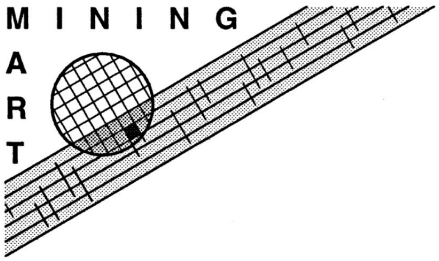


Enabling End-User Datawarehouse Mining
Contract No. IST-1999-11993
Deliverable No. D11.3

Best Practices Report

Experiences with using the Mining Mart system
Deliverable D11.3

Olaf Rem and Marten Trautwein

Perot Systems Nederland B.V.
P.O. box 2729, NL-3800 GG Amersfoort, The Netherlands
{Olaf.Rem, Marten.Trautwein}@ps.net

December 20, 2002

Abstract

This best practices report should be helpful in supporting future users with using the Mining Mart system. The main focus is on the Mining Mart preprocessing operators. The Mining Mart system provides an environment for advanced preprocessing. It stores meta data about business data and preprocessing chains, thus, making it possible to build a repository of cases, which may be shared and reused. It also supports making more intelligent choices during preprocessing by providing learning operators, which may be used within this stage.

In this report we have provided information, ideas and experiences with respect to working with the early prototype of the Mining Mart system. We constructed an extensive overview of general preprocessing problems from the literature. For a selection of these common preprocessing problems we describe our experiences in dealing with them using the Mining Mart system. We also describe the application of combinations of operators, which can be seen as "templates" or design patterns for solving specific problems.

We have found the Mining Mart system to be a valuable tool for preprocessing data. With its meta data model and user interface it is possible to conveniently create a reusable repository of preprocessing cases. From our experiences with the current Mining Mart system we expect that early adopters will already benefit from this system and that future users will do so even more.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This best practices report is one of the deliverables of workpackage 11 ("Exploitation and Transfer of Results") in the Mining Mart project. This report should be helpful in supporting future users with using the (prototype) Mining Mart system. The main focus is on the Mining Mart preprocessing operators. Other important aspects of the Mining Mart system are underexposed and only briefly addressed in Chapter 8. The meta model, concept hierarchy and reuse of cases form an integral part of the Mining Mart system, but are only mentioned indirectly as a consequence of the chose strategy in the construction of this report. We expect that the intuitive nature of these aspects eliminates the need for an as elaborate discussion as devoted to the preprocessing operators.

In general "best practices" provide practical solutions to common problems in a certain field. These solutions have proved themselves in the real world. Useful best practices need a certain level of abstraction for generalizing both the problem and the solution.

In this report we focus on data preprocessing best practices using the Mining Mart system. This means discussing how common preprocessing problems (in data selection, data cleaning and data transformation) can be handled best using the Mining Mart system.

In preparing this document the following strategy was chosen:

- Provide an overview in the literature of common preprocessing problems. This overview has become Appendix A.
- Build up experience by using the various operators and see how they work. A list of currently available operators with screen shots is provided in Appendix B.
- Make a selection of preprocessing problems (or categories) and describe how the Mining Mart system can be used in solving these problems. If possible find generalizations of the preprocessing solutions that can

be presented as templates or design patterns. These descriptions have become Chapters 3 to 7.

We have used an early prototype version of the Mining Mart system. During our practices minor and major shortcomings to the initial system were identified. These were reported to the responsible parties and most of these shortcomings were overcome in subsequent fixes. In some cases the functionality of the prototype system was extended. At the time of writing this report some operators defined in [Eul02] were being incorporated in the Mining Mart prototype. The added operators included a general feature construction operator, various feature selection operators, two discretization operators and some learning operators. (We have sometimes taken an advance on these additions and described the use of operators before they were present in the prototype.) The creation of this best practices report, thus, greatly contributed to the maturation of the Mining Mart system. We believe the current prototype system will already prove profitable to the early adopters. The larger public will definitely benefit from a future matured Mining Mart system.

The next chapter provides an overview of the Mining Mart system. Chapters 3 through 7 focus on common preprocessing problems and how to solve these using the Mining Mart system. Chapter 8 lists tips for the early adopters of the prototype Mining Mart system. While practicing with the prototype system, we have found these simple tricks and conventions of great value. Chapter 9 provides an analysis of the strong and weak points of the Mining Mart system and lists some recommendations for future versions. The report has two appendices. The first is the result of a literature study in data preprocessing. The used resources are mentioned in the preceding bibliography. The second appendix gives a brief overview of the available operators in the Mining Mart system.

Chapter 2

Overview Mining Mart System

This chapter gives a brief overview of the goals, principles and architecture of the Mining Mart system.

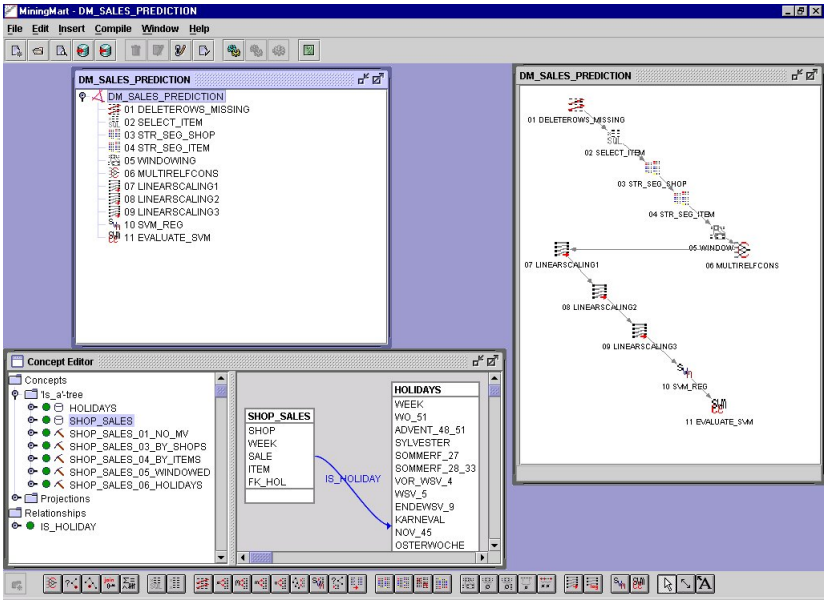
2.1 Goal

The success criteria for the Mining Mart project are

- Create a user-friendly access to data mining for non-expert users.
- Speed-up the discovery process.
- Minimize the amount of data kept within KDD procedures.
- Improve the quality of mining by improving the quality of data.

The Mining Mart system developed in workpackage 12 plays a crucial role in fulfilling these criteria. Workpackage 12 depends heavily on the work done in other workpackages and integrates many results of these workpackages into one system.

The main objective for the Mining Mart system is to provide a user-friendly user interface (see Figure 2.1) for enhanced preprocessing of data for a knowledge discovery task. It should also be easy to make changes to the preprocessing steps and then re-run the case. Further it should be possible with the system to re-apply preprocessing to different data (but with the same structure as the original data). A last important goal is that it should be possible to import and export cases, thus enabling parties to exchange cases with each other.



This screen shot of the Mining Mart HCI (human computer interface) depicts three internal windows. The upper and right windows (the Chain Editor) show the operators and the order in which they are applied. The lower left window (the Concept Editor) shows concepts and relationships that are inputs or outputs for the operators. At the bottom of the screen, buttons represent the different operators that can be added.

Figure 2.1: Screen shot of the Mining Mart HCI.

2.2 Principles

A key part in the Mining Mart system is the storage of meta data. This is data about the data itself and about the preprocessing process. The Mining Mart Meta Model (also known as the M4) defines the meta data model used by the Mining Mart system. It consists of several objects, including: Case, Step, Operator, Parameter, Concept, Relationship and BaseAttribute. The meta data allows to easily adapt and reuse cases.

The Mining Mart system contains various operators for preprocessing. New operators are still being added to the system. The Mining Mart project recognizes that applying machine learning does not have to be restricted to the learning stage: it can also be applied in the preprocessing stage. Currently the system provides learning operators for dealing with missing values and for feature construction. Applying learning operators in the preprocessing stage can help to improve the quality of the data.

Working with the Mining Mart system typically involves the following steps:

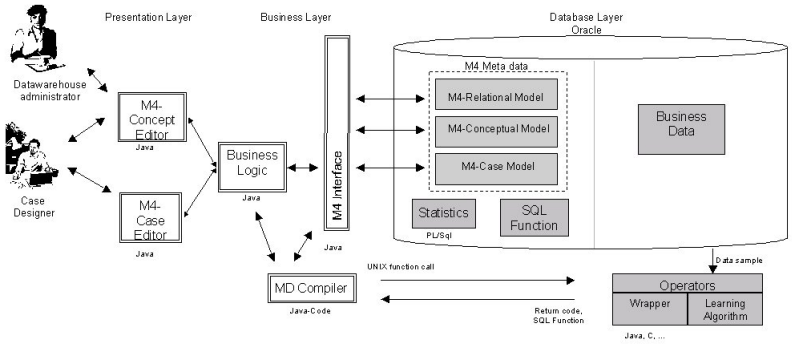
- Create the conceptual level and connect it to the business data. First, we create a concept and base attributes for every relevant table in the business data. Next, we define the relationships between concepts. These concepts and relationships define the initial conceptual model. This conceptual model represents the business data that should be preprocessed. This business data should be stored in a relational database (currently only Oracle is supported).

- Define a chain of preprocessing operators. The Mining Mart system provides a number of operators that can be used for preprocessing. For every operator an input concept should be selected and some additional information may be required. The output of an operator is either a new concept or a new base attribute depending on the type of operator. By applying more operators sequentially a preprocessing chain is defined.

- Compile the chain. This step executes the defined preprocessing chain. Each operator in the chain is executed. The execution of an operator results in the generation of SQL code and in a set of views (and sometimes tables) The generated SQL code is stored in the meta data, the views and tables are stored in the business data. The original business data tables themselves are not altered. In general the result of a preprocessing chain is one (or maybe more) view(s) in the business data schema that can be used as input for a separate mining tool.

2.3 Architecture

The Mining Mart system architecture (see Figure 2.2) consists of the following main components: Concept Editor, Chain Editor, Compiler, meta model schema and business data schema. The Concept Editor and Chain Editor act closely together and are both part of the Mining Mart system HCI (human computer interface). The HCI is written in Java. It uses the so-called M4Interface to access the meta model schema and the business data schema. Part of the M4Interface is implemented on the client and another part uses Enterprise Java Beans running in a JBoss application server. The Compiler is implemented as an RMI server and its methods are called by the HCI. Because various parts of the system are implemented using Java (HCI, Compiler, M4Interface) it should be possible to use the system on various operating systems, like Windows, Unix, Linux or Mac OS. The operators in the project, however, were required to work on Unix. Although many will work on for example Windows and Linux, others may not. Not all combinations of operators and operating systems have been thoroughly tested at this point in time. The currently most used configuration is to have both the HCI and the JBoss application server running on a Windows



Schematic view of the Mining Mart architecture. The Concept Editor and Case Editor are part of the HCI. The M4 Interface provides a Java object interface to the M4 and is divided over the client (Java Swing) and the application server (JBoss). The Compiler (Java RMI server) executes operators and creates resulting tables and views. The database (Oracle) contains the M4 and the business data.

Figure 2.2: Mining Mart architecture.

(NT or 2000) client and to have the Compiler and Oracle database running on a Unix (Solaris) server.

2.4 Further Information

There are various other sources available that provide more information about the Mining Mart project and the Mining Mart system. A good place to start is the Mining Mart website[1] which offers a good overview of the available documentation. Here also many documents can be downloaded directly.

The Mining Mart approach is described in: [MoS02], [ZKV01] and [KZV00]. Further information about the Mining Mart system can be found in: [PeF02] (requirements, system overview), [Eul02] (Compiler and operators), [LaR02] (M4Interface), [VKZ01] (the Mining Mart Meta Model), [Zu01b] (Compiler) and [Zu01a] (M4 schema).

[1] *http://www-ai.cs.uni-dortmund.de/FORSCHUNG/PROJEKTE/MININGMART/index.eng.html*

Chapter 3

Data Selection

3.1 Problem Description

The Mining Mart system assumes that all business data that is to be used in a case is stored in one schema in an Oracle database. Any data selection or preparation that has been applied to the business data to get it into this form is beyond the scope of the system. When using this data within the Mining Mart system it is, however, still very likely that some form of data selection will have to be applied.

Selecting data means reducing the amount of data available for a case by reducing the depth (that is, the number of rows) or the width (that is, the number of columns). It may be convenient to work only with part of the data. In general a balance needs to be found between what can be done and the time available. As the amount of data may considerably influence the performance of any data preparation or analysis system it may be a good idea to work only with part of the data. A key point to keep in mind when taking a sample of the data is, is the sample still representative enough for the original data? Using domain knowledge it may become clear that certain data is not at all relevant for the learning task at hand. This may be another reason to reduce data. Also the data exploration stage may give reasons to remove data (for example because of sparsity of data, noisy data or constants).

In the Mining Mart system the user can specify a sample to use less data, but samples are also important in using learning operators. All these operators work on a sample of the original data. The user can specify the sample size for these operators. For the learning operators to be effective, the specified sample should not be too small.

In this chapter we will further focus on how data can be selected (or reduced) using the Mining Mart system.

3.2 Reducing Depth

With reducing depth we mean reducing the number of records. This can be done using the row selection operators: RowSelByQuery, RowSelByRandom and DelRecWMVal.

The RowSelByQuery is a loopable operator that allows selection of rows by specifying a query. For one or more base attributes of a concept one can specify a condition and thus restrict the number of rows that will be available in the output concept. For example, one could select only employees with high salaries by specifying *Salary > 100000*. The SQL-operator parameter can contain Oracle comparison operators like: IN, NOT IN, =, !=, >, <, LIKE. It is also possible to specify more conditions per attribute by adding another loop and using the same attribute again with a different condition. Adding more conditions by adding loops effectively adds an "AND" to the resulting SQL statement. Note that with this dialog it is not possible to build SQL statements containing "OR" operators.

The RowSelByRandom, as could be expected from the name, randomly selects rows from a table. With the "How Many Rows" parameter one can specify how many rows *approximately* should be selected from the table. More precise it specifies a probability per row to be selected. This probability is defined by the "How Many Rows" parameter and the total number of rows. The number of selected rows may deviate a bit from the number of rows specified by the "How Many Rows" parameter. The operator is based on the java.util.Random class.

The operator for deleting missing values (DelRecWMVal) is a specialized row selection operator. It only selects rows where the specified attribute is not null. It is further described in Section 4.2.

3.2.1 Sample Size, Variability and Confidence

When taking a sample of the data an important point is if the sample is still representative enough. In other words: is the variability in the sample close enough to the variability in the original data? But when is the data sample too small? When is it large enough? The variability in a sample is measured by the variance. In general the changes in the variance of a sample will become smaller when the sample size increases. Repeatedly calculating the variance for an increasing sample size and comparing it with the variance of the original data can give a certain confidence level that the variability of the variable is caught. The number of positive tests needed for a certain confidence and equal variability level is given by the simplified formula $n = \log(1-c)/\log(c)$ where n is the number of tests and c is the confidence and variabilitylevel ([Pyl99]). For example, to obtain a 95% confidence that 95% of the variability has been captured one needs 59 consequative positive tests.

Now this is a very exact way of getting a certain confidence, but unless

one has a tool that calculates the variance for increasing sample sizes and determines the confidence level from that it is not very practically applicable. At this time the Mining Mart system does not have an operator that determines the required sample size that is needed to obtain a certain confidence level. However, the point we want to make is that the change in variance with changing sample size is important. It may not be practical to determine the variance for a large number of sample sizes, but it is very much feasible to do this for a small number of samples. The RowSelectionByRandom operator in combination with the *update statistics* menu item can be used for this. Update statistics determines basic statistics for the specified concept including the variance. These values can be viewed with the *show statistics* menu item. Creating random samples of increasing size (say 10%, 20%, 30%,...,100%) and comparing the variance will not give a clearly defined confidence level, but at least it will give some feeling if the variability of the sample resembles that of the original data.

The variance can only be determined for numerical attributes, so for nominal variables it does not work. An alternative measure that can be used here is to look at the change in the relative occurrence of values with increasing sample size. Suppose for example that a variable representing employee skill can have values "analyst", "specialist", and "senior specialist". Suppose further that for a certain data set these values occur for respectively 47%, 35% and 18% of the instances. When using a small sample the relative occurrence will likely be different, but as the sample size increases it will converge to what was present in the original data set. The update statistics option will also count the number of occurrences of different values for a certain attribute. It will not show the relative proportion of the values however, so this must be done manually. This is obviously not an ideal way of determining sample size, but if no other tool is at hand it could be used to decide on sample size.

3.3 Reducing Width

Another form of data selection is selecting features. There are two basic ways to select features in Mining Mart:

1. When connecting a concept to a database object only select relevant columns.

2. Use a feature selection operator.

Concepts of type DB (database) are directly based on a database object. They determine which data is available for preprocessing within the Mining Mart system. By only creating base attributes for those columns that are relevant one is in effect doing feature selection. A small example may illustrate this. Consider a table customer with columns id, name, date

of birth, age, type, nr_items_purchased, value_items_purchased. On the conceptual level one would create a concept Customer and base attributes that may be mapped to columns. In this case the case designer decides not to create a base attribute for the date of birth field as the (derived) age field is considered more useful.

The second way of selecting features is by using an operator. At the time of writing the only implemented operator which could be used for feature selection is MultiRelationalFeatureConstruction. It allows to combine features from several concepts into one output concept. The features may be selected from the input concepts. Several other feature selection operators have been defined already for the Mining Mart system (see [Eul02]), but at this time are waiting to be implemented into the Mining Mart HCI. These are:

- FeatureSelectionByAttributes. This operator simply allows the user to select certain features from the input concept.
- StochasticFeatureSelection. Uses a stochastic correlation measure to select a subset of the attributes.
- GeneticFeatureSelection. Uses a genetic algorithm to select a subset of the attributes. Individuals of the population are evaluated using C4.5.
- SGFeatureSelection. Here first StochasticFeatureSelection is applied and then GeneticFeatureSelection is applied.

The last three operators are learning operators that work with a sample of the data. They can be useful when it is not clear (from domain knowledge) which features are the most informative.

Chapter 4

Missing Values

4.1 Problem Description

Missing values frequently occur in operational data. That values are missing can have various causes. They can be for example the result of faults in the technical measuring infrastructure, human errors or the variable may not have been measured in the first place.

Although many modeling tools have some way of dealing with incomplete data, these mechanisms are not always robust and in general the tools will benefit from receiving cleaned data.

Various preprocessing methods exist to handle missing values including:

- Ignore (delete) records with missing values.
- Fill in the missing values manually.
- Replace the missing value with a constant, the variable mean, or the most probable value.

One should always be careful in handling missing values. When records with missing values are deleted information is removed. This information could well be critical to the discovery task. Typically in fraud detection lack of information is a valuable indication of interesting patterns. Also when replacing missing values with default values one should realize that this may damage the data set structure.

In the following sections support for dealing with missing values given by the Mining Mart system will be discussed.

4.2 Delete Records with Missing Values

The Mining Mart system has an operator called DelRecWMVal that deletes records with missing values for a given attribute. Its use is simple and

straightforward: specify the input concept, target attribute and output concept and as a result the output concept is automatically created and all records that have a NULL value for the given target attribute are removed. It is not a loopable operator so to delete records with missing values in other attributes the operator needs to be reapplied.

4.3 Basic Replacement Operators for Missing Values

There are currently four operators in Mining Mart that do a basic replacement of missing values:

- AssDefValue: Assigns a specified (default) value to all missing values of an attribute.
- AssAvgValue: Assigns the average value of all present values to all missing values of an attribute. If statistics are stored for the concept these are used to find the average value otherwise the average is calculated. This operator of course only works with numerical attributes.
- AssMedValue: Assigns the median value of all present values to all missing values of an attribute. The median value is the middle value taken from the list of ordered values. If the list is even it is the average of the two middle values. Statistics should have been calculated for the concept for this operator to work. This operator only works with numerical attributes.
- AssModValue: Assigns the modal value to all missing values of an attribute. The modal value is the most frequently occurring value in the list of values. This operator also requires statistics to have been calculated for the concept.

One must realize when replacing missing values with these operators that one is actually distorting the data set. One is saying that for every missing value every time the same value was measured (which is very unlikely). Therefore these operators should be used with care.

The average, median and modal operators are operators to find the "center value" for an attribute. So effectively one is seeking to replace missing values with the center value. The idea is that this "center value" is the best (or least worst) estimate for the missing values.

Depending on the form of the distribution of values one operator may better represent the "center value" than the other. In a strictly normal distribution the average, median and modal values are the same. In a skewed (not symmetric) distribution the median better represents the center value than the average. In other cases the modal value may be preferred.

4.4 Advanced Replacement Operators for Missing Values

The Mining Mart system contains three additional learning operators for dealing with missing values. These are learning operators that build a model using selected predictive attributes and the values that are present in the attribute that also has missing values. The resulting model is then applied to predict the values that were missing. The operators mentioned here work with a sample of the business data. The user can specify how large the sample should be. This provides an efficient way for dealing with large amounts of data.

The learning operators available are:

- Missing values with decision tree (MvwDecTree).
- Missing values with decision rules (MvwDecRules).
- Missing values with SVM for regression (MvwRegSVM).

The decision tree operator uses the C4.5 algorithm to build a decision tree. Predicting attributes are input for the model and the model then predicts the value of the attribute that is missing. The predicting attributes may be nominal or numeric and the target attribute should be nominal.

In general learning operators may benefit from having input values normalized. It prevents one predicting variable to outweigh another. Therefore it is good practice to consider normalizing predicting attributes before applying a learning operator.

The operator has a pruning confidence parameter (value 0-100). Higher values will lead to more pruning. Very high pruning values may lead to a model that is learning noise (see Chapter 6). One could for example apply the decision tree operator or decision rules operator multiple times (using the looping functionality) specifying a different pruning confidence (say 60, 70, ..., 100) with a different output attribute. In order to obtain a better model the sample size could be increased or the pruning setting could be changed.

The decision rules operator also uses the C4.5 algorithm. It determines a set of rules from the predicting attributes and uses those to predict the value of the target attribute. It can work with the same data types as the decision tree operator.

For the SVM in regression mode the predicting attributes should be numeric.

How well applying these operators works will depend among other things on the amount of data, the percentage of missing values for the target attribute and the relations between the predicting attributes and the target attribute.

Chapter 5

Time Series

5.1 Problem Description

Series variables always are at least two-dimensional, although one of the dimensions may be implicit. Series data enfolds their information in the ordering of the data. Preserving the ordering is the main reason that series data has to be prepared differently from non-series data.

Any series shape can be thought of as being constructed from simple wave forms, each of a separate single frequency. The most common type of series variable is a time series, in which a series of values of some event are recorded over a period of time. The issues and techniques described about time series also apply to any other displacement series.

Mining Mart provides five operators specifically for manipulating series. Windowing is a basic technique to create an interval in the series. SimpleMovingFunction, WeightedMovingFunction and ExponentialMovingFunction corresponds to the simple, weighted, and exponential moving average technique for smoothing series, respectively. SignalToSymbolProcessing is a time series generalization/abstraction operator.

This chapter uses a sales concept as a running example. The "Sales" concept contains four attributes: shop ID (of type numeric), year + week number (of type time), sale number (numeric), and item ID (numeric). In Table 5.1, we present a data fragment of the corresponding database table. The description of the columns as well as the connection with the concept attributes should be clear from the context. The scenario that is described in the text is shown graphically in Figure 5.1 on page 32.

The section below describes how to handle the non-uniform displacement in the time series. Next, the various time series techniques the Mining Mart system supports are illustrated. We conclude this chapter with a brief recapitulation.

SHOP	WEEK	SALE	ITEM
55	199548	9	3269
55	199548	2	13150
55	199549	4	3269
55	199549	3	13150
55	199550	5	3269
55	199550	3	13150
55	199551	10	3269
55	199551	4	13150
55	199552	4	3269
55	199552	4	13150
141	199548	7	3269
141	199548	2	13150
141	199549	11	3269
141	199549	3	13150
141	199550	4	3269
141	199550	5	13150
141	199551	7	3269
141	199551	4	13150
141	199552	4	3269
141	199552	3	13150

Table 5.1: Data fragment of the running time series example ("Sales")

5.2 Handling Non-uniform Displacement

The time series operators presuppose that the time variable is uniformly increasing. The time variable in our running example (Table 5.1) is not uniformly increasing. This non-uniform displacement has the following two causes.

1. The time attribute makes a huge jump at the end of the year, for example, from 199552 to 199601 (not shown in the data fragment). Therefore a mapping has to be created from the non-uniform displacement variable to a uniform displacement variable.

2. The "Sales" concept contains sales data from multiple shops and multiple items (two shops and two items are shown in the data fragment). The sales data of a week are aggregated per shop and per item. The total sales of an item in a shop in a particular week is recorded. Each week appears approximately as many times as the number of shops multiplied by the number of items. As a consequence only within a shop/item segment the displacement can be uniformly increasing.

Conceptually, one would first segment the "Sales" concept according to the values of the shop and item attributes. Next one would create for each segment a mapping from the non-uniform displacement variable to a uniform

displacement variable. In practice, however, we can minimize the number of operators applied by first mapping and next segmenting.

The mapping concept, "Uniform Time Scale", contains two attributes: week ID (time) and time step (time). In Table 5.2, we present a data fragment of the corresponding database table. The description of the columns as well as the connection with the concept attributes should be clear from the context.

WEEK_ID	STEP
199548	1
199549	2
199550	3
199551	4
199552	5
199601	6
199602	7
199603	8
199604	9

Table 5.2: Data fragment of the uniform time scale mapping ("Uniform Time Scale")

The JoinByKey operator can now be used to create a new concept which joins the "Sales" concept and the "Uniform Time Scale" concept. The resulting concept "Uniform Sales" is an extension of the "Sales" concept with a uniformly increasing displacement variable. The new concept contains five attributes: shop ID (numeric), year + week number (time), sale number (numeric), item ID (numeric) and time step (time). In Table 5.3, we present a data fragment of the corresponding database table. The description of the columns as well as the connection with the concept attributes should be clear from the context.

The "Uniform Sales" concept contains sales data of multiple shops and items. As a consequence only within segments of one shop and one item the displacement is uniformly increasing. Thus the concept should be segmented per shop and item to get an overview of the weekly sales per shop and item. Therefore, we apply the segmentation operator SegmentationStratified twice (once for the shop and once for the item attribute) to the "Uniform Sales" concept. The resulting concept "Uniform Segmented Sales" contains the remaining three attributes of the unsegmented concept. The attributes shop and item have been removed from the concept. In contrast the new concept contains four segments, that is, one column set for each shop/item pair.

In Table 5.4, we present a data fragment of the corresponding database tables. The description of the columns as well as the connection with the concept attributes should be clear from the context.

SHOP	WEEK	SALE	ITEM	STEP
55	199548	9	3269	1
55	199548	2	13150	1
55	199549	4	3269	2
55	199549	3	13150	2
55	199550	5	3269	3
55	199550	3	13150	3
55	199551	10	3269	4
55	199551	4	13150	4
55	199552	4	3269	5
55	199552	4	13150	5
141	199548	7	3269	1
141	199548	2	13150	1
141	199549	11	3269	2
141	199549	3	13150	2
141	199550	4	3269	3
141	199550	5	13150	3
141	199551	7	3269	4
141	199551	4	13150	4
141	199552	4	3269	5
141	199552	3	13150	5

Table 5.3: Data fragment of the running uniformly increasing time series example ("Uniform Sales")

5.3 Windowing

The Windowing operator is applicable to uniform displaced time series data. Windowing takes two attributes from the input concept: the time stamps and the values. Each row of the output concept gives a time window. Two time stamps define the beginning and ending of each time window. Further, there will be as many value attributes as specified by the window size.

The running example "Uniform Segmented Sales" concept contains segmented sales data per shop and item. We apply the windowing operator to produce a new output concept "Sales Window3" with five attributes (start week, end week, week 1, week 2, week 3) based on a window of length 3 (for step attribute of type time) and a displacement distance of 1 (week). The corresponding data fragment for one segment (shop 55, item 3269) is given in Table 5.5.

Windowing is always assigned to time periods. The output values of windowing are the actual values in the time period. The week numbers of the time period are helpful information to put the results into perspective. Therefore we map the time step to the year + week number of the time steps. The JoinByKey operator in combination with the "Uniform Segmented Sales" is well suited to perform this task. Finally, we apply the Unsegment operator twice (for the segmented attributes shop and item).

Shop 55 - item 3269 segment

WEEK	SALE	STEP
199548	9	1
199549	4	2
199550	5	3
199551	10	4
199552	4	5

Shop 55 - item 13150 segment

WEEK	SALE	STEP
199548	2	1
199549	3	2
199550	3	3
199551	4	4
199552	4	5

Shop 141 - item 3269 segment

WEEK	SALE	STEP
199548	7	1
199549	11	2
199550	4	3
199551	7	4
199552	4	5

Shop 141 - item 13150 segment

WEEK	SALE	STEP
199548	2	1
199549	3	2
199550	5	3
199551	4	4
199552	3	5

Table 5.4: Data fragment of the running uniform segmented time series example "Uniform Segmented Sales"

START	END	WK1	WK2	WK3	Denotes that the weekly sales
1	3	9	4	5	in periode 1995-48 to 1995-50 are 9, 4, and 5, respectively
2	4	4	5	10	in periode 1995-49 to 1995-51 are 4, 5, and 10, respectively
3	5	5	10	4	in periode 1995-50 to 1995-52 are 5, 10, and 4, respectively

Table 5.5: Data fragment of the Windowing example ("Sales Window3")

The resulting concept is the an unsegmented concept, "Unsegmented Sales Window3", with seven attributes. The corresponding data fragment for is given in Table 5.6.

5.4 Simple Moving Average

The Mining Mart operator SimpleMovingFunction implements the simple moving average technique. The SimpleMovingFunction operator combines windowing (see Section 5.3) with the computation of the average value in each window. The average of the values in the window is stored in one output attribute. The user specifies the size of the window and the displacement distance in the window. The SimpleMovingFunction operator assigns the average value of a window to the time interval associated with the window.

The running example "Uniform Segmented Sales" concept contains seg-

SHOP	ITEM	START	END	WK1	WK2	WK3
55	3269	199548	199550	9	4	5
55	3269	199549	199551	4	5	10
55	3269	199550	199552	5	10	4
55	13150	199548	199550	2	3	3
55	13150	199549	199551	3	3	4
55	13150	199550	199552	3	4	4
141	3269	199548	199550	7	11	4
141	3269	199549	199551	11	4	7
141	3269	199550	199552	4	7	4
141	13150	199548	199550	2	3	5
141	13150	199549	199551	3	5	4
141	13150	199550	199552	5	4	3

Table 5.6: Data fragment of the unsegmented Windowing example ("Unsegmented Sales Window3")

mented sales data per shop and item. We applied the SimpleMovingFunction operator to produce a new output concept "Sales-SMA3" with three attributes (start week, end week, sma3 sale) based on a window of length 3 (for the time step attribute of type time) and a displacement distance of 1 (week). The corresponding data fragment for one segment (shop 55, item 3269) is given in Table 5.7.

START	END	SALE_SMA3	Denotes the sales average
1	3	6	of 9, 4, and 5 over the period 199548 – 199550
2	4	6.333	of 4, 5, and 10 over the period 199549 – 199551
3	5	6.333	of 5, 10, and 4 over the period 199550 – 199552

Table 5.7: Data fragment of the SimpleMovingFunction example

Moving averages are often used to remove noise and reduce the variance in a series. In order to reduce the variance, we would like to assign the computed average value of the SimpleMovingFunction operator to a particular time point (for example, the center) in the time period instead of to the time period as a whole. The JoinByKey operator in combination with the "Uniform Segmented Sales" and a window mapping concept is well suited to perform this task.

Suppose a mapping concept, "Window 3", that contains three time attributes: first window step, mid window step, and last window step. The concept[1] indicates that a window of size three has, three time points. The beginning of the window, the middle of the window and the end of the window. In Table 5.8, we present a data fragment of the corresponding database table. The description of the columns as well as the connection with the concept attributes should be clear from the context.

[1]The concept can in a trivial way be extended to windows of other sizes.

FIRST	MID	LAST
1	2	3
2	3	4
3	4	5
4	5	6
5	6	7

Table 5.8: Data fragment of the Window 3 mapping

Joining the "Uniform Segmented Sales" and "Window 3" concepts with the JoinByKey operator at the time step and the mid window step yields an auxiliary concept that places the "Uniform Segmented Sales" in the center of a window of size 3. Joining this auxiliary concept and the simple moving average concept ("Sales-SMA3") concept by means of the JoinByKey operator at the first window step and the window start yields a SMA3 smoothed uniform segmented sales concept with seven attributes: window start (i.e, the first window step), window end, time step (i.e, the mid window step), the last window step, the year + week number, the sales number and the simple moving average of the sales. The FeatureSelectionByAttributes operator finally can remove all auxiliary windowing attributes, resulting in a smoothed segmented sales concept, "Smoothed Uniform Segmented Sales", with three attributes: the year + week number, the actual sales number and the smoothed sales number. The corresponding data fragment for one segment (shop 55, item 3269) is given in Table 5.9.

WEEK	SALE	SALE_SMA3
199549	4	6
199550	5	6.333
199551	10	6.333

Table 5.9: Data fragment of the (SMA3) Smoothed Uniform Segmented Sales example

Finally, we apply the Unsegment operator twice (for the segmented attributes shop and item). The resulting concept is the unsegmented concept, "Unsegmented Smoothed Uniform Sales", with the two additional attributes shop ID and item ID.

5.5 Weighted Moving Average

The Mining Mart operator WeightedMovingFunction implements the weighted moving average technique. The WeightedMovingFunction operator combines windowing (see Section 5.3) with the computation of the weighted average value for the window. The weighted average of the window is stored in one output attribute. The user specifies the weights of the window val-

ues and the displacement distance in the window. The number of weights determines the size of the window. The weights must sum to 1. The WeightedMovingFunction operator assigns a weighted average value for a window to the time interval associated with the window.

The running example "Uniform Segmented Sales" concept contains segmented sales data per shop and item. We applied the WeightedMovingFunction operator to produce a new output concept "Sales-WMA3" with three attributes (start week, end week, wma3 sale) based on a window with weights 0.2, 0.3, and 0.5 of length 3 (for the time step attribute of type time) and a displacement distance of 1 (week). The corresponding data fragment for one segment (shop 55, item 3269) is given in Table 5.10.

START	END	SALE_WMA3	Denotes the weighted average
1	3	5.5	$(9 \times 0.2 + 4 \times 0.3 + 5 \times 0.5)$ over the period 199548 – 199550
2	4	7.3	$(4 \times 0.2 + 5 \times 0.3 + 10 \times 0.5)$ over the period 199549 – 199551
3	5	6	$(5 \times 0.2 + 10 \times 0.3 + 4 \times 0.5)$ over the period 199550 – 199552

Table 5.10: Data fragment of the WeightedMovingFunction example

Moving averages are often used to remove noise and reduce the variance in a series. In order to reduce the variance, we would like to assign the computed average value of the WeightedMovingFunction operator to a particular time point (for example the weighted center) in the time period instead of to the time period as a whole. The JoinByKey operator in combination with the "Uniform Segmented Sales" and the "Window 3" mapping concept (see Section 5.4) is well suited to perform this task.

Joining the "Uniform Segmented Sales" and "Window 3" concepts with the JoinByKey operator at the time step and the window step nearest to the weighted center of the window (that is, the last window step) yields an auxiliary concept that places the "Uniform Segmented Sales" in the weighted center of a window of size 3. Joining this auxiliary concept and the weighted moving average concept ("Sales-WMA3") concept by means of the JoinByKey operator at the first window step and the window start yields a WMA3 smoothed uniform segmented sales concept with seven attributes: window start (i.e, the first window step), window end, mid window step, time step (i.e, the last window step), the year + week number, the sales number and the weighted moving average of the sales. The FeatureSelectionByAttributes operator finally can remove all auxiliary windowing attributes, resulting in a smoothed segmented sales concept, "Smoothed Uniform Segmented Sales", with three attributes: the year + week number, the actual sales number and the smoothed sales number. The corresponding data fragment for one segment (shop 55, item 3269) is given in Table 5.11.

WEEK	SALE	SALE_WMA235
199550	5	5.5
199551	10	7.3
199552	4	6

Table 5.11: Data fragment of the (WMA235) Smoothed Uniform Segmented Sales example

Finally, we apply the Unsegment operator twice (for the segmented attributes shop and item). The resulting concept is the unsegmented concept, "Unsegmented Smoothed Uniform Sales", with the two additional attributes shop ID and item ID.

5.6 Exponential Moving Average

The Mining Mart operator ExponentialMovingFunction implements the exponential moving average technique. The ExponentialMovingFunction operator is a time series smoothing operator. The smoothed value for the window is stored in one output attribute. The distance determines the size of the window. The last value in the window is multiplied by the head weight. The first value is multiplied by the tail weight. The smoothed value becomes the first value of the next window. The head and tail weights must sum to 1. The ExponentialMovingFunction operator assigns a moving average value for a window to the final time point of the time interval associated with the window.

The running example "Uniform Segmented Sales" concept contains segmented sales data per shop and item. We applied the ExponentialMovingFunction operator to produce a new output concept "Sales-EMA55" with two attributes (ema55 time step, ema55 sale) based on a window with head and tails weights 0.55 and 0.45, respectively (for the time step attribute of type time) and a displacement distance of 1 (week), that is of window length 2. The corresponding data fragment is given in Table 5.12.

Moving averages are often used to remove noise and reduce the variance in a series. Exponential moving averages do not suffer from the delay problem. In contrast with the simple and weighted moving averages, exponential moving averages are already assigned to time points and not to time periods. All that remains to be done is to map the time step to the year + week number and add the actual sales. The JoinByKey operator in combination with the "Uniform Segmented Sales" is well suited to perform this task.

Joining the "Uniform Segmented Sales" and the concept "Sales-EMA55" concepts with the JoinByKey operator at the time steps yields a EMA55 smoothed uniform segmented sales concept with four attributes: time step, the year + week number, the actual sales number and the moving average of the sales. The FeatureSelectionByAttributes operator finally can remove

STEP	SALE_EMA55	
1	9	Denotes the actual sale of week 1995-48 by default
2	6.25	weighted average of actual sale of week 1995-49 and the weighted average of week 1995-48 ($4 \times 0.55 + 9 \times 0.45$)
3	5.5625	weighted average of actual sale of week 1995-50 and the weighted average of week 1995-49 ($5 \times 0.55 + 6.25 \times 0.45$)
4	8.003125	weighted average of actual sale of week 1995-51 and the weighted average of week 1995-50 ($10 \times 0.55 + 5.5625 \times 0.45$)
5	5.80140625	weighted average of actual sale of week 1995-52 and the weighted average of week 1995-51 ($4 \times 0.55 + 8.003125 \times 0.45$)

Table 5.12: Data fragment of the ExponentialMovingFunction example

the auxiliary time step attribute, resulting in a smoothed segmented sales concept, "Smoothed Uniform Segmented Sales", with three attributes: the year + week number, the actual sales number and the smoothed sales number. The corresponding data fragment for one segment (shop 55, item 3269) is given in Table 5.13.

WEEK	SALE	SALE_EMA
199548	9	9
199549	4	6.25
199550	5	5.5625
199551	10	8.003125
199552	4	5.80140625

Table 5.13: Data fragment of the (EMA55) Smoothed Uniform Segmented Sales example

Finally, we apply the Unsegment operator twice (for the segmented attributes shop and item). The resulting concept is the unsegmented concept, "Unsegmented Smoothed Uniform Sales", with the two additional attributes shop ID and item ID.

5.7 Signal to Symbol Processing

The Mining Mart operator SignalToSymbolProcessing implements the signal to symbol processing approach. This approach bridges the gap between numerical sensor data and symbol approaches to learning. The approach transforms a stream of numeric sensor measurements into a sequence of symbolic descriptions.

SignalToSymbolProcessing is a time series abstraction operator. Two

output attributes form the bounds of the interval. The average value and average increase of each interval is stored in one output attribute. The user specifies the tolerance that determines when the average increase, interpolated from the last interval, deviates thus that a new interval has to be created.

The SignalToSymbolProcessing operator creates time intervals in the time series based on a tolerance measure on the average increase in the time interval. The SignalToSymbolProcessing operator creates the largest time interval whose variance in increase is less than the tolerance. We applied the SignalToSymbolProcessing operator to produce a new output concept "Sales-S2S4" with four attributes (start time, end time, average value and average increase value) based on a tolerance of 4. The SignalToSymbolProcessing operator learned three intervals in the initial fragment of the running example. The corresponding learned data fragment for one segment (shop 55, item 3269) is given in Table 5.14.

START	END	VALUE	INCREASE	
1	1	9	0	Denotes that the average sale in week 1995-48 is 9
2	4	6.333	3	in period 1995-49 to 1995-51 is 6.333 and sale increased weekly with 3 on average
5	5	4	0	in week 1995-52 is 4

Table 5.14: Data fragment of SignalToSymbolProcessing example

The data fragment of the running example is too small to show major new insights, but serves to illustrate the possibilities of signal to symbol processing. In the running example the period of 1995-48 to 1995-52 is partitioned into three intervals. The first and last intervals show stable sales, for the trivial reason that they span just one week. The middle interval shows an increase in sale for the period 1995-49 to 1995-51.

SignalToSymbolProcessing is a time series abstraction operator that can be used to partition a time period in intervals of stability, growth and decline. In contrast with the various moving averages, signal to symbol processing is always assigned to time periods. The output values of signal to symbol processing are averages over the time period. The actual sales at the start (or end or both) of the time period are helpful information to put the results into perspective. All that remains to be done is to map the time step to the year + week number of the time steps and add the actual sales. The JoinByKey operator in combination with the "Uniform Segmented Sales" is well suited to perform this task.

Joining the "Uniform Segmented Sales" and the concept "Sales-S2S4" concepts with the JoinByKey operator twice[2] at the interval start and end

[2]Alas, without manual manipulation of the concept base attribute names, the current Mining Mart prototype can only join concepts once, due to name clashes in the underlying

time steps yields a uniform segmented sales concept with eight attributes: interval start time step (that is the time step of the first join), interval end time step (that is the time step of the second join), the interval start year + week number, the interval end year + week number, the interval start actual sales number, the interval end actual sales number, the average sale in the time period and the average increase in sale in the time period. The FeatureSelectionByAttributes operatorfinally can remove the auxiliary time step attributes, resulting in an abstracted segmented sales concept, "Abstracted Uniform Segmented Sales", with six attributes: the starting year + week number, the starting sales number, the ending year + week number, the ending sales number, the average sale and the average increase. The corresponding data fragment for one segment (shop 55, item 3269) is given in Table 5.15.

START	END	SALE_1	SALE_2	VALUE	INCREASE
199548	199548	9	9	9	0
199549	199551	4	10	6.333	3
199552	199552	4	4	4	0

Table 5.15: Data fragment of the Abstracted Uniform Segmented Sales example

Finally, we apply the Unsegment operator twice (for the segmented attributes shop and item). The resulting concept is the unsegmented concept, "Unsegmented Smoothed Uniform Sales", with the two additional attributes shop ID and item ID.

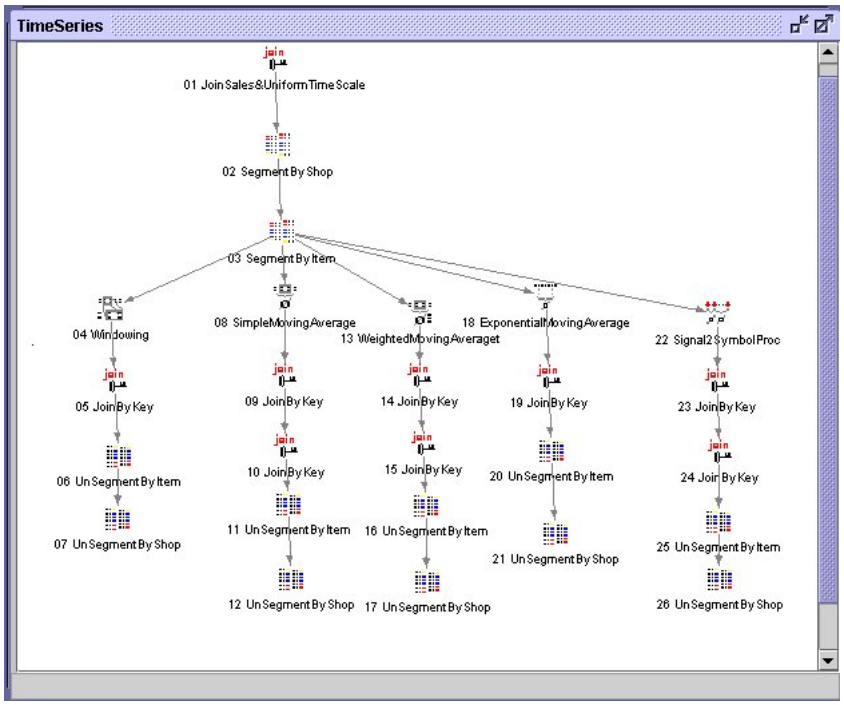
5.8 Summarizing

This chapter described the five time series operators in the Mining Mart system: one generic windowing operator, three moving average operators and one signal to symbol abstraction operator (the scenario is shown in Figure 5.1). All five operators require a uniformly increasing distributed displacement (time) variable. The following two operations were needed to fulfill this requirement.

- Join the "Sales" input concept with a "Uniform Time Scale" concept.
- Segment the "Sales" input concept according to the values of the identification attributes shop and item.

Series variables always are at least two-dimensional. Thus, all time series operators operate on a time base attribute and a value base attribute of an

database column names. Currently, only the actual sale at the start or the end of the interval can be added.



The figure shows the scenario that is described in this chapter. Several time series operators are applied to handle shop sales data.

Figure 5.1: Example of applying time series operators.

input concept. The time series operators yield a time period as output. The user specifies the size of the time period, except in the case of the Signal-ToSymbolProcessing operator which determines the size automatically.

The Windowing operator simply gave an overview of the values of the target attribute. Moving averages are often used to remove noise and reduce the variance in a series. In order to reduce the variance, we assigned the computed average value of the moving average operators to a particular time point in the time period instead of to the time period as a whole. The JoinByKey operator in combination with the "Uniform Segmented Sales" and a window mapping concept performed this task.

The Windowing and SignalToSymbolProcessing operators only apply to intervals of time. The results of these two operators were joined with the "Uniform Segmented Sales" to add the actual sales of the interval start and end times as a reference.

The FeatureSelectionByAttributes operator operator was used to remove all auxiliary time window attributes from the output concept. Finally we applied the Unsegment operator to combine the results of the various segments to one concept again.

Chapter 6

Noise

6.1 Problem Description

Operational data tend to be noisy: the data record errors and unusual values. There are many possible reasons for noisy data. Most mining routines have procedures for dealing with noisy data, but they are not always robust. Noisy data may be due to faulty data collection instruments that produced unusual values. There may have been human or computer errors at data transmission or entry.

Noise is related to other data cleaning preprocessing problems, like pollution, missing values, and empty values. Pollution is a result of people who try to stretch a system beyond its original intended functionality. This may be a result of a changing business environment while the data model remains fixed. Missing and empty values are discussed in Chapter 4. Chapter 5 handles about time series, where smoothing is one of the most common techniques to reduce noise in time series.

This chapter concerns noise in non-series data. We focus on learning techniques that predicts the value of an attribute from a set of predicting attributes. The Mining Mart system provides three learning operators for creating a support vector machine, a set of decision rules or a decision tree.

SupportVectorMachineForRegression: A Support Vector Machine is trained in regression mode using the predicting attributes.

PredictionWithDecisionRules: A set of decision rules is trained using the predicting attributes.

PredictionWithDecisionTree: A decision tree is trained using the predicting attributes.

6.2 Running Example

The three learning operators have a lot in common from the end-user's point of view. Each operator takes a set of predicting attributes as input to learn a user specified target attribute. The operators differ in the operator specific parameters, like the error threshold "epsilon" (Section 6.3) or the pruning confidence level (Section 6.4).

As a running example consider an "Employee" concept containing the attributes "age", "salary", "department", "hire date", "commission", and other (identification) attributes, like "employee number" and "name". One might try to predict the employee's commission from the learning attributes "age", "salary", "department", and "hire date".

The learning operators are only able to learn discrete values. Thus, the "commission" attribute has to be discretized. One may want to normalize (see Chapter 7) the "commission" attribute before discretization. Linear scaling, for instance, is a technique to limit the range of possible values of the attribute, which will simplify the construction of numeric intervals for the discretization operator.

The number of defined intervals in the discretization operators depends on the required degree of granularity. Typically, one would define equiwidth intervals, for example covering a range of 100 values ($0 – $100, $100 – $200, $200 – $300, ...). In special circumstances one might define, guided by domain knowledge, intervals of alternating sizes (for example $0 – $100, $100 – $250, $250 – $1000, ...).

After applying the normalization and discretization feature construction operators, the "Employee" concept contains the additional attributes "normalized commision" and "commission category". Next, any of the three learning operator may be applied to learn the normalized discretized target attribute "commission category".

6.3 Support Vector Machine

The SupportVectorMachineForRegression operator is a data mining operator. The operator trains a Support Vector Machine (SVM) in regression mode using the indicated predicting attributes. The indicated target attribute serves as the target function to train the Support Vector Machine. A new output attribute contains the predicted values for the target attribute.

A special evaluation operator, the ComputeSVMError operator, evaluates the result of the Support Vector Machine. Values in the target attribute are compared with the predicted values.

Both operators are loopable, feature construction operators. Thus, the user can specify multiple loops for one operator. The user can vary the SVM-specific parameters in each loop to determine the best learning model.

The user should cautiously examine the results and be aware of overfitting.

The goal of the SVM is to find a function that has at most "epsilon" deviation from the actual target values for all the training data. In other words, we do not care about errors as long as they are less than "epsilon". A normalisation parameter balances training error against generalisation error. The bias towards predicting too high or too low values is determined by the ratio between the positive and negative loss function parameters.

After applying the normalization and discretization feature construction operators to the running example, the "Employee" concept contains the additional attributes "normalized commision" and "commission category". We will predict the employee's commission from the learning attributes "age", "salary", "department", and "hire date" with a SVM. Thus, the SupportVectorMachineForRegression operator is applied next to learn the "commission category" target attribute.

Suppose the user specifies five loops for the operator. As input concept the user select the "Employee" concept. This parameter, by definition, stays the same in all loops. In each of the five loops the user has to select the "commission category" as target attribute, the attributes "age", "salary", "department", and "hire date" as predicting attributes. The user varies the allowed error parameter ("epsilon") stepwise in each loop, say from 0.10 in the first loop to 0.50 in the fifth loop, and specifies a separate output attribute for each loop, say "predicted commission category 0.10", "predicted commission category 0.20", ... The user will have to assign values for the remaining attributes (such as kernel type and sample size) and consistently use these values in all loops. Next the user applies the evaluation operator, ComputeSVMError, to evaluate the result of the Support Vector Machine. The user has to specify exactly as many loops for this evaluation operator and copy the corresponding values from the SupportVectorMachineForRegression operator for each loop. Then the operator compares the values in the target attribute with the predicate values.

6.4 Decision Rules and Decision Tree

The PredictionWithDecisionRules and the PredictionWithDecisionTree operators are two similar data mining operators. The former operator trains a set of decision rules using the indicated learning attributes. The latter operator trains a decision tree using the indicated learning attributes. The indicated target attribute of both operators contains the labels to be learned. In both cases, a new output attribute contains the predicted labels for the target attribute.

The user will have to compare manually the predicated values of the output attribute and the actual values of the target attribute.

Both operators are loopable, feature construction operators. Thus, the

user can specify multiple loops for each operator. The user can vary the pruning confidence level parameter in each loop to determine the best learning model. The user should cautiously examine the results and be aware of overfitting.

Both the decision rules and the decision tree operators learn labels, that is, the target attribute must be discrete. The Mining Mart system provides the discretization operator "NumericIntervalManualDiscretization" that maps a numeric attribute to a user specified category. The user specifies the mapping in terms of lower and upper bounds of the intervals and the values to which the intervals are mapped. Input values that do not belong to any interval are mapped to a user specified default value.

After applying the normalization and discretization feature construction operators to the running example, the "Employee" concept contains the additional attributes "normalized commision" and "commission category". We will try to predict the employee's commission from the learning attributes "age", "salary", "department", and "hire date" with either decision rules or a decision tree. Thus, either the PredictionWithDecisionRules or the PredictionWithDecisionTree operator is applied next to learn the "commission category" target attribute.

Suppose the user specifies five loops for either operator. As input concept the user select the "Employee" concept. This parameter, by definition, stays the same in all loops. In each of the five loops the user has to select the "commission category" as target attribute, the attributes "age", "salary", "department", and "hire date" as predicting attributes and has to use the same integer value as sample size. The user varies the pruning confidence level parameter stepwise in each loop, say from 60% in the first loop to 100% in the fifth loop, and specifies a separate output attribute for each loop, say "predicted commission category 60", "predicted commission category 70", ...

The model that the decision rules and decision tree operators learn will become more specific, with respect to the training data, with an increase in the pruning confidence level. We may thus expect that the model will generalize more, that is, neglect unusual values and potential errors more often, when the pruning confidence level is low. The user has to manually examine the outcome of the prediction operators and determine the best learning model. The attribute for the lowest pruning confidence level will typically misclassify distinct commission category values as one category. The attribute for the highest pruning confidence level will typically do the opposite, fail to classify erroneously distinct commission category values as one category.

Chapter 7

Normalization

7.1 Problem Description

A variable is normalized by scaling its values within a small specified range (for example the interval 0 to 1). It is convenient and sometimes necessary to normalize the range of a variable. It makes comparing variables easier and some modeling tools require variables to be normalized. Other tools that do not have this requirement may still benefit considerably from using normalized variables. Normalization is particularly useful for classification algorithms involving neural networks, or distance measurements such as nearest neighbor classification and clustering. It can speed up the learning phase of neural networks and it helps prevent variables with initially large ranges from outweighing variables with initially smaller ranges in distance-based methods.

In general it makes sense to normalize attributes before applying a learning operator. Examples have been described for missing values (Chapter 4) and noise (Chapter 6).

The Mining Mart system currently provides two scaling operators: a linear scaling operator and a logarithmic scaling operator.

7.2 Linear Scaling

The linear scaling operator simply scales a specified attribute to lie between *RangeMin* and *RangeMax*. Often used values are 0 or -1 for *RangeMin* and 1 for *RangeMax*. In order to scale the attribute the minimum and maximum values are found for the attribute. It is a loopable operator and therefore multiple attributes can be scaled with this operator in one step.

7.3 Logarithmic Scaling

The logarithmic scaling operator scales an attribute by taking the logarithm of the value with *logBase* as base value. If an attribute varies exponentially with another attribute then this scaling operator can be useful. For example many processes in nature depend exponentially on time like the growth in time of an initially small colony of bacteria. After applying logarithmic scaling the linear scaling operator may be applied again in order to restrict values for example to the [0, 1] range.

Chapter 8

Tips for Using the Mining Mart System

In this chapter we give some tips for early adopters for using the Mining Mart system. These tips mainly concern the underexposed aspects of the Mining Mart system: the meta data, the concept hierarchy and the reuse possibilities.

Although the software has become more stable in the last months it should still be classified as a prototype. Errors may occur when the program is being used. Also some functionality could be made more user-friendly. The tips mentioned here are meant to enable early users to use the system more conveniently and more effectively. As development of the software will still continue at least for a short time after delivery of this report (but hopefully for much longer!), some of the remarks made in this chapter may have become obsolete at the time of reading.

8.1 Keeping Operators Nicely Listed

It is likely that operators will be ordered in a case. The user inserts operators, edits properties and defines connections between the operators. There are two views that show operators (see Figure 2.1 on page 10): one listing the operator names in alphabetical order and another showing a graphical representation of operators and their connections.

We have found it convenient to have the operators listed in the order that the operators should be applied. As the operators are ordered alphabetically this means using a naming strategy. We have used a simple naming strategy starting operator names with numbers (01, 02, 03, ...) for the first, second, third and following operators (see again Figure 2.1). In this way when a case is opened operators are ordered like we would want to see them. More elaborate naming strategies are possible of course, but we already found this simple one to be useful.

8.2 Keeping Track of Concepts

Operators have a concept as output or have an attribute as output. We used a naming strategy to keep track of the output concepts and attributes. Concepts and attributes can be viewed in the Concept Editor (see again Figure 2.1). In order to be able to relate a certain concept to a certain operator it is convenient to have the operator number (see Section 8.1) and an abbreviation for the operator type in the concept name. For example, the output of a RowSelection operator with number 06 and input concept *Sales* is named *Sales_06_RS*. As concept names are also ordered alphabetically in the Concept Editor this will nicely order all different Sales concepts. The same can be done for attributes that are output of an operator.

8.3 Reusing Cases

Having a meta data repository of cases, an interesting option is of course to reuse a case. One can reuse a case by first exporting it and then importing it again with a different name. The import/export functionality also allows to share cases with other users. This import/export functionality was a major requirement for the Mining Mart system to create a "market place" of cases.

8.4 Reusing Concepts

Concepts are important in cases as they form the bridge to the business data that is to be preprocessed. If business data should be used in several cases, it may be convenient to create a separate concept repository. A concept repository can be created as a case that only contains concepts and no operators. The concept repository case may be the first case a user would want to create. Once a concept repository case has been defined its concepts can be used in other cases. To use a concept from the concept repository in the current case choose the menu item `File\Import\Concepts from case...`. It allows to specify the case, concepts and whether the columnsets and columns should also be imported. In this way new cases, where the main focus is on operators, can be created quicker and more efficient.

Reuse of concepts can also be stimulated by using abstraction. Thinking in concept hierarchies and defining general concepts at the highest level makes concepts more reusable. Examples of high level concepts are: customer, product, price, transaction, region, and profession. Sub concepts for product are, for example, loan, insurance policy, TV, mobile phone, and Internet access. Sub concepts for customer are for example: high potential customer, medium potential customer, and low potential customer. Concept hierarchies are often strongly connected to a domain (for example finance, telecommunications, biology, IT services, and sports). Therefore initially it

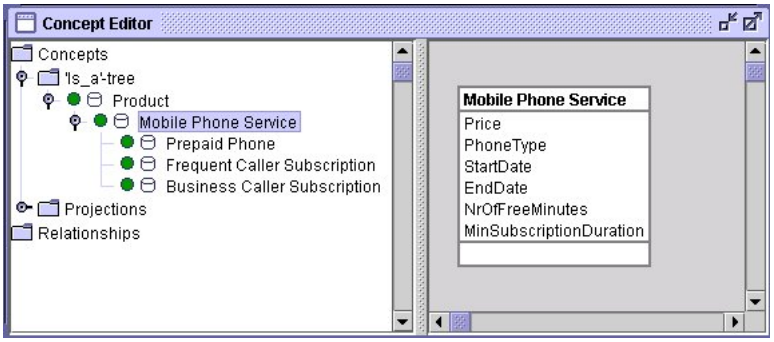


Figure 8.1: Example of a concept hierarchy.

makes sense to start building concept hierarchies that are domain specific. The Mining Mart system supports construction of concept hierarchies. The hierarchy is shown in the Concept Editor as a tree of concepts (see Figure 8.1). Also relationships can be defined in a hierarchy. Only abstract concepts (type BASE) and database concepts (type DB) can be placed in a concept hierarchy by the user. Concepts of type MINING result from a preprocessing step and therefore cannot be put into a concept hierarchy by the user.

8.5 Building a Chain of Operators

Creating a valid chain of operators is an inherently complex task. Not so much from a user standpoint (it is quite easy to specify operators and input and outputs), but from a process standpoint. The HCI is still a bit weak, however, in handling changes with operators. When the input concept or other parameters are changed the operator may become "stuck". The state of the operator is not correct anymore and compilation of the step will fail. The situation is worst in chains. Operators in the end of the chain respond badly to changes made in the start of the chain. Therefore we suggest to use a very cautious approach when creating cases.

Before starting to create cases one should get experience with the system by trying out the various operators and concepts and see how they work (the Mining Mart system also provides some helpful online information about operators). When starting on a real case one should make an effort of thinking out the case in order to do it right the first time. If one has a clear idea what operators should be applied and what concepts should be used, lesser changes will be needed during the process. Then iteratively follow a step-by-step approach:

- Insert operator

- Specify input and output
- Compile step
- Evaluate result
- Insert next operator

In general when one needs to change the input concept of an operator it is better to delete the operator and create a new one (with a new name) instead of modifying the existing operator.

8.6 General Problem Solving

8.6.1 Compilation Issues

Sometimes compilation of an operator may fail and an error code is produced. There are basically a few main reasons why compilation may fail:

- The meta data is giving problems.
- The compiler has a problem with one (or more) of the input and output parameters.
- The HCI failed to correctly store the input and output parameters for the operator.
- The compiler is connected to the wrong meta data schema.
- The operator is not available for the user's platform.

Meta data problem. Sometimes compilation may fail because the compiler finds something in the meta data it cannot deal with. The error message given by the compiler may then give a clue where the problem lies. By close inspection of the data stored in the M4 schema such problems sometimes can be resolved. At various points the HCI does check input in order to prevent entering invalid data, but it's not full proof. For example, at one time we created a multicolumn feature consisting of two base attributes. Later we deleted one of the base attributes, forgetting that it was part of a multicolumn feature. This made the multicolumn feature invalid and caused compilation to fail for steps where the concept containing this multicolumn feature was used.

Input and output parameter problem. We have seen with some operators that the compiler had trouble with handling nominal values. Although these cases were looked at and fixed, it cannot be assured now that all operators work correctly in this respect. When one encounters problems with nominal values, one should try to enter the parameters differently. For example, one could enter a nominal value *ANALYST* also as *'ANALYST'* (between single quotes).

Incorrect storage of meta data. We have also seen cases where the HCI did not correctly store parameter names. In this case a workaround may help to still be able to compile the operator. This involves checking and perhaps modifying the meta data directly in the schema. The STEP_T, PARAMETER_T and VALUE_T tables work together in defining operator inputs and outputs. Using the Mining Mart system online help about operators (choose the menu item `Help\Contents...`), one can check if the parameter names that have been stored are correct. If they are not correct, they should be replaced manually and one could recompile the operator.

Meta data schema. If one is using more instances of the M4 schema, it may be possible that errors are occurring because the compiler is connected to a different schema than the HCI. This can be checked by comparing the *db.config* files. The db.config file for the HCI is located in the installation directory for the HCI; the db.config file for the compiler is located in the path `$HOME$/etc` where HOME is the installation directory for the compiler.

User's platform. Although the goal is to have all operators available on Unix, Linux and Windows, some operators are currently only available for Unix (operators using decision trees, decision rules, KMeans and stochastic correlation measures). One should check the Mining Mart site[1] for the latest operators or see if it is possible to install the compiler on a Unix platform.

8.6.2 Unlocking a Locked Case

It may occur that the Mining Mart system reaches an error state, where it doesn't react anymore to the user. Terminating the system will cause the case to become locked when the user tries to open it the next time. The user will need to modify the M4 schema directly (using *sqlplus* for example) to unlock the case . The SQL command `DELETE FROM M4_ACCESS_T` will unlock all locked cases. To unlock a specific case with name CaseName use `DELETE FROM M4_ACCESS_T WHERE OBJECT_ID='CaseName'`.

[1] *http://www-ai.cs.uni-dortmund.de/FORSCHUNG/PROJEKTE/MININGMART/index.eng.html*

Chapter 9

Conclusions

We have found the Mining Mart system to be a valuable tool for preprocessing data. With its meta data model and HCI it is possible to conveniently create a repository of cases. Other important features are:

- A good level of platform independence (Oracle is, however, required and for some operators Unix is needed).
- Ability to handle large amounts of data.
- Learning operators applied in the preprocessing stage.
- Re-usability of cases and concepts.

A good variety of operators is now available in the system that makes the system useful for dealing with most of the common preprocessing activities. Just as this report was being finalized other important operators were added to the system: a general feature construction operator, various types of feature selection operators, two types of discretization operators and more learning operators. Unfortunately it was not possible for us to also include reviews of all these additions within this report. It shows nicely, however, that the Mining Mart system is still a very much evolving tool.

One of the important trends in data mining ([HaB02]) is the use of a visual language (for example UML) for modeling data mining tasks. The Mining Mart system fits this trend. The concept editor gives a static representation of the concepts a user wants to use and the relations between these concepts.

There are some areas where the Mining Mart system could be improved. One point of improvement is the stability of the platform. As we started working with operators we have encountered many minor and some major problems. Most of these have been solved along the way and we are glad we have been able to contribute to the maturation of the system in this way. Improvements in this field are still possible though.

The Mining Mart system gives basic data exploration support by providing views of the raw data and basic statistics. As data understanding is an important requirement before starting any data preparation some more support for exploring the data would be nice to have within the system (for example by providing histograms).

Other areas of improvement are being able to copy a series of operators, being able to copy concepts, being able to define an abstraction for a series of operators (Chain), more support in creating and maintaining a concept hierarchy, and a more extensive help. Some of these functionalities are expected to be added in the near future.

In this report we have provided information, ideas and experiences with respect to working with the Mining Mart system prototype. From the literature an extensive overview was made of general preprocessing problems (see Appendix A). Then for a selection of common preprocessing problems (data selection, missing values, time series, noise and normalization) we described our experiences in dealing with them using the Mining Mart system. We also described the application of combinations of operators in these sections to solve preprocessing problems. These combinations of operators can be seen as "templates" or design patterns for solving specific problems.

This report should, however, not be seen as the definitive guide on "best practices using the Mining Mart system." The Mining Mart system is still evolving and functionality is still being added. More and more users are now building up experience with the system and will continue to do so in the future. Therefore there will be certainly room in the future for a followup on this report. From our experiences with the current Mining Mart system we can say that we expect that early adopters will already benefit from this system and that future users will do so even more.

Acknowledgements

This work has partially been funded by the European Commission, IST-1999-11993(MiningMart).

Bibliography

[AdZ96] Adriaans, P., Zantinge, D., *Data Mining*, Addison-Wesley, Harlow, 1996.

[Eul02] Euler, T., *Compiler Constraints and Operator Parameters*, Technical report TR 12-02, October 30, 2002.

[HaB02] de Haas, M., Brandt, N., "Relational Data Mining", in J. Meij (ed), *Dealing with the data flood: Mining data, text and multimedia*, STT publication 65, Stichting Toekomstbeeld der Techniek, Den Haag, 2002.

[HaK01] Han, J., Kamber, M., *Data Mining: Concepts and Techniques*, Morgan Kaufmann Publishers, San Francisco, California, 2001.

[KZV00] Kietz, J.-U., Zücker, R., Vaduva, A., "Mining Mart: Combining case-based reasoning and multistrategy learning into a framework for reusing kdd-applications", in *Proc. of the Int. Conf. on Multi-Strategy Learning, MSL-2000*, 2000.

[LaR02] Laverman, B., Rem, O., *Description of the M4 Interface used by the HCI of WP12*, Deliverable D12.2, July 16, 2002.

[MoS02] Morik, K., Scholz, M., *The MiningMart Approach.*, Workshop Management des Wandels der 32. GI Jahrestagung, 2002, to appear.

[PeF02] Perot Systems Netherlands, Fraunhofer Institute AiS, *Mining Mart Human Computer Interface*, Technical report TR 12-01, March 15, 2002.

[Pyl99] Pyle,D., *Data Preparation for Data Mining*, Morgan Kaufmann Publishers, San Francisco, California, 1999.

[VKZ01] Vaduva, A., Kietz, J.-U., Zücker, R., Dittrich, K., Morik, K., Marco, B., Luigi, P., *M4 - The MiningMart Meta Model.*, Deliverable, D8/9, IST Project MiningMart, IST-11993, 2001.

[Zu01a] Zücker, R., *Description of the M4-Relational Metadata-Schema within the Database.*, Deliverable D7a, IST Project MiningMart, IST-11993, 2001.

[Zu01b] Zücker, R., *Description of the Metadata-Compiler using the M4-Relational Metadata-Schema.*, Deliverable, D7b, IST Project MiningMart, IST-11993, 2001.

[ZKV01] Zücker, R., Kietz, J.-U., Vaduva, A., "MiningMart: Metadata-Driven Preprocessing.", in *Proceedings of the ECML/PKDD Workshop on Database Support for KDD*, 2001.

Appendix A

General Preprocessing Problems

This chapter briefly describes various preprocessing operations. Its purpose is to provide a general overview of preprocessing problems. The descriptions and examples have been freely taken from [AdZ96], [HaK01] and [Pyl99]. The purpose of data preprocessing is to transform data sets such that their information is best exposed to the mining tool ([Pyl99]). Preprocessing operations can be categorized into the following stages:

- Data selection
- Data cleaning
- Data integration and enrichment
- Data transformation and coding

The following sections describe each preprocessing stage in turn. A separate section is devoted to displacement series (such as time series). Often special attention has to be given when preprocessing series, due to the enclosed information in the ordering of the data.

A.1 Data Selection

A.1.1 Getting Enough Data

Description

Measured values should reflect some entity in the real world. Often it is impossible to measure all possible values of an entity and therefore the data that has been captured must be seen as a sample of all values occurring in the real world. The question then should be: has enough data been measured in order to properly reflect the variable?

It is also common in data mining to use only a sample of the available data for building a model. In Mining Mart samples are also used in the preprocessing stage when applying learning operators. In these cases again the question should be answered if the sample is representative enough for the variable.

A certain confidence that indeed the variability has been captured to a certain extent can be obtained by looking at the change in variance with increasing sample size. The variance measure can be used for numeric attributes. For nominal attributes the change in relative occurrence of values with increasing sample size plays a role in determining confidence.

A.1.2 Reducing Depth

Description

The data set is presumed to be in a table format. Reducing depth means reducing the number of rows. It is not always needed to use all of the data all of the time. It may be more convenient to work only with part of the data. If a selection of the data is taken, then it must be assured that the sample still reflects all of the relationships that are present in the full data set.

A.1.3 Reducing Width

Description

The data set is presumed to be in a table format. Reducing width means reducing the number of variables (columns). More variables means more information, but too many variables may bring any mining algorithm to its knees.

Every variable in the data set should be inspected closely. A conclusion of this inspection might be that the variable needs to be removed. There are various reasons why a variable should be removed. Variables with no information content should be removed. These are for example variables with only missing values or with only one value. Also duplication of information in variables can be a reason for removing variables. A variable may be redundant if it can be derived from another variable. Some redundancies can be detected by correlation analysis. Such analyses measure how strongly one variable implies the other, based on the available data.

Another problem is highly sparse data. Sparse data is difficult to mine. It should be decided if a sparse variable should be removed or be processed in some way.

Mining on the reduced data set should be more efficient, yet produce (almost) the same analytical results. Typically (heuristic) variable subset selection methods are used to find the minimum set of variables such that

the resulting probability distribution of the data classes is as close as possible to the original distribution obtained using all variables.

Example

Consider one database that contains a variable "customer_id" and another database that contains a variable "cust_number". Correlation analysis can detect whether the variables are highly correlated.

Consider another example where a client database contains both the client names and client number. The client's name provides the same information as their client number. Moreover we want to identify certain types of client and are not interested in their particular names or client numbers. Therefore the client's name and client number are removed from the sample.

If the task is to classify customers as to whether they are likely to purchase a popular new CD when notified of a sale, variables such as the customer's telephone number are likely to be irrelevant, unlike variables such as "age" or "music_taste".

A.2 Data Cleaning

A.2.1 Data Pollution, Noisy Data

Description

When variables have values that they originally were not supposed to have one speaks of data pollution.

Operational data tend to be noisy: the data records contain errors and unusual values. There are many possible reasons for noisy data. Most mining routine have procedures for dealing with noisy data, but they are not always robust. Noisy data can often be smoothed by means of the following techniques: binning, clustering, combined computer and human inspection, or regression.

Often pollution is a result of people who try to stretch a system beyond its original intended functionality. This may be a result of a changing business environment while the data model remains fixed. Human resistance is another source of data pollution.

Example

Noisy data may be due to faulty data collection instruments that produced unusual values.

Another example is a gender field that is filled with "business" because a bank started to issue corporate credit cards.

Consider a transaction database recording the purchase dates of transactions. The database may contain records dated January 1st, 1901 (01-01-01),

although the company probably did not even exist at that time. In some databases, analysis shows an unexpected high number of people born on November 11th, 1911. When people are forced to fill in a birth date on a screen and they do not know it, they are inclined to type in "11-11-11", whereas it should be represented as being unknown instead.

A.2.2 De-duplication

Description

In a normal database several records may represent the same object. In many cases this duplication is the result of negligence, such as typing errors. There are also cases in which deliberately spelling errors are introduced with fraudulent objectives in mind. There is no automatic way to determine whether the records represent the same object. A de-duplication algorithm using pattern analysis techniques could, however, identify the situation and present it to the user to make a decision.

Example

Consider a Mr. Johnson and a Mr. Jonson in a client database. They have different client numbers but the same address. They could be the same person where one of the entries contains a spelling error.

A.2.3 Missing and Empty Values

Description

For empty values there is no corresponding real-world value. For missing values the underlying value has not been captured. There may be predictive or inferential information content in missing values.

Operational data tend to be incomplete: variables for various records have no recorded value. Incomplete data can occur from a number of reasons. Most mining routines have procedures for dealing with incomplete data, but they are not always robust. Various preprocessing methods exist to handle missing values, including, ignore the record and fill in the missing value manually, with a global constant, the variable mean, or the most probable value.

One should be careful, however, with replacing missing values with default values as they can damage the data set structure. In order to be able to use other values of the instance, it is best to replace the missing value with a value that least influences the data set structure.

Example

Some variables of interest, like customer income, may not always be available. Other data may not be considered important at the time of entry. Relevant data may not be included due to equipment malfunctioning. Data that were inconsistent with other data may have been deleted manually.

Typically in fraud detection lack of information is a valuable indication of interesting patterns.

Consider a client database that is enriched with demographic information on average income and car and house ownership for certain neighborhoods from an external party. Some clients may live in neighborhoods for which no demographic information is available. After a thorough analysis of the consequences, the purchase records for these clients are deleted.

A.2.4 Outliers

Description

Outliers concern variable values that lie some distance apart of the rest of the values. The question here is whether the value(s) are a mistake. If they are found to be a mistake, they can be treated as missing values; if not the variable can be remapped.

Example

Insurance data typically suffers considerably from the problem of outliers. Most insurance claims are small, but occasionally a claim will come in for a large sum. As this is not an error, it must be included in modeling. This may, however, distort the remaining data.

A.2.5 Anachronistic Variables

Description

An anachronistic variable is a variable containing information that is not available at the time prediction is needed. It is set after the value of the predicted variable is known. An indicator for such variables are models that have a near 100% prediction rate. During the initial data survey these kind of variables may be difficult to locate. If such a variable is found it should be removed.

Example

In a data set to predict who will take a specific type of bank account various predicting variables are selected. If “account number” is also included, this will be an anachronistic variable as it is only present after the account has been created.

A.3 Data Integration and Enrichment

A.3.1 Concurrency

Description

Different data streams may be captured at different times. When these streams are merged concurrency should be considered. Because of the difference in time certain relationships found in the data may not be valid.

Example

Merging demographic data which is one year old with current credit information may not produce a useful data set.

A.3.2 Data Consistency

Description

Different things may be represented by the same name in different systems and the same thing may be represented by different names in different systems. This can lead to problems when combining data from different systems. This is also referred to as the entity identification problem.

Example

A company's payroll system and personnel system both use the variable "employee", but each have their own notion of what an employee is. Asking the systems how many employees there are, may give two different answers.

One database may contain a variable "customer_id" and another database may contain a variable "cust_number". How can we be sure that both variables refer to the same entity?

A.3.3 Data Value Conflict

Description

Variable values from different sources may differ. This may be due to differences in representation, scaling or encoding.

Example

A weight variable may be stored in metric units in one system and British imperial units in another. The price of different hotels may involve not only different currencies, but also different services and taxes.

A.4 Data Transformation and Coding

A.4.1 Reverse Pivoting

Description

In many cases some kind of transactions of customers are recorded. If the behavior of the customer needs to be modeled, then the information from the transactions table needs to be aggregated into derived fields representing customer activity. A reverse pivot is done from the transactions table to the customer table.

Example

When a customer asks for a loan at a bank, the bank would like to estimate if the customer will pay back the loan by examining the account transactions of the customer. By comparing this customer behavior with behaviors seen previously for other customers, it can be predicted if the customer will pay back the loan without problems or not. Reverse pivoting extracts the customer behavior from the transactions of the customer.

A.4.2 Monotonic Variables

Description

Monotonic variables are variables that increase without an upper bound. The problem is that as soon as actual data for modeling is used from another source, the monotonic variable will very likely soon take on values outside the range sampled. Even if the data is within the sampled range the distribution of that data will almost certainly be totally different. This may lead to invalid results when using a previously made model. A transformation is needed to a non-monotonic variable for the data to be useful. There are ways to automatically give a good estimate if a variable is monotonic or not.

Example

Examples of monotonic variables are: date, social security id and employee id. By transforming date to season, it is transformed to an non-monotonic variable.

A.4.3 Remapping Nominal Values

Description

Nominal values most often should be transformed to numerical values, because many modeling techniques are bad in dealing with nominal variables.

Nominal values should not naively be substituted by assigning numbers. This naive substitution can introduce artificial patterns. One should use numbers that represent the "natural order" of the nominal values.

In addition, the organization of operational data is often not suited for data mining. The records provide one dimension to view the data. We often want to view the data along the dimension of a particular variable instead. The record orientation of the operational data is inefficient when one wants to find relationships between the records.

Remapping is useful when there should be no implication of ordering among the labels. Remapping (in particular flattening) also can transpose data and provides an overview of the values for a particular variable. The remapping techniques for nominals include one-of-n remapping, m-of-n remapping and remapping circular discontinuity.

In one-of-n remapping (also called flattening) a variable with cardinality n is replaced by n binary variables and only one of them is "on" (set to one) for every instance (other fields are set to zero).

In m-of-n remapping n new variables are introduced and more than one of these fields can be "on" for every instance.

Circular discontinuity is present in some time variables. Numbering the weeks of the year shows this problem. After week 52 week 1 follows. This may prevent a modeling tool from finding cyclic information. Remapping this variable makes it much easier for modeling tools to find cyclic patterns.

Example

Consider a variable representing the marital status, measured as: married, single, windowed, divorced, or never married. Simply assigning numeric values 1, 2, 3, 4, 5 to these nominal values is totally destructive of the natural structure of the data.

Consider a transaction database that records per transaction the purchase date, the client number, the client's age and income and item purchased. This operational database is transaction-oriented and not suited to give a characterization of the clients that purchase particular combinations of items. The purchase date is considered to be irrelevant for this characterization and ignored. We replace the variable that represents the items purchased, by one binary variable for each possible value of that variable. If a client has purchased the item the corresponding variable is given the value 1 and 0 otherwise. After this transformation the sample contains one record for each client, describing the items purchased.

An example of m-of-n is where a grocery product name variable is replaced by a few variables describing common characteristics like "Fruit", "Vegetable", "Leafy" or "Root crop".

A.4.4 Conversion of Binary Variables to 0-1

Description

In data mining applications it is sometimes easier to code binary variables into one bit. Conversion of variables that take one of two values into 0 or 1 facilitates an efficient execution of pattern recognition algorithms.

Example

Consider variables car and house that represent ownership. The two variables take value "yes" for car owners and house owners respectively and "no" otherwise. We can convert these variables to the values 0 and 1 instead of "no" and "yes".

A.4.5 Aggregation

Description

Data from operational databases often need to be transformed or consolidated into a form that is appropriate for mining. Summary or aggregation operations are applied to the data. Aggregation in addition reduces the data set, which enables more efficient processing yet produces (almost) the same analytical results.

Example

Consider a data set of quarterly sales for the years 1997 to 1999. You are interested in annual sales, rather than totals per quarter. Thus the data can be aggregated so that the resulting data summarize the total sales per year instead of per quarter. The resulting data set is smaller in volume, without loss of information necessary for the analysis task.

A.4.6 Data Enhancement

Description

Feature construction is a way of adding domain knowledge and enhancing the data. It can speed up creation of better and more understandable models. It can also avoid "feature swamping". Feature construction adds new variables to a given set of variables to help the mining process. New variables are constructed from the given variables and added in order to help improve the accuracy and the understanding of the structure in high dimensional data. By combining variables, feature construction can discover missing information about the relationships between data variables that can be useful for knowledge discovery.

Example

In trying to predict stock market performance, it is believed that the trend of the market is important. If this is indeed the case, the modeling tool will develop a "trend detector". The trend might be determined by comparing the mean of the last three days' closing prices with the previous three days. Thus, the trend can be "up", "down" or "flat". By providing the trend variable the modeling tool does not have to learn addition, subtraction and division, which will save considerable time.

Another example is the modeling of the value of share prices. The data contains information about the price and the earnings of shares. It is believed that the price per earning ratio has predictive value and therefore this ratio is also added to the data.

A customer subscription date can be transformed in month numbers starting from a certain date. This coding enables us to find patterns in time series of customer transactions. The subscription date can also be transformed in seasonal codes to try to find patterns in seasonal influence on customer behavior.

A.4.7 Generalization

Description

The information in operational data is often much too detailed for pattern recognition algorithms. The information has to be transformed into course-grained codes. The way in which information is coded greatly determines the type of patterns found.

Data generalization is a process that abstracts a large set of task-relevant data in a database from a relatively low conceptual level to higher conceptual levels. One method for the efficient and flexible generalization of large data sets is the variable-oriented induction approach. The general idea of variable-oriented induction is to first collect the task-relevant data and then perform generalization based on the number of distinct values of each variable. The generalization is performed by either variable removal or variable generalization.

Example

The removal of a variable eliminates a constraint and thus generalizes the relation. Consider the variable "house_number", which has a large set of distinct values but no generalization. This variable should be removed because it cannot be generalized and contradicts the goal of generating concise rules. Consider another variable "street", with also a large set of distinct values, whose higher-level concepts are represented in terms of the other variables "city", "state", and "country". The variable "street" may be removed to

generalize the relation. The variable "birth_date" also has a large set of distinct values, but can be generalized to the concept "age" and further to concept "age_range". The variable "birth_date" should be generalized to, that is, replaced by, the concept "age_range".

A.4.8 Discretization and Concept Hierarchy Generation

Description

Raw data values for variables are replaced by ranges or higher conceptual levels to obtain a reduced representation of the data set that is much smaller in volume, yet maintains the integrity of the original data. Mining on the reduced data set should be more efficient, yet produce (almost) the same analytical results. Concept hierarchies allow the mining of data at multiple levels of abstraction and are a powerful tool for data mining. Discretization techniques are used to reduce the number of values for a continuous variable, by dividing the range of the variable into intervals. Concept hierarchies can be defined for both numeric and categorical data. A concept hierarchy for a numeric variable defines a discretization of the variable. Concept hierarchies also reduce the data by collecting and replacing low-level concepts by higher-level concepts.

Example

Low-level concepts such as numeric values for the variable age are replaced by higher-level concepts such as young, middle-aged and senior. Consider a numeric variable "price" whose values range from $0 to $1000. A concept hierarchy for the variable "price" could at the highest-level start with the concept "from $0 to $1000". At a lower level the concepts "from $0 to $200", "from $200 to $400", "from $400 to $600", "from $600 to $800" and "from $800 to $1000" appear. At the next lower levels the concepts "from $0 - $100", "from $100 - $200", etcetera are defined. The variables "street", "country", "city" and "state" are concepts in the dimension "location". The concept hierarchy for "location" can be automatically generated by sorting the variables in ascending order based on the number of distinct values in each variable.

A.4.9 Normalization

Description

It is convenient to normalize the range of a variable. It makes comparing variables easier and some modeling tools require variables to be normalized. A variable is normalized by scaling its values so that they fall within a small specified range. Normalization is particularly useful for classification algorithms involving neural networks, or distance measurements such

as nearest neighbor classification and clustering. Normalized input values speed up the learning phase of neural networks. Normalization helps prevent variables with initially large ranges from outweighing variables with initially smaller ranges in distance-based methods. Other modeling tools do not require variables to be normalized, but may benefit considerably from it.

Another form of normalization is normalizing the distribution of variable values. The distribution of variable values over its range may cause problems for modeling tools. Many modeling tools have difficulty with varying density within a distribution. Even tools that are able to handle irregular distributions may benefit from regularizing (balancing) a distribution.

Example

Consider a client database containing birth dates, income and credit. The birth dates may be coded as age classes with an interval of 10 years, yielding the values 1 to 10. After dividing the income and credit by 1000, most customers will have an income class and credit class between 10 and 100. Comparing the age, income and credit information is now easier, because the numbers are close to each other.

Consider a variable "income" with minimum and maximum values \$12,000 and \$98,000, respectively, and mean and standard deviation \$54,000 and \$16,000, respectively. Suppose that we would like to map "income" to the range [0.0, 1.0] by means of the min-max normalization. The min-max normalization will transform value \$73,600 to $(73,600 - 12,000)/(98,000 - 12,000) \times (1.0 - 0.0) + 0.0 = 0.716$. The zero-mean normalization uses the mean and standard deviation to transform value \$73,600 to $(73,600 - 54,000)/16,000 = 1.225$.

A.4.10 Data Compression

Description

Data compression is used to reduce the amount of data. It can be useful when the amount of data that is available is too large, thus, making complex analysis impractical or infeasible. Encoding mechanisms are used to obtain a reduced representation of the data set that is much smaller in volume, yet maintains the integrity of the original data. Mining on the reduced data set should be more efficient, yet produce (almost) the same analytical results. Typically (heuristic) variable subset selection methods are used to find the minimum set of variables such that the resulting probability distribution of the data classes is as close as possible to the original distribution obtained using all variables.

Example

The discrete wavelet transform is a linear signal processing technique that, when applied to a data vector transforms it to a numerically different vector of wavelet coefficients. The wavelet transformed data can be truncated. Storing only a small fraction of the strongest of the wavelet coefficients retains a compressed approximation of the data. For example, all wavelet coefficients larger than some user-specified threshold can be retained. The remaining coefficient is set to 0. The resulting data representation is very sparse, so that operations that can take advantage of data sparsity are computationally very fast.

A.4.11 Numerosity Reduction

Description

Numerosity reduction is also used to reduce data when the data set available for analysis is impractically large. The original data are replaced by a model. Typically the model or the parameters of the model and outliers are stored instead of the actual data. The model is used to estimate the data.

Example

Regression models can be used to estimate the original data. In linear regression the data are modeled to fit a straight line. One variable is modeled as a linear function of another variable.

A.5 Displacement Series

Series variables always are at least two-dimensional, although one of the dimensions may be implicit. Series data enfolds their information in the ordering of the data. Preserving the ordering is the main reason that series data has to be prepared differently from non-series data.

Any series shape can be thought of as being constructed from simple wave forms, each of a separate single frequency. The most common type of series variable is a time series, in which a series of values of some event are recorded over a period of time. The issues and techniques described about time series also apply to any other displacement series.

A form of time series analysis known as classical decomposition looks at the series as being built from four separate components: trend, seasonality, cycles, and noise. Trend moves in a consistent direction, monotonically increasing or decreasing. Seasonality reflects the insight that certain seasons are inherently different. Cycles are fluctuations in the level of the series that have some identifiable repetitive form and structure. Noise is the component that is left after the trend, cyclic and seasonal components have

been extracted. Noise can be generated from a variety of sources. A noise source may have a characteristic signature. Noise sources can sometimes be identified.

A.5.1 Missing Values

Description

Filling in the holes with self-similar patterns from other parts of the series reinforces the self-similarity. So replacing missing values necessarily enhances a pattern. One way to ameliorate the problem to some extent is to add noise to the replacement values.

A.5.2 Outliers

Description

Outliers in displacement series come as individual occurrences and as clusters of consecutive values. The miner will need to ask hard questions about why the outliers exists. If no rationale can account for the outliers, replace the outliers exactly as for missing values.

A.5.3 Non-uniform Displacement

Description

Many of the analytical techniques assume that measurements are taken at regular increments. If the displacement is not constant, the values must be adjusted to reflect what they would have been had they been taken with a uniform displacement. Treating the non-uniform measured values as if they were sampled at uniform displacement intervals causes distortions. These distortions appear as noise and techniques for removing noise work well to estimate the original series.

A.5.4 Trend

Description

Leaving a trend present in a wave form causes problems for most modeling methods. Detrending a series is an absolute necessity.

Detrending has sever dangers. Incomplete series can indicate an artifact trend even though they have no trends. Detrending non-trended data can do enormous damage. Always the miner should ask if there is a reason to expect the existence of the discovered trend.

A.5.5 Attenuation

Description

Any series shape can be thought of as being constructed from simple wave forms, each of a separate single frequency. Attenuation reduces the amplitude of the lower-frequency wave form and thus leaves the higher frequencies more visible.

Removing trend corresponds to lower-frequency filtering at the lowest possible frequency. In addition to the zero frequency component, the components consisting of fractional frequencies are usually identified and removed from series. Some of the more common fractional frequency components include exponential growth curves, logistic function curves, logarithmic growth curves and power-law growth curves.

A.5.6 Moving Average

Description

Moving averages are used for general purpose filtering and well-suited to remove noise in series. Moving averages come in an enormous range and variety: simple moving average, weighted moving average, exponential moving average. The moving averages reduce the variance in the series. The longer the period of the average, the more the variance is reduced. A drawback of long weighting periods is that the average cannot begin to be calculated until the number of periods in the weighting has passed. Exponential moving average solves this delay problem.

In general an increase in the period for the simple moving average and weighted moving average or in the tail weight for the exponential moving average, makes the average react more slowly to the changes in the series. Slow changes correspond to longer wavelength, that is, lower frequencies. The simple moving average is the slowest to change of the averages. The weighted moving average moves similar to the simple moving average, but clearly responds more to the recent values. The exponential moving average is the most responsive to the actual series value.

The ability to effectively change the frequency at which the moving average reacts to the series makes them useful filters.

Example

Consider the series of ten positions in the table below. The table shows a lag-five simple moving average (SMA5), a lag-five weighted moving average (WMA5) and a lag-five exponential moving average (EMA5). The weighted moving average has the following weight distribution, which sums up to 1.0: 0.066, 0.132, 0.198, 0.264, 0.340. The exponential moving average uses head

and tail weight 0.576766 and 0.423234, respectively. The SMA5 value[1] is assigned to the center position in the weighting period. The WMA5 value[2] is centered one advanced on SMA5, because the weights favor the most recent values. The EMA5 value[3] tends to center round about the value of the SMA5. By definition the first EMA5 value is set to the initial value of the series. A series of length 10 is not sufficient to show the effect clearly.

Position	Value	SMA5	WMA5	EMA5	Head	Tail
1	0.1338			0.1338		
2	0.4622			0.3232	0.2666	0.0566
3	0.1448	0.2940		0.2203	0.0835	0.1368
4	0.6538	0.3168	0.2966	0.4703	0.3771	0.0932
5	0.0752	0.3067	0.2833	0.2424	0.0434	0.1991
6	0.2482	0.3497	0.3161	0.2458	0.1432	0.1026
7	0.4114	0.3751	0.3331	0.3413	0.2373	0.1040
8	0.3598	0.4673	0.4796	0.3519	0.2075	0.1444
9	0.7809		0.5303	0.5993	0.4504	0.1490
10	0.5362			0.5629	0.3092	0.2537

A.5.7 Smoothing

Description

Smoothing removes noise from the series. Peak-valley-mean (PVM) smoothing uses the mean of peaks and valleys as the estimate of the underlying waveform. The shortest possible peak-valley-mean, the lag-three PVM, covers three data points: last peak, last valley and estimated mean. Longer lags that take more peaks and valleys in consideration exist as well.

Median smoothing uses windows, a group of contiguous data points that are manipulated in some way. Median smoothing uses the median of the values in the window in place of the actual value. In many ways median smoothing is similar to average smoothing except that the median is used instead of the average. Using the median makes the smoothed value less sensitive to extremes in the window. A single extreme value will never appear in the median.

Resmoothing is a technique for smoothing the smoothed values. One form of resmoothing continues until there is no change in the resmoothed waveform. Other resmoothing techniques use a fixed number of resmooths, but vary the window size from smoothing to smoothing.

Hanning is a form of weighted averaging, which uses a lag-three. The three data points in the lag are multiplied by the weights 0.25, 0.5, and

[1] SMA value at position 3: $\frac{0.1338+0.4622+0.1448+0.6538+0.0752}{5} = 0.2940$

[2] WMA value at position 4: $0.1338 \times 0.066 + 0.4622 \times 0.132 + 0.1448 \times 0.198 + 0.6538 \times 0.264 + 0.0752 \times 0.340 = 0.2966$

[3] EMA value at position 2: $0.4622 \times 0.576766 + 0.1338 \times 0.42323 = 0.2666 + 0.0566 = 0.3232$

0.25, respectively. The hanning operation removes any final spikes left after smoothing or resmoothing.

Example

Consider the series of ten positions in the table below. The table shows a lag-three peak-valley-mean[4] (PVM), with its peaks and valleys, and a Median smoothing[5] with window of size five (MS5).

Position	Value	PVM	Peak	Valley	MS5
1	0.1338	0.1338	0.1338	0.1338	
2	0.4622	0.2980	0.4622	0.1338	
3	0.1448	0.3035	0.4622	0.1448	0.1448
4	0.6538	0.3993	0.6538	0.1448	0.2482
5	0.0752	0.3645	0.6538	0.0752	0.2482
6	0.2482	0.3645	0.6538	0.0752	0.3598
7	0.4114	0.2433	0.4114	0.0752	0.3598
8	0.3598	0.3856	0.4114	0.3598	0.4114
9	0.7809	0.5704	0.7809	0.3598	
10	0.5362	0.5704	0.7809	0.3598	

There are very many types of resmoothing. Two examples are "3R2H" and "4253H". The first is a median smooth with a window of three, repeated until no change in the waveform occurs; then a median smoothing with a window of two; then a hanning operation. The latter has four median smoothing operations with windows of four, two, five and three respectively, followed by a hanning operation.

A.5.8 Extraction

Description

Smoothing and filtering methods can be combined in numerous ways. All these methods intend to separate information in the waveform into its component parts. The extraction techniques separate the high and lower frequencies. Smoothing and filtering extract the lower frequencies. The remainder forms the higher frequency part and is found by subtracting the filtered waveform from the original waveform. When further extraction is made on either extracted waveform, this is called re-extraction.

[4]PMV value at position 2: $\frac{0.4622+0.1338}{2} = 0.2980$

[5]MS5 value at position 3: median of $\{0.1338, 0.4622, 0.1448, 0.6538, 0.0752\}$ = median of $\{0.0752, 0.1338, 0.1448, 0.4622, 0.6538\} = 0.1448$

A.5.9 Differencing

Description

The differencing method takes the difference between each value and some previous value, and analyzes the differences. A lag value determines exactly which previous value is used. The actual differences tend to appear noisy. Looking at the spectra and correlograms of the difference plots, however, reveals information. Differencing amplifies the higher frequencies and attenuates the lower frequencies. So differencing serves as a high-pass filter.

The difference of a composite waveform without noise contains little spectral energy at any of the frequencies. The correlogram shows a high correlation. A noise waveform shows high energy at high frequencies but the correlogram shows little correlation at any lag. Differencing a random walk shows low spectral energy at all frequencies, and the correlogram shows a rather high correlation.

A.5.10 Numerating Alpha Values

Description

Numeration of alpha values in a series presents some difficulties. On the rare occasions when alpha values occur, numerating them using the non-series techniques provides better than numeration without any rationale. Arbitrary assignment of values to alpha labels is always damaging. Non-series techniques, however, do not use the ordering information in the numeration. With time series in particular, it seems easier to find an appropriate rationale for numerating alpha values from a domain expert than for non-series data.

A.5.11 Distribution

Description

A series variable has a distribution that exists without reference to the ordering. The displacement variable can be redistributed in exactly the same manner as a non series variable. The rationale and methods for redistribution are similar for series and may be even more applicable in some ways. The distribution should be normalized after removing any trend from the series.

Redistribution removes all nonlinearity from the series. A curved waveform is translated into a linear representation with only straight lines. Redistribution goes a long way toward equalizing the variance.

Example

The exact shape of the waveform may be important to the modeling tool. The modeler is in a position to know for sure if this is the case at modeling time.

Appendix B

Mining Mart Operators

The usefulness of the Mining Mart system is greatly determined by the available operators. This chapter contains two lists of operator names and a series of screen shots, which give a quick impression of the available operators and the required input. The operators listed in Tables B.1 and B.2 are the currently available operators (November 2002). These lists of operators are expected to grow further in the future.

There are two kinds of operators, distinguished by their output on the conceptual level:

- Concept operators have an output concept, and
- Feature construction operators have an output attribute.

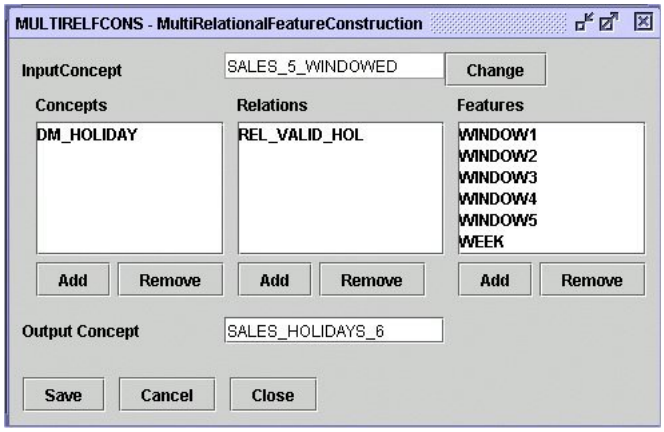
The next two sections discuss the concept and feature construction operators, respectively.

B.1 Concept Operators

This section lists the concept operators (Table B.1) and displays the corresponding screen shots (Figures B.1 through B.15). All concept operators take an input concept and create a new output concept. All concept operators contain the two fields *input concept* and *output concept* in their dialog box.

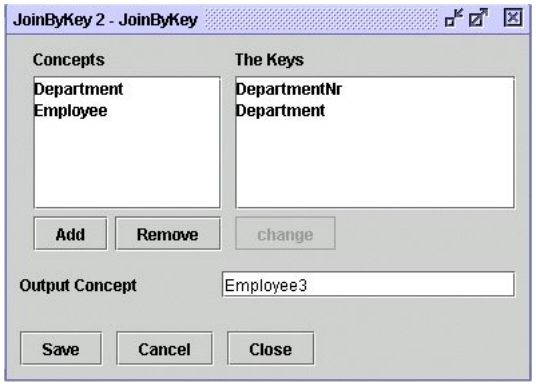
Name	Type	Short description	See
DelRecWMVal	Replace missing values	Deletes missing values	Fig. B.6
ExpMovFct	Time series operator	Smooths time series	Fig. B.14
JoinByKey	Data integration	Joins multiple concepts	Fig. B.2
MrfConstr	Feature construction	Select features from multiple concepts	Fig. B.1
RowSelByQuery	Row selection	Select rows by query	Fig. B.4
RowSelByRandom	Row selection	Select randomly a specified number of rows	Fig. B.5
SegmByPart	Segmentation	Segment data in specified number of partitions	Fig. B.8
SegmStrat	Segmentation	Segment data by specified attribute	Fig. B.7
SegmKMean	Segmentation	Segment data by KMeans	Fig. B.9
Sig2SymbProc	Time series operator	Time series abstraction operator	Fig. B.15
SimpleMovFct	Time series operator	Determines average value for each window	Fig. B.12
SpecStat	Statistics operator	Creates basic statistics for a specified concept	Fig. B.3
Unsegment	Segmentation	Unsegment previously segmented concepts	Fig. B.10
WeightMovFct	Time series operator	Determines weighted average value for each window	Fig. B.13
Windowing	Time series operator	Copies time related information into a number of attributes	Fig. B.11

Table B.1: Alphabetical list of concept operators



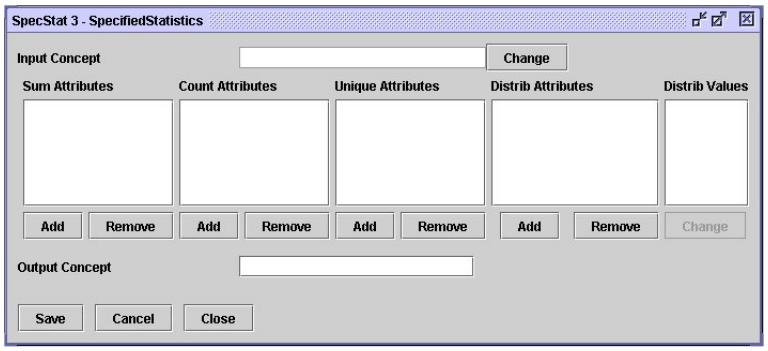
Creates a new concept with features that can be selected from the input concept and all concepts listed in the Concepts box. All these concepts should have a relation that connects the concept to the input concept.

Figure B.1: MultiRelationalFeatureConstruction dialog.



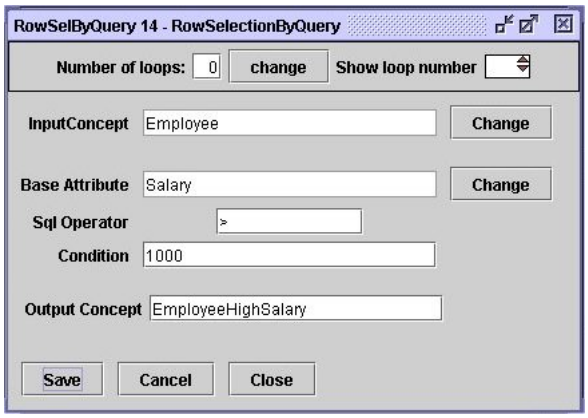
Joins two or more concepts using the specified keys.

Figure B.2: JoinByKey dialog.



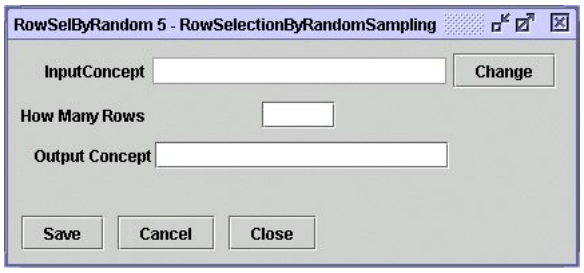
Creates an output concept with one row of information. For the respectively specified attributes the sum, count, number of unique values and distribution values are determined.

Figure B.3: SpecifiedStatistics dialog.



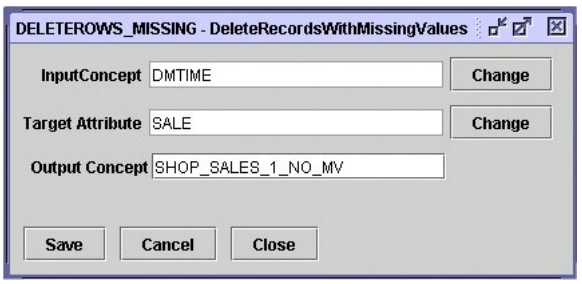
Selects records based on the query specified.

Figure B.4: RowSelectionByQuery dialog.



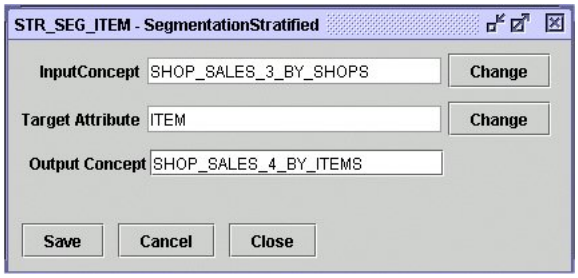
Selects randomly the specified number of records.

Figure B.5: RowSelectionByRandomSampling dialog.



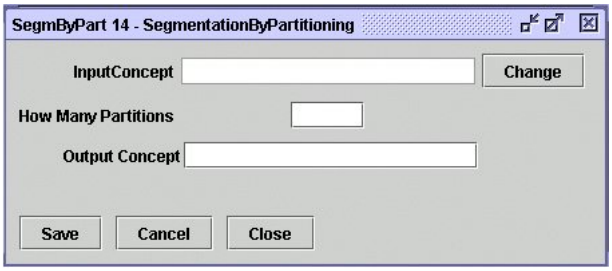
Deletes records with missing values. The records are not really deleted from the business data. Effectively a new view is created selecting records where the specified attribute is not null.

Figure B.6: DeleteRecordsWithMissingValues dialog.



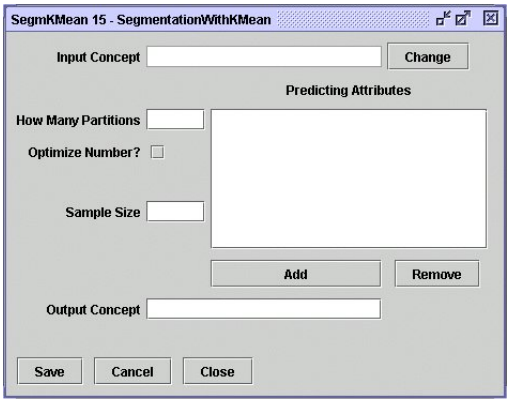
Divides data for a concept into segments. For every value of the specified target attribute a new segment is created. After this step a concept may have multiple columnsets: for each segment one.

Figure B.7: SegmentationStratified dialog.



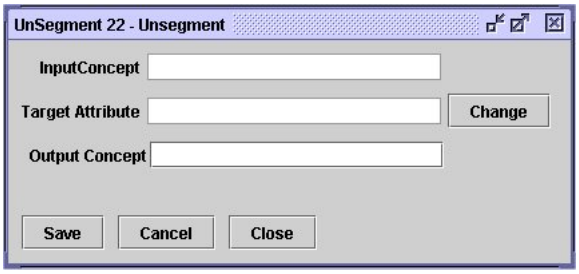
Randomly creates the specified number of segments. For every segment in the output concept a columnset is created.

Figure B.8: SegmentationByPartitioning dialog.



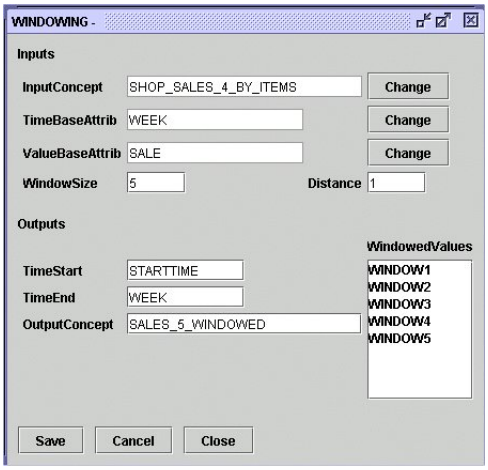
The input concept is segmented using the clustering algorithm KMeans. One can specify the number of records that are used to determine the clusters (sample size); the maximum number of partitions; the predicting attributes and if the algorithm should try to optimize the number of partitions.

Figure B.9: SegmentationWithKMeans dialog.



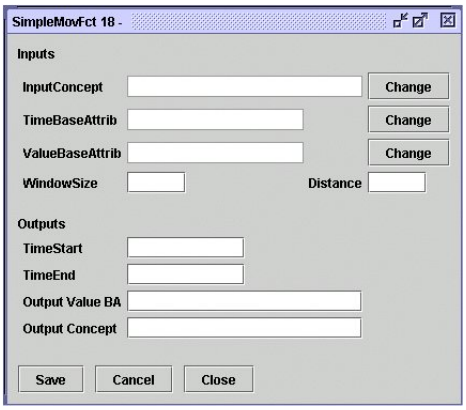
Unsegments data for the input concept for the specified attribute. This operator can only be specified if it is first placed in a chain of operators which contains at least one segmentation operator.

Figure B.10: Unsegment dialog.



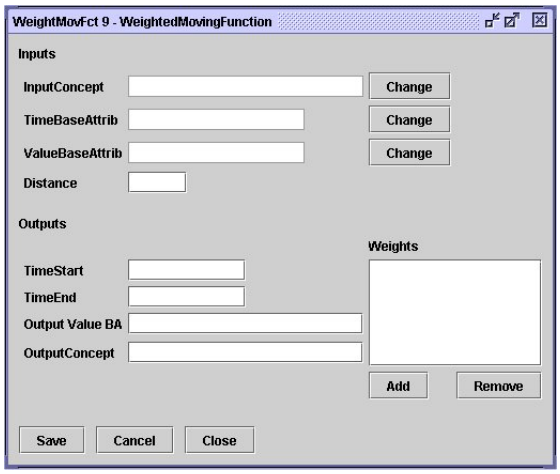
The Windowing operator is applicable to time series data. Windowing takes two attributes from the input concept: the time stamps and the values. Each row of the output concept gives a time window. Two time stamps define the beginning and ending of each time window. Further, there will be as many value attributes as specified by the window size and displacement distance.

Figure B.11: Windowing dialog.



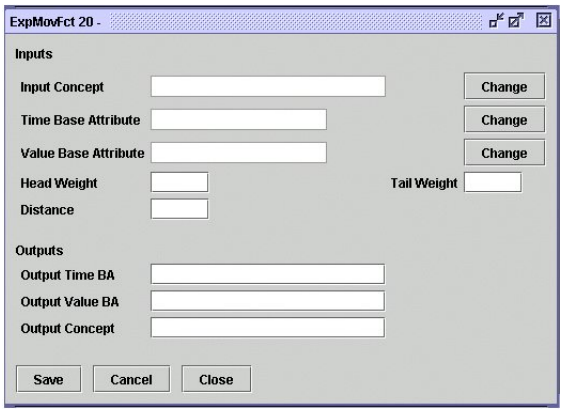
The SimpleMovingFunction operator combines windowing with the computation of the average values in each window. The average of the values in the window is stored in one output attribute. The user specifies the size of the window and the displacement distance in the window.

Figure B.12: SimpleMovingFunction dialog.



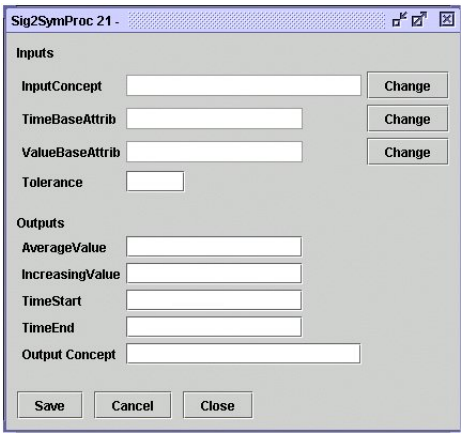
The WeightedMovingFunction operator combines windowing with the computation of the weighted average value for the window. The weighted average of the window is stored in one output attribute. The user specifies the the weights of the window values and the displacement distance in the window. The number of weights determines the size of the window. The weights must sum to 1.

Figure B.13: WeightedMovingFunction dialog.



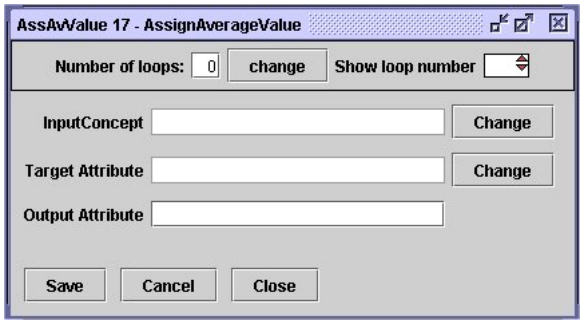
The ExponentialMovingFunction operator is a time series smoothing operator. The smoothed value for the window is stored in one output attribute. The distance determines the size of the window. The last value in the window is multiplied by the head weight. The first value is multiplied by the tail weight. The smoothed valued becomes the first value of the next window. The head and tail weights must sum to 1.

Figure B.14: ExponentialMovingFunction dialog.



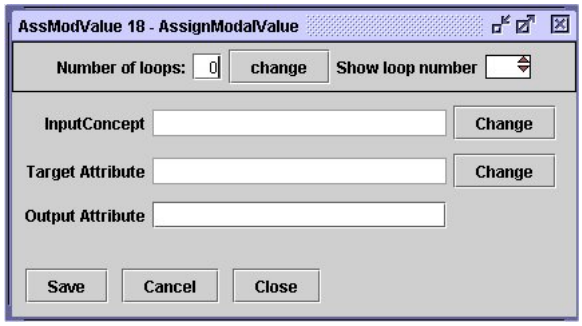
SignalToSymbolProcessing is a time series abstraction operator. Two output attributes form the bounds of the interval. The average of each interval is stored in one output attribute. The user specifies the average increase, and when the average increase, interpolated from the last interval, deviates thus that a new interval is created.

Figure B.15: SignalToSymbolProcessing dialog.



Replaces missing values with the average value for that column. Statistics for the concept should first have been calculated.

Figure B.16: AssignAverageValue dialog.



Replaces missing values with the modal value for that column. Statistics for the concept should first have been calculated.

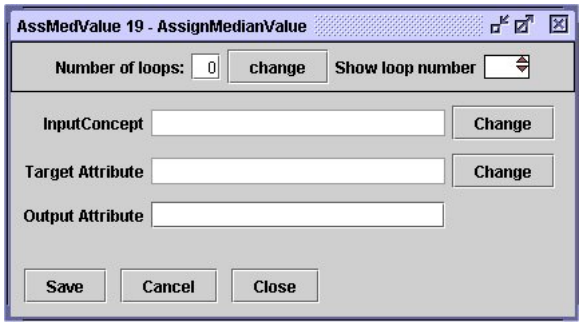
Figure B.17: AssignModalValue dialog.

B.2 Feature Construction Operators

This section lists the feature construction operators (Table B.2) and displays the corresponding screen shots (Figures B.16 through B.29). All feature construction operators are loopable. Looping means that the operator is applied to several target attributes (one after the other). The input concept remains the same in each loop, while the target attribute, the output attribute and further operator-specific parameters change from loop to loop.

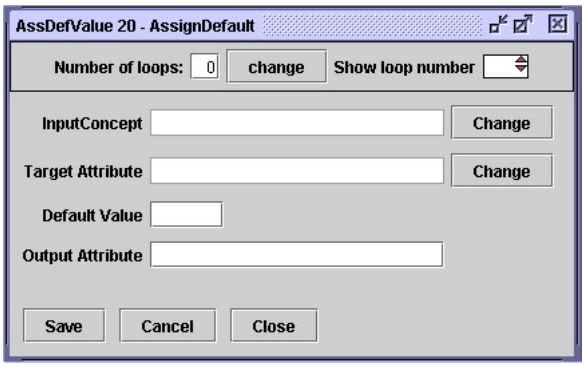
Name	Type	Short description	See
AssAvValue	Replace missing values	Replace missing values with average value	Fig. B.16
AssDefValue	Replace missing values	Replace missing values with default value	Fig. B.19
AssMedValue	Replace missing values	Replace missing values with median value	Fig. B.18
AssModValue	Replace missing values	Replace missing values with modal value	Fig. B.17
AssPredValue	Replace missing values	Replace missing values with a predicted value	Fig. B.29
CompSVMError	Evaluate learning operator	Evaluation operator for Support Vector Machine	Fig. B.26
LinScal	Scaling	Linear scaling of specified attributes	Fig. B.23
LogScal	Scaling	Logarithmic scaling of specified attributes	Fig. B.24
MvwRegSVM	Replace missing values	Replace missing values using a Support Vector Machine	Fig. B.22
MvwDecRules	Replace missing values	Replace missing values using a Decision Rules	Fig. B.21
MvwDecTree	Replace missing values	Replace missing values using a Decision Tree	Fig. B.20
PredWDecRules	Feature construction	Use Decision rules to add attribute that is predicted from specified attributes	Fig. B.27
PredWDecTree	Feature construction	Use Decision tree to add attribute that is predicted from specified attributes	Fig. B.28
SVMForRegr	Learning operator	Support Vector Machine for Regression	Fig. B.25

Table B.2: Alphabetical list of feature construction operators



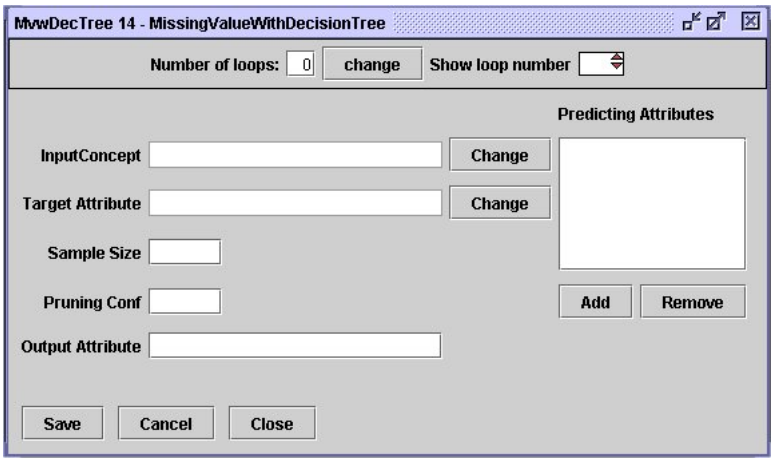
Replaces missing values with the median value for that column. Statistics for the concept should first have been calculated.

Figure B.18: AssignMedianValue dialog.



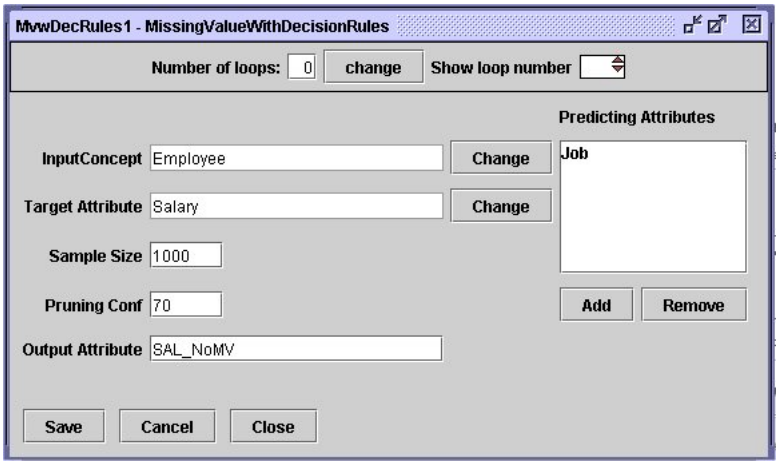
Replaces missing values with the specified value.

Figure B.19: AssignDefaultValue dialog.



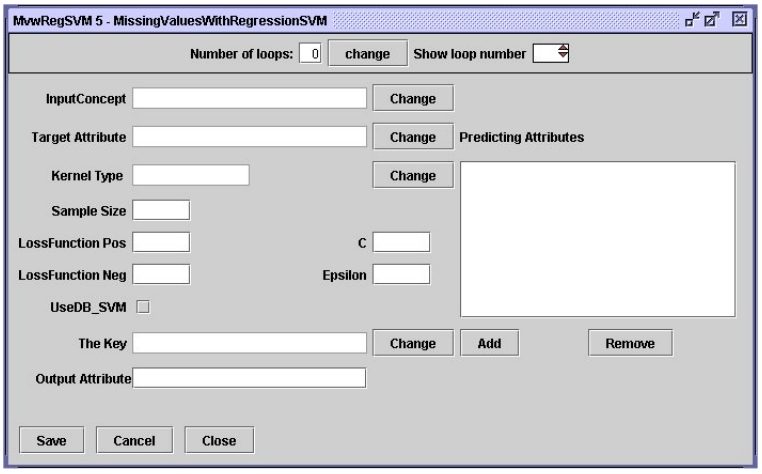
Replaces missing values with a predicted value using a Decision Tree. A Decision Tree is learned from the predicting attributes.

Figure B.20: MissingValuesWithDecisionTree dialog.



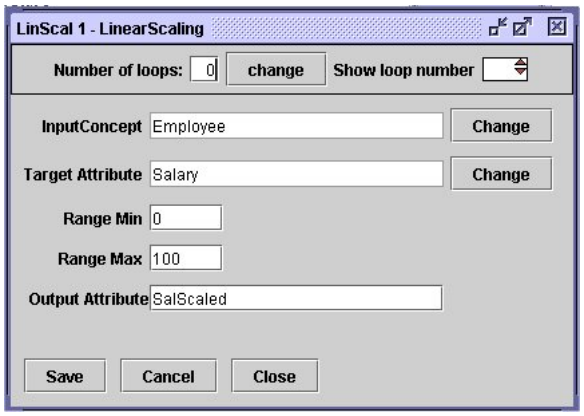
Replaces missing values with a predicted value using a set of Decision Rules. A set of Decision Rules is learned from the predicting attributes.

Figure B.21: MissingValueWithDecisionRules dialog.



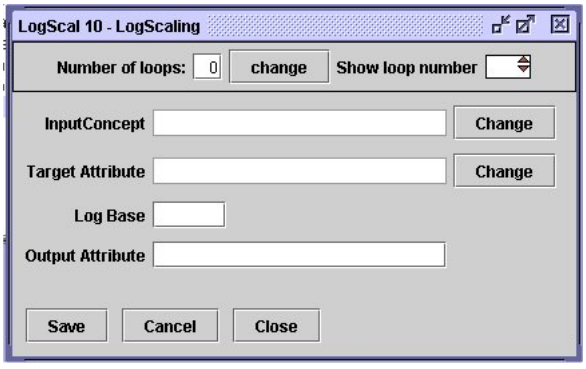
Replaces missing values with a predicted value using a Support Vector Machine. A Support Vector Machine is trained in regression mode from the predicting attributes.

Figure B.22: MissingValuesWithRegressionSVM dialog.



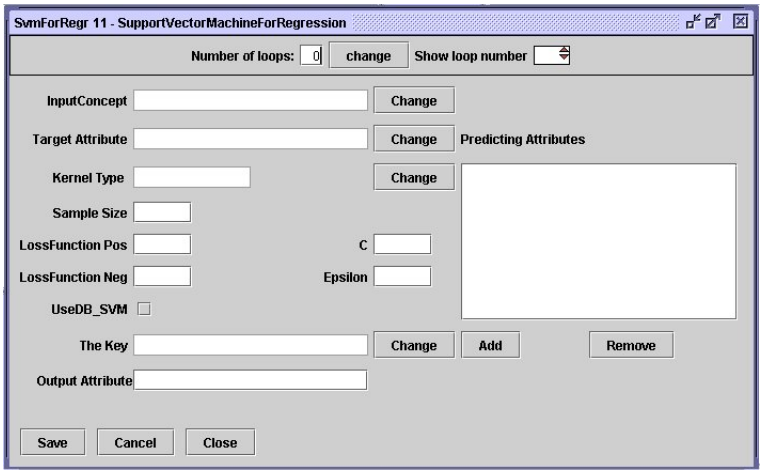
Scales the specified attribute between the minimum and the maximum value.

Figure B.23: LinearScaling dialog.



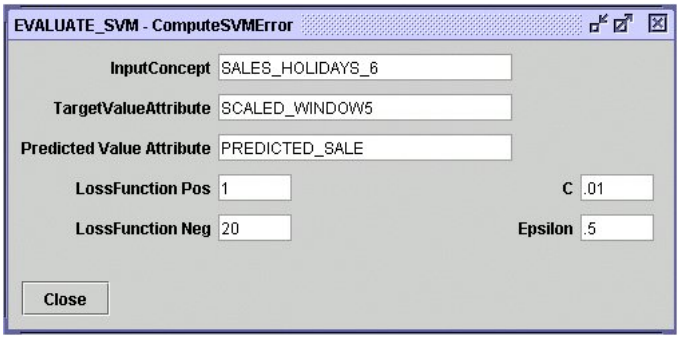
Determines the logarithm with the specified base for the specified attribute. Useful for attributes that show exponential behavior.

Figure B.24: LogScaling dialog.



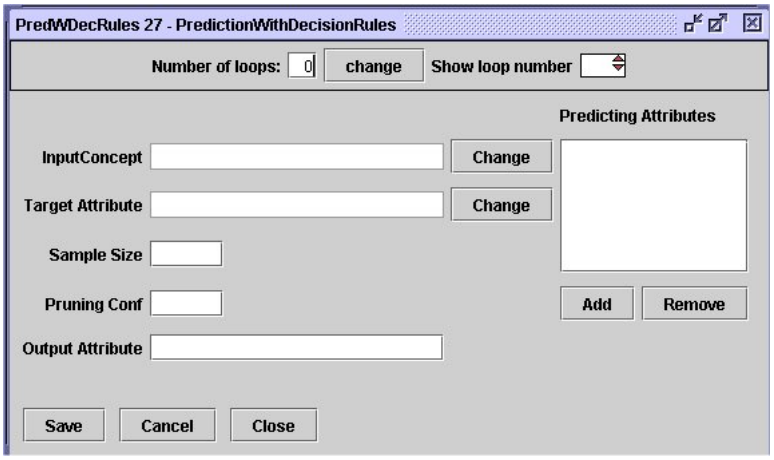
A data mining operator. A Support Vector Machine is trained in regression mode using the predicting attributes. It predicts the value for the output attribute. Several parameters can be set to manipulate the behavior of the operator.

Figure B.25: SVMForRegression dialog.



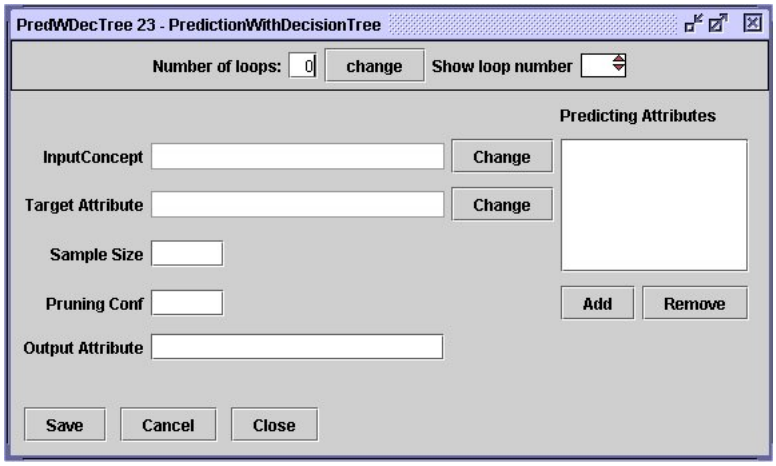
A special operator to evaluate the results obtained from the Support Vector Machine.

Figure B.26: ComputeSVMError dialog.



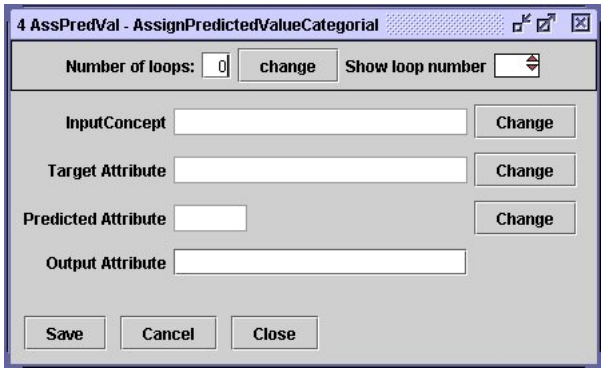
A data mining operator. A set of Decision Rules is learned from the predicting attributes. It predicts the value for the output attribute. Several parameters can be set to manipulate the behavior of the operator.

Figure B.27: PredictionWithDecisionRules dialog.



A data mining operator. A Decision Tree is learned from the predicting attributes. It predicts the value for the output attribute. Several parameters can be set to manipulate the behavior of the operator.

Figure B.28: PredictionWithDecisionTree dialog.



Replaces missing values in the target attribute with the value of the predicted attribute. The predicted attribute results from the PredictionWithDecisionRules or PredictionWithDecisionTree operator.

Figure B.29: AssignPredictedValueCategorial dialog.