sale`, the operator is *IN*, and the right condition is $(50, 60)$.

If this operator is applied in a loop, only the three parameters modelling the condition change from loop to loop, while input and output Concept remain the same.

ParameterName	ObjType	Type	Remarks
TheInputConcept	CON	IN	inherited (same in all loops)
TheLeftCondition	BA	IN	any BA of input concept
TheConditionOperator	V	IN	an SQL operator: $<$, $=$, ...
TheRightCondition	V	IN	
TheOutputConcept	CON	OUT	inherited (same in all loops)

3.2.7 RowSelectionByRandomSampling

Puts atmost as many rows into the output Concept as are specified in the parameter *HowMany*. Selects the rows randomly.

ParameterName	ObjType	Type	Remarks
TheInputConcept	CON	IN	inherited
HowMany	V	IN	max. no. of rows
TheOutputConcept	CON	OUT	inherited

3.2.8 DeleteRecordsWithMissingValues

Puts only those rows into the output Concept that have an entry which is NOT NULL in the Column for the specified *TheTargetAttribute*.

ParameterName	ObjType	Type	Remarks
TheInputConcept	CON	IN	inherited
TheTargetAttribute	BA	IN	may have NULL entries
TheOutputConcept	CON	OUT	inherited

3.2.9 SegmentationStratified

A MultiStep operator (creates several ColumnSets for the output Concept). The input Concept is segmented according to the values of the specified attribute, so that each resulting Columnset corresponds to one value of the attribute. For numeric attributes, intervals are built automatically (this makes use of the statistics tables and the functions that compute the statistics).

ParameterName	ObjType	Type	Remarks
TheInputConcept	CON	IN	inherited
TheAttribute	BA	IN	
TheOutputConcept	CON	OUT	inherited

3.2.10 SegmentationByPartitioning

A MultiStep operator (creates several ColumnSets for the output Concept). The input Concept is segmented randomly into as many Columnsets as are specified by the parameter *HowManyPartitions*.

ParameterName	ObjType	Type	Remarks
TheInputConcept	CON	IN	inherited
HowManyPartitions	V	IN	positive integer
TheOutputConcept	CON	OUT	inherited

3.2.11 SegmentationWithKMean

A MultiStep operator (creates several `ColumnSets` for the output Concept). The input Concept is segmented according to the clustering method KMeans (an external learning algorithm). The number of `ColumnSets` in the output concept is therefore not known before the application of this operator. However, the parameter *HowManyPartitions* specifies a maximum for this number. The parameter *OptimizePartitionNum* is a boolean that specifies if this number should be optimized by the learning algorithm (but it will not exceed the maximum). The parameter *SampleSize* gives a maximum number of learning examples for the external algorithm. The algorithm (KMeans) uses *ThePredictingAttributes* for clustering; these attributes must belong to *TheInputConcept*.

ParameterName	ObjType	Type	Remarks
TheInputConcept	CON	IN	inherited
HowManyPartitions	V	IN	positive integer
OptimizePartitionNum	V	IN	*true* or *false*
ThePredictingAttributes	BA *List*	IN	
SampleSize	V	IN	positive integer
TheOutputConcept	CON	OUT	inherited

3.2.12 Windowing

Windowing is applicable to time series data. It takes two BaseAttributes from the input Concept; one of contains time stamps, the other values. In the output Concept each row gives a time window; there will be two time stamp BaseAttributes which give the beginning and the end of each time window. Further, there will be as many value attributes as specified by the *WindowSize*; they contain the values for each window. *Distance* gives the distance between windows in terms of number of time stamps.

While *TimeBaseAttrib* and *ValueBaseAttrib* are `BaseAttributes` that belong to *TheInputConcept*, *OutputTimeStartBA*, *OutputTimeEndBA* and the *WindowedValuesBA*s belong to *TheOutputConcept*.

ParameterName	ObjType	Type	Remarks
TheInputConcept	CON	IN	inherited
TimeBaseAttrib	BA	IN	time stamps
ValueBaseAttrib	BA	IN	values
WindowSize	V	IN	positive integer
Distance	V	IN	positive integer
OutputTimeStartBA	BA	OUT	start time of window
OutputTimeEndBA	BA	OUT	end time of window
WindowedValuesBA	BA *List*	OUT	as many as *WindowSize*
TheOutputConcept	CON	OUT	inherited

3.2.13 SimpleMovingFunction

This operator combines windowing with the computation of the average value in each window. There is only one *OutputValueBA* which contains the average of the values in a window of the given *WindowSize*; windows are computed with the given *Distance* between each window. See also the description of the Windowing operator in section 3.2.12.

ParameterName	ObjType	Type	Remarks
TheInputConcept	CON	IN	inherited
InputTimeBA	BA	IN	
InputValueBA	BA	IN	
WindowSize	V	IN	
Distance	V	IN	
OutputTimeStartBA	BA	OUT	
OutputTimeEndBA	BA	OUT	
OutputValueBA	BA	OUT	
TheOutputConcept	CON	OUT	inherited

3.2.14 WeightedMovingFunction

This operator works like SimpleMovingFunction (section 3.2.13), but the weighted average is computed. The window size is not given explicitly, but is determined from the number of *Weights* given. The sum of all *Weights* must be 1.

ParameterName	ObjType	Type	Remarks
TheInputConcept	CON	IN	inherited
InputTimeBA	BA	IN	
InputValueBA	BA	IN	
Weights	V *List*	IN	sum must be 1
Distance	V	IN	positive integer
OutputTimeStartBA	BA	OUT	
OutputTimeEndBA	BA	OUT	
OutputValueBA	BA	OUT	
TheOutputConcept	CON	OUT	inherited

3.2.15 ExponentialMovingFunction

A time series smoothing operator. For two values with the given *Distance*, the first one is multiplied with *TailWeight* and the second one with *HeadWeight*. The resulting average is written into *OutputValueBA* and becomes the new tail value. *HeadWeight* and *TailWeight* must sum to 1.

ParameterName	ObjType	Type	Remarks
TheInputConcept	CON	IN	inherited
InputTimeBA	BA	IN	
InputValueBA	BA	IN	
HeadWeight	V	IN	
TailWeight	V	IN	
Distance	V	IN	positive integer
OutputTimeBA	BA	OUT	
OutputValueBA	BA	OUT	
TheOutputConcept	CON	OUT	inherited

3.2.16 SignalToSymbolProcessing

A time series abstraction operator. Creates intervals, their bounds are given in *OutputTimeStartBA* and *OutputTimeEndBA*. The average value of every interval will be in *AverageValueBA*. The average increase in that interval is in *IncreaseValueBA*. *Tolerance* determines when an interval is closed and a new one is opened: if the average increase, interpolated from the last interval, deviates from a value by more than *Tolerance*, a new interval begins.

ParameterName	ObjType	Type	Remarks
TheInputConcept	CON	IN	inherited
InputTimeBA	BA	IN	
InputValueBA	BA	IN	
Tolerance	V	IN	non-negative real number
AverageValueBA	BA	OUT	
IncreaseValueBA	BA	OUT	
OutputTimeStartBA	BA	OUT	
OutputTimeEndBA	BA	OUT	
TheOutputConcept	CON	OUT	inherited

3.2.17 Apriori

An implementation of the well known Apriori algorithm for the data mining step. It works on a sample read from the database. The sample size is given by the parameter *SampleSize*.

The input format is fixed. There is one input concept (*TheInputConcept*) having a BaseAttribute for the customer ID (parameter: *CustID*), one for the transaction ID (*TransID*), and one for an item part of this customer/transaction's itemset (*Item*). The algorithm expects all entries of these `BaseAttributes` to be integers. No null values are allowed.

It then finds all frequent (parameter: *MinSupport*) rules with at least the specified confidence (parameter: *MinConfidence*). Please keep in mind that these settings (especially the minimal support) are applied to a sample!

The output is specified by three parameters. *TheOutputConcept* is the concept the output table is attached to. It has two `BaseAttributes`, *PremiseBA* for the premises of rules and *ConclusionBA* for the conclusions. Each entry for one of these attributes contains a set of whitespace-separated item IDs (integers).

ParameterName	ObjType	Type	Remarks
TheInputConcept	CON	IN	inherited
CustID	BA	IN	customer id (integer, not NULL)
TransID	BA	IN	transaction id (integer, not NULL)
Item	BA	IN	item id (integer, not NULL)
MinSupport	V	IN	minimal support (integer)
MinConfidence	V	IN	minimal confidence (in $[0, 1]$)
SampleSize	V	IN	the size of the sample to be used
PremiseBA	BA	OUT	premises of rules
ConclusionBA	BA	OUT	conclusions of rules
TheOutputConcept	CON	OUT	inherited

3.3 Feature selection operators

Feature selection operators are also concept operators in that their output is a `Concept`, but they are listed in their own section since they have some common special properties. All of them (except *FeatureSelectionByAttributes*, see 3.3.1) use external algorithms to determine which features are taken over to the output concept. This means that at the time of designing an operating chain, it is not known which features will be selected. How can a complete, valid chain be designed then, since the input of later operators may depend on the output of a feature selection operator, which is only determined at compile time?

The answer is that conceptually, **all** possible features are present in the output concept of a feature selection operator, while the compiler creates `Columns` for only some of them (the selected ones). This means that in later steps, some of the features that are used for the input of an operator may not have a `Column`. If the operator depends on a certain feature, the compiler checks whether a `Column` is present, and shows an error message if no `Column` is found. If the operator is executable without that `Column`, no error occurs.

All feature selection operators have a parameter *TheAttributes* which specifies the set of features from which some are to be selected. (Again this is not true for *FeatureSelectionByAttributes*, see 3.3.1.) The parameter is needed because not all of the features of *TheInputConcept* can be used, as they may include a key attribute or the target attribute for a data mining step, which should not be deselected. This means that all attributes from *TheInputConcept* that are *not* listed as one of *TheAttributes* will be present in *TheOutputConcept* both on the conceptual and on the relational level.

3.3.1 FeatureSelectionByAttributes

This operator can be used for manual feature selection, which means that the user specifies all features to be selected. This is done by providing all and only the features that are to be selected in *TheOutputConcept*. The operator then simply copies those features from *TheInputConcept* to *TheOutputConcept* which are present in *TheOutputConcept*. It can be used to get rid of features that are not needed in later parts of the operator chain. All features in *TheOutputConcept* must have a corresponding feature (with the same name) in *TheInputConcept*.

ParameterName	ObjType	Type	Remarks
TheInputConcept	CON	IN	inherited
TheOutputConcept	CON	OUT	inherited

3.3.2 StatisticalFeatureSelection

A Feature Selection operator. This operator uses the stochastic correlation measure to select a subset of *TheAttributes*. All of *TheAttributes* must be present in *TheOutputConcept*. The parameter *Threshold* is a real number between 0 and 1 (default is 0.7). *SampleSize* specifies a maximum number of examples that are fed into the external algorithm.

ParameterName	ObjType	Type	Remarks
TheInputConcept	CON	IN	inherited
TheAttributes	BA *list*	IN	see section 3.3
SampleSize	V	IN	positive integer
Threshold	V	IN	real between 0 and 1
TheOutputConcept	CON	OUT	inherited

3.3.3 GeneticFeatureSelection

A Feature Selection operator.This operator uses a genetic algorithm to select a subset of *TheAttributes*. It calls C4.5 to evaluate the individuals of the genetic population. *TheTargetAttribute* specifies which attribute is the target attribute for the learning algorithm whose performance is used to select the best feature subset. *PopDim* gives the size of the population for the genetic algorithm. *StepNum* gives the number of generations. The probabilities of mutation and crossover are specified with *ProbMut* and *ProbCross*.

ParameterName	ObjType	Type	Remarks
TheInputConcept	CON	IN	inherited
TheAttributes	BA *list*	IN	see section 3.3
SampleSize	V	IN	positive integer
PopDim	V	IN	positive integer; try 30
StepNum	V	IN	positive integer; try 20
ProbMut	V	IN	real between 0 and 1; try 0.001
ProbCross	V	IN	real between 0 and 1; try 0.9
TheOutputConcept	CON	OUT	inherited

3.3.4 SGFeatureSelection

A Feature Selection operator. This operator is a combination of *StochasticFeatureSelection* (see 3.3.2), which is applied first, and *GeneticFeatureSelection* (see 3.3.3), applied afterwards. The parameter descriptions can be found in the sections about these operators (3.3.2 and 3.3.3).

ParameterName	ObjType	Type	Remarks
TheInputConcept	CON	IN	inherited
TheAttributes	BA *list*	IN	see section 3.3
SampleSize	V	IN	
PopDim	V	IN	
StepNum	V	IN	
ProbMut	V	IN	
ProbCross	V	IN	
Threshold	V	IN	real, between 0 and 1
TheOutputConcept	CON	OUT	inherited

3.3.5 FeatureSelectionWithSVM

A Feature Selection operator. This operator uses the $\xi\alpha$-estimator as computed by a Support Vector Machine training run to compare the classification performance of different feature subsets. Searching either forward or backward, it finds the best feature subset according to this criterion. Thus it performs a simple beam search of width 1.

TheTargetAttribute must be binary as Support Vector Machines can only solve binary classification problems. (The $\xi\alpha$-estimator can only be computed for classification problems.) The parameter *PositiveTargetValue* specifies the class label of the positive class. There are some SVM-specific parameters; the table gives reasonable values to choose if nothing is known about the data or SVMs. For the *KernelType*, only the following values (Strings) are possible: *dot, polynomial, neural, radial, anova. Dot* is the linear kernel and can be taken as default.

This operator can use two different versions of the Support Vector Machine algorithm. One runs in main memory; it needs the parameter *SampleSize* to determine a maximum number of training examples. The other runs in the database; it is used if the optional parameter *UseDB_SVM* is set to the String

`true`. When this version is used, an additional parameter *TheKey* is needed which gives the `BaseAttribute` whose column is the primary key of *TheInputConcept*. (*TheKey* can be left out only if the `ColumnSet` that belongs to *TheInputConcept* represents a table rather than a view.) The database algorithm restricts the possible kernel types to *dot* and *radial*. It can also use the parameter *SampleSize*.

ParameterName	ObjType	Type	Remarks
TheInputConcept	CON	IN	inherited
TheAttributes	BA *list*	IN	see section 3.3
TheTargetAttribute	BA	IN	must be binary
PositiveTargetValue	V	IN	the positive class label
KernelType	V	IN	see explanation above
SampleSize	V	IN	see explanation above
C	V	IN	positive real; try 1.0
Epsilon	V	IN	positive real; try 0.1
UseDB_SVM	V	IN	optional; one of *true*, *false*
TheKey	BA	IN	optional
SearchDirection	V	IN	one of *forward*, *backward*
TheOutputConcept	CON	OUT	inherited

3.3.6 SimpleForwardFeatureSelectionGivenNoOfAttributes

A Feature Selection operator. This operator adds one feature a time starting from the empty set until the required number of features *NoOfAttributes* is reached. The attributes are selected with respect to *TheClassAttribute*, the group optimises the information dependence criterion. Use this operator if only a small number of original attributes is to be selected. The selection is done from the set of *TheAttributes*, attributes not specified in this set are selected automatically.

ParameterName	ObjType	Type	Remarks
TheInputConcept	CON	IN	inherited
TheAttributes	BA *list*	IN	see section 3.3
TheClassAttribute	BA	IN	must be categorial
NoOfAttributes	V	IN	positive integer
SampleSize	V	IN	positive integer
TheOutputConcept	CON	OUT	inherited

3.3.7 SimpleBackwardFeatureSelectionGivenNoOfAttributes

A Feature Selection operator. This operator removes one feature a time starting from all attributes until the required number of features *NoOfAttributes* is reached. The attributes are selected with respect to *TheClassAttribute*, the group optimises the information dependence criterion. Use this operator if a large number of original attributes is to be selected. The selection is done from the set of *TheAttributes*, attributes not specified in this set are selected automatically.

ParameterName	ObjType	Type	Remarks
TheInputConcept	CON	IN	inherited
TheAttributes	BA *list*	IN	see section 3.3
TheClassAttribute	BA	IN	must be categorial
NoOfAttributes	V	IN	positive integer
SampleSize	V	IN	positive integer
TheOutputConcept	CON	OUT	inherited

3.3.8 FloatForwardFeatureSelectionGivenNoOfAtt

A Feature Selection operator. This operator adds one feature a time starting from empty set until the required number of features *NoOfAttributes* is reached. The attributes are selected with respect to *TheClassAttribute*, the group optimises the information dependence criterion. Unlike the simple operator, after adding a feature a check is performed if another feature should be removed. Use this operator if only a small number of original attributes is to be selected. The selection is done from the set of *TheAttributes*, attributes not specified in this set are selected automatically.

ParameterName	ObjType	Type	Remarks
TheInputConcept	CON	IN	inherited
TheAttributes	BA *list*	IN	see section 3.3
TheClassAttribute	BA	IN	must be categorial
NoOfAttributes	V	IN	positive integer
SampleSize	V	IN	positive integer
TheOutputConcept	CON	OUT	inherited

3.3.9 FloatBackwardFeatureSelectionGivenNoOfAtt

A Feature Selection operator. This operator removes one feature a time starting from all attributes until the required number of features *NoOfAttributes* is reached. The attributes are selected with respect to *TheClassAttribute*, the group optimises the information dependence criterion. Unlike the simple operator, after removing a feature a check is performed if another feature should be added. Use this operator if a large number of original attributes is to be selected. The selection is done from the set of *TheAttributes*, attributes not specified in this set are selected automatically.

ParameterName	ObjType	Type	Remarks
TheInputConcept	CON	IN	inherited
TheAttributes	BA *list*	IN	see section 3.3
TheClassAttribute	BA	IN	must be categorial
NoOfAttributes	V	IN	positive integer
SampleSize	V	IN	positive integer
TheOutputConcept	CON	OUT	inherited

3.3.10 UserDefinedFeatureSelection

A Feature Selection operator. This operator copies exactly those features from *TheInputConcept* to *TheOutputConcept* that are specified in *TheSelectedAttributes*. It can be used for the same task as the operator *FeatureSelectionByAttributes*, see 3.3.1, namely when the user knows which features to select. The difference is that *FeatureSelectionByAttributes* copies all features that are present in *TheOutputConcept*, while this operator copies those that are specified in the extra parameter *TheSelectedAttributes*.

ParameterName	ObjType	Type	Remarks
TheInputConcept	CON	IN	inherited
TheSelectedAttributes	BA *list*	IN	the user's selection
TheOutputConcept	CON	OUT	inherited

3.4 Feature construction operators

All operators in this section are loopable. For loops, *TheInputConcept* remains the same (`par_stloopnr = 0`) while *TheTargetAttribute*, *TheOutputAttribute* and further operator-specific parameters change from loop to loop (loop numbers start with 1).

3.4.1 AssignAverageValue

A MissingValue operator. Each missing value in *TheTargetAttribute* is replaced by the average value of that Column. The operator computes the column statistics if they are not computed yet, which may take some time.

ParameterName	ObjType	Type	Remarks
TheInputConcept	CON	IN	inherited
TheTargetAttribute	BA	IN	must be numeric
TheOutputAttribute	BA	OUT	inherited

3.4.2 AssignModalValue

A MissingValue operator. Each missing value in *TheTargetAttribute* is replaced by the modal value of that Column. The operator computes the column statistics if they are not computed yet, which may take some time.

ParameterName	ObjType	Type	Remarks
TheInputConcept	CON	IN	inherited
TheTargetAttribute	BA	IN	
TheOutputAttribute	BA	OUT	inherited

3.4.3 AssignMedianValue

A MissingValue operator. Each missing value in *TheTargetAttribute* is replaced by the median of that Column. The operator computes the column statistics if they are not computed yet, which may take some time.

ParameterName	ObjType	Type	Remarks
TheInputConcept	CON	IN	inherited
TheTargetAttribute	BA	IN	
TheOutputAttribute	BA	OUT	inherited

3.4.4 AssignDefaultValue

A MissingValue operator. Each missing value in *TheTargetAttribute* is replaced by the *DefaultValue*.

ParameterName	ObjType	Type	Remarks
TheInputConcept	CON	IN	inherited
TheTargetAttribute	BA	IN	inherited
DefaultValue	V	IN	
TheOutputAttribute	BA	OUT	inherited

3.4.5 AssignStochasticValue

A MissingValue operator. Each missing value in *TheTargetAttribute* is replaced by a value which is randomly selected according to the distribution of present values in this attribute. For example, if half of the entries in *TheTargetAttribute* have a specific value, this value is chosen with a probability of 0.5. The operator computes the column statistics if they are not computed yet, which may take some time.

ParameterName	ObjType	Type	Remarks
TheInputConcept	CON	IN	inherited
TheTargetAttribute	BA	IN	inherited
TheOutputAttribute	BA	OUT	inherited

3.4.6 MissingValuesWithRegressionSVM

A MissingValue operator. Each missing value in *TheTargetAttribute* is replaced by a predicted value. For prediction, a Support Vector Machine (SVM) is trained in regression mode from *ThePredictingAttributes* (taking *TheTargetAttribute* values that are not missing as target function values). All *ThePredictingAttributes* must belong to *TheInputConcept*. *TheOutputAttribute* contains the original values, plus the predicted values where the original ones were missing.

There are some SVM-specific parameters; the table gives reasonable values to choose if nothing is known about the data or SVMs. For the *KernelType*, only the following values (Strings) are possible: *dot, polynomial, neural, radial, anova*. *Dot* is the linear kernel and can be taken as default.

This operator can use two different versions of the Support Vector Machine algorithm. One runs in main memory; it needs the parameter *SampleSize* to determine a maximum number of training examples. The other runs in the database; it is used if the optional parameter *UseDB_SVM* is set to the String `true`. When this version is used, an additional parameter *TheKey* is needed

which gives the `BaseAttribute` whose column is the primary key of *TheInputConcept*. (*TheKey* can be left out only if the `ColumnSet` that belongs to *TheInputConcept* represents a table rather than a view.) The database algorithm restricts the possible kernel types to *dot* and *radial*. It can also use the parameter *SampleSize*.

With the parameters *LossFunctionPos* and *LossFunctionNeg*, the loss function that is used for the regression can be biased such that predicting too high is more expensive ($LossFunctionPos > LossFunctionNeg$) or less expensive ($LossFunctionNeg > LossFunctionPos$)than predicting too low. If both values are equal, no bias is used. The parameter C balances training error against generalisation quality; positive values between 0.01 and 1000 have been used successfully in the literature. *Epsilon* limits the allowed error an example may produce; small values under 0.5 should be used.

ParameterName	ObjType	Type	Remarks
TheInputConcept	CON	IN	inherited
TheTargetAttribute	BA	IN	inherited
ThePredictingAttributes	BA *List*	IN	
KernelType	V	IN	see explanation above
SampleSize	V	IN	see explanation above
LossFunctionPos	V	IN	positive real; try 1.0
LossFunctionNeg	V	IN	positive real; try 1.0
C	V	IN	positive real; try 1.0
Epsilon	V	IN	positive real; try 0.1
UseDB_SVM	V	IN	optional; one of *true*, *false*
TheKey	BA	IN	optional
TheOutputAttribute	BA	OUT	inherited

3.4.7 LinearScaling

A scaling operator. Values in *TheTargetAttribute* are scaled to lie between *NewRangeMin* and *NewRangeMax*.

ParameterName	ObjType	Type	Remarks
TheInputConcept	CON	IN	inherited
TheTargetAttribute	BA	IN	inherited
NewRangeMin	V	IN	new min value
NewRangeMax	V	IN	new max value
TheOutputAttribute	BA	OUT	inherited

3.4.8 LogScaling

A scaling operator. Values in *TheTargetAttribute* are scaled to their logarithm to the given *LogBase*.

ParameterName	ObjType	Type	Remarks
TheInputConcept	CON	IN	inherited
TheTargetAttribute	BA	IN	inherited
LogBase	V	IN	
TheOutputAttribute	BA	OUT	inherited

3.4.9 SupportVectorMachineForRegression

A data mining operator. Values in *TheTargetAttribute* are used as target function values to train the SVM on examples that are formed with *ThePredicting-Attributes*. All *ThePredictingAttributes* must belong to *TheInputConcept*. *The-OutputAttribute* contains the predicted values.

There are some SVM-specific parameters; the table gives reasonable values to choose if nothing is known about the data or SVMs. For the *KernelType*, only the following values (Strings) are possible: *dot, polynomial, neural, radial, anova*. *Dot* is the linear kernel and can be taken as default.

This operator can use two different versions of the Support Vector Machine algorithm. One runs in main memory; it needs the parameter *SampleSize* to determine a maximum number of training examples. The other runs in the database; it is used if the optional parameter *UseDB_SVM* is set to the String `true`. When this version is used, an additional parameter *TheKey* is needed which gives the `BaseAttribute` whose column is the primary key of *TheInputConcept*. (*TheKey* can be left out only if the `ColumnSet` that belongs to *TheInputConcept* represents a table rather than a view.) The database algorithm restricts the possible kernel types to *dot* and *radial*. It can also use the parameter *SampleSize*.

With the parameters *LossFunctionPos* and *LossFunctionNeg*, the loss function that is used for the regression can be biased such that predicting too high is more expensive ($LossFunctionPos > LossFunctionNeg$) or less expensive ($LossFunctionNeg > LossFunctionPos$)than predicting too low. If both values are equal, no bias is used. The parameter C balances training error against generalisation quality; positive values between 0.01 and 1000 have been used successfully in the literature. *Epsilon* limits the allowed error an example may produce; small values under 0.5 should be used.

ParameterName	ObjType	Type	Remarks
TheInputConcept	CON	IN	inherited
TheTargetAttribute	BA	IN	inherited
ThePredictingAttributes	BA *List*	IN	
KernelType	V	IN	see explanation above
SampleSize	V	IN	see explanation above
LossFunctionPos	V	IN	positive real; try 1.0
LossFunctionNeg	V	IN	positive real; try 1.0
C	V	IN	positive real; try 1.0
Epsilon	V	IN	positive real; try 0.1
UseDB_SVM	V	IN	optional; one of *true*, *false*
TheKey	BA	IN	optional
TheOutputAttribute	BA	OUT	inherited

3.4.10 SupportVectorMachineForClassification

A data mining operator. Values in *TheTargetAttribute* are used as target function values to train the SVM on examples that are formed with *ThePredicting-Attributes*. *TheTargetAttribute* must be binary as Support Vector Machines can only solve binary classification problems. The parameter *PositiveTargetValue* specifies the class label of the positive class. All *ThePredictingAttributes* must belong to *TheInputConcept*. *TheOutputAttribute* contains the predicted values.

There are some SVM-specific parameters; the table gives reasonable values to choose if nothing is known about the data or SVMs. For the *KernelType*, only the following values (Strings) are possible: *dot, polynomial, neural, radial, anova*. *Dot* is the linear kernel and can be taken as default.

This operator can use two different versions of the Support Vector Machine algorithm. One runs in main memory; it needs the parameter *SampleSize* to determine a maximum number of training examples. The other runs in the database; it is used if the optional parameter *UseDB_SVM* is set to the String `true`. When this version is used, an additional parameter *TheKey* is needed which gives the `BaseAttribute` whose column is the primary key of *TheInputConcept*. (*TheKey* can be left out only if the `ColumnSet` that belongs to *TheInputConcept* represents a table rather than a view.) The database algorithm restricts the possible kernel types to *dot* and *radial*. It can also use the parameter *SampleSize*.

The parameter *C* balances training error against generalisation quality; positive values between 0.01 and 1000 have been used successfully in the literature. *Epsilon* limits the allowed error an example may produce; small values under 0.5 should be used.

ParameterName	ObjType	Type	Remarks
TheInputConcept	CON	IN	inherited
TheTargetAttribute	BA	IN	inherited; must be binary
ThePredictingAttributes	BA *List*	IN	
KernelType	V	IN	see explanation above
SampleSize	V	IN	see explanation above
C	V	IN	positive real; try 1.0
Epsilon	V	IN	positive real; try 0.1
UseDB_SVM	V	IN	optional; one of *true*, *false*
TheKey	BA	IN	optional
PositiveTargetValue	V	IN	the positive class label
TheOutputAttribute	BA	OUT	inherited

3.4.11 MissingValueWithDecisionRules

A Missing value operator. Each missing value (NULL value) in *TheTargetAttribute* is replaced by a predicted value. For prediction, a set of Decision Rules is learned from *ThePredictingAttributes*, which must belong to *TheInputConcept*. The pruning confidence level is given in *PruningConf* as a percentage.

ParameterName	ObjType	Type	Remarks
TheInputConcept	CON	IN	inherited
TheTargetAttribute	BA	IN	inherited
ThePredictingAttributes	BA *List*	IN	
SampleSize	V	IN	positive integer
PruningConf	V	IN	between 0 and 100
TheOutputAttribute	BA	OUT	inherited

3.4.12 MissingValueWithDecisionTree

A Missing value operator. Each missing value (NULL value) in *TheTargetAttribute* is replaced by a predicted value. For prediction, a Decision Tree is learned from *ThePredictingAttributes*, which must belong to *TheInputConcept*. The pruning confidence level is given in *PruningConf* as a percentage.

ParameterName	ObjType	Type	Remarks
TheInputConcept	CON	IN	inherited
TheTargetAttribute	BA	IN	inherited
ThePredictingAttributes	BA *List*	IN	
SampleSize	V	IN	positive integer
PruningConf	V	IN	between 0 and 100
TheOutputAttribute	BA	OUT	inherited

3.4.13 PredictionWithDecisionRules

A Feature Construction operator. Decision rules are learned using *ThePredicting-Attributes* as learning attributes and *TheTargetAttribute* as label. *TheOutputAttribute* contains the labels that the decision rules predict. The operator may be

used to compare predicted and actual values, or in combination with the operator AssignPredictedValueCategorial (see section 3.4.15). All *ThePredictingAttributes* must belong to *TheInputConcept*. The pruning confidence level is given in *PruningConf* as a percentage.

ParameterName	ObjType	Type	Remarks
TheInputConcept	CON	IN	inherited
TheTargetAttribute	BA	IN	inherited
ThePredictingAttributes	BA *List*	IN	
SampleSize	V	IN	positive integer
PruningConf	V	IN	between 0 and 100
TheOutputAttribute	BA	OUT	inherited

3.4.14 PredictionWithDecisionTree

A Feature Construction operator. A Decision Tree is learned using *ThePredictingAttributes* as learning attributes and *TheTargetAttribute* as label. *TheOutputAttribute* contains the labels that the decision tree predicts. The operator may be used to compare predicted and actual values, or in combination with the operator AssignPredictedValueCategorial (see section 3.4.15). All *ThePredictingAttributes* must belong to *TheInputConcept*. The pruning confidence level is given in *PruningConf* as a percentage.

ParameterName	ObjType	Type	Remarks
TheInputConcept	CON	IN	inherited
TheTargetAttribute	BA	IN	inherited
ThePredictingAttributes	BA *List*	IN	
SampleSize	V	IN	positive integer
PruningConf	V	IN	between 0 and 100
TheOutputAttribute	BA	OUT	inherited

3.4.15 AssignPredictedValueCategorial

A Missing Value operator. Any missing value of *TheTargetAttribute* is replaced by the value of the same row from *ThePredictedAttribute*. The latter may have been filled by the operator PredictionWithDecisionRules (3.4.13) or PredictionWithDecisionTree (3.4.14). It must belong to *TheInputConcept*.

ParameterName	ObjType	Type	Remarks
TheInputConcept	CON	IN	inherited
TheTargetAttribute	BA	IN	inherited
ThePredictedAttribute	BA	IN	
TheOutputAttribute	BA	OUT	inherited

3.4.16 GenericFeatureConstruction

This operator creates an output attribute on the basis of a given SQL definition (Parameter *SQL_String*). The definition must be well-formed SQL defining how

values for the output attribute are computed based on one of the attributes in *TheInputConcept*. To refer to the attributes in *TheInputConcept*, the names of the `BaseAttributes` are used—and not the names of any `Columns`. For example, if there are two `BaseAttributes` named "INCOME" and "TAX" in *TheInputConcept*, this operator can compute their sum if *SQL_String* is defined as "(INCOME + TAX)". Since the operator must resolve names of `BaseAttributes`, it cannot be used if there are two or more `BaseAttributes` in *TheInputConcept* with the same name.

TheTargetAttribute is needed to have a blueprint for *TheOutputAttribute*. The operator ignores *TheTargetAttribute*, except that it uses the relational datatype of its column to specify the relational datatype for the column of *TheOutputAttribute*.

ParameterName	ObjType	Type	Remarks
TheInputConcept	CON	IN	inherited
TheTargetAttribute	BA	IN	inherited; specifies datatype
SQL_String	V	IN	see text
TheOutputAttribute	BA	OUT	inherited

3.4.17 TimeIntervalManualDiscretization

This operator can be used to discretize a time attribute manually. The looped parameters specify a mapping to be performed from *TheTargetAttribute*, a BaseAttribute of type *TIME* to a set of user specified categories. As for all FeatureConstruction operators a BaseAttribute *TheOutputAttribute* is added to the *TheInputConcept*.

The mapping is defined by looped parameters. An interval is specified by its lower bound *IntervalStart*, its upper bound *IntervalEnd* and two additional parameters *StartIncExc* and *EndIncExc*, stating if the interval bounds are included (value: "I") or excluded (value: "E"). The value an interval is mapped to is given by the looped parameter *MapTo*. If an input value does not belong to any interval, it is mapped to the value *DefaultValue*.

To be able to cope with various time formats (e.g. 'HH-MI-SS') the operator reads the given format from the parameter *TimeFormat* (ORACLE-specific).

ParameterName	ObjType	Type	Remarks
TheInputConcept	CON	IN	inherited
TheTargetAttribute	BA	IN	inherited, type: TIME
IntervalStart	V	IN	"looped", lower bound of interval
IntervalEnd	V	IN	"looped", upper bound of interval
MapTo	V	IN	value to map time interval to
StartIncExc	V	IN	one of "I" and "E"
EndIncExc	V	IN	one of "I" and "E"
DefaultValue	V	IN	value if no mapping applies
TimeFormat	V	IN	ORACLE specific time format
TheOutputAttribute	BA	OUT	inherited

3.4.18 NumericIntervalManualDiscretization

This operator can be used to discretize a numeric attribute manually. It is very similar to the operator TimeIntervalManualDiscretization described in 3.4.17. The looped parameters *IntervalStart*, *IntervalEnd*, *StartIncExc*, *EndIncExc*, and *MapTo*. again specify a mapping to be performed. If an input value does not belong to any interval, it is mapped to the value *DefaultValue*. *TheTargetAttribute* needs to be of type ordinal.

ParameterName	ObjType	Type	Remarks
TheInputConcept	CON	IN	inherited
TheTargetAttribute	BA	IN	inherited, type: ORDINAL
IntervalStart	V	IN	"looped", lower bound of interval
IntervalEnd	V	IN	"looped", upper bound of interval
MapTo	V	IN	value to map time interval to
StartIncExc	V	IN	one of "I" and "E"
EndIncExc	V	IN	one of "I" and "E"
DefaultValue	V	IN	value if no mapping applies
TimeFormat	V	IN	ORACLE specific time format
TheOutputAttribute	BA	OUT	inherited

3.4.19 EquidistantDiscretizationGivenWidth

A discretization operator. Numeric attributes are discretized and the output is a categorial attribute. 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 values out of range.

ParameterName	ObjType	Type	Remarks
TheInputConcept	CON	IN	inherited
TheTargetAttribute	BA	IN	must be numeric
StartPoint	V	IN	optional
IntervalWidth	V	IN	a positive real number
ClosedTo	V	IN	one of *LEFT* or *RIGHT*
TheOutputAttribute	BA	OUT	should be categorial

3.4.20 EquidistantDiscretizationGivenNoOfIntervals

A discretization operator. Numeric attributes are discretized and the output is a categorial attribute. This operator divides the range of *TheTargetAttribute* into the given number of intervals *NoOfIntervals* with the same width. The first and the last interval cover also the values out of range. Values of *TheOutputAttribute* can be specified in the parameter *Label*.

ParameterName	ObjType	Type	Remarks
TheInputConcept	CON	IN	inherited
TheTargetAttribute	BA	IN	must be numeric
NoOfIntervals	V	IN	integer
ClosedTo	V	IN	one of *LEFT* or *RIGHT*
Label	V *List*	IN	optional
TheOutputAttribute	BA	OUT	should be categorial

3.4.21 EquifrequentDiscretizationGivenCardinality

A discretization operator. Numeric attributes are discretized and the output is a categorial attribute. This operator divides the range of *TheTargetAttribute* into intervals with given *Cardinality* (number of examples whose values are in the interval). The first and the last interval cover also the values out of range. *CardinalityType* decides how the parameter *Cardinality* is to be interpreted. Values of *TheOutputAttribute* can be specified in the parameter *Label* (this makes sense only if *CardinalityType* is *RELATIVE*).

ParameterName	ObjType	Type	Remarks
TheInputConcept	CON	IN	inherited
TheTargetAttribute	BA	IN	must be numeric
CardinalityType	V	IN	*ABSOLUTE* or *RELATIVE*
Cardinality	V	IN	positive
ClosedTo	V	IN	one of *LEFT* or *RIGHT*
Label	V *List*	IN	optional
TheOutputAttribute	BA	OUT	should be categorial

3.4.22 EquifrequentDiscretizationGivenNoOfIntervals

A discretization operator. Numeric attributes are discretized and the output is a categorial attribute. This operator divides the range of *TheTargetAttribute* into the 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 values out of range. Values of *TheOutputAttribute* can be specified in the parameter *Label*.

ParameterName	ObjType	Type	Remarks
TheInputConcept	CON	IN	inherited
TheTargetAttribute	BA	IN	must be numeric
NoOfIntervals	V	IN	positive integer > 1
ClosedTo	V	IN	one of *LEFT* or *RIGHT*
Label	V *List*	IN	optional
TheOutputAttribute	BA	OUT	should be categorial

3.4.23 UserDefinedDiscretization

A discretization operator. Numeric attributes are discretized and the output is a categorial attribute. This operator divides the range of *TheTargetAttribute*

into intervals according to user given cutpoints *TheCutpoints*, which is a list of values which each give a cutpoint for the intervals to be created. The cutpoints must be given in ascending order. Values of *TheOutputAttribute* can be specified in the parameter *Label*.

ParameterName	ObjType	Type	Remarks
TheInputConcept	CON	IN	inherited
TheTargetAttribute	BA	IN	must be numeric
TheCutpoints	V	IN	see text
ClosedTo	V	IN	one of *LEFT* or *RIGHT*
Label	V *List*	IN	optional
TheOutputAttribute	BA	OUT	should be categorial

3.4.24 ImplicitErrorBasedDiscretization

A discretization operator. Numeric attributes are discretized and the output is a categorial attribute. This operator divides the range of *TheTargetAttribute* into intervals by merging subsequent values with the same majority class (or classes) given in *TheClassAttribute*. *TheClassAttribute* contains the labels of an example as in a Machine Learning setting. 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. The parameter *SampleSize* gives a maximum number of learning examples for the external algorithm.

ParameterName	ObjType	Type	Remarks
TheInputConcept	CON	IN	inherited
TheTargetAttribute	BA	IN	must be numeric
TheClassAttribute	BA	IN	must be categorial
ClosedTo	V	IN	one of *LEFT* or *RIGHT*
FullMerge	V	IN	one of *YES* or *NO*
SampleSize	V	IN	optional; positive integer
TheOutputAttribute	BA	OUT	should be categorial

3.4.25 ErrorBasedDiscretizationGivenMinCardinality

A discretization operator. Numeric attributes are discretized and the output is a categorial attribute. This operator divides the range of *TheTargetAttribute* into intervals with cardinality greater or equal to *MinCardinality*. *MinCardinalityType* decides if *MinCardinality* values are read as absolute values (integers) or relative values (real, between 0 and 1). *TheTargetAttribute* 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. *TheClassAttribute* contains the labels of an example as in a Machine Learning setting. The parameter *SampleSize* gives a maximum number of learning examples for the external algorithm.

ParameterName	ObjType	Type	Remarks
TheInputConcept	CON	IN	inherited
TheTargetAttribute	BA	IN	must be numeric
TheClassAttribute	BA	IN	must be categorial
MinCardinalityType	V	IN	*ABSOLUTE* or *RELATIVE*
MinCardinality	V	IN	positive
ClosedTo	V	IN	one of *LEFT* or *RIGHT*
SampleSize	V	IN	optional; positive integer
TheOutputAttribute	BA	OUT	should be categorial

3.4.26 ErrorBasedDiscretizationGivenNoOfInt

A discretization operator. Numeric attributes are discretized and the output is a categorial attribute. This operator divides the range of *TheTargetAttribute* into at most *NoOfIntervals* intervals. *TheTargetAttribute* is divided into intervals with respect to *TheClassAttribute*, but unlike the implicit discretization, if the number of interval exceeds *NoOfIntervals*, intervals are further merged. This will increase the classification error. *TheClassAttribute* contains the labels of an example as in a Machine Learning setting. Values of *TheOutputAttribute* can be specified in the parameter *Label*. The parameter *SampleSize* gives a maximum number of learning examples for the external algorithm.

ParameterName	ObjType	Type	Remarks
TheInputConcept	CON	IN	inherited
TheTargetAttribute	BA	IN	must be numeric
TheClassAttribute	BA	IN	must be categorial
NoOfIntervals	V	IN	positive integer > 1
ClosedTo	V	IN	one of *LEFT* or *RIGHT*
Label	V *List*	IN	optional
SampleSize	V	IN	optional; positive integer
TheOutputAttribute	BA	OUT	should be categorial

3.4.27 GroupingGivenMinCardinality

A grouping operator. Values of *TheTargetAttribute* are grouped under a certain label which is stored in *TheOutputAttribute*, which must be categorial. 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. *MinCardinalityType* decides if *MinCardinality* values are read as absolute values (integers) or relative values (real, between 0 and 1).

ParameterName	ObjType	Type	Remarks
TheInputConcept	CON	IN	inherited
TheTargetAttribute	BA	IN	must be numeric
MinCardinalityType	V	IN	*ABSOLUTE* or *RELATIVE*
MinCardinality	V	IN	positive
TheOutputAttribute	BA	OUT	should be categorial

3.4.28 GroupingGivenNoOfGroups

A grouping operator. Values of *TheTargetAttribute* are grouped under a certain label which is stored in *TheOutputAttribute*, which must be categorial. 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 the parameter *Label*.

ParameterName	ObjType	Type	Remarks
TheInputConcept	CON	IN	inherited
TheTargetAttribute	BA	IN	must be numeric
NoOfGroups	V	IN	positive integer
Label	V *List*	IN	optional
TheOutputAttribute	BA	OUT	should be categorial

3.4.29 UserDefinedGrouping

A grouping operator. Values of *TheTargetAttribute* are grouped under a certain label which is stored in *TheOutputAttribute*, which must be categorial. This operator creates groups of *TheTargetAttribute* according to specifications given by the user in *TheGroupings*, which is a list of values. Each of the values in the list in turn is a String that lists values of *TheTargetAttribute* which should be grouped together, separating them with a comma. Values not specified for grouping retain their original values. Values of *TheOutputAttribute* can be specified in the parameter *Label*.

ParameterName	ObjType	Type	Remarks
TheInputConcept	CON	IN	inherited
TheTargetAttribute	BA	IN	must be numeric
TheGroupings	V *List*	IN	see text
Label	V *List*	IN	optional
TheOutputAttribute	BA	OUT	should be categorial

3.4.30 UserDefinedGroupingWithDefaultValue

A grouping operator. Values of *TheTargetAttribute* are grouped under a certain label which is stored in *TheOutputAttribute*, which must be categorial. This operator creates groups of *TheTargetAttribute* values according to specifications given by the user in *TheGroupings*, which is a list of values. Each of the values in the list in turn is a String that lists values of *TheTargetAttribute* which should

be grouped together, separating them with a comma. Values not specified for grouping are grouped into default group *Default*. Values of *TheOutputAttribute* can be specified in the parameter *Label*.

ParameterName	ObjType	Type	Remarks
TheInputConcept	CON	IN	inherited
TheTargetAttribute	BA	IN	must be numeric
Default	V	IN	default group
Label	V *List*	IN	optional
TheOutputAttribute	BA	OUT	should be categorial

3.4.31 ImplicitErrorBasedGrouping

A grouping operator. Values of *TheTargetAttribute* are grouped under a certain label which is stored in *TheOutputAttribute*, which must be categorial. 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 has the same majority class. The resulting grouping minimizes the classification error. *TheClassAttribute* contains the labels of an example as in a Machine Learning setting. The parameter *SampleSize* gives a maximum number of learning examples for the external algorithm.

ParameterName	ObjType	Type	Remarks
TheInputConcept	CON	IN	inherited
TheTargetAttribute	BA	IN	must be numeric
TheClassAttribute	BA	IN	must be categorial
FullMerge	V	IN	one of *YES* or *NO*
SampleSize	V	IN	optional; positive integer
TheOutputAttribute	BA	OUT	should be categorial

3.4.32 ErrorBasedGroupingGivenMinCardinality

A grouping operator. Values of *TheTargetAttribute* are grouped under a certain label which is stored in *TheOutputAttribute*, which must be categorial. This operator merges the values of *TheTargetAttribute* into groups with the cardinality above the given threshold *MinCardinality*. *MinCardinalityType* decides if *MinCardinality* values are read as absolute values (integers) or relative values (real, between 0 and 1). The grouping is performed with respect to *TheClassAttribute*, but unlike implicit grouping, groups with a single majority class are further merged if they do not have the required cardinality. This will increase the classification error. *TheClassAttribute* contains the labels of an example as in a Machine Learning setting. The parameter *SampleSize* gives a maximum number of learning examples for the external algorithm.

ParameterName	ObjType	Type	Remarks
TheInputConcept	CON	IN	inherited
TheTargetAttribute	BA	IN	must be numeric
TheClassAttribute	BA	IN	must be categorial
SampleSize	V	IN	optional; positive integer
MinCardinalityType	V	IN	*ABSOLUTE* or *RELATIVE*
MinCardinality	V	IN	positive
TheOutputAttribute	BA	OUT	should be categorial

3.4.33 ErrorBasedGroupingGivenNoOfGroups

A grouping operator. Values of *TheTargetAttribute* are grouped under a certain label which is stored in *TheOutputAttribute*, which must be categorial. 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 the parameter *Label*. *TheClassAttribute* contains the labels of an example as in a Machine Learning setting. The parameter *SampleSize* gives a maximum number of learning examples for the external algorithm.

ParameterName	ObjType	Type	Remarks
TheInputConcept	CON	IN	inherited
TheTargetAttribute	BA	IN	must be numeric
TheClassAttribute	BA	IN	must be categorial
NoOfGroups	V	IN	integer > 1
Label	V *List*	IN	optional
SampleSize	V	IN	optional; positive integer
TheOutputAttribute	BA	OUT	should be categorial

3.5 Other Operators

3.5.1 ComputeSVMError

A special evaluation operator used for obtaining some results for the regression SVM. Values in *TheTargetValueAttribute* are compared to those in *ThePredictedValueAttribute*. The average loss is determined taking the asymmetric loss function into account. That is why the SVM parameters are needed here as well. **Note** that they must have the same value as for the operator `SupportVectorMachineForRegression`, which must have preceded this evaluation operator in the chain.

ParameterName	ObjType	Type	Remarks
TheInputConcept	CON	IN	inherited
TheTargetValueAttribute	BA	IN	actual values
ThePredictedValueAttribute	BA	IN	predicted values
LossFunctionPos	V	IN	(same values
LossFunctionNeg	V	IN	as in SVM-
Epsilon	V	IN	ForRegression)

3.5.2 SubgroupMining

A special operator without output on the conceptual level. The output of the algorithm is a textual description of discovered subgroups which will be printed to the compiler output (log file). The operator is only applicable to a table which is suitable for spatial subgroup discovery. Thus, *ThePredictingAttributes* must only contain categorial data. Therefore only features with a finite (and small) number of distinct values should be selected.

TheTargetAttribute and *TheKey* must belong to *TheInputConcept*; *TheKey* must refer to the primary key column. *ThePredictingAttributes* are used to learn from. *TargetValue* is one value from *TheTargetAttribute*. *SearchDepth* limits the search for generating hypotheses. *MinSupport* and *MinConfidence* give minimum values between 0 and 1 for support and confidence of the generated subgroups. *NumHypotheses* specifies the number of hypotheses to be generated. *RuleClusters* is a boolean parameter specifying whether or not clustering should be performed on the generated rules.

ParameterName	ObjType	Type	Remarks
TheInputConcept	CON	IN	inherited
TheTargetAttribute	BA	IN	
TheKey	BA	IN	
ThePredictingAttributes	BA *List*	IN	
TargetValue	V	IN	from TheTargetAttribute
SearchDepth	V	IN	positive integer
MinSupport	V	IN	real between 0 and 1
MinConfidence	V	IN	real between 0 and 1
NumHypotheses	V	IN	positive integer
RuleClusters	V	IN	one of *YES*, *NO*